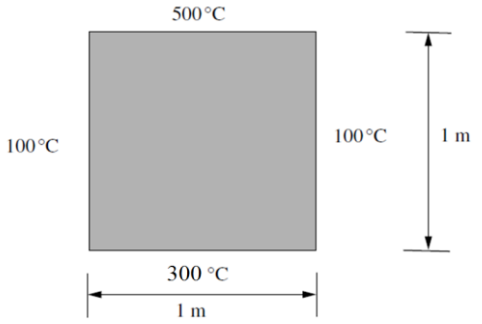
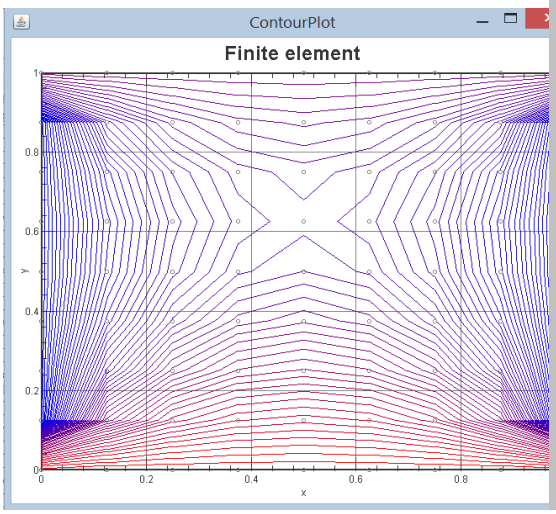
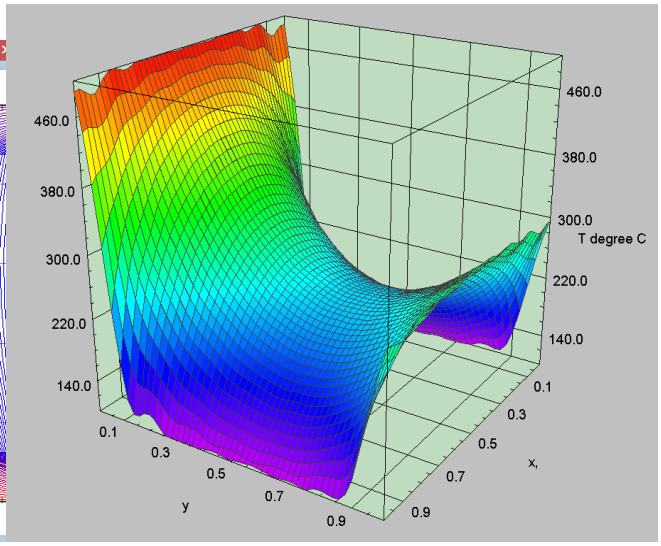
NUMERICAL HEAT TRANSFER



data output(veri çıktısı) :

0	1	2	3	4	5	6	7	8
500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0
100.0	296.517...	370.812...	400.520...	408.681...	400.520...	370.812...	296.517...	100.0
100.0	215.257...	286.210...	322.587...	333.686...	322.587...	286.210...	215.257...	100.0
100.0	178.301...	236.185...	269.933...	280.888...	269.933...	236.185...	178.301...	100.0
100.0	161.764...	210.294...	240.073...	250.000...	240.073...	210.294...	161.764...	100.0
100.0	158.462...	203.153...	230.066...	238.964...	230.066...	203.153...	158.462...	100.0
100.0	168.933...	213.789...	238.073...	245.725...	238.073...	213.789...	168.933...	100.0
100.0	203.482...	244.996...	262.714...	267.788...	262.714...	244.996...	203.482...	100.0
300.0	300.0	300.0	300.0	300.0	300.0	300.0	300.0	300.0





M. Turhan ÇOBAN

EGE UNIVERSITY

School of Engineering

Department of Mechanical Engineering

Bornova, Izmir, TURKEY

www.turhancoban.com

INDEX

PART I HEAT CONDUCTION

1. **HEAT CONDUCTION INTRODUCTION**
   - **1.1 HEAT CONDUCTION FORMULATION IN CARTESIAN COORDINATE SYSTEM**
   - **1.2 HEAT CONDUCTION FORMULATION IN GENERAL COORDINATE SYSTEM**
   - **1.3 BOUNDARY CONDITIONS**
   - **1.4 PROPERTIES OF SOLID MATERIALS**

2. **ONE DIMENSIONAL HEAT CONDUCTION SOLUTIONS**
   - **2.1 THE WALL**
   - **2.2 CYLINDER-PIPE**
   - **2.3 LUMPED SOLUTION OF FINS**
   - **2.4 ONE DIMENSIONAL TRANSIENT (TIME DEPENDENT) LUMPED SOLUTION**
   - **2.5 STEADY STATE SHAPE FACTOR SOLUTIONS**

3. **ANALYTICAL SOLUTIONS: SEPARATIONS OF VARIABLES**
   - **3.1 SEPARATION OF VARIABLES FOR UNIFORM MEDIUM**
   - **3.2 SEPARATION OF VARIABLES FOR COMPOSITE MEDIUM**

4. **FINITE DIFFERENCE METHODS**
   - **4.1 FINITE DIFFERENCES**
   - **4.2 METHOD OF LINES: A METHOD COMBINE FINITE DIFFERENCE AND ONE DIMENSIONAL NUMERICAL D:E: SOLUTIONS**

5. **FINITE ELEMENT METHODS**
   - **5.1 FINITE ELEMENT**
   - **5.2 SOLUTION OF STEADY STATE HEAT CONDUCTION BY FINITE ELEMENT METHOD**
   - **5.3 SOLUTION OF TRANSIENT HEAT CONDUCTION BY FINITE ELEMENT METHOD**

6. **FURTHER ANALYTICAL SOLUTIONS**
   - **6.1 DUHAMEL'S THEOREM**
   - **6.2 GREEN'S FUNCTIONS**
   - **6.3 LAPLACE TRANSFORMS**

7. **INTEGRAL TRANSFORM METHOD**

8. **APPROXIMATE SOLUTIONS: INTEGRAL EQUATIONS**

9. **PHASE CHANGE PROBLEMS**

10. **HEAT CONDUCTION IN ANISOTROPIC SOLIDS**

PART II HEAT CONVECTION

11. **NAVIER-STOKES EQUATIONS**
   - **11.1 CONSERVATION OF MASS**
   - **11.2 CONSERVATION OF MOMENTUM**
   - **11.3 CONSERVATION OF ENERGY**
   - **11.4 TURBULENT FLOW**
   - **11.5 PROPERTIES OF LIQUIDS AND GASES**

12. **CONVECTIVE HEAT TRANSFER OVER BODIES**
   - **12.1 BOUNDARY LAYER THEORY AND SIMILARITY SOLUTIONS OF BLASSIUS FLAT PLATE PROBLEM AND WEDGE FLOW PROBLEM**
   - **12.2 CONVECTIVE HEAT TRANSFER EQUATIONS AND EXAMPLES OVER A FLAT PLATE**
   - **12.3 CONVECTIVE HEAT TRANSFER EQUATIONS AND EXAMPLES ACROSS THE CYLINDER**
   - **12.4 CONVECTIVE HEAT TRANSFER EQUATIONS AND EXAMPLES ACROSS THE SPHERE**

12.5 CONVECTIVE HEAT TRANSFER EQUATIONS AND EXAMPLES OVER A BANK OF PIPES

13. CONVECTIVE HEAT TRANSFER IN PIPES AND TUBES
13.1 ONE PHASE PRESSURE DROP IN PIPES
13.2 CONVECTIVE HEAT TRANSFER IN CIRCULAR PIPES
13.3 CONVECTIVE HEAT TRANSFER IN NON-CIRCULAR TUBES

14. NATURAL (FREE) CONVECTION
14.1 NATURAL CONVECTION SIMILARITY SOLUTION
14.2 NATURAL CONVECTION HEAT TRANSFER EQUATIONS IN VERTICAL PLATE
14.3 NATURAL CONVECTION HEAT TRANSFER EQUATIONS IN INCLINED PLATE
14.4 NATURAL CONVECTION HEAT TRANSFER EQUATIONS IN HORIZONTAL PLATE
14.5 NATURAL CONVECTION HEAT TRANSFER EQUATIONS IN HORIZONTAL PIPES
14.6 NATURAL CONVECTION HEAT TRANSFER EQUATIONS IN CAVITIES
14.7 NATURAL CONVECTION HEAT TRANSFER EQUATIONS IN CHANNELS

15. BOILING AND CONDENSATION
15.1POOL BOILING HEAT TRANSFER
15.2 INSIDE PIPES CONVECTIVE BOILING HEAT TRANSFER
15.3 CONDENSATION HEAT TRANSFER
15.4 INSIDE PIPES MULTIPHASE PRESSURE DROP

16. HEAT EXCHANGERS
16.1 HEAT TRANSFER CLASSIFICATION
16.2 OVERALL HEAT TRANSFER COEFFICIENT
16.3 LOGARITHMIC MEAN TEMPERATURE DIFFERENCE
16.4 ε-NUMBER OF TRANSFER UNITS (NTU) METHOD
16.5 FINITE DIFFERENCE APPROACH
16.6 FINNED SURFACES HEAT TRANSFER FOR HEAT EXCHANGER APPLICATIONS
16.7 CHEVRON PLATE HEAT EXCHANGER HEAT TRANSFER

PART III HEAT RADIATION

17. RADIATION BASIC DEFINITIONS

18. BLACK BODY RADIATION FUNCTION

19. SPECTRAL AND TOTAL EMMISSIVITY

20. ABSORTIVITY, REFLEVTIVITY AND TRANSMISSIVITY OF REAL SURFACES

21. SHAPE FACTOR (CONFUGURATION FACTOR) FOR SURFACES SEPERATED BY A TRANSPARENT MEDIUM

22. SURFACE EXCHANGE – NO PARTICIPATING MEDIUM

23. SURFACE EXCHANGE – PARTICIPATING MEDIUM

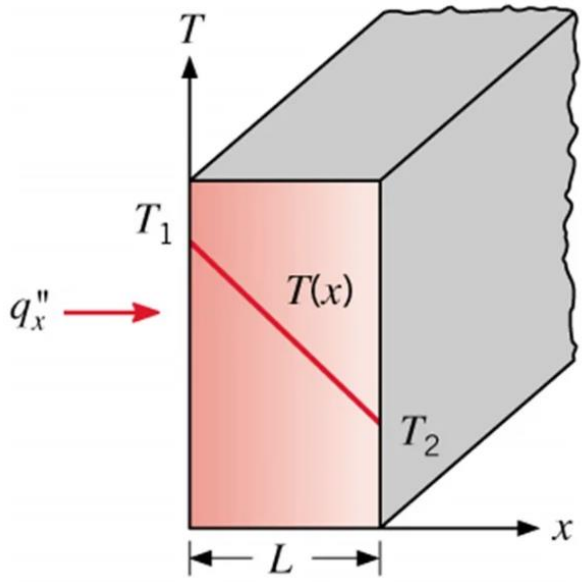
Mustafa Turhan Çoban, Borned in Bolu, Seben, Turkey in 1957. He received his B.Sc. degree from Ege University School of Mechanical Engineering, department of mechanical engineering in 1978, his M.Sc. degree in Mechanical Engineering from Michigan Technological University(U.S.A.) in 1982. He Graduated From University of Utah(U.S.A.) in 1986 with a Ph.D. in Mechanical Engineering. In 1995, he received Postgraduate degree in Computer Science from Victoria Technological University(Australia). He worked as a teacher and researcher at University of Nebraska, Lincoln(U.S.A.), Victoria Technological University(Australia), Ballarat University (Australia), Dokuz Eylül University, School of Engineering, Department of mechanical engineering(Turkey), Gebze Yüksek Teknoloji Enstitüsü, Computer science(Turkey), Ege University, School of Engineering, Department of mechanical engineering. He worked at Aras compressors (Turkey), Mineral Research and Exploration Institute of Turkey, division of drilling, section of geothermal energy, TUBITAK (Turkish Technological and Scientific Research Foundation) Institute of Energy, Imperial Chemical Industries (Australia), Ceramic Fuel Cells Limited (Australia) as an engineer/senior engineer/researcher; TUBITAK UME (National Metrology Institute of Turkey) as technical vice-chairman.

PREFACE

I started to teach heat transfer classes in Ege University in 2005 B.Sc. and Ph.D. levels and doing so since then. Class notes and programs to calculate several heat transfer problems are accumulated since then, but they has not been converted into a textbook format until I was in a hospital for a gall blader operation. Hospital a bussy place but as far as you are concerned, you do not have much to do except laying down and waiting for te recovery. So I begin to convert the collected material from class works into a book form and afford continued after the hospital and ended up as this book. The basic difference of this book is that it offers solutions of the problems in the form of computer programs. Due to the large extension of the program lists, all of them are not listed in the book, you can find the extension in my web site www.turhancoban.com. You will find three subchapters: conduction, convection and radiation. In classical textbooks conduction heat transfer is usually subgroup according to number of independent variables of the problem. In this book, for conduction solution methods, such as analytical solution methods, finite element methods, finite difference methods .., are considered for subgrouping. In convection and radiation topics it is more or less similar to classical approaches of grouping such as external forced flow, internal forced flow, natural convection, boiling, heat exchangers etc. The book is aimed to give relatively simple computer solutions to students and heat transfer community while trying to teach the topics of heat transfer. In modern times this dummy calculators called computer should be use effectively in engineering calculation and I am hoping that this book could be a useful tool in this voyage. Another nice effect of the computer solutions is that it opens a good way of visualisation of temperatures and heat flux. Visualisation of temperature and heat flux variations make it simpler to understanding heat transfer problems. Have a good trip through the world of heat transfer. I hope you enjoy it as much as ı do.

Dr. M. Turhan ÇOBAN, Ege University, 23.01.2020

I. HEAT CONDUCTION



1. HEAT CONDUCTION INTRODUCTION

1.1 HEAT CONDUCTION FORMULATION IN CARTESIAN COORDINATE SYSTEM

Fourier's Law of Conduction(in x direction) ;



Jean-Baptiste Joseph Fourier (1768-1830)

The Fourier law of thermal conductive heat transfer states that the time rate of heat transfer in solid materials is proportional to the negative gradient of temperature and to area. In cartesian coordinates, x direction thermal conductive heat transfer then expressed as

$$q_x = \frac{Q_x}{A} = -k_x \frac{dT}{dx}$$

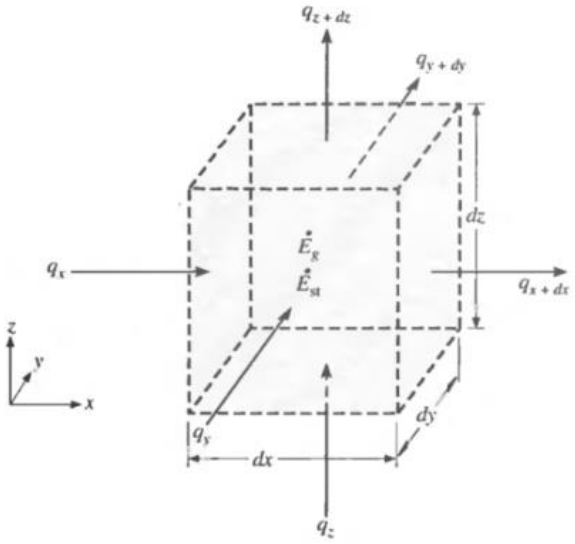
Where Q_x is total conductive heat transfer in Watt, q_x is the conductive heat transfer per unit area in Watt/m^2 , k_x is called thermal conductivity coefficient (W/(mK), T is temperature (degree K) and x is the distance in x coordinates (meters).

First law of thermodynamics (conservation of energy) states that:

$E_{in} - E_{out} + E_{generated} = E_{stored}$

Where E_{in} energy input, E_{out} is energy output, $E_{generated}$ energy generated inside the solid material and E_{stored} is the energy stored as a function of time.

Let us condider infinitely small rectangular prism with sides dx,dy and dz and apply the first law of thermodynamics for this infinitesimal prism.



$$(q_x(dydz)+q_y(dxdz)+q_z(dxdy))-(q_{x+dx}(dydz)+q_{y+dy}(dxdz)+q_{z+dz}(dxdy))+E_{generated}=E_{stored}$$

$$q_{x+dx}=q_x+\frac{\partial q_x}{\partial x}dx+\frac{1}{2!}\frac{\partial^2 q_x}{\partial x^2}dx^2+\frac{1}{3!}\frac{\partial^3 q_x}{\partial x^3}dx^3+\frac{1}{4!}\frac{\partial^4 q_x}{\partial x^4}dx^4+\cdots$$

Taylor's formula

Due to small (differential nature of prism all upper terms of Taylor's formula can be ignored so:

$$q_{x+dx}=q_x+\frac{\partial q_x}{\partial x}dx$$

$$q_{y+dy}=q_y+\frac{\partial q_y}{\partial y}dy$$

$$q_{z+dz}=q_z+\frac{\partial q_z}{\partial z}dz$$

So energy equation becomes:

$$-\left[\frac{\partial q_x}{\partial x}+\frac{\partial q_y}{\partial y}+\frac{\partial q_z}{\partial z}\right](dxdydz)+g(dxdydz)=\rho(dxdydz)\frac{\partial U}{\partial t}$$

Substituting Fourier law of conduction into the First law of thermodynamics for solid in differential cartesian element

$$q_x=-k_x(T)\frac{dT}{dx}\quad q_y=-k_y(T)\frac{dT}{dx}\quad q_z=-k_z(T)\frac{dT}{dz}$$

Equation becomes

$$\left[\frac{\partial}{\partial x}\left(k_x(T)\frac{dT}{dx}\right)+\frac{\partial}{\partial y}\left(k_y(T)\frac{dT}{dx}\right)+\frac{\partial}{\partial z}\left(k_x(T)\frac{dT}{dx}\right)\right]+g=\rho\frac{\partial U}{\partial t}$$

If internal energy be approximated as (note that specific heat at constant pressure is equala to specific heat at constant temperature, $C_p(T)=C_v(T)$ for solids)

$\partial U\cong C_p(T)dT$ equation becomes

$$\left[\frac{\partial}{\partial x}\left(k_x(T)\frac{dT}{dx}\right)+\frac{\partial}{\partial y}\left(k_y(T)\frac{dT}{dy}\right)+\frac{\partial}{\partial z}\left(k_z(T)\frac{dT}{dz}\right)\right]+g=\rho C_p(T)\frac{\partial T}{\partial t}$$

If it is assumed that thermal conductivity coefficient is independent of time and homogeneous in all direction $k(T)=k_x(T)=k_y(T)=k_z(T)$ equation becomes

$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}+\frac{\partial^2 T}{\partial z^2}+\frac{g}{k}=\frac{\rho(T)C_p(T)}{k(T)}\frac{\partial T}{\partial t}=\frac{1}{\alpha}\frac{\partial T}{\partial t}$$

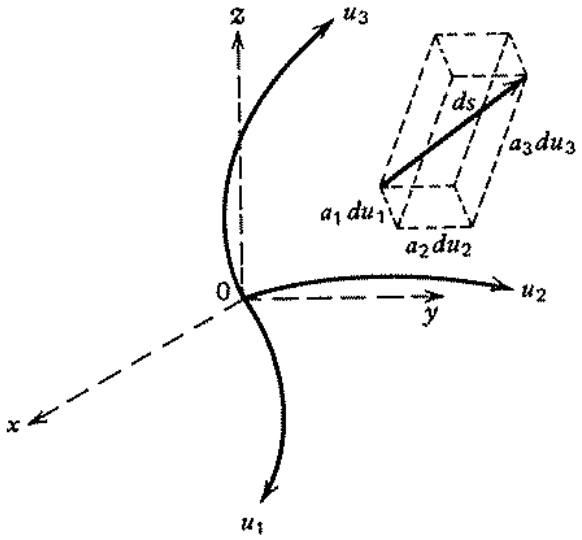
Where

$$\alpha(T)=\frac{k(T)}{\rho(T)C_p(T)}$$

is called thermal diffusivity. Most of the metals has homogeneous thermal conductivity coefficients, materials like wood has non homegeneous (direction dependent) thermal conductivity.

1.2 HEAT CONDUCTION FORMULATION IN GENERAL COORDINATE SYSTEM

Let us consider a general coordinate system of (u_1, u_2, u_3) and investigate its relation with respect to cartesian coordinate system (x,y,z)



For a general coordinate system a path dS can be expressed as a function of cartesian coordinate system as

$$(dS)^2 = (dx)^2 + (dy)^2 + (dz)^2$$

If the relation between orthagonal coordinate system (u_1, u_2, u_3) and cartesian coordinate system(x,y,z) is given as

$$x = X(u_1, u_2, u_3) \quad y = Y(u_1, u_2, u_3) \;\; z = Z(u_1, u_2, u_3)$$

Then the differential lengths dx,dy and dz can be obtained as:

$$dx = \sum_{i=1}^{3} \frac{\partial X(u_1,u_2,u_3)}{\partial u_i} \quad dy = \sum_{i=1}^{3} \frac{\partial Y(u_1,u_2,u_3)}{\partial u_i} \quad dz = \sum_{i=1}^{3} \frac{\partial Z(u_1,u_2,u_3)}{\partial u_i}$$

Substituting these back into the path equation

$$(dS)^2 = a_1^2 (du_1)^2 + a_2^2 (du_2)^2 + a_3^2 (du_3)^2$$

where

$$a_i^2 = \left(\frac{\partial x}{\partial u_i}\right)^2 + \left(\frac{\partial y}{\partial u_i}\right)^2 + \left(\frac{\partial z}{\partial u_i}\right)^2$$

Heat transfer in orthagonal coordinate system becomes:

$$q = -k\nabla T = -k \sum_{i=1}^{3} u_i \frac{\partial T}{\partial u_i}$$

Or

$$q = -k\nabla T = -k \frac{1}{a_i} \frac{\partial T}{\partial u_i}$$

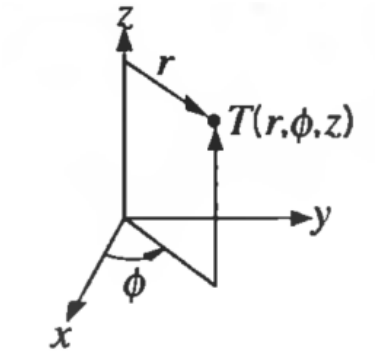
And

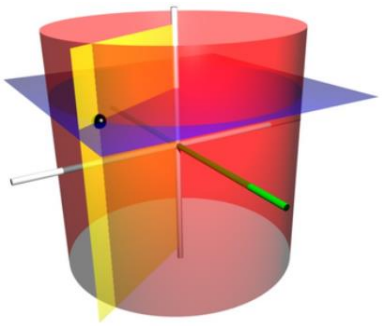
$$a = a_1 a_2 a_3$$

In this case general first law of thermodynamics for this coordinate system becomes:

$$\frac{1}{a}\left[\frac{\partial}{\partial u_1}\left(k_{u_1}(T)\frac{a}{a_1^2}\frac{\partial T}{\partial u_1}\right) + \frac{\partial}{\partial u_2}\left(k_{u_2}(T)\frac{a}{a_2^2}\frac{\partial T}{\partial u_2}\right) + \frac{\partial}{\partial u_3}\left(k_{u_3}(T)\frac{a}{a_3^2}\frac{\partial T}{\partial u_3}\right)\right] + g = \rho C_p(T)\frac{\partial T}{\partial t}$$

Example case:
Relation between cartesian coordinate system and **cylindrical coordinate system** given as:
$\boldsymbol{x = r\cos(\phi) \;\; y = r\sin(\phi)}$ **z=z**





$u_1 = r \;\; u_2 = \phi \;\; u_3 = z$, The scale factors:

$a_1^2 = a_r^2 = \left(\frac{\partial x}{\partial r}\right)^2 + \left(\frac{\partial y}{\partial r}\right)^2 + \left(\frac{\partial z}{\partial r}\right)^2 = cos^2(\phi) + sin^2(\phi) + 0 = 1$

$a_2^2 = a_\phi^2 = \left(\frac{\partial x}{\partial \phi}\right)^2 + \left(\frac{\partial y}{\partial \phi}\right)^2 + \left(\frac{\partial z}{\partial \phi}\right)^2 = \left(-rcos(\phi)\right)^2 + \left(rsin(\phi)\right)^2 + 0 = r^2$

$a_3^2 = a_z^2 = \left(\frac{\partial x}{\partial z}\right)^2 + \left(\frac{\partial y}{\partial z}\right)^2 + \left(\frac{\partial z}{\partial z}\right)^2 = 0 + 0 + 1 = 1$ so

$a_r = 1 \;\; a_\phi = r \; a_z = 1$

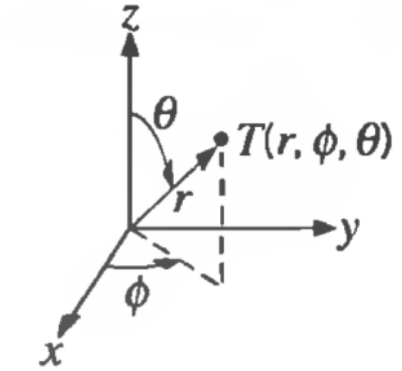
$a = a_r a_\phi a_z = r$ so heat transfer equations becomes:

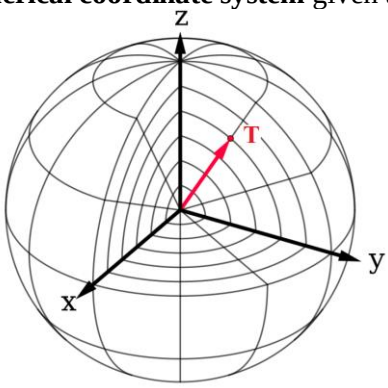
$q_r = -k_r(T)\frac{\partial T}{\partial r} \quad q_\phi = -\frac{k_\phi(T)}{r}\frac{\partial T}{\partial \phi} \quad q_z = -k_z(T)\frac{\partial T}{\partial z}$ and

$\frac{1}{r}\left[\frac{\partial}{\partial r}\left(k_r(T)\frac{r}{1^2}\frac{\partial T}{\partial r}\right) + \frac{\partial}{\partial \phi}\left(k_\phi(T)\frac{r}{r^2}\frac{\partial T}{\partial \phi}\right) + \frac{\partial}{\partial z}\left(k_z(T)\frac{r}{1^2}\frac{\partial T}{\partial z}\right)\right] + g = \rho C_p(T)\frac{\partial T}{\partial t}$

$\frac{1}{r}\frac{\partial}{\partial r}\left(k_r(T) r \frac{\partial T}{\partial r}\right) + \frac{1}{r^2}\frac{\partial}{\partial \phi}\left(k_\phi(T)\frac{\partial T}{\partial \phi}\right) + + \frac{\partial}{\partial z}\left(k_z(T)\frac{\partial T}{\partial z}\right) + g = \rho C_p(T)\frac{\partial T}{\partial t}$

Relation between cartesian coordinate system and **spherical coordinate system** given as:





$\boldsymbol{x = rsin(\theta)cos(\phi) \;\; y = rsin(\theta)sin(\phi) \;\; z = rcos(\theta)}$

$u_1 = r \;\; u_2 = \phi \;\; u_3 = \theta$, The scale factors:

$a_1^2 = a_r^2 = \left(\frac{\partial x}{\partial r}\right)^2 + \left(\frac{\partial y}{\partial r}\right)^2 + \left(\frac{\partial z}{\partial r}\right)^2 = \left(\sin(\theta)cos(\phi)\right)^2 + \left(\sin(\theta)sin(\phi)\right)^2 + (\cos(\theta))^2 = 1$

$a_2^2 = a_\phi^2 = \left(\frac{\partial x}{\partial \phi}\right)^2 + \left(\frac{\partial y}{\partial \phi}\right)^2 + \left(\frac{\partial z}{\partial \phi}\right)^2 = r^2\left(\sin(\theta)sin(\phi)\right)^2 + r^2\left(\sin(\theta)cos(\phi)\right)^2 + 0 = r^2\sin^2(\theta)$

$a_3^2 = a_\theta^2 = \left(\frac{\partial x}{\partial \theta}\right)^2 + \left(\frac{\partial y}{\partial \theta}\right)^2 + \left(\frac{\partial z}{\partial \theta}\right)^2 = r^2\left(\cos(\theta)cos(\phi)\right)^2 + r^2\left(\cos(\theta)sin(\phi)\right)^2 + r^2\sin^2(\phi) = r^2$

$a_r = 1 \;\; a_\phi = rsin(\theta) \quad a_\theta = r$

$a = a_r a_\phi a_\theta = r^2 sin(\theta)$ so heat transfer equations becomes:

$q_r = -k_r(T)\frac{a}{a_r^2}\frac{\partial T}{\partial r} = -k_r(T)\frac{r^2 sin(\theta)}{1^2}\frac{\partial T}{\partial r} \quad q_\phi = -k_\phi(T)\frac{a}{a_\phi^2}\frac{\partial T}{\partial \phi} = -k_\phi(T)\frac{r^2 sin(\theta)}{r^2 sin^2(\theta)}\frac{\partial T}{\partial \phi}$

$q_\theta = -k_\theta(T)\frac{a}{a_\theta^2}\frac{\partial T}{\partial \theta} = -k_\theta(T)\frac{r^2 sin(\theta)}{r^2}\frac{\partial T}{\partial \theta}$ and

$\frac{1}{r^2 sin(\theta)}\left[\frac{\partial}{\partial r}\left(k_r(T)\frac{r^2 sin(\theta)}{1^2}\frac{\partial T}{\partial r}\right) + \frac{\partial}{\partial \phi}\left(k_\phi(T)\frac{r^2 sin(\theta)}{r^2 sin^2(\theta)}\frac{\partial T}{\partial \phi}\right) + \frac{\partial}{\partial \theta}\left(k_\theta(T)\frac{r^2 sin(\theta)}{r^2}\frac{\partial T}{\partial \theta}\right)\right] + g = \rho C_p(T)\frac{\partial T}{\partial t}$

$$\frac{1}{r^2}\frac{\partial}{\partial r}\left(k_r(T)r^2\frac{\partial T}{\partial r}\right)+\frac{1}{r^2sin^2(\theta)}\frac{\partial}{\partial \phi}\left(k_\phi(T)\frac{\partial T}{\partial \phi}\right)+\frac{1}{r^2sin(\theta)}\frac{\partial}{\partial \theta}\left(k_\theta(T)sin(\theta)\frac{\partial T}{\partial \theta}\right)+g=\rho C_p(T)\frac{\partial T}{\partial t}$$

1.3 BOUNDARY CONDITIONS

In order to solve above differential equations, baundary values on the surfaces of the solid are also required. In the boundary heat can be supplied directly(for example as electrical heating), or through convective or radiative heat transfer. Surface temmperature variation can also be specified as a boundary condition

Convective heat flux in the surface can be expressed as

$$q_{convection}=h(T-T_\infty)$$

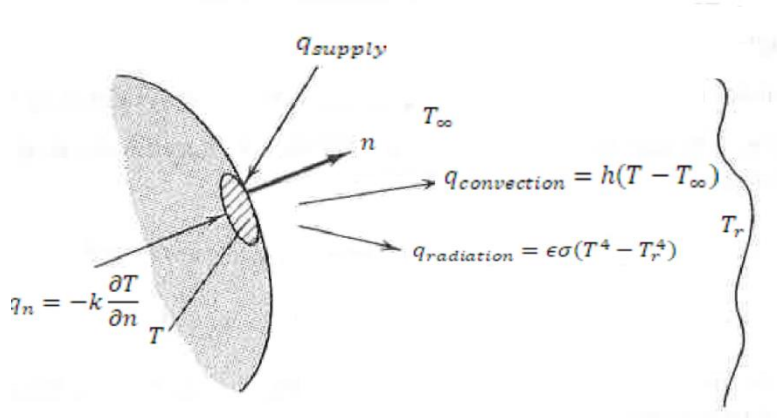
In this equation T is the temperature of solid in the surface, T_∞ is the temperature of the fluid at the boundary of the solid surface, and h is the thermal convection coefficient. The thermal convectivity or thermal convection coefficients will change with pysical conditions of boundary fluid sech as temperature velocity, density etc. In second part of this book it will be investigated more throughly. For most of boundary problems of thermal conduction, it will be assumed to be constant to simplify equation solving process.

Radiative heat flux in the surface can be expressed as

$$q_{radiation}=\varepsilon\sigma(T^4-T_r^4)$$

In this equation ε is called emmissivity coefficient. Emmisisvity is surface characteristicscify emitted radiative energy. When the energy coming from outside and absorbed by the surface absorbtivity should be used for this term, but According to Kirchoff law, if the system is in thermal equilibrium, it can be shown that absorbtivity coefficient(α) is equal to emmissivity. $\alpha=\varepsilon$. Radiative energy is also wvavelenth dependent, but it is ommitted in here. For more details please refer to radiation section of the book. σ is Stephan-Boltsman constant σ =5.6703744192x10^{-8} W/m^{2}K^4. T_r is the temperature of the surface radiation emmitted. T_r does not have to be in vicinity of the surface, it can be far away as long as it see the boundary surface of our solid. T is the surface temmperature of the solid.

On the solid part of the surface heat transfer can be expressed as fourier conduction normal to the surface. In order to investigate changing values of internal conduction and external boundary heat transfer should be considered constant for a small area of boundary value application. Conditions can be change throughout the surface. These changes can be taken into acount by considering cchanges as function of surface coordiantes or as constant vales in exach small part of the surface by considering finite difference or finite element solutions of the energy equation.



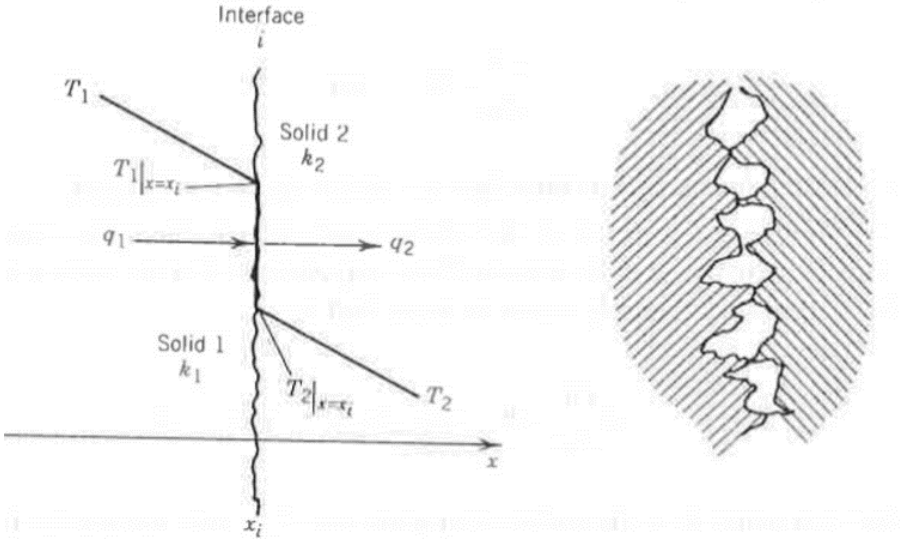
For a differential surface element

$q_n=-k\nabla T=-k\left(\frac{\partial T}{\partial x}l_x+\frac{\partial T}{\partial y}l_y+\frac{\partial T}{\partial z}l_z\right)=-k\frac{\partial T}{\partial n}$ where l_x, l_y and l_z are unit vectors in x,y and z directions.

Energy balance at the surface: Heat supply=heat loss

T is surface temperature

T_∞ outside fluid temperature
T_r radiative surface temperature
n surface normal vector
$q_n = -k\frac{\partial T}{\partial n}$
$q_n + q_{supply} = q_{convection} + q_{radiation}$
$-k\frac{\partial T}{\partial n} + q_{supply} = h(T - T_\infty) + \epsilon\sigma(T^4 - T_r^4)$ As addition to this generalised case, a temperature distribution on the surface can also be specified, such as:
T=f(x,t) on S or T=0 or T=T_i on S
In addition to the surface type of boundary, interphase of the contacting surfaces can also specify a boundary. When two differnt material id connected by some physical processes (welding, brazing or just contact with applied pressure) some small gap usually existed between this two solid layer interphase. This gaps cause additional heat resistance, that should be considered as additional boundary conditions.



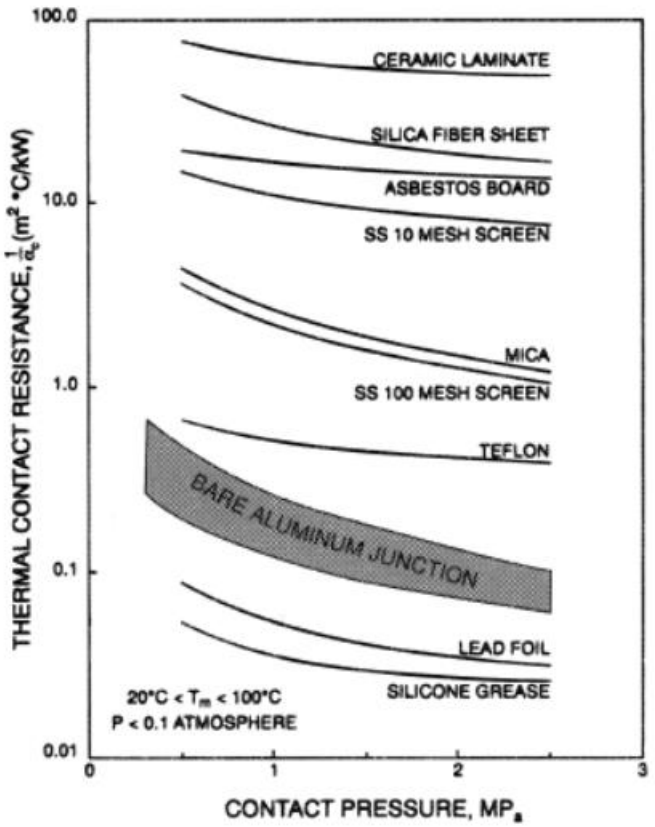
(Heat conduction through solid 1)=(heat conduction through solid 2)=(heat transfer across the gap)

$$-k_1 \frac{\partial T_1}{\partial x}|_i = h(T_1 - T_2)|_i = -k_2 \frac{\partial T_2}{\partial x}|_i$$

Instead of h term, thermal contact resistance R=1/h is usually given . Thermal contact resistance are function of the gap between surfaces, therefore applied pressure when surfaces attached to each other effects this parameter. Thermal contact resistances of some common materials are listed below:

Surface type	**R=1/h m^2K/W**
416 Stainless,ground, air	2.64
416 Stainless,ground, 0.0254 mm thickness brass sheet in contact, air	3.52
304 Stainless,ground, air	5.28
Aliminium,ground, air	0.88
Aliminium ,ground, 0.0254 mm thickness brass sheet in contact, air	1.23
Copper, ground, air	0.07
Copper, milled, air	0.18
Copper,milled,vacuum	0.88

Thermal contact resistance can be effected or changed by using some interfacial material between the contact surfaces. Application of different interface material between aliminium junction is shown in the figure below (Source: www.thermopedia.com/content/1188)



1.4 PROPERTIES OF SOLID MATERIALS

1.4.1 THERMAL CONDUCTIVITY

Fourier's law of conductivity is the basic equation defining thermal conductivity.

$Q = -k(T)A\frac{dT}{dx}$ therefore thermal conductivity expressed as

$$k(T) = \frac{Q}{A\frac{dT}{dx}}$$

Thermal conductivity is a temperature dependent term. In metals thermal conductivity is usually independent of orientations, but in composite materials and some other type of materials orientation is also an important variable for thermal conductivity. Thermal conductivities of some materials are listed below.

Table 1.4.1-1 Thermal conductivity of metals

Metal	Temperature - *t* - (*°C*)	Thermal - *k* - (*W/m K*)
Admiralty Brass	20	111
Aluminum, pure	20	204
Aluminum, pure	93	215
Aluminum, pure	204	249
Aluminum Bronze	20	76
Antimony	20	19
Beryllium	20	218
Beryllium Copper	20	66
Bismuth	20	8.5
Cadmium	20	93
Carbon Steel, max 0.5% C	20	54
Carbon Steel, max 1.5% C	20	36
Carbon Steel, max 1.5% C	400	33
Carbon Steel, max 1.5% C	1,200	29
Cartridge brass (UNS C26000)	20	120
Cast Iron, gray	21	47 – 80

Chromium	20	90
Cobalt	20	69
Copper, pure	20	386
Copper, pure	300	369
Copper, pure	600	353
Copper bronze (75% Cu, 25% Sn)	20	26
Copper brass (70% Cu, 30% Zi)	20	111
Cupronickel	20	29
Gold	20	315
Hastelloy B		10
Hastelloy C	21	8.7
Inconel	21 - 100	15
Incoloy	0 – 100	12
Iridium	20	147
Iron, nodular pearlitic	100	31
Iron, pure	20	73
Iron, pure	300	55
Iron, pure	1,000	35
Iron, wrought	20	59
Lead	20	35
Lead	300	30
Manganese Bronze	20	106
Magnesium	20	159
Mercury	20	8.4
Molybdenum	20	140
Monel	0 – 100	26
Nickel	20	90
Nickel Wrought	0 – 100	61 – 90
Niobium (Columbium)	20	52
Osmium	20	61
Phosphor bronze (10% Sn, UNS C52400)	20	50
Platinum	20	73
Plutonium	20	8
Potassium	20	100
Red Brass	20	159
Rhodium	20	150
Selenium	20	0.52
Silicon	20	84
Silver, pure	20	407
Sodium	20	134
Stainless Steel	20	12 – 45
Tantalum	20	54
Thorium	20	42
Tin	0	62 – 68
Titanium	20	19 – 23
Tungsten	20	163 – 173
Uranium	20	24
Vanadium	20	61
Wrought Carbon Steel	0	59
Yellow Brass	20	116
Zinc		116
Zirconium	0	23

Table 1.4.1-2 Thermal conductivity of some materials

Material/Substance	**Temperature**		
	25 °C	***125 °C***	***225 °C***
Diamond	1000		
slica eorogel	0.003		
Acetals	0.23		
Acetone	0.16		

Acetylene (gas)	0.018		
Acrylic	0.2		
Air, atmosphere (gas)	0.0262	0.0333	0.0398
Air, elevation 10000 m	0.02		
Agate	10.9		
Alcohol	0.17		
Aluminum	205	215	250
Aluminum Brass	121		
Aluminum Oxide	30		
Ammonia (gas)	0.0249	0.0369	0.0528
Antimony	18.5		
Apple (85.6% moisture)	0.39		
Argon (gas)	0.016		
Asbestos-cement board	0.744		
Asbestos-cement sheets	0.166		
Asbestos-cement	2.07		
Asbestos, loosely packed	0.15		
Asbestos mill board	0.14		
Asphalt	0.75		
Balsa wood	0.048		
Bitumen	0.17		
Bitumen/felt layers	0.5		
Beef, lean (78.9 % moisture)	0.43 - 0.48		
Benzene	0.16		
Beryllium	218		
Bismuth	8.1		
Bitumen	0.17		
Blast furnace gas (gas)	0.02		
Boiler scale	1.2 - 3.5		
Brass	109		
Breeze block	0.10 - 0.20		
Brick dense	1.31		
Brick, fire	0.47		
Brick, insulating	0.15		
Brickwork, common (Building Brick)	0.6 -1.0		
Brickwork, dense	1.6		
Bromine (gas)	0.004		
Bronze	110		
Brown iron ore	0.58		
Butter (15% moisture content)	0.2		
Cadmium	92.1		
Calcium silicate	0.05		
Carbon	1.7		
Carbon dioxide (gas)	0.0146		
Carbon monoxide	0.0232		
Cast iron	58		
Cellulose, cotton, wood pulp and regenerated	0.23		
Cellulose acetate, molded, sheet	0.17 - 0.33		
Cellulose nitrate, celluloid	0.12 - 0.21		
Cement, Portland	0.29		
Cement, mortar	1.73		
Ceramic materials			
Chalk	0.09		
Charcoal	0.084		
Chlorinated poly-ether	0.13		
Chlorine (gas)	0.0081		
Chrome Nickel Steel (18% Cr, 8 % Ni)	16.3		
Chromium	94		
Chrom-oxide	0.42		
Clay, dry to moist	0.15 - 1.8		
Clay, saturated	0.6 - 2.5		

Coal	0.2		
Cobalt	69		
Cod (83% moisture content)	0.54		
Coke	0.184		
Concrete, lightweight	0.1 - 0.3		
Concrete, medium	0.4 - 0.7		
Concrete, dense	1.0 - 1.8		
Concrete, stone	1.7		
Constantan	23.3		
Copper	401	400	398
Corian (ceramic filled)	1.06		
Cork board	0.043		
Cork, re-granulated	0.044		
Cork	0.07		
Cotton	0.04		
Cotton wool	0.029		
Carbon Steel	54	51	47
Cotton Wool insulation	0.029		
Diamond	1000		
Diatomaceous earth (Sil-o-cel)	0.06		
Diatomite	0.12		
Duralium	129		
Earth, dry	1.5		
Ebonite	0.17		
Emery	11.6		
Engine Oil	0.15		
Ethane (gas)	0.018		
Ether	0.14		
Ethylene (gas)	0.017		
Epoxy	0.35		
Ethylene glycol	0.25		
Feathers	0.034		
Felt insulation	0.04		
Fiberglass	0.04		
Fiber insulating board	0.048		
Fiber hardboard	0.2		
Fire-clay brick 500^{o}C	1.4		
Fluorine (gas)	0.0254		
Foam glass	0.045		
Dichlorodifluoromethane R-12 (gas)	0.007		
Dichlorodifluoromethane R-12 (liquid)	0.09		
Gasoline	0.15		
Glass	1.05		
Glass, Pearls, dry	0.18		
Glass, Pearls, saturated	0.76		
Glass, window	0.96		
Glass, wool Insulation	0.04		
Glycerol	0.28		
Gold	310	312	310
Granite	1.7 - 4.0		
Graphite	168		
Gravel	0.7		
Ground or soil, very moist area	1.4		
Ground or soil, moist area	1		
Ground or soil, dry area	0.5		
Ground or soil, very dry area	0.33		
Gypsum board	0.17		
Hairfelt	0.05		
Hardboard high density	0.15		
Hardwoods (oak, maple..)	0.16		
Helium (gas)	0.142		
Honey (12.6% moisture content)	0.5		

Hydrochloric acid (gas)	0.013		
Hydrogen (gas)	0.168		
Hydrogen sulfide (gas)	0.013		
Ice (273.15 K)	2.18		
Ingot iron	47 - 58		
Insulation materials	0.035 - 0.16		
Iodine	0.44		
Iridium	147		
Iron	80	68	60
Iron, wrought	59		
Iron, cast	55		
Iron-oxide	0.58		
Kapok insulation	0.034		
Kerosene	0.15		
Krypton (gas)	0.0088		
Lead Pb	35		
Leather, dry	0.14		
Limestone	1.26 - 1.33		
Lithium	301		
Magnesia insulation (85%)	0.07		
Magnesite	4.15		
Magnesium	156		
Magnesium alloy	70 - 145		
Marble	2.08 - 2.94		
Mercury, liquid	8.3		
Methane (gas)	0.03		
Methanol	0.21		
Mica	0.71		
Milk	0.53		
Mineral wool insulation materials, wool blankets ..	0.04		
Molybdenum	138		
Monel	26		
Neon (gas)	0.046		
Neoprene	0.05		
Nickel	91		
Nitric oxide (gas)	0.0238		
Nitrogen (gas)	0.024		
Nitrous oxide (gas)	0.0151		
Nylon 6, Nylon 6/6	0.25		
Oil, machine lubricating SAE 50	0.15		
Olive oil	0.17		
Oxygen (gas)	0.024		
Palladium	70.9		
Paper	0.05		
Paraffin Wax	0.25		
Peat	0.08		
Perlite, atmospheric pressure	0.031		
Perlite, vacuum	0.00137		
Phenolic cast resins	0.15		
Phenol-formaldehyde moulding compounds	0.13 - 0.25		
Phosphorbronze	110		
Pinchbeck	159		
Pitch	0.13		
Pit coal	0.24		
Plaster light	0.2		
Plaster, metal lath	0.47		
Plaster, sand	0.71		
Plaster, wood lath	0.28		
Plasticine	0.65 - 0.8		
Plastics, foamed (insulation materials)	0.03		
Platinum	70	71	72
Plutonium	6.7		

Plywood	0.13		
Polycarbonate	0.19		
Polyester	0.05		
Polyethylene low density, PEL	0.33		
Polyethylene high density, PEH	0.42 - 0.51		
Polyisoprene natural rubber	0.13		
Polyisoprene hard rubber	0.16		
Polymethylmethacrylate	0.17 - 0.25		
Polypropylene, PP	0.1 - 0.22		
Polystyrene, expanded styrofoam	0.03		
Polystyrol	0.043		
Polyurethane foam	0.03		
Porcelain	1.5		
Potassium	1		
Potato, raw flesh	0.55		
Propane (gas)	0.015		
Polytetrafluoroethylene (PTFE)	0.25		
Polyvinylchloride, PVC	0.19		
Pyrex glass	1.005		
Quartz mineral	3		
Radon (gas)	0.0033		
Red metal	128		
Rhenium	71		
Rhodium	88		
Rock, solid	2.Tem		
Rock, porous volcanic (Tuff)	0.5 - 2.5		
Rock Wool insulation	0.045		
Rosin	0.32		
Rubber, cellular	0.045		
Rubber, natural	0.13		
Rubidium	58		
Salmon (73% moisture content)	0.5		
Sand, dry	0.15 - 0.25		
Sand, moist	0.25 - 2		
Sand, saturated	2.Nis		
Sandstone	1.7		
Sawdust	0.08		
Selenium	0.2		
Sheep wool	0.039		
Silica aerogel	0.02		
Silicon cast resin	0.15 - 0.32		
Silicon carbide	120		
Silicon oil	0.1		
Silver	429		
Slag wool	0.042		
Slate	2.01		
Snow (temp < 0^{o}C)	0.05 - 0.25		
Sodium	135 (solid)	86 (liquid)	
Softwoods (fir, pine ..)	0.12		
Soil, clay	1.1		
Soil, with organic matter	0.15 - 2		
Soil, saturated	0.6 - 4		
Soot	0.07		
Steam, saturated	0.0184		
Steam, low pressure	0.0188		
Steatite	2		
Steel, Carbon 1%	43		
Stainless Steel	16	17	19
Straw slab insulation, compressed	0.09		
Styrofoam	0.033		
Sulfur dioxide (gas)	0.0086		
Sulfur, crystal	0.2		

Sugars	0.087 - 0.22		
Tantalum	54		
Tar	0.19		
Tellurium	4.9		
Thorium	38		
Timber, alder	0.17		
Timber, ash	0.16		
Timber, birch	0.14		
Timber, larch	0.12		
Timber, maple	0.16		
Timber, oak	0.17		
Timber, pitchpine	0.14		
Timber, pockwood	0.19		
Timber, red beech	0.14		
Timber, red pine	0.15		
Timber, white pine	0.15		
Timber, walnut	0.15		
Tin Sn	67		
Titanium	22		
Tungsten	174		
Uranium	27.6		
Urethane foam	0.021		
Perfect Vacuum	**0**		
Vermiculite granules	0.065		
Vinyl ester	0.25		
Water	0.606		
Water, vapor (steam)		0.0267	0.0359
Wax	0.084		
Wheat flour	0.45		
White metal	35 - 70		
Wood across the grain, white pine	0.12		
Wood across the grain, balsa	0.055		
Wood across the grain, yellow pine, timber	0.147		
Wood, oak	0.17		
Wool, felt	0.07		
Wood wool, slab	0.1 - 0.15		
Xenon (gas)	0.0051		
Zinc Zn	116		

Table 1.4.1-3 Thermal conductivities of some materials as a function of temperature[93],[94]

C9				**C 0.57**	**Mn 0.55**	**Si 0.38**					
Steel 3	**T K**	**k W/mK**									
	273.1	51									
	373.2	47.7									
	473.2	44.8									
	573.2	41.8									
	673.2	39.2									
	773.2	38.1									
C129	**T K**	**k W/mK**	**C 0.8**	**Cu 0.07**	**Mn 0.32**	**P 0.08**	**S 0.09**	**Si 0.13**	**Ni 0.13**	**Cr 0.11**	**As 0.02**
British steel 7	273.1	49.8									
	323.1	49.4									
	373.1	48.3									
	423.1	46.9									
	473.1	45.2									
	523.1	43.1									
	573.1	41.4									
	623.1	39.8									
	673.1	38.1									
	723.1	36.6									
	773.1	35.4									

	823.1	34.1											
	873.1	29.9											
	923.1	31.6											
	973.1	30.3											
	993.1	29.7											
	1013.	23.9											
	1023.	23.9											
	1073.	24.3											
	1123.	24.7											
	1173.	25.1											
	1223.	25.9											
	1273.	26.8											
C124	**T K**	**k W/mK**	**C 0.64**	**Cu 0.09**	**Mn 0.27**	**P 0.01**	**S 0.028**	**Si 0.06**					
12 Forged	283.1	43.9											
	303.1	43.9											
	391.1	43.9											
	420.1	43.9											
	469.1	43.5											
	521.1	43.1											
	527.1	41.4											
	641.1	41											
	690.1	41											
	796.1	36.8											
	851.1	35.8											
	912.1	36											
	967.1	38.5											
	999.1	29.7											
	1044.	28.9											
	1100.	30.5											
	1134.	28.9											
	1199.	31.8											
C127	**T K**	**k W/mK**	**C 1.3**	**Cu 0.06**	**Mn 0.4**	**P 0.04**	**S0.018**	**Si 0.08**					
26	283.1	36											
	303.1	36											
	383.1	36											
	407.1	36.8											
	460.1	36.8											
	487.1	36.4											
	541.1	36.8											
	595.1	36.8											
	638.1	34.3											
	693.1	35.6											
	728.1	35.6											
	760.1	34.3											
	803.1	32.6											
	863.1	31.8											
	910.1	31											
	973.1	41.4											
	1022.	27.6											
	1102.	26.8											
	1147.	28											
C132	**T K**	**k W/mK**	**C 0.8**	**Cu 0.07**	**Mn 0.32**	**P 0.00**	**S 0.009**	**Si 0.13**	**Ni 0.13**	**Cr 0.11**	**As 0.02**	**Mo <0.0**	**Al 0.00**
British steel 7	273.1	49.8											
	323.1	49.4											
	373.1	48.3											
	423.1	46.9											
	473.1	45.1											
	523.1	43.1											
	573.1	41.4											
	623.1	40.1											

T328 C26	T K	k W/mK		Cr 12.95	C 0.13	Mn 0.25	Ni 0.14	P 0.018	S 0.024	Si 0.17	Al 0.034	Cu 0.06	V 0.012	As 0.015
chromium steel	273.1	27												
	323.1	27.2												
	373.1	27.6												
	423.1	27.6												
	473.1	27.8												
	523.1	27.8												
	573.1	28												
	623.1	28												
T328 C32	**T K**	**k W/mK**		**Cr 26**	**C 0.13**	**Mn 0.56**	**Ni 0.1**	**P 0.01**	**S 0.007**	**Si 0.5**	**Al 0.03**	**Cu 0.06**	**V 0.01**	**As 0.01**
chromium steel	273.1	22.8												
	373.1	23.8												
	473.1	25.5												
	573.1	27.2												
	673.1	28.5												
	773.1	30.1												
	873.1	31.8												
	973.1	33.5												
	1073.	35.1												
	1173.	36.4												
T330 C2	**T K**	**k W/mK**		**Cr 17-**	**Nı 7-9**	**C 0.11**								
chromium-nickel	273.2	14.9												
SS 304	373.2	16.3												
	405.9	17.2												
	423.2	17.6												
	444.1	17.8												
	457.2	17.9												
	463.2	18.1												
	473.2	18.1												
	483.3	18.4												
	512.2	18.8												
	523.2	18.8												
	534.2	19.2												
	547.8	19.3												
	583.7	19.7												
	599.6	19.9												
	614.3	20.1												
	637.8	20.5												
	670.5	20.7												
	678.7	21												
	695	21.2												
	757.7	22												
	773.2	22.4												
	839	23.1												
	851	23.3												
	904.1	24.2												
	920.5	24.5												
	934.1	24.8												
	991.4	25.4												
	1008	25.9												
T331 C3	**T K**	**k W/mK**	**Cr 16-18**	**Ni 6-8**	**C 0.08**	**Mn 2**	**Si 0.11**		**P 0.002**					
AISI 301 SS	136.8	10.9												
	184	13												
	225	13.4												
	266	14.2												
	329.8	16.3												
	355.5	16.7												
	380.3	17.6												
	405	16.7												

	536.4	18.8								
	602.5	19.7								
	722.3	20.9								
	831.8	22.6								
	887.8	23.4								
	1032.	25.5								
	1038.	25.1								
	1216.	26.4								
T331 C4	**T K**	**k W/mK**	**Cr 16.82**	**Ni 11.6**	**C 0.10**	**Mn 1.59**	**Mo 2.18**	**P 0.01**	**Si 0.26**	**S 0.02**
AISI 316 SS	135.7	10.5								
	184.4	11.3								
	228.9	12.6								
	270	13.4								
	367.8	14.6								
	433.8	15.5								
	482.1	15.1								
	497.8	15.9								
	552.5	15.9								
	560.9	18								
	592.5	18.4								
	660.7	18.8								
	759.8	20.1								
	812.6	20.5								
	825.6	20.9								
	944.7	22.2								
	1031.	23.4								
	1114	23.8								
	1220.	25.1								

T331 C5	**T K**	**k W/mK**	**Cr 17.65**	**Ni 10.9**	**C 0.06**	**Co 0.09**	**Mn 1.64**	**Mo 0.02**	**P 0.013**	**Nb 0.73**	**Si 0.58**	**S 0.01**
AISI 347 SS	137.7	11.7										
	185.4	13										
	228.6	14.2										
	269.2	14.6										
	361.9	15.1										
	427.9	15.9										
	429.5	16.7										
	490.8	16.7										
	540.7	18.4										
	550.1	17.6										
	552.1	16.7										
	576.5	18.8										
	662.8	18.8										
	741.5	20.5										
	768.9	20.1										
	801	21.3										
	919.1	23.4										
	996.9	23.4										
	1079.	24.7										
	1173.	25.9										

T331 C64	**T K**	**k W/mK**	**Cr 17.45**	**Ni 12.6**	**C 0.06**	**Cu 0.09**	**Mn 1.59**	**Mo 2.55**	**P 0.023**	**Si 0.6**	**Co 0.19**	**S 0.01**
AISI 316 3A SS	273.7	13										
	334.8	14.2										
	343.7	14.6										
	388.7	15.4										
	391.5	15.4										
	413.7	15.8										
	417.6	15.7										
	430.9	16.5										
	431.5	15.9										
	457.1	16.3										

	460.9	16.5										
	481.5	17.1										
	500.9	17.3										
	514.3	17.5										
	520.4	17.5										
	589.8	18.9										
	605.4	19.6										
	613.7	18.9										
	659.3	19.6										
	670.4	19.5										
	679.8	19.9										
	712.6	20										
	722.1	20.4										
	799.8	22.1										
	800.4	22										
	897.1	23.6										
	958.2	24.4										
	982.6	24.6										
	1113.	26.5										
	1185.	27.7										
T331 C65	**T K**	**k W/mK**	**Cr 17.45**	**Ni 12.6**	**C 0.06**		**Mn 1.88**	**Mo 2.70**	**P 0.021**	**Si 0.54**		**S 0.01**
AISI 316 4A SS	272.6	13.3										
	335.4	14.6										
	346.5	14.8										
	375.4	15.7										
	394.3	15.7										
	408.7	15.4										
	433.7	16.6										
	438.2	16										
	439.8	16.5										
	459.3	17										
	472.6	16.9										
	494.3	17										
	507.1	17										
	558.2	17.9										
	597.6	18.2										
	600.9	18.7										
	657.6	19.2										
	662.6	19.2										
	667.6	19.4										
	703.7	19.6										
	743.7	21.1										
	792.1	22.3										
	796.5	21.1										
	822.1	22.3										
	892.6	22.8										
	952.6	23.7										
	1012.	25.1										
	1140.	28.2										
	1218.	29.1										
TS331R C A	**T K**	**k W/mK**	**Cr 18-20**	**Ni 8-12**	**Mn 2**							
SS304	25	2.7										
	50	5.8										
	75	8										
	100	9.5										
	150	11.5										
	200	13										
	250	14.2										
	273.1	14.7										
	300	15.2										
	350	16.2										

	400	17				
	450	17.7				
	500	18.4				
	600	19.8				
	700	21.2				
	800	22.5				
	900	23.9				
	1000	25.3				
	1100	26.7				
	1200	28.1				
	1300	29.5				
	1400	30.9				
	1500	32.3				
	1600	33.7				
	1665	34.7				
TS331R C B	**T K**	**k W/mK**	**Cr 17-19**	**Ni 9-13**	**Mn 2**	**Si 1**
SS347	25	2.5				
	50	5.4				
	75	7.6				
	100	9.1				
	150	11.1				
	200	12.6				
	250	13.8				
	273.1	14.3				
	300	14.8				
	350	15.7				
	400	16.5				
	450	17.2				
	500	17.9				
	600	19.2				
	700	20.5				
	800	21.9				
	900	23.2				
	1000	24.6				
	1100	25.9				
	1200	27.3				
	1300	28.6				
	1400	30				
	1500	31.3				
	1600	32.7				
	1665	33.6				
T12R Pure copper	**T K**	**k W/mK**				
	0	0				
	10	19600				
	20	10500				
	30	4300				
	40	2050				
	50	1220				
	60	850				
	70	670				
	80	570				
	90	514				
	100	483				
	150	428				
	200	413				
	273.1	401				
	300	398				
	350	394				
	400	392				
	500	388				
	600	383				

	700	377
	800	371
	900	364
	1000	357
	1100	350
	1200	342
	1300	334
	1356	330
T12R Pure Aluminium	**T K**	**k W/mK**
	0	0
	10	23500
	20	11700
	30	5180
	40	2380
	50	1230
	60	754
	70	532
	80	414
	90	344
	100	302
	150	248
	200	237
	273.1	236
	300	237
	350	394
	400	340
	500	237
	600	232
	700	226
	800	220
	900	213
	933.2	211

T270 C1	**T K**	**k W/mK**	**Cu 60.7**	**Zn 38.5**	**Mn 0.32**	**Sn 0.5**
Bronze bar 4	293	78.7				
	348	84.9				
	373	87.9				
	423	93.7				
	473	99.6				
	523	105.4				

T99 C1	**T K**	**k W/mK**	**Cu 90**	**Al 10**
Aluminium bronze	293.1	51		
	348.1	55.2		
	373.1	57.3		
	423.1	41.5		
	473.1	65.3		
	523.1	69.5		

T100 C1	**T K**	**k W/mK**	**Cu 99.73**	**Ni 0.23**
cupro nickel	293	321		
	473	324.3		

T100 C49	**T K**	**k W/mK**	**Cu 99.96**	**Ni 0.30**
cupro nickel	273.2	345.6		
	353.2	352		
	373.2	353.5		
	403.2	356		

T100 C50	**T K**	**k W/mK**	**Cu 99.96**	**Ni 0.30**
cupro nickel	273.2	331.8		
	353.2	338.9		
	373.2	340.6		

	403.2	343.1		
T118 C15-16	**T K**	**k W/mK**	**Cu 70**	**Zn 30**
Brass 70/30	108.2	74.5		
	123.2	77.8		
	135.2	80.8		
	152.2	83.3		
	170.2	88.7		
	188.2	92.5		
	204.2	95.4		
	238.2	100.4		
	260.2	105.4		
	281.2	105.9		
	296.2	109.6		
	298.2	109.2		
	351.2	127.6		
	367.2	128		
	372.2	132.6		
	386.2	130.1		
	394.2	135.1		
	413.2	140.6		
	437.2	142.3		
	448.2	141		
	481.2	143.1		
	511.2	149.4		
	541.2	143.1		
	571.2	145.6		
	610.2	149.4		
	634.2	150.2		
	648.2	147.7		
	703.2	146.4		
T363 C1	**T K**	**k W/mK**		
GaAs (solar cell)	90	39.7		
	100	39.1		
	110	38.4		
	120	37.8		
	130	37.2		
	140	36.8		
	150	36.4		
	160	36.1		
	170	35.8		
	180	35.6		
	190	35.3		
	200	35.1		
	210	34.9		
	220	34.6		
	230	34.4		
	240	34.3		
	250	34.1		
	260	34		
	270	33.7		
	280	33.6		
	290	33.5		
	300	33		
T50R	**T K**	**k W/mK**		
Single crystal silisium	10	2110		
pure	20	4940		
	30	4810		
	40	3530		
	50	2680		
	60	2110		
	70	1680		

	80	1340
	90	1080
	100	884
	150	409
	200	264
	250	191
	273.1	168
	300	148
	350	119
	400	98.9
	500	76.2
	600	61.9
	700	50.8
	800	42.2
	900	35.9
	1000	31.2
	1100	27.9
	1200	25.7
	1300	24.4
	1400	23.5
	1500	22.7
	1600	22.1
	1685	22
T52R Pure silver	**T K**	**k W/mK**
	10	16800
	20	5100
	30	1930
	40	1050
	50	700
	60	550
	70	497
	80	471
	90	460
	100	450
	150	432
	200	430
	250	428
	273.1	428
	300	427
	350	424
	400	420
	500	413
	600	405
	700	397
	800	389
	900	382
	1000	374
	1100	366
	1200	358
	1234	355
T19 C1	**T K**	**k W/mK**
Gold 99.999% pure	10	2820
	20	1500
	30	760
	40	520
	50	420
	60	380
	70	358
	80	352
	90	348
	100	345
	150	335

	200	327
	250	320
	273.1	319
	300	315
	350	313
	400	312
	500	309
	600	304
	700	298
	800	292
	900	285
	1000	278
	1100	271
	1200	262
	1300	251
	1336.	247
T239R	**T K**	**k W/mK**
Platinium pure 99.999%	10	1230
	20	495
	30	215
	40	139
	50	109
	60	94.7
	70	86.2
	80	81.5
	90	78.9
	100	77.5
	150	74
	200	72.4
	250	71.7
	273.1	71.5
	300	71.4
	350	71.5
	400	71.6
	500	72.2
	600	73
	700	74.1
	800	75.5
	900	77
	1000	78.6
	1100	80.5
	1200	82.5
	1300	84.8
	1400	87.1
	1500	89.6
	1600	92.1
	1700	94.7
	1800	97.3

T287 C15	**T K**	**k W/mK**	**Ni 67.1**	**Cu 29.1**	**Fe 1.72**	**Al 0.04**	**Mn 0.98**	**Si 0.01**	**Co 0.33**	**S 0.01**	**Mg 0.13**	**P 0.02**
Monel	273.1	21.3										
	373.1	24.3										
	473.1	27.6										
	573.1	30.5										
	673.1	33.5										
	773.1	36.8										
	873.1	39.8										
	973.1	42.9										
	1073.	46										
2T367 C1	**T K**	**k W/mK**										
Cotton fabric	313.2	0.0753										
2T367 C1	T K	k W/mK										

Cotton slicate	400.9	0.0582
	402.7	0.0715
	527.2	0.0885
	582.2	0.1074
2T368	T K	k W/mK
Cotton wool	92.2	0.0138
	136	0.017
	155.9	0.0211
	188	0.0256
	204.4	0.0275
	217.2	0.0298
	251.2	0.036
	263.5	0.0384
	278	0.0406
	286.2	0.0422
2T373	**T K**	**k W/mK**
Asbestos cement board	338.7	0.77
	366.5	0.757
	394.3	0.749
	422.1	0.742
	449.8	0.739
	477.6	0.736
	505.4	0.736
	533.2	0.736
	560.9	0.733
	588.7	0.731
2T374	**T K**	**k W/mK**
Asphalt-Glass wool	222.2	0.0304
	233.7	0.0326
	255.4	0.0348
	272.2	0.0369
	288.9	0.0394
	305.5	0.0414
2T375	T K	k W/mK
cardboard	273.2	0.138
density 0.79 g/cm^3	293.2	0.143
	323.2	0.167
2T376	**T K**	**k W/mK**
lignin impregnated	222	0.241
	238.7	0.247
	255.4	0.251
	272.2	0.255
	288.9	0.261
	305.5	0.265
2T380	**T K**	**k W/mK**
Plywood	222	0.101
density 0.552 g/cm^3	238.7	0.104
	255.4	0.107
	272.2	0.111
	288.9	0.114
	305.5	0.117
2T381 C34	**T K**	**k W/mK**
Fiberglass	260.7	0.036
density 0.033 g/cm^3	270.2	0.0389
	275.7	0.0395
2T381 C36	T K	k W/mK
Fiberglass	263.2	0.0291
density 0.12 g/cm^3	270.7	0.03
	275.2	0.0307
	280.2	0.0314
2T381 C37	**T K**	**k W/mK**
Fiberglass	260.7	0.0291

density 0.088 g/cm^3	271.7	0.0325
	277.2	0.0316
	281.6	0.0322
	286.2	0.0331
2T382	**T K**	**k W/mK**
glass fiber board	289.1	0.0499
density 0.215 g/cm3	289.7	0.0495
	290.1	0.0494
	311.2	0.0324
	311.6	0.0524
	311.7	0.0518
	315.7	0.053
2T385	**T K**	**k W/mK**
Paper	294.7	0.18
	297.2	0.18
	310.2	0.184
	321.2	0.184
	335.2	0.186
	339.2	0.18
	353.2	0.186
	359.2	0.182
	360.2	0.184
	361.7	0.186
	365.2	0.184
	385.2	0.18
2T387 C1	**T K**	**k W/mK**
Vegetable fibreboard	222	0.0495
0.75 in thick 0.251 g/cm3	238	0.0516
moisture 24.1 %	255.4	0.0542
	272.2	0.0565
	288.9	0.0588
	305.5	0.0611
2T387 C2	**T K**	**k W/mK**
Vegetable fibreboard	222	0.0424
0.75 in thick 0.251 g/cm3	238	0.0441
moisture 3.1 %	255.4	0.046
	272.2	0.0477
	288.9	0.0495
	305.5	0.0513
2T387 C3	**T K**	**k W/mK**
Vegetable fibreboard	222	0.0411
0.75 in thick 0.251 g/cm3	238.7	0.043
moisture 0 %	255.4	0.0447
	272.2	0.0466
	288.9	0.0483
	305.5	0.0502
2T389 C1	**T K**	**k W/mK**
wood fiber blanket	222	0.0286
density 0.056 g/cm3	238.7	0.0304
	255.4	0.0322
	272.2	0.034
	288.9	0.0361
	305.5	0.0384
2T389 C2	**T K**	**k W/mK**
wood fiber mat	222	0.027
density 0.056 g/cm3	238.7	0.0293
	255.4	0.0316
	272.2	0.034
	288.9	0.0363
	305.5	0.0392
2T390	**T K**	**k W/mK**
wool felt	313.2	0.0623

SG 0.15	343.2	0.0732
2T393 C1	**T K**	**k W/mK**
Mineral fiber	297.1	0.042
0.0264 g/cm3	311	0.0453
	324.8	0.0494
2T393 C2	**T K**	**k W/mK**
Mineral fiber	297.1	0.0398
0.0561 g/cm3	311	0.0428
	324.8	0.0463
2T393 C3	**T K**	**k W/mK**
Mineral fiber	297.1	0.0375
0.0396 g/cm3	311	0.0404
	324.8	0.0438
2T393 C4	**T K**	**k W/mK**
Mineral fiber	297.1	0.0361
0.0529 g/cm3	311	0.0389
	324.8	0.0418
2T393 C5	**T K**	**k W/mK**
Mineral fiber	297.1	0.0378
0.0681 g/cm3	311	0.0401
	324.8	0.0433
2T393 C6	**T K**	**k W/mK**
Mineral fiber	297.1	0.0362
0.112 g/cm3	311	0.0388
	324.8	0.0412
2T394 C5	**T K**	**k W/mK**
Mineral wool	222	0.0366
density 0.252 g/cm3	238.7	0.0387
	255.4	0.0405
	272.2	0.0424
	288.9	0.0444
	305.5	0.0464
2T394 C6	**T K**	**k W/mK**
Mineral wool	222	0.0346
density 0.232-0.264	238.7	0.0361
	255.4	0.0389
	272.2	0.0404
	288.9	0.0418
	305.5	0.0464
2T400	**T K**	**k W/mK**
Bitumen	292.1	0.166
	355.9	0.173
2T402	**T K**	**k W/mK**
charcoal	293	0.36
2T264	T K	k W/mK
olivine Basalt	209.3	0.142
density 1.49 g/cm3	249.9	0.254
	267.1	0.165
	279.9	0.246
	338.2	0.157
	367.1	0.228
2T268	**T K**	**k W/mK**
dolomite	484	1.1
	521	1.08
	565	1.36
	619	1.37
	678	1.24
	729	1.25
	773	1.25
	835	1.14
	888	0.983
	932	1.09

	970	1.16
	1051	0.699
	1098	0.64
	1173	0.356
	1208	0.452
2T269	**T K**	**k W/mK**
earth	293.2	0.186
	333.2	0.199
	403.2	0.234
	503.2	0.289
	618.2	0.337
	708.2	0.383
	813.2	0.423
2T271 C4	**T K**	**k W/mK**
granite	330	1.51
	383	1.63
	448	1.76
	483	1.87
	573	1.76
	658	2.01
	728	1.74
	793	1.74
	811	1.52
	873	1.64
	943	1.8
2T272	**T K**	**k W/mK**
limestone	392.1	1.43
	395.9	1.42
	450.4	1.41
	472	1.19
	512	1.2
	513	1.28
	546	1.15
	553	1.21
	610	1.21
	683	1.19
	736	1.12
	813	1.11
	878	1.11
	952	1.12
	1013	1.07
	1075	1.03
	1181	0.619
	1253	0.573
	1324	0.54
2T273	**T K**	**k W/mK**
mica	92.2	0.0306
	130.3	0.0374
	155.2	0.0396
	176.2	0.0436
	192.2	0.0478
	206.2	0.0509
	216.4	0.053
	225.2	0.0554
	234.2	0.0571
	248.2	0.0602
	260.2	0.0635
	265.2	0.0647
	275.2	0.0656
	288.9	0.0659
	305.5	0.0687
2T274	**T K**	**k W/mK**

expanded perlite	170.7	0.025
density 0.048	228.2	0.0325
	283.2	0.0395
2T275		
rock	287.3	0.606
consisted of fine grade	**T K**	**k W/mK**
2T276	332	2.91
salt	375	2.83
	439	2.89
	464	2.78
	496	2.6
	532	2.65
	600	2.51
2T277 C60	**T K**	**k W/mK**
sand	180.4	0.209
	212.6	0.243
	235.4	0.296
	279.3	0.172
	279.9	0.138
	303.8	0.236
	312.1	0.22
	332.1	0.222
	342.6	0.246
2T278	**T K**	**k W/mK**
sandstone	303.2	0.0264
2T302	T K	k W/mK
nylon	357.9	0.0138
2T303	**T K**	**k W/mK**
phenolic resin	293.5	0.3
2T305 C7	**T K**	**k W/mK**
Polyvinylchloride (PVC)	118.6	0.15
	123.2	0.15
	127.3	0.151
	131.8	0.151
	136.1	0.153
	139.6	0.153
	144.2	0.154
	148.1	0.154
	151.5	0.155
	156.3	0.156
	158.9	0.156
	162.6	0.157
	166.6	0.157
	169.5	0.158
	173.2	0.158
	177.6	0.158
	181	0.158
	183.6	0.159
	188	0.159
	190.7	0.159
	193.7	0.16
	197.5	0.161
	201.2	0.161
	203.7	0.161
	207.2	0.161
	209.8	0.161
	213.4	0.162
	216.3	0.162
	219.4	0.162
	223.1	0.162
	231.2	0.162
	233.4	0.162

	236.9	0.161
	242.7	0.162
	245.3	0.161
	247.9	0.16
	252.8	0.16
	253.7	0.16
	258.8	0.16
	260.9	0.159
	263.7	0.159
	266.6	0.159
	268.8	0.158
	271.5	0.157
	272.5	0.158
	276.4	0.157
	278.7	0.157
	281.7	0.157
	283.9	0.156
	286.6	0.156
	289.1	0.156
	291.3	0.155
	294.2	0.155
	296.8	0.155
	298.8	0.155
	301.4	0.155
	304.2	0.154
	306.1	0.154
	308.4	0.154
	310.7	0.153
	312.7	0.153
	316.7	0.153
	317.3	0.153
	320.7	0.152
	322.7	0.152
	324.8	0.152
	326.8	0.151
	329.4	0.151
	332.7	0.151
	335.1	0.15
	336.8	0.151
	339.4	0.15
	341.7	0.15
	345.1	0.15
	349.1	0.15
	350.6	0.149
	352.7	0.149
	354.5	0.148
	356.7	0.148
	359.4	0.148
	362.1	0.148
2T306 C1	**T K**	**k W/mK**
Polyethylene MW 21000	305.2	0.28
	317.2	0.278
	322.2	0.268
	333.7	0.264
	343.7	0.268
	352.7	0.276
	362.7	0.274
2T306 C2	**T K**	**k W/mK**
Polyethylene MW 70000-	293.2	0.349
	313.2	0.331
	319.2	0.32
	326.2	0.316

	328.2	0.314
	333.2	0.31
	344.7	0.301
	349.7	0.295
	360.2	0.285
	371.2	0.272
2T308	**T K**	**k W/mK**
Plexiglass	183.3	0.105
polymethylmethacrylate	184.4	0.106
	244.8	0.134
	245.6	0.135
	300.3	0.154
	332.1	0.158
	332.2	0.157
2T309	**T K**	**k W/mK**
Polystyrene	249.2	0.109
	273.2	0.116
	306.2	0.126
	339.2	0.129
	350.2	0.118
	363.2	0.128
	373.2	0.129
	423.2	0.136
	466.2	0.139
2T316	**T K**	**k W/mK**
Teflon	358.7	0.42
	394.3	0.411
	439.3	0.401
	470.9	0.317
	481.5	0.346
	485.4	0.346
	527.6	0.352
	532.1	0.337
	553.2	0.375
	563.2	0.426
	572.1	0.404
2T311	**T K**	**k W/mK**
polytrifluorochloroethyle	317.6	0.146
	350.7	0.185
	370.9	0.211
	383.2	0.245
	390.9	0.232
	402.6	0.237
	408.2	0.249
	435.9	0.235
	437.6	0.251
	460.9	0.267
	464.6	0.248
2T312	**T K**	**k W/mK**
Rubber	109.2	0.0941
	113.2	0.1
	117.2	0.105
	143.2	0.117
	149.2	0.119
	156.2	0.118
	157.2	0.118
	171.2	0.123
	181.2	0.126
	193.2	0.13
	217.2	0.146
	239.2	0.14
	263.2	0.142

	285.2	0.159
	301.2	0.172

Values can also put into a program format. The following program uses cubic spline interpolation formula to fit temperature dependent curve fitting.

```
public class thermal_conductivity_CS_Data extends Object
{
//this class reads and store thermal conductivity data
public double a[][];
public String solidName;
public String additional_info;
double kmin,kmax;
public double kl[][]; //  thermal conductivity
public int n;
// order of data
// a[0] temperature degree K
//a [1] kl thermal conductivity
   public thermal_conductivity_CS_Data(String solidNamei,String additional_infoi,double ai[][])
   {
   solidName=solidNamei;
   a=ai;
   n=a[0].length;
   additional_info=additional_infoi;
   int n=a[0].length;
   int i,j;
   if(n>2) kl=cubic_spline(a[0],a[1],0.0,0.0);
   }
   public void additional_info(String s)
   {additional_info=s;}
  public static double [] thomas(double a[][],double r[])
  {
  int n=a.length;
  double f[]=new double[n];
  double e[]=new double[n];
  double g[]=new double[n];
  double x[]=new double[n];
  for(int i=0;i<n;i++)     {f[i]=a[i][i];}
  for(int i=0;i<(n-1);i++) {g[i]=a[i][i+1];}
  for(int i=0;i<(n-1);i++) {e[i+1]=a[i+1][i];}
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
     }
  for(int k=1;k<n;k++)
     {r[k]=r[k]-e[k]*r[k-1];
     }
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
     {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
  return x;
  }

  public static double [] thomas(double f[],double e[],double g[],double r[])
  {
  int n=f.length;
  double x[]=new double[n];
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
     }
  for(int k=1;k<n;k++)
     {r[k]=r[k]-e[k]*r[k-1];
     }
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
     {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
  return x;
  }
  //cubic spline curve fitting
  public static double [][] cubic_spline(double xi[],double yi[],double c0,double cn)
  {
  int n=xi.length;
  double h[]=new double[n];
  double w[]=new double[n];
```

```
 double f[]=new double[n];
 double e[]=new double[n];
 double g[]=new double[n];
 double d[]=new double[n];
 double x[]=new double[n];
 double S[][]=new double[4][n];
 int k;
 for(k=0;k<(n-1);k++)
    {h[k]=xi[k+1]-xi[k];
     w[k]=(yi[k+1]-yi[k])/h[k];
                }
 d[0]=c0;
 d[n-1]=cn;
 for(k=1;k<(n-1);k++)
    {d[k]=6.0*(w[k]-w[k-1]);}
 f[0]=1.0;
 f[n-1]=1.0;
 g[0]=0.0;
 g[n-1]=0.0;
 e[0]=0.0;
 e[n-1]=0.0;
 for(k=1;k<(n-1);k++)
 {f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];}
 S[2]=thomas(f,e,g,d);
 S[3]=xi;
 for(k=0;k<(n-1);k++)
   {S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
    S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
   }
 return S;
 }
 //cubic spline curve fitting function
 public static double funcSpline(double S[][],double x)
 {
 int n=S[0].length;
 double xx1=0;
 double xx2=0;
 double y=0;
 double hk=0;
 for(int k=0;k<(n-1);k++)
    {if(S[3][k]<=x && x<=S[3][k+1])
     {hk=(S[3][k+1]-S[3][k]);
      xx1=(x-S[3][k]);
      xx2=(S[3][k+1]-x);
      y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
      break;
      }
    }
      if(y==0 && S[3][n-2]<=x )
      {
                int k=n-2;
                hk=(S[3][k+1]-S[3][k]);
      xx1=(x-S[3][k]);
      xx2=(S[3][k+1]-x);
      y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
      }
 return y;
 }
//Cubic spline interpolation output
public static double[][]  funcSpline(double xi[],double yi[],int numberofmidpoints)
{
              //numberofmidpoints:in  x--o--o--x--o--o--x if x is x
              // are experimental points number of midpoints are 2
              int n=xi.length;
              int nn=(n-1)*(numberofmidpoints+1)+1;
              double z[][]=new double[2][nn];
   double S[][]=cubic_spline(xi,yi,0,0);
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
     {  z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);k++;
        for(int j=0;j<numberofmidpoints;j++)
           {dx=(xi[i+1]-xi[i])/((double)numberofmidpoints+1.0);
        z[0][k]=z[0][k-1]+dx;z[1][k]=funcSpline(S,z[0][k]);k++;}
     }
     z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);
```

```
    return z;
  }
  // derivative formulas
 // =================
  public static double dfSpline(double xi[],double yi[],double c0,double cn,double x)
  { //kübik şerit türev formülü
              double S[][]=cubic_spline(xi,yi,c0,cn);
    return dfSpline(S,x);
  }

  public static double dfSpline(double S[][],double x)
  {
    //cubic spline derivative formula
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y=0;
  double hk=0;
  for(int k=0;k<(n-1);k++)
     {if(S[3][k]<=x && x<=S[3][k+1])
      {hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]-S[1][k]+(xx1*xx1*S[2][k+1]-xx2*xx2*S[2][k])/(2.0*hk);
       break;
       }
     }
       if(y==0 && S[3][n-2]<=x )
       {
                int k=n-2;
                hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
       }
  return y;
  }

  public static double  dfSpline(double xi[],double yi[],double x)
  {
    //natural cubic spline derivative formula
              double S[][]=cubic_spline(xi,yi,0,0);
    return dfSpline(S,x);
  }

  public static double intSpline(double S[][],double a,double b)
  {
  //cubic spline function integral
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y1,y2;
  double hk=0;
  double toplam=0;
  y1=0;y2=0;
  for(int k=0;k<(n-1);k++)
     { hk=(S[3][k+1]-S[3][k]);
      if(a>S[3][k+1])
      {toplam=0;
      }//condition 1
              else if(S[3][k]<=a && a<=S[3][k+1] && b>S[3][k+1])//şart 2
       {
       xx1=(a-S[3][k]);
       xx2=(S[3][k+1]-a);
       y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       xx1=hk;
       xx2=0;
       y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       toplam+=(y2-y1);
       }
       else if(S[3][k]<=a && a<=S[3][k+1] && S[3][k]<=b && b<=S[3][k+1])//şart 3
       { //condition 3
       xx1=(a-S[3][k]);
       xx2=(S[3][k+1]-a);
       y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       xx1=(b-S[3][k]);
       xx2=(S[3][k+1]-b);
```

```
    y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
    toplam+=(y2-y1);
    }
    else if(a<S[3][k] && b>=S[3][k+1]) //condition 4
    {
    xx1=0;
    xx2=hk;
    y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
    xx1=hk;
    xx2=0;
    y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
    toplam+=(y2-y1);
    }
    else if(S[3][k]<=b && b<=S[3][k+1])//condition 5
    {xx1=0;
     xx2=hk;
     y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
     xx1=(b-S[3][k]);
     xx2=(S[3][k+1]-b);
     y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
     toplam+=(y2-y1);
     }
     else break;
  }
return toplam;
}

public static double s0(double S[][],double a,double b)
{
// cubic spline function integral of (S(x)/x)
int n=S[0].length;
double xx1=0;  //x
//ak S[0][k]
//bk S[1][k]
//ck S[2][k]
//xk s[3][k]
double xx2=0;
double y1,y2;
double hk=0;
double toplam=0;
y1=0;y2=0;
double xk1=0,xk=0,ak=0,bk=0,ck=0,ck1=0;
for(int k=0;k<(n-1);k++)
  { hk=(S[3][k+1]-S[3][k]);
    xk=S[3][k];
    xk1=S[3][k+1];
    ak=S[0][k];
    bk=S[1][k];
    ck=S[2][k];
    ck1=S[2][k+1];
   if(a>xk1)
   {toplam=0;
   }//şart1
            else if(xk<=a && a<=xk1 && b>xk1)//şart 2
    {
    xx1=(xk1-a);
    xx2=(xk1/a);
    y1=(ak-bk)*xx1+(bk*xk1-ak*xk)*Math.log(xx2);
    xx1=xk1;
    xx2=a;
    y1+=ck1*((xx1*xx1*xx1-xx2*xx2*xx2)/3.0-
    1.5*(xx1*xx1-xx2*xx2)*xk+
    3.0*xk*xk*(xx1-xx2)-
    xk*xk*xk*Math.log(xx1/xx2))/6/hk;
    y1+=ck*(xk1*xk1*xk1*Math.log(xx1/xx2)-
    3.0*xk1*xk1*(xx1-xx2)+1.5*xk1*(xx1*xx1-xx2*xx2)-
    1.0/3.0*(xx1*xx1*xx1-xx2*xx2*xx2))/6/hk;
    toplam+=y1;
    }
    else if(S[3][k]<=a && a<=S[3][k+1] && S[3][k]<=b && b<=S[3][k+1])//şart 3
    {
    xx1=(b-a);
    xx2=(b/a);
     y1=(ak-bk)*xx1+(bk*xk1-ak*xk)*Math.log(xx2);
    xx1=b;
    xx2=a;
    y1+=ck1*((xx1*xx1*xx1-xx2*xx2*xx2)/3.0-
```

```
      1.5*(xx1*xx1-xx2*xx2)*xk+
      3.0*xk*xk*(xx1-xx2)-
      xk*xk*xk*Math.log(xx1/xx2))/6/hk;
      y1+=ck*(xk1*xk1*xk1*Math.log(xx1/xx2)-
      3.0*xk1*xk1*(xx1-xx2)+1.5*xk1*(xx1*xx1-xx2*xx2)-
      1.0/3.0*(xx1*xx1*xx1-xx2*xx2*xx2))/6/hk;
      toplam+=y1;
      }
      else if(a<S[3][k] && b>=S[3][k+1]) //şart 4
      {
      xx1=(xk1-xk);
      xx2=(xk1/xk);
       y1=(ak-bk)*xx1+(bk*xk1-ak*xk)*Math.log(xx2);
      xx1=xk1;
      xx2=xk;
      y1+=ck1*((xx1*xx1*xx1-xx2*xx2*xx2)/3.0-
      1.5*(xx1*xx1-xx2*xx2)*xk+
      3.0*xk*xk*(xx1-xx2)-
      xk*xk*xk*Math.log(xx1/xx2))/6/hk;
      y1+=ck*(xk1*xk1*xk1*Math.log(xx1/xx2)-
      3.0*xk1*xk1*(xx1-xx2)+1.5*xk1*(xx1*xx1-xx2*xx2)-
      1.0/3.0*(xx1*xx1*xx1-xx2*xx2*xx2))/6/hk;
      toplam+=y1;
      }
      else if(S[3][k]<=b && b<=S[3][k+1])//şart 5
      {       xx1=(b-xk);
      xx2=(b/xk);
      y1=(ak-bk)*xx1+(bk*xk1-ak*xk)*Math.log(xx2);
      xx1=b;
      xx2=xk;
      y1+=ck1*((xx1*xx1*xx1-xx2*xx2*xx2)/3.0-
      1.5*(xx1*xx1-xx2*xx2)*xk+
      3.0*xk*xk*(xx1-xx2)-
      xk*xk*xk*Math.log(xx1/xx2))/6/hk;
      y1+=ck*(xk1*xk1*xk1*Math.log(xx1/xx2)-
      3.0*xk1*xk1*(xx1-xx2)+1.5*xk1*(xx1*xx1-xx2*xx2)-
      1.0/3.0*(xx1*xx1*xx1-xx2*xx2*xx2))/6/hk;
      toplam+=y1;
       }
       else break;
    }
  return toplam;
  }

  public static double intSpline(double xi[],double yi[],double c0,double cn,double a,double b)
  {double S[][]=cubic_spline(xi,yi,c0,cn);
   return intSpline(S,a,b);
  }

   public double k_l(double t)
   {   if(n==1) return a[1][0];
       else if(n==2)
       {double R=(t-a[0][0])/(a[0][1]-a[0][0]);
        return a[1][0]+R*(a[1][1]-a[1][0]);}
                  else
                  { if(t<a[0][0]) return a[1][0];
                    else if(t>a[0][n-1]) return a[1][n-1];
                    else{
          return funcSpline(kl,t);}
       }
    }
}
```

```
//=========================================================
// java Termodynamic- heat transfer package
// Class thermal_conductivity_CS thermal conducticity of solids
// Dr. Turhan Coban
// Ege Üniversitesi Mühendislik fakultesi
// Makina Bölümü
//
//=========================================================
// program name   : thermalconductivity_CS.java
// =========================================================

import java.io.*;
import java.net.*;
```

```
import java.util.*;
import javax.swing.*;

public class thermal_conductivity_CS
{ private static final long serialVersionUID =76986793L;
String s[];
public thermal_conductivity_CS_Data rd;
public String solidName;
public String additional_info;

public thermal_conductivity_CS(String rName)
{
String s2[]={
"C9 Carbon Steel 3",
"C129 British steel 7",
"C124 12 Forged steel",
"C127 26 carbon  steel",
"C132 British carbon steel 7",
"T328 C26 chromium steel",
"T328 C32 chromium steel",
"T330 C2 chromium-nickel stainless steel SS304",
"T331 C3 AISI 301 SS stainless steel",
"T331 C4 AISI 316 SS stainless steel",
"T331 C5 AISI 347 SS stainless steel",
"T331 C64 AISI 316 3A SS stainless steel",
"T331 C65 AISI 316 4A SS stainless steel",
"TS331R C A SS304 stainless steel",
"TS331R C B SS347 stainless steel",
"T12R Pure copper %99.999",
"T12R Pure Aluminium %99.999",
"T270 C1 Bronze bar 4",
"T99 C1 Aluminium bronze",
"T100 C1 cupro nickel",
"T100 C50 cupro nickel",
"T118 C15-16 Brass 70/30",
"T363 C1 GaAs(solar cell)",
"T50R Single crystal silisium pure",
"T52R Pure silver",
"T19 C1 Gold 99.999% pure",
"T23R Platinium pure 99.999%",
"T287 C15 Monel",
"2T367 C1 Cotton fabric",
"2T367 C1 cotton slicate",
"2T368 Cotton wool",
"2T373 Asbestos cement board",
"2T374 Asphalt-Glass wool",
"2T375 cardboard density 790 kg/m^3",
"2T376 lignin impregnated fireboard",
"2T380 Plywood density 552 kg/m^3",
"2T381 C34 Fiberglass density 33 kg/m^3",
"2T381 C36 Fiberglass density 120 kg/m^3",
"2T381 C37 Fiberglass density 88 kg/m^3",
"2T382 glass fiber board density 215 kg/m^3",
"2T385 paper",
"2T387 C1 Vegetable fibreboard 0.75 in thick 251 kg/m^3 moisture 24.1 %",
"2T387 C1 Vegetable fibreboard 0.75 in thick 251 kg/m^3 moisture 3.1 %",
"2T387 C1 Vegetable fibreboard 0.75 in thick 251 kg/m^3 moisture 0 %",
"2T389 C1 wood fiber blanket density 56 kg/m^3",
"2T389 C2 wood fiber mat density 56 kg/m^3",
"2T390 wood felt SG 0.15",
"2T393 C1 mineral fiber density 26.4 kg/m^3",
"2T393 C2 mineral fiber density 56.1 kg/m^3",
"2T393 C3 mineral fiber density 39.6 kg/m^3",
"2T393 C4 mineral fiber density 52.9 kg/m^3",
"2T393 C5 mineral fiber density 68.1 kg/m^3",
"2T393 C6 mineral fiber density 112 kg/m^3",
"2T394 C5 mineral wool density 252 kg/m^3",
"2T394 C5 mineral wool density 232-264 kg/m^3",
"2T400 Bitumen",
"2T402 charcoal",
"2T264 olivine basalt density 1490 kg/m^3",
"2T268 dolomite",
"2T269 earth",
"2T271 C4 granite",
"2T272 limestone",
"2T273 mica",
"2T274 expanded perlite density 48 kg/m^3",
```

```
"2T275 rock consisted fine grade quartz diorite",
"2T276 salt",
"2T277 C60 sand",
"2T278 sandstone",
"2T302 nylon",
"2T303 phenolic resin",
"2T305 C7 polyvinylcholoride",
"2T306 C1 Polyethylene MW 21000",
"2T306 C2 Polyethylene MW 70000-80000",
"2T308 Plexiglass (polymethylmethacrylate)",
"2T309 Polystyrene",
"2T316 teflon",
"2T311 polytrifluorochloroethylene",
"2T312 rubber"};
s=s2;
readref(rName);
}
public String[] list()
{ return s;}

public String readNames()
{ String s1="";
  for(int i=0;i<s.length;i++) {s1+=s[i]+"\n";}
  return s1;
}

public boolean readref(String rName)
{  String s="";
   boolean b2=false;
   if(readref1(rName))      {b2=true;}
   solidName=rd.solidName;
   additional_info=rd.additional_info;
   return b2;
}
public boolean readref1(String rName)
{
boolean b1=false;
if(rName.equals("C9 Carbon Steel 3"))
{double a[][]={{ 273.15,373.2,473.2,573.2,673.2,773.2},
{51,47.7,44.8,41.8,39.2,38.1}};
rd=new thermal_conductivity_CS_Data("C9 Carbon Steel 3","C 0.57          Mn 0.55    Si 0.38",a);
b1=true;}
else if(rName.equals("C129 British steel 7"))
{double
a[][]={{273.15,323.15,373.15,423.15,473.15,523.15,573.15,623.15,673.15,723.15,773.15,823.15,873.15,923.15,973.15,993.15,1013.15,10
23.15,1073.15,1123.15,1173.15,1223.15,1273.15},
{49.8,49.4,48.3,46.9,45.2,43.1,41.4,39.8,38.1,36.6,35.4,34.1,29.9,31.6,30.3,29.7,23.9,23.9,24.3,24.7,25.1,25.9,26.8}};
rd=new thermal_conductivity_CS_Data("C129 British steel 7",
"C 0.8      Cu 0.07     Mn 0.32    P 0.08      S 0.09      Si 0.13     Ni0.13      Cr 0.11     As0.021",a);
b1=true;}
else if(rName.equals("C124 12 Forged steel"))
{double
a[][]={{283.15,303.15,391.15,420.15,469.15,521.15,527.15,641.15,690.15,796.15,851.15,912.15,967.15,999.15,1044.15,1100.15,1134.15,
1199.15},
{43.9,43.9,43.9,43.9,43.5,43.1,41.4,41,41,36.8,35.8,36,38.5,29.7,28.9,30.5,28.9,31.8,}};
rd=new thermal_conductivity_CS_Data("C124 12 Forged steel",
"C0.64     Cu0.09     Mn0.27     P0.01      S 0.028     Si 0.06",a);
b1=true;}
else if(rName.equals("C127 26 carbon  steel"))
{double
a[][]={{283.15,303.15,391.15,420.15,469.15,521.15,527.15,641.15,690.15,796.15,851.15,912.15,967.15,999.15,1044.15,1100.15,1134.15,
1199.15},
{43.9,43.9,43.9,43.9,43.5,43.1,41.4,41,41,36.8,35.8,
36,38.5,29.7,28.9,30.5,28.9,31.8}};
rd=new thermal_conductivity_CS_Data("C127 26 carbon  steel",
"C1.3      Cu0.06     Mn0.4      P0.046     S0.018     Si0.08",a);
b1=true;}
else if(rName.equals("C132 British carbon steel 7"))
{double a[][]={{273.15,323.15,373.15,423.15,473.15,523.15,573.15,623.15},
{49.8,49.4,48.3,46.9,45.1,43.1,41.4,40.1}};
rd=new thermal_conductivity_CS_Data("C132 British carbon steel 7",
"C0.8      Cu0.07     Mn0.32     P0.008     S0.009     Si0.13     Ni 0.13     Cr 0.11     As 0.021   Mo <0.01  Al 0.004",a);
b1=true;}
else if(rName.equals("T328 C26 chromium steel"))
{double a[][]={{273.15,323.15,373.15,423.15,473.15,523.15,573.15,623.15},
{27,27.2,27.6,27.6,27.8,27.8,28,28}};
rd=new thermal_conductivity_CS_Data("T328 C26 chromium steel",
```

```
"Cr12.95  C0.13      Mn0.25    Ni0.14     P0.018     S0.024     Si 0.17     Al 0.034   Cu 0.06    V 0.012    As 0.015",a);
b1=true;}
else if(rName.equals("T330 C2 chromium-nickel stainless steel SS304"))
{double
a[][]={{273.2,373.2,405.9,423.2,444.1,457.2,463.2,473.2,483.3,512.2,523.2,534.2,547.8,583.7,599.6,614.3,637.8,670.5,678.7,695,757.7,77
3.2,839,851,904.1,920.5,934.1,991.4,1008},
{14.9,16.3,17.2,17.6,17.8,17.9,18.1,18.1,18.4,18.8,18.8,19.2,19.3,19.7,19.9,20.1,20.5,20.7,21,21.2,22,22.4,23.1,23.3,24.2,24.5,24.8,25.4,25
.9}};
rd=new thermal_conductivity_CS_Data("T330 C2 chromium-nickel stainless steel SS304",
"Cr 17-19  Ni 7-9      C 0.11",a);
b1=true;}
else if(rName.equals("T331 C3 AISI 301 SS stainless steel"))
{double a[][]={{136.8,184,225,266,329.8,355.5,380.3,405,536.4,602.5,722.3,831.8,887.8,1032.4,1038.8,1216.7},
{10.9,13,13.4,14.2,16.3,16.7,17.6,16.7,18.8,19.7,20.9,22.6,23.4,25.5,25.1,26.4}};
rd=new thermal_conductivity_CS_Data("T331 C3 AISI 301 SS stainless steel",
"Cr 16.82  Ni 11.66   C 0.108    Mn 1.59    Mo 2.18    P 0.018    Si 0.26     S 0.023",a);
b1=true;}
else if(rName.equals("T331 C4 AISI 316 SS stainless steel"))
{double a[][]={{135.7,184.4,228.9,270,367.8,433.8,482.1,497.8,552.5,560.9,592.5,660.7,759.8,812.6,825.6,944.7,1031.5,1114,1220.4},
{10.5,11.3,12.6,13.4,14.6,15.5,15.1,15.9,15.9,18,18.4,18.8,20.1,20.5,20.9,22.2,23.4,23.8,25.1}};
rd=new thermal_conductivity_CS_Data("T331 C4 AISI 316 SS stainless steel",
"Cr 16.82  Ni 11.66   C 0.108    Mn 1.59    Mo 2.18    P 0.018    Si 0.26     S 0.023",a);
b1=true;}
else if(rName.equals("T331 C5 AISI 347 SS stainless steel"))
{double
a[][]={{137.7,185.4,228.6,269.2,361.9,427.9,429.5,490.8,540.7,550.1,552.1,576.5,662.8,741.5,768.9,801,919.1,996.9,1079.9,1173.6},
{11.7,13,14.2,14.6,15.1,15.9,16.7,16.7,18.4,17.6,16.7,18.8,18.8,20.5,20.1,21.3,23.4,23.4,24.7,25.9}};
rd=new thermal_conductivity_CS_Data("T331 C5 AISI 347 SS stainless steel",
"Cr 17.65  Ni 10.94   C 0.06     Co 0.09    Mn1.64     Mo0.02     P 0.013    Nb0.73     Si 0.58     S 0.017",a);
b1=true;}
else if(rName.equals("T331 C64 AISI 316 3A SS stainless steel"))
{double
a[][]={{273.7,334.8,343.7,388.7,391.5,413.7,417.6,430.9,431.5,457.1,460.9,481.5,500.9,514.3,520.4,589.8,605.4,613.7,659.3,670.4,679.8,
712.6,722.1,799.8,800.4,897.1,958.2,982.6,1113.2,1185.9},
{13,14.2,14.6,15.4,15.4,15.8,15.7,16.5,15.9,16.3,16.5,17.1,17.3,17.5,17.5,18.9,19.6,18.9,19.6,19.5,19.9,20,20.4,22.1,22,23.6,24.4,24.6,26.5
,27.7}};
rd=new thermal_conductivity_CS_Data("T331 C64 AISI 316 3A SS stainless steel",
"Cr 17.45  Ni12.60    C0.063     Cu0.09     Mn 1.59    Mo 2.55    P 0.023    Si 0.6      Co 0.19    S 0.017",a);
b1=true;}
else if(rName.equals("TS331R C A SS304 stainless steel"))
{double a[][]={{25,75,100,150,200,250,273.15,300,350,400,450,500,600,700,800,900,1000,1100,1200,1300,1400,1500,1600,1665},
{2.7,5.8,8,9.5,11.5,13,14.2,14.7,15.2,16.2,17,17.7,18.4,19.8,21.2,22.5,23.9,25.3,26.7,28.1,29.5,30.9,32.3,33.7,34.7}};
rd=new thermal_conductivity_CS_Data("TS331R C A SS304 stainless steel",
"Cr 18-20  Ni 8-12     Mn 2 max ",a);
b1=true;}
else if(rName.equals("TS331R C B SS347 stainless steel"))
{double a[][]={{25,50,75,100,150,200,250,273.15,300,350,400,450,500,600,700,800,900,1000,1100,1200,1300,1400,1500,1600,1665,},
{2.5,5.4,7.6,9.1,11.1,12.6,13.8,14.3,14.8,15.7,16.5,17.2,17.9,19.2,20.5,21.9,23.2,24.6,25.9,27.3,28.6,30,31.3,32.7,33.6}};
rd=new thermal_conductivity_CS_Data("TS331R C B SS347 stainless steel",
"Cr 17-19  Ni9-13     Mn 2 max Si 1 (mx)",a);
b1=true;}
else if(rName.equals("T12R Pure copper %99.999"))
{double a[][]={{10,20,30,40,50,60,70,80,90,100,150,200,273.15,300,350,400,500,600,700,800,900,1000,1100,1200,1300,1356},
{19600,10500,4300,2050,1220,850,670,570,514,483,428,413,401,398,394,392,388,383,377,371,364,357,350,342,334,330}};
rd=new thermal_conductivity_CS_Data("T12R Pure copper %99.999",
"Cu %99.999",a);
b1=true;}
else if(rName.equals("T12R Pure Aluminium %99.999"))
{double a[][]={{10,20,30,40,50,60,70,80,90,100,150,200,273.15,300,350,400,500,600,700,800,900,933.2},
{23500,11700,5180,2380,1230,754,532,414,344,302,248,237,236,237,394,340,237,232,226,220,213,211}};
rd=new thermal_conductivity_CS_Data("T12R Pure Aluminium %99.999",
"Al %99.999",a);
b1=true;}
else if(rName.equals("T270 C1 Bronze bar 4"))
{double a[][]={{293,348,373,423,473,523},
{78.7,84.9,87.9,93.7,99.6,105.4}};
rd=new thermal_conductivity_CS_Data("T270 C1 Bronze bar 4",
"Cu 60.7   Zn 38.5     Mn 0.32    Sn0.5",a);
b1=true;}
else if(rName.equals("T99 C1 Aluminium bronze"))
{double a[][]={{293.15,348.15,373.15,423.15,473.15,523.15},
{51,55.2,57.3,41.5,65.3,69.5}};
rd=new thermal_conductivity_CS_Data("T99 C1 Aluminium bronze","Cu 90 Al 10",a);
b1=true;}
else if(rName.equals("T100 C1 cupro nickel"))
{double a[][]={{293,473},
{321,324.3}};
```

```
rd=new thermal_conductivity_CS_Data("T100 C1 cupro nickel",
"Cu 99.73 Ni 0.23",a);
b1=true;}
else if(rName.equals("T100 C50 cupro nickel"))
{double a[][]={{273.2,353.2,373.2,403.2},
{345.6,352,353.5,356}};
rd=new thermal_conductivity_CS_Data("T100 C50 cupro nickel",
"Cu 99.96 Ni 0.303",a);
b1=true;}
else if(rName.equals("T118 C15-16 Brass 70/30"))
{double
a[][]={{108.2,123.2,135.2,152.2,170.2,188.2,204.2,238.2,260.2,281.2,296.2,298.2,351.2,367.2,372.2,386.2,394.2,413.2,437.2,448.2,481.2,
511.2,541.2,571.2,610.2,634.2,648.2,703.2},
{74.5,77.8,80.8,83.3,88.7,92.5,95.4,100.4,105.4,105.9,109.6,109.2,127.6,128,132.6,130.1,135.1,140.6,142.3,141,143.1,149.4,143.1,145.6,1
49.4,150.2,147.7,146.4}};
rd=new thermal_conductivity_CS_Data("T118 C15-16 Brass 70/30",
"Cu 70      Zn 30",a);
b1=true;}
else if(rName.equals("T363 C1 GaAs(solar cell)"))
{double a[][]={{90,100,110,120,130,140,150,160,170,180,190,200,210,220,230,240,250,260,270,280,290,300},
{39.7,39.1,38.4,37.8,37.2,36.8,36.4,36.1,35.8,35.6,35.3,35.1,34.9,34.6,34.4,34.3,34.1,34,33.7,33.6,33.5,33}};
rd=new thermal_conductivity_CS_Data("T363 C1 GaAs(solar cell)",
" ",a);
b1=true;}
else if(rName.equals("T50R Single crystal silisium pure"))
{double
a[][]={{10,20,30,40,50,60,70,80,90,100,150,200,250,273.15,300,350,400,500,600,700,800,900,1000,1100,1200,1300,1400,1500,1600,168
5},
{2110,4940,4810,3530,2680,2110,1680,1340,1080,884,409,264,191,168,148,119,98.9,76.2,61.9,50.8,42.2,35.9,31.2,27.9,25.7,24.4,23.5,22
.7,22.1,22}};
rd=new thermal_conductivity_CS_Data("T50R Single crystal silisium pure",
"Si % 99.999 ",a);
b1=true;}
else if(rName.equals("T52R Pure silver"))
{double a[][]={{10,20,30,40,50,60,70,80,90,100,150,200,250,273.15,300,350,400,500,600,700,800,900,1000,1100,1200,1234},
{16800,5100,1930,1050,700,550,497,471,460,450,432,430,428,428,427,424,420,413,405,397,389,382,374,366,358,355}};
rd=new thermal_conductivity_CS_Data("T52R Pure silver",
" ",a);
b1=true;}
else if(rName.equals("T19 C1 Gold 99.999% pure"))
{double a[][]={{10,20,30,40,50,60,70,80,90,100,150,200,250,273.15,300,350,400,500,600,700,800,900,1000,1100,1200,1300,1336.2},
{2820,1500,760,520,420,380,358,352,348,345,335,327,320,319,315,313,312,309,304,298,292,285,278,271,262,251,247}};
rd=new thermal_conductivity_CS_Data("T19 C1 Gold 99.999% pure",
" ",a);
b1=true;}
else if(rName.equals("T23R Platinium pure 99.999%"))
{double
a[][]={{10,20,30,40,50,60,70,80,90,100,150,200,250,273.15,300,350,400,500,600,700,800,900,1000,1100,1200,1300,1400,1500,1600,170
0,1800},
{1230,495,215,139,109,94.7,86.2,81.5,78.9,77.5,74,72.4,71.7,71.5,71.4,71.5,71.6,72.2,73,74.1,75.5,77,78.6,80.5,82.5,84.8,87.1,89.6,92.1,9
4.7,97.3}};
rd=new thermal_conductivity_CS_Data("T23R Platinium pure 99.999%",
" ",a);
b1=true;}
else if(rName.equals("T287 C15 Monel"))
{double a[][]={{273.15,373.15,473.15,573.15,673.15,773.15,873.15,973.15,1073.15},
{21.3,24.3,27.6,30.5,33.5,36.8,39.8,42.9,46}};
rd=new thermal_conductivity_CS_Data("T287 C15 Monel",
"Ni 67.1 Cu 29.18 Fe 1.72 Al 0.04 Mn 0.98 Si 0.01 Co 0.33 S 0.014 Mg 0.13 P 0.024",a);
b1=true;}
else if(rName.equals("2T367 C1 Cotton fabric"))
{double a[][]={{313.2},{0.0753}};
rd=new thermal_conductivity_CS_Data("2T367 C1 Cotton fabric",
" ",a);
b1=true;}
else if(rName.equals("2T367 C1 cotton slicate"))
{double a[][]={{400.9,402.7,527.2,582.2},{0.0582,0.0715,0.0885,0.1074}};
rd=new thermal_conductivity_CS_Data("2T367 C1 cotton slicate"," ",a);
b1=true;}
else if(rName.equals("2T368 Cotton wool"))
{double a[][]={{92.2,136,155.9,188,204.4,217.2,251.2,263.5,278,286.2},
{0.0138,0.017,0.0211,0.0256,0.0275,0.0298,0.036,0.0384,0.0406,0.0422}};
rd=new thermal_conductivity_CS_Data("2T368 Cotton wool"," ",a);
b1=true;}
else if(rName.equals("2T373 Asbestos cement board"))
{double a[][]={{338.7,366.5,394.3,422.1,449.8,477.6,505.4,533.2,560.9,588.7},
{0.77,0.757,0.749,0.742,0.739,0.736,0.736,0.736,0.733,0.731}};
```

```
rd=new thermal_conductivity_CS_Data("2T373 Asbestos cement board"," ",a);
b1=true;}
else if(rName.equals("2T374 Asphalt-Glass wool"))
{double a[][]={{222.2,233.7,255.4,272.2,288.9,305.5},
{0.0304,0.0326,0.0348,0.0369,0.0394,0.0414}};
rd=new thermal_conductivity_CS_Data("2T374 Asphalt-Glass wool"," ",a);
b1=true;}
else if(rName.equals("2T375 cardboard density 790 kg/m^3"))
{double a[][]={{273.2,293.2,323.2},{0.138,0.143,0.167}};
rd=new thermal_conductivity_CS_Data("2T375 cardboard density 790 kg/m^3"," ",a);
b1=true;}
else if(rName.equals("2T376 lignin impregnated fireboard"))
{double a[][]={{222,238.7,255.4,272.2,288.9,305.5},{0.241,0.247,0.251,0.255,0.261,0.265}};
rd=new thermal_conductivity_CS_Data("2T376 lignin impregnated fireboard"," ",a);
b1=true;}
else if(rName.equals("2T380 Plywood density 552 kg/m^3"))
{double a[][]={{222,238.7,255.4,272.2,288.9,305.5},{0.101,0.104,0.107,0.111,0.114,0.117}};
rd=new thermal_conductivity_CS_Data("2T380 Plywood density 552 kg/m^3"," ",a);
b1=true;}
else if(rName.equals("2T376 lignin impregnated fireboard"))
{double a[][]={{222,238.7,255.4,272.2,288.9,305.5},{0.241,0.247,0.251,0.255,0.261,0.265}};
rd=new thermal_conductivity_CS_Data("2T376 lignin impregnated fireboard"," ",a);
b1=true;}
else if(rName.equals("2T380 Plywood density 552 kg/m^3"))
{double a[][]={{222,238.7,255.4,272.2,288.9,305.5},{0.101,0.104,0.107,0.111,0.114,0.117}};
rd=new thermal_conductivity_CS_Data("2T380 Plywood density 552 kg/m^3","density 552 kg/m^3",a);
b1=true;}
else if(rName.equals("2T381 C34 Fiberglass density 33 kg/m^3"))
{double a[][]={{260.7,270.2,275.7},{0.036,0.0389,0.0395}};
rd=new thermal_conductivity_CS_Data("2T381 C34 Fiberglass density 33 kg/m^3","density 33 kg/m^3",a);
b1=true;}
else if(rName.equals("2T381 C36 Fiberglass density 120 kg/m^3"))
{double a[][]={{263.2,270.7,275.2,280.2},{0.0291,0.03,0.0307,0.0314}};
rd=new thermal_conductivity_CS_Data("2T381 C36 Fiberglass density 120 kg/m^3","density 120 kg/m^3",a);
b1=true;}
else if(rName.equals("2T381 C37 Fiberglass density 88 kg/m^3"))
{double a[][]={{260.7,271.7,277.2,281.6,286.2},{0.0291,0.0325,0.0316,0.0322,0.0331}};
rd=new thermal_conductivity_CS_Data("2T381 C37 Fiberglass density 88 kg/m^3","density 88 kg/m^3",a);
b1=true;}
else if(rName.equals("2T382 glass fiber board density 215 kg/m^3"))
{double a[][]={{289.1,289.7,290.1,311.2,311.6,311.7,315.7},{0.0499,0.0495,0.0494,0.0324,0.0524,0.0518,0.053}};
rd=new thermal_conductivity_CS_Data("2T382 glass fiber board density 215 kg/m^3","density 215 kg/m^3",a);
b1=true;}
else if(rName.equals("2T385 paper"))
{double a[][]={{294.7,297.2,310.2,321.2,335.2,339.2,353.2,359.2,360.2,361.7,365.2,385.2},
{0.18,0.18,0.184,0.184,0.186,0.18,0.186,0.182,0.184,0.186,0.184,0.18}};
rd=new thermal_conductivity_CS_Data("2T385 paper"," ",a);
b1=true;}
else if(rName.equals("2T387 C1 Vegetable fibreboard 0.75 in thick 251 kg/m^3 moisture 24.1 %"))
{double a[][]={{222,238,255.4,272.2,288.9,305.5},{0.0495,0.0516,0.0542,0.0565,0.0588,0.0611}};
rd=new thermal_conductivity_CS_Data("2T387 C1 Vegetable fibreboard 0.75 in thick 251 kg/m^3 moisture 24.1 %","0.75 in thick 251
kg/m^3 moisture 24.1 %",a);
b1=true;}
else if(rName.equals("2T387 C1 Vegetable fibreboard 0.75 in thick 251 kg/m^3 moisture 3.1 %"))
{double a[][]={{222,238,255.4,272.2,288.9,305.5},{0.0424,0.0441,0.046,0.0477,0.0495,0.0513}};
rd=new thermal_conductivity_CS_Data("2T387 C1 Vegetable fibreboard 0.75 in thick 251 kg/m^3 moisture 3.1 %","0.75 in thick 251
kg/m^3 moisture 3.1 %",a);
b1=true;}
else if(rName.equals("2T387 C1 Vegetable fibreboard 0.75 in thick 251 kg/m^3 moisture 0 %"))
{double a[][]={{222,238,255.4,272.2,288.9,305.5},{0.0411,0.043,0.0447,0.0466,0.0483,0.0502}};
rd=new thermal_conductivity_CS_Data("2T387 C1 Vegetable fibreboard 0.75 in thick 251 kg/m^3 moisture 0 %","0.75 in thick 251
kg/m^3 moisture 3.1 %",a);
b1=true;}
else if(rName.equals("2T389 C1 wood fiber blanket density 56 kg/m^3"))
{double a[][]={{222,238,255.4,272.2,288.9,305.5},{0.0286,0.0304,0.0322,0.034,0.0361,0.0384}};
rd=new thermal_conductivity_CS_Data("2T389 C1 wood fiber blanket density 56 kg/m^3","density 56 kg/m^3",a);
b1=true;}
else if(rName.equals("2T389 C2 wood fiber mat density 56 kg/m^3"))
{double a[][]={{222,238,255.4,272.2,288.9,305.5},{0.027,0.0293,0.0316,0.034,0.0363,0.0392}};
rd=new thermal_conductivity_CS_Data("2T389 C2 wood fiber mat density 56 kg/m^3"," ",a);
b1=true;}
else if(rName.equals("2T390 wood felt SG 0.15"))
{double a[][]={{313.2,343.2},{0.0623,0.0732}};
rd=new thermal_conductivity_CS_Data("2T390 wood felt SG 0.15"," ",a);
b1=true;}
else if(rName.equals("2T393 C1 mineral fiber density 26.4 kg/m^3"))
{double a[][]={{297.1,311,324.8},{0.042,0.0453,0.0494}};
```

```
rd=new thermal_conductivity_CS_Data("2T393 C1 mineral fiber density 26.4 kg/m^3","density 26.4 kg/m^3",a);
b1=true;}
else if(rName.equals("2T393 C2 mineral fiber density 56.1 kg/m^3"))
{double a[][]={{297.1,311,324.8},{0.0398,0.0428,0.0463}};
rd=new thermal_conductivity_CS_Data("2T393 C3 mineral fiber density 39.6 kg/m^3","density 56.1 kg/m^3",a);
b1=true;}
else if(rName.equals("2T393 C1 mineral fiber density 26.4 kg/m^3"))
{double a[][]={{297.1,311,324.8},{0.0375,0.0404,0.0438}};
rd=new thermal_conductivity_CS_Data("2T393 C3 mineral fiber density 39.6 kg/m^3","density 26.4 kg/m^3",a);
b1=true;}
else if(rName.equals("2T393 C4 mineral fiber density 52.9 kg/m^3"))
{double a[][]={{297.1,311,324.8},{0.0361,0.0389,0.0418}};
rd=new thermal_conductivity_CS_Data("2T393 C4 mineral fiber density 52.9 kg/m^3","density 52.9 kg/m^3",a);
b1=true;}
else if(rName.equals("2T393 C4 mineral fiber density 52.9 kg/m^3"))
{double a[][]={{297.1,311,324.8},{0.0361,0.0389,0.0418}};
rd=new thermal_conductivity_CS_Data("2T393 C4 mineral fiber density 52.9 kg/m^3","density 52.9 kg/m^3",a);
b1=true;}
else if(rName.equals("2T393 C5 mineral fiber density 68.1 kg/m^3"))
{double a[][]={{297.1,311,324.8},{0.0378,0.0401,0.0433}};
rd=new thermal_conductivity_CS_Data("2T393 C5 mineral fiber density 68.1 kg/m^3","density 568.1 kg/m^3",a);
b1=true;}
else if(rName.equals("2T393 C6 mineral fiber density 112 kg/m^3"))
{double a[][]={{297.1,311,324.8},{0.0362,0.0388,0.0412}};
rd=new thermal_conductivity_CS_Data("2T393 C6 mineral fiber density 112 kg/m^3","density 112 kg/m^3",a);
b1=true;}
else if(rName.equals("2T394 C5 mineral wool density 252 kg/m^3"))
{double a[][]={{222,238.7,255.4,272.2,288.9,305.5},{0.0366,0.0387,0.0405,0.0424,0.0444,0.0464}};
rd=new thermal_conductivity_CS_Data("2T394 C5 mineral wool density 232-264 kg/m^3","density 252 kg/m^3",a);
b1=true;}
else if(rName.equals("2T393 C6 mineral fiber density 112 kg/m^3"))
{double a[][]={{222,238.7,255.4,272.2,288.9,305.5},{0.0346,0.0361,0.0389,0.0404,0.0418,0.0464}};
rd=new thermal_conductivity_CS_Data("2T394 C5 mineral wool density 232-264 kg/m^3","density 232-264 kg/m^3",a);
b1=true;}
else if(rName.equals("2T400 Bitumen"))
{double a[][]={{292.1,355.9},{0.166,0.173}};
rd=new thermal_conductivity_CS_Data("2T400 Bitumen"," ",a);
b1=true;}
else if(rName.equals("2T402 charcoal"))
{double a[][]={{293},{0.166,0.36}};
rd=new thermal_conductivity_CS_Data("2T402 charcoal"," ",a);
b1=true;}
else if(rName.equals("2T264 olivine basalt density 1490 kg/m^3"))
{double a[][]={{209.3,249.9,267.1,279.9,338.2,367.1},{0.142,0.254,0.165,0.246,0.157,0.228}};
rd=new thermal_conductivity_CS_Data("2T264 olivine basalt density 1490 kg/m^3","density 1490 kg/m^3 ",a);
b1=true;}
else if(rName.equals("2T268 dolomite"))
{double a[][]={{484,521,565,619,678,729,773,835,888,932,970,1051,1098,1173,1208},
{1.1,1.08,1.36,1.37,1.24,1.25,1.25,1.14,0.983,1.09,1.16,0.699,0.64,0.356,0.452}};
rd=new thermal_conductivity_CS_Data("2T268 dolomite","  ",a);
b1=true;}
else if(rName.equals("2T269 earth"))
{double a[][]={{293.2,333.2,403.2,503.2,618.2,708.2,813.2},{0.186,0.199,0.234,0.289,0.337,0.383,0.423}};
rd=new thermal_conductivity_CS_Data("2T269 earth","  ",a);
b1=true;}
else if(rName.equals("2T271 C4 granite"))
{double a[][]={{330,383,448,483,573,658,728,793,811,873,943},{1.51,1.63,1.76,1.87,1.76,2.01,1.74,1.74,1.52,1.64,1.8}};
rd=new thermal_conductivity_CS_Data("2T271 C4 granite","  ",a);
b1=true;}
else if(rName.equals("2T272 limestone"))
{double a[][]={{392.1,395.9,450.4,472,512,513,546,553,610,683,736,813,878,952,1013,1075,1181,1253,1324},
{1.43,1.42,1.41,1.19,1.2,1.28,1.15,1.21,1.21,1.19,1.12,1.11,1.11,1.12,1.07,1.03,0.619,0.573,0.54}};
rd=new thermal_conductivity_CS_Data("2T272 limestone","  ",a);
b1=true;}
else if(rName.equals("2T273 mica"))
{double a[][]={{92.2,130.3,155.2,176.2,192.2,206.2,216.4,225.2,234.2,248.2,260.2,265.2,275.2,288.9,305.5},
{0.0306,0.0374,0.0396,0.0436,0.0478,0.0509,0.053,0.0554,0.0571,0.0602,0.0635,0.0647,0.0656,0.0659,0.0687}};
rd=new thermal_conductivity_CS_Data("2T273 mica","  ",a);
b1=true;}
else if(rName.equals("2T274 expanded perlite density 48 kg/m^3"))
{double a[][]={{170.7,228.2,283.2},{0.025,0.0325,0.0395}};
rd=new thermal_conductivity_CS_Data("2T274 expanded perlite density 48 kg/m^3","density 48 kg/m^3",a);
b1=true;}
else if(rName.equals("2T275 rock consisted fine grade quartz diorite"))
{double a[][]={{287.3},{0.606}};
rd=new thermal_conductivity_CS_Data("2T275 rock consisted fine grade quartz diorite"," ",a);
b1=true;}
```

```
else if(rName.equals("2T276 salt"))
{double a[][]={{332,375,439,464,496,532,600},{2.91,2.83,2.89,2.78,2.6,2.65,2.51}};
rd=new thermal_conductivity_CS_Data("2T276 salt"," ",a);
b1=true;}
else if(rName.equals("2T277 C60 sand"))
{double a[][]={{180.4,212.6,235.4,279.3,279.9,303.8,312.1,332.1,342.6},
{0.209,0.243,0.296,0.172,0.138,0.236,0.22,0.222,0.246}};
rd=new thermal_conductivity_CS_Data("2T277 C60 sand"," ",a);
b1=true;}
else if(rName.equals("2T278 sandstone"))
{double a[][]={{303.2},{0.0264}};
rd=new thermal_conductivity_CS_Data("2T278 sandstone"," ",a);
b1=true;}
else if(rName.equals("2T302 nylon"))
{double a[][]={{357.9},{0.0138}};
rd=new thermal_conductivity_CS_Data("2T278 sandstone"," ",a);
b1=true;}
else if(rName.equals("2T303 phenolic resin"))
{double a[][]={{293.5},{0.3}};
rd=new thermal_conductivity_CS_Data("2T303 phenolic resin"," ",a);
b1=true;}
else if(rName.equals("2T305 C7 polyvinylcholoride"))
{double
a[][]={{118.6,123.2,127.3,131.8,136.1,139.6,144.2,148.1,151.5,156.3,158.9,162.6,166.6,169.5,173.2,177.6,181,183.6,188,190.7,193.7,197.
5,201.2,203.7,207.2,209.8,213.4,216.3,219.4,223.1,231.2,233.4,236.9,242.7,245.3,247.9,252.8,253.7,
258.8,260.9,263.7,266.6,268.8,271.5,272.5,276.4,278.7,281.7,283.9,286.6,289.1,291.3,294.2,296.8,298.8,301.4,304.2,306.1,308.4,310.7,31
2.7,316.7,317.3,320.7,322.7,324.8,326.8,329.4,332.7,335.1,336.8,339.4,341.7,345.1,349.1,350.6,352.7,354.5,356.7,359.4,362.1},
{0.15,0.15,0.151,0.151,0.153,0.153,0.154,0.154,0.155,0.156,0.156,0.157,0.157,0.158,0.158,0.158,0.158,0.159,0.159,0.159,0.16,0.161,0.16
1,0.161,0.161,0.161,0.162,0.162,0.162,0.162,0.162,0.162,0.161,0.162,0.161,0.16,0.16,0.16,0.16,0.159,
0.159,0.159,0.158,0.157,0.158,0.157,0.157,0.157,0.156,0.156,0.156,0.155,0.155,0.155,0.155,0.155,0.154,0.154,0.154,0.153,0.153,0.153,0.
153,0.152,0.152,0.152,0.151,0.151,0.151,0.15,0.151,0.15,0.15,0.15,0.15,0.149,0.149,0.148,0.148,0.148,0.148}};
rd=new thermal_conductivity_CS_Data("2T305 C7 polyvinylcholoride"," ",a);
b1=true;}
else if(rName.equals("2T306 C1 Polyethylene MW 21000"))
{double a[][]={{305.2,317.2,322.2,333.7,343.7,352.7,362.7},
{0.28,0.278,0.268,0.264,0.268,0.276,0.274}};
rd=new thermal_conductivity_CS_Data("2T306 C1 Polyethylene MW 21000"," ",a);
b1=true;}
else if(rName.equals("2T306 C2 Polyethylene MW 70000-80000"))
{double a[][]={{293.2,313.2,319.2,326.2,328.2,333.2,344.7,349.7,360.2,371.2},
{0.349,0.331,0.32,0.316,0.314,0.31,0.301,0.295,0.285,0.272}};
rd=new thermal_conductivity_CS_Data("2T306 C2 Polyethylene MW 70000-80000"," ",a);
b1=true;}
else if(rName.equals("2T308 Plexiglass (polymethylmethacrylate)"))
{double a[][]={{183.3,184.4,244.8,245.6,300.3,332.1,332.2},
{0.105,0.106,0.134,0.135,0.154,0.158,0.157}};
rd=new thermal_conductivity_CS_Data("2T308 Plexiglass (polymethylmethacrylate)"," ",a);
b1=true;}
else if(rName.equals("2T309 Polystyrene"))
{double a[][]={{249.2,273.2,306.2,339.2,350.2,363.2,373.2,423.2,466.2},
{0.109,0.116,0.126,0.129,0.118,0.128,0.129,0.136,0.139}};
rd=new thermal_conductivity_CS_Data("2T309 Polystyrene"," ",a);
b1=true;}
else if(rName.equals("2T316 teflon"))
{double a[][]={{358.7,394.3,439.3,470.9,481.5,485.4,527.6,532.1,553.2,563.2,572.1},
{0.42,0.411,0.401,0.317,0.346,0.346,0.352,0.337,0.375,0.426,0.404}};
rd=new thermal_conductivity_CS_Data("2T316 teflon"," ",a);
b1=true;}
else if(rName.equals("2T311 polytrifluorochloroethylene"))
{double a[][]={{317.6,350.7,370.9,383.2,390.9,402.6,408.2,435.9,437.6,460.9,464.6},
{0.146,0.185,0.211,0.245,0.232,0.237,0.249,0.235,0.251,0.267,0.248}};
rd=new thermal_conductivity_CS_Data("2T311 polytrifluorochloroethylene"," ",a);
b1=true;}
else if(rName.equals("2T312 rubber"))
{double a[][]={{109.2,113.2,117.2,143.2,149.2,156.2,157.2,171.2,181.2,193.2,217.2,239.2,263.2,285.2,301.2},
{0.0941,0.1,0.105,0.117,0.119,0.118,0.118,0.123,0.126,0.13,0.146,0.14,0.142,0.159,0.172}};
rd=new thermal_conductivity_CS_Data("2T312 rubber"," ",a);
b1=true;}
return b1;
}
public String[][] toString(double t)
{
  String s1[][]=new String[4][4];
   s1[0][0]=" name ";
   s1[1][0]=" additional info";
   s1[2][0]=" temperature";
   s1[2][1]=" "+t;
```

```
    s1[3][0]=" k ,thermal conductivity";
    s1[2][2]="degree Celcius";
    s1[3][2]="W/(mK)";
    s1[0][1]=rd.solidName;
    s1[1][1]=rd.additional_info;
    s1[3][1]=""+k(t);
  return s1;
  }
  double k(double t)
  {double T=t+273.15;return rd.k_l(T);}
  public static void main(String arg[])
  {thermal_conductivity_CS tc=new thermal_conductivity_CS("C9 Carbon Steel 3");
   double t=100.0;
   System.out.println("k="+tc.k(t)+" W/mK");
  } }
```

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import javax.swing.event.*;

class thermal_conductivity_CS_Model extends AbstractTableModel
{ private static final long serialVersionUID =9579657756L;
  Object[][] veri;
  String[] baslik={"Name ","Value ","Units"};
  thermal_conductivity_CS g;

  public thermal_conductivity_CS_Model()
  {
  g=new thermal_conductivity_CS("C9 Carbon Steel 3");
  veri=g.toString(20);
  }

  public thermal_conductivity_CS_Model(String s,double t)
  {
  g=new thermal_conductivity_CS(s);
  veri=g.toString(t);
  }

  public thermal_conductivity_CS_Model(String s)
  {
  g=new thermal_conductivity_CS(s);
  veri=g.toString(20);
  }

  public int getRowCount() {return veri.length;}
  public int getColumnCount() {return baslik.length;}
  public Object getValueAt(int satir,int sutun) {return veri[satir][sutun];}
  public String getColumnName(int c) {return baslik[c];}
  public void setValueAt(Object val, int row, int col)
  {
    veri[row][col] = val;
   }

  public void setValues(String s,double t)
  {
  g=new thermal_conductivity_CS(s);
  veri=g.toString(t);
  }

  public void setValues(double t)
  {veri=g.toString(t);}

  public boolean isCellEditable(int row, int col) {return true;}
}
```

```
import java.lang.Integer;
import java.awt.*;
import java.awt.event.*;
import java.awt.font.*;
import java.awt.geom.*;
import java.awt.image.*;
import javax.swing.*;
```

```
import java.util.Locale;
import java.text.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.table.*;

public class thermal_conductivity_CS_Table extends JApplet implements ItemListener,ActionListener
{ private static final long serialVersionUID =9576876L;
  Container c;
  Color bg;
  JTabbedPane tabPane = new JTabbedPane();
  JScrollPane skrolPane;
  JScrollPane skrolPane1;
  final static String say1 = "substance ";
  final static String say2 = "additional information     ";
  //ImageIcon sogutucuakiskanlar;
  JPanel panel1;
  JPanel girisPaneli;
  JScrollPane cikisPaneli;
  JLabel l1[]=new JLabel[2];          // Label prompt unit
  JLabel l2[]=new JLabel[2];          // Label prompt unit
  JTextField t1[]=new JTextField[2];
  JComboBox<String>  c1;
  JTextArea output;
  JTable jt;
  thermal_conductivity_CS_Model samodel;
  //condYTOK  cond;
  double aa[]=new double[10];  // çıktı değerleri
  //Icon turkoglu_logo;
  JButton b1;
  double ts;
  String isim;
  String s1[];
  JTextArea LSK;
    public void init()
    {
    isim="C9 Carbon Steel 3";
                samodel=new thermal_conductivity_CS_Model();
                ts=20.0;
                s1=samodel.g.list();
    c =getContentPane();
    panel1 = new JPanel();
    panel1.setBackground(c.getBackground());
    panel1.setLayout(new BorderLayout());
      //==========================
      output=new JTextArea(toString());
      output.setBackground(bg);

      l1[0]=new JLabel("name of the solid : ");
      l1[0].setBorder(BorderFactory.createEtchedBorder(Color.blue,Color.yellow));
      Font fo16=l1[0].getFont();
      fo16=new Font(fo16.getFontName(),Font.BOLD,10);
      Font fo12=new Font(fo16.getFontName(),Font.BOLD,12);
      Font fo10=new Font(fo16.getFontName(),Font.BOLD,10);
      l1[0].setFont(fo10);
      output.setFont(fo10);
      //panel1.add(b1,BorderLayout.NORTH);
      panel1.add(output,BorderLayout.NORTH);
      LSK=new JTextArea(toString());
      l1[1]=new JLabel("  temperature");
      l1[1].setBorder(BorderFactory.createEtchedBorder(Color.blue,Color.yellow));
      l2[0]=new JLabel(" ");
      l2[0].setBorder(BorderFactory.createEtchedBorder(Color.blue,Color.yellow));
      l2[1]=new JLabel(" degree Celcius");
      l2[1].setBorder(BorderFactory.createEtchedBorder(Color.blue,Color.yellow));
      c1=new JComboBox<String>(s1);
       c1.setBorder(BorderFactory.createEtchedBorder(Color.blue,Color.yellow));
      t1[1]=new JTextField(""+ts);
      t1[1].setBorder(BorderFactory.createEtchedBorder(Color.blue,Color.yellow));
      jt=new JTable(samodel);
      girisPaneli = new JPanel();
      girisPaneli.setLayout(new GridLayout(2,3));
      girisPaneli.setBorder(BorderFactory.createLineBorder(Color.blue));
      cikisPaneli=new JScrollPane(jt);
      girisPaneli.add(l1[0]);
```

```
    girisPaneli.add(c1);
    girisPaneli.add(l2[0]);
    girisPaneli.add(l1[1]);
    girisPaneli.add(t1[1]);
    girisPaneli.add(l2[1]);
   t1[1].setBackground(bg);
   l1[0].setBackground(bg);
   l1[1].setBackground(bg);
   c1.setBackground(bg);
   l2[0].setBackground(bg);
   l2[1].setBackground(bg);
   jt.setBackground(bg);
   t1[1].setFont(fo12);
   jt.setFont(fo12);
   c1.addItemListener(this);
   t1[1].addActionListener(this);
   panel1.add(girisPaneli,BorderLayout.CENTER);
   //cikisPaneli.add(jt,BorderLayout.NORTH);
   panel1.add(cikisPaneli,BorderLayout.SOUTH);

    //=========================
    skrolPane=new JScrollPane(panel1);
    skrolPane1=new JScrollPane(output);
    tabPane.addTab(say1,skrolPane);

    tabPane.addTab(say2,skrolPane1);
    c.add(tabPane);
   }

   public void itemStateChanged(ItemEvent e)
   {
            isim=(String)c1.getSelectedItem();
            Double V1=new Double(t1[1].getText());
   ts=V1.doubleValue();
   t1[1].setText(""+ts);
            samodel.setValues(isim,ts);
            repaint();
   }

   public void actionPerformed(ActionEvent e)
   {
            if(e.getSource()==t1[1])
            {Double V1=new Double(t1[1].getText());
    ts=V1.doubleValue();}
    t1[1].setText(""+ts);
    samodel.setValues(ts);
    repaint();
  }

  public static String toString(double left, int w, int d)
  // converts a double to a string with given width and decimals.
  {
            NumberFormat df=NumberFormat.getInstance(Locale.US);
            df.setMaximumFractionDigits(d);
   df.setMinimumFractionDigits(d);
   df.setGroupingUsed(false);
   String s = df.format(left);
   while (s.length() < w)
     s = " " + s;
   if (s.length() > w)
   {
      s = "";
      for (int i=0; i<w; i++)
        s = s + "-";
   }
   return s;
   }
   public String toString()
   {
            String s="  Thermal conductivity of solids\n\n";
            s+="  M. Turhan ÇOBAN\n";
            s+="  EGE Üniversity, Schoool of Engineering. Department of mechanical eng. Izmir Turkey\n";
            s+="  e-mail : turhan_coban@yahoo.com\n";
            s+="  web : www.turhancoban.com\n";
            s+="=================================================================\n\n";
            s+="list of solids : \n";
            s+="============================================================\n";
```

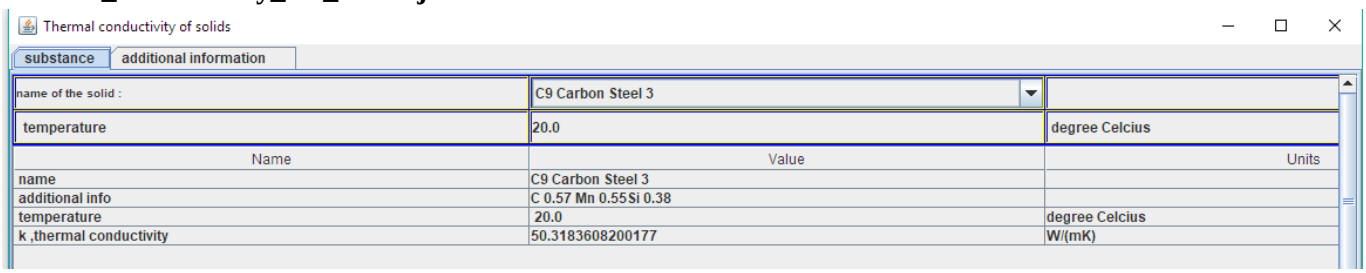
```
            s+=samodel.g.readNames();
     return s;
            }

   public static void main(String s[])
   {
      //main program
            JFrame f = new JFrame("Thermal conductivity of solids");
      f.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {System.exit(0);}
      });
      JApplet applet = new thermal_conductivity_CS_Table();
      f.getContentPane().add("Center", applet);
      applet.init();
      f.pack();
      f.setSize(new Dimension(400,400));
      f.setVisible(true);
   }
 }
```

```
--------- Capture Output ---------
> "D:\co\java\bin\java.exe" thermal_conductivity_CS
k=47.70153484397837 W/mK

 > Terminated with exit code 0.
```

thermal_conductivity_CS_Table.java



In this particular program cubic spline curve fitting method is applied. Polynomial Least square curve fitting method can also be applied. This approach prefferred by most due to simplistic representation of the temperature dependency equation. An example case is given here. As an example case **2T309 Polystyrene** data is taken. A general curve fitting program SCO11B4.java is used here to calculate polynomial coefficients.

```
 import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.Color;

class SCO11B4
{
//least square curve fitting
  public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //partial pivoting

            p=k;
            buyuk=Math.abs(a[k][k]);
            for(ii=k+1;ii<n;ii++)
            {  dummy=Math.abs(a[ii][k]);
```

```
                if(dummy > buyuk) {buyuk=dummy;p=ii;}
               }
               if(p!=k)
               { for(jj=k;jj<n;jj++)
                 { dummy=a[p][jj];
                   a[p][jj]=a[k][jj];
                   a[k][jj]=dummy;
                 }
                 dummy=b[p];
                 b[p]=b[k];
                 b[k]=dummy;
               }
               //back substitution
   for(i=k+1;i<n;i++)
   {  carpan=a[i][k]/a[k][k];
      a[i][k]=0;
      for(j=k+1;j<n;j++)
      {   a[i][j]-=carpan*a[k][j]; }
          b[i]   =b[i]   -carpan*b[k];
               }
  }
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}
//least square curve fitting
public static double[] EKK(double xi[],double yi[],int n)
{
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  for(j=0;j<n+1;j++)
     {if(i==0 && j==0) A[i][j]=l;
     else  for(k=0;k<l;k++) A[i][j] += Math.pow(xi[k],(i+j));
     }
     for(k=0;k<l;k++) { if(i==0)  B[i]+= yi[k];
                  else      B[i] += Math.pow(xi[k],i)*yi[k];}
  }
X=pivotlugauss(A,B);
//X=B/A;
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
 return X;
}

public static double funcEKK(double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=e.length;
double ff;
if(n!=0.0)
  { ff=e[n-1];
  for(int i=n-2;i>=0;i--)
   { ff=ff*x+e[i]; }
  }
else
  ff=0;
return ff;
}
```

```
public static double hata(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcEKK(e,x[k]))*(y[k]-funcEKK(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

public static double[][]  funcEKK(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
 {
                //number of subdivision:  x--o--o--x--o--o--x in the given chain if x is experimental data
                // number of subdivision is 2
                 int n=xi.length;
                int nn=(n-1)*(aradegersayisi+1)+1;
                double z[][]=new double[2][nn];
   double E[]=EKK(xi,yi,polinomkatsayisi);
   String ss[]={"a0","a1","a2","a3","a4","a5","a6","a7","a8","a9","a10","a11","a12","a13","a14","a15","a16","a17"};
   String ss1="";
   for(int i=0;i<polinomkatsayisi;i++)
   {ss1+=ss[i]+" = "+E[i]+"\n";}
   System.out.println(ss1);
   double hata=hata(xi,yi,E);
   //System.out.println("hata="+hata+"\nkatsayılar :\n"+Matrix.toStringT(E));
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
      {z[0][k]=xi[i];z[1][k]=funcEKK(E,z[0][k]);k++;
      for(int j=0;j<aradegersayisi;j++)
      {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
      z[0][k]=z[0][k-1]+dx;z[1][k]=funcEKK(E,z[0][k]);k++;}
      }
      z[0][k]=xi[i];z[1][k]=funcEKK(E,z[0][k]);
   return z;
 }
 public static double yavg(double y[])
 {
 int n=y.length;
 double total=0;
 for(int i=0;i<n;i++) total+=y[i];
 return total/n;
 }
 public static double R2(double y[],double f[])
 {
 double yavg=yavg(y);
 int n=y.length;
 double SStot=0;
 double SSerr=0;
 for(int i=0;i<n;i++)
 {SStot=(y[i]-yavg)*(y[i]-yavg);
  SSerr=(y[i]-f[i])*(y[i]-f[i]);
 }
 return (1-SSerr/SStot);
 }

 public static double R2(double x[],double y[],double e[])
 { int n=y.length;
 double f[]=new double[n];
   for(int k=0;k<n;k++)
    {f[k]=funcEKK(e,x[k]);}
  return R2(y,f);
 }

  public static void main(String args[]) throws IOException
  {
  double x[];
  double y[];
  String s1=JOptionPane.showInputDialog("file name (aaa.txt)");
  //JFileChooser fc=new JFileChooser();
  //if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file = fc.getSelectedFile();s1=file.getName(); }
  double a[][]=Text.readDoubleT(s1);
```

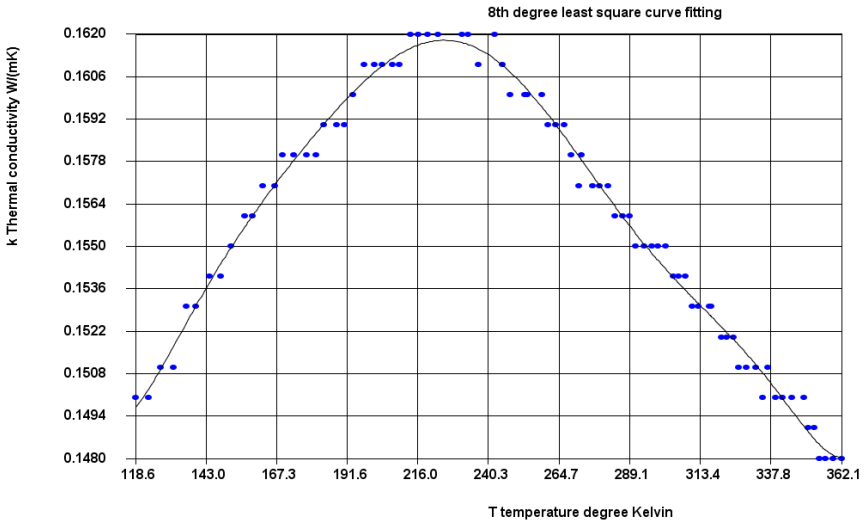
```
  x=a[0];
  y=a[1];
  int n=Integer.parseInt(JOptionPane.showInputDialog("degree of polynomial n:"));
  double z[][]=funcEKK(x,y,n,4);
  Text.printT(z);
  //System.out.println("E.K.K.\n"+Matrix.toStringT(z));
  Plot pp=new Plot(a[0],a[1]);
  pp.setPlabel(""+n+"th degree least square curve fitting");
  pp.setXlabel("T temperature degree Kelvin");
  pp.setYlabel("k Thermal conductivity W/(mK)");
  pp.setPlotType(0,23);
  pp.addData(z[0],z[1]);
  pp.setGrid(1,1);
  pp.setColor(0,Color.BLUE);
  pp.plot();
  }
}
```

2T309 Polystyrene

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" SCO11B4
a0 = 2.8019055476787833
a1 = -0.10631833488379154
a2 = 0.0018150980667831751
a3 = -1.7277946213139326E-5
a4 = 1.004258052091839E-7
a5 = -3.6486771049148165E-10
a6 = 8.090773720795419E-13
a7 = -1.0016212077805614E-15
```

So k(T) for polystyrene can be represented as an 8th degree polynomial as:

$$k(T) = a0 + a1T + a2T^2 + a2T^3 + a2T^4 + a2T^5 + a2T^6 + a2T^7 + a2T^8$$

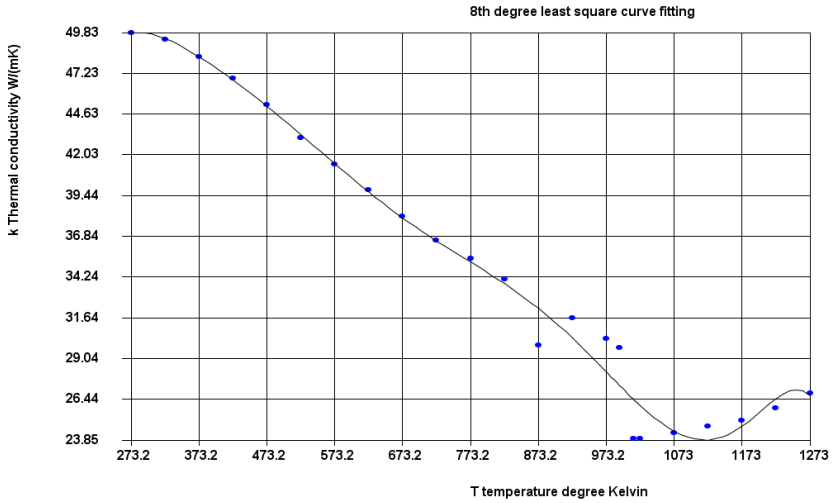


Graphic representation of data and 8th degree polynomial curve fitting line is shown in the plot.

Let us check out least square solution for few other cases:

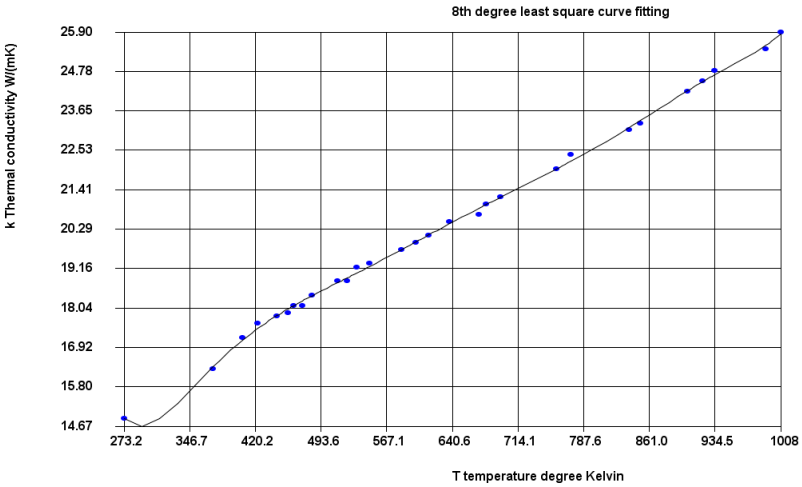
C129 British steel 7

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" SCO11B4
a0 = -155.75784551326365
a1 = 2.886754338818323
a2 = -0.017156888192827912
a3 = 5.6797499876122244E-5
a4 = -1.1478717651898844E-7
a5 = 1.440049314436008E-10
a6 = -1.0906823369518688E-13
a7 = 4.552235513057068E-17
```



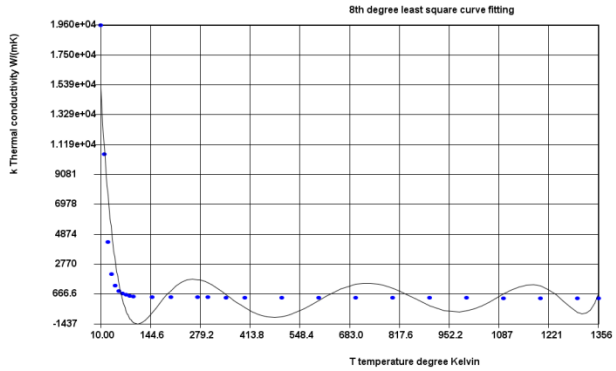
T330 C2 chromium-nickel steel SS304

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" SCO11B4
a0 = 629.219544213965
a1 = -8.785673170339718
a2 = 0.05300263682552713
a3 = -1.7711104936578227E-4
a4 = 3.603113322948669E-7
a5 = -4.5810482232978316E-10
a6 = 3.561454476094692E-13
a7 = -1.5503051690194676E-16
```

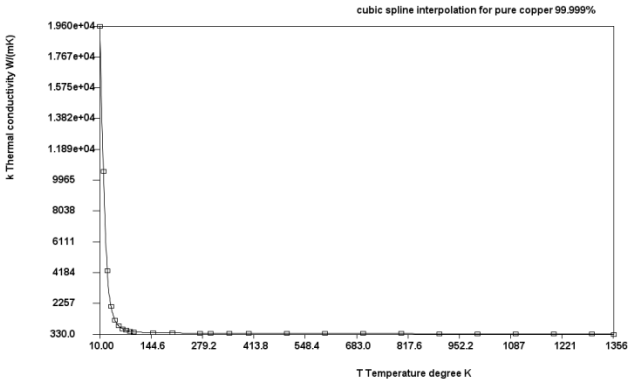


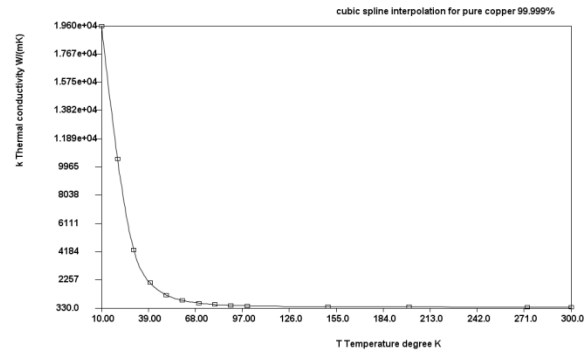
T12R pure Copper

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" SCO11B4
a0 = 20293.272431383157
a1 = -562.0064310567664
a2 = 5.3430057235494175
a3 = -0.02393680237368781
a4 = 5.7953278916051136E-5
a5 = -8.025743120123928E-8
a6 = 6.354980774994162E-11
a7 = -2.6752151949041078E-14
```



As it is seen from the plot 8th degree curve fitting is not very successful to represent this data. But when cubic spline interpolation used, data is successfully represented





Only problem is big coefficient set in cubic spline, but as long as computer handles coefficients, it will not give any problems. Coefficient set for cubic spline for pure copper:

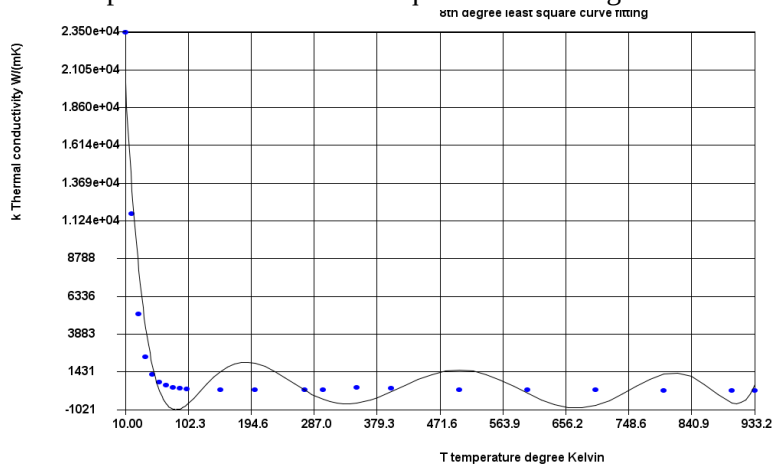
data set no : 0 x ref : 0 y ref : 1

dataset number : k	a[k]	b[k]	c[k]	x[k]
0.0	998.1361393763322	1960.0	0.0	10.0
1.0	347.45544249467133	998.1361393763322	31.11831637420071	20.0
2.0	192.04209064498232	347.45544249467133	49.526734503197176	30.0
3.0	114.37619492539929	192.04209064498232	7.774745613010601	40.0
4.0	82.45312965342049	114.37619492539929	4.574283044760423	50.0
5.0	65.81128646091875	82.45312965342049	1.5281222079477073	60.0
6.0	56.3017245029045	65.81128646091875	0.7132281234487483	70.0
7.0	50.98181552746325	56.3017245029045	0.4189652982572999	80.0
8.0	48.17101338724252	50.98181552746325	0.25091068352205265	90.0
9.0	8.526023825626536	9.015066936212591	0.07739196765448914	100.0
10.0	8.240837761281263	8.526023825626536	0.004077140924815569	150.0
11.0	5.474302376656038	5.617898659108555	0.002299468646248611	200.0
12.0	14.819546873320455	14.93203929577812	6.220771998970242E-4	273.15
13.0	7.871341349952576	7.953399675322559	7.920389612930107E-4	300.0
14.0	7.841234924867137	7.871341349952576	0.0010390380056908499	350.0
15.0	3.881249100844603	3.9224698497342736	-1.4819098405640993E-4	400.0
16.0	3.8325337468873135	3.881249100844603	-7.494605067619501E-5	500.0
17.0	3.7686159116061426	3.8325337468873135	-1.5202481323881018E-4	600.0
18.0	3.713002606688115	3.7686159116061426	8.304530363143594E-5	700.0
19.0	3.639373661641395	3.713002606688115	-1.8015640128693356E-4	800.0
20.0	3.5695027467463047	3.639373661641395	3.7580301516297724E-5	900.0
21.0	3.5026153513733878	3.5695027467463047	2.9835195221742663E-5	1000.0
22.0	3.4200358477601442	3.5026153513733878	-1.569210824032684E-4	1100.0
23.0	3.3372412575860344	3.4200358477601442	-2.150865608668752E-6	1200.0
24.0	5.892857142857143	5.962740818533893	1.6552454483794338E-4	1300.0

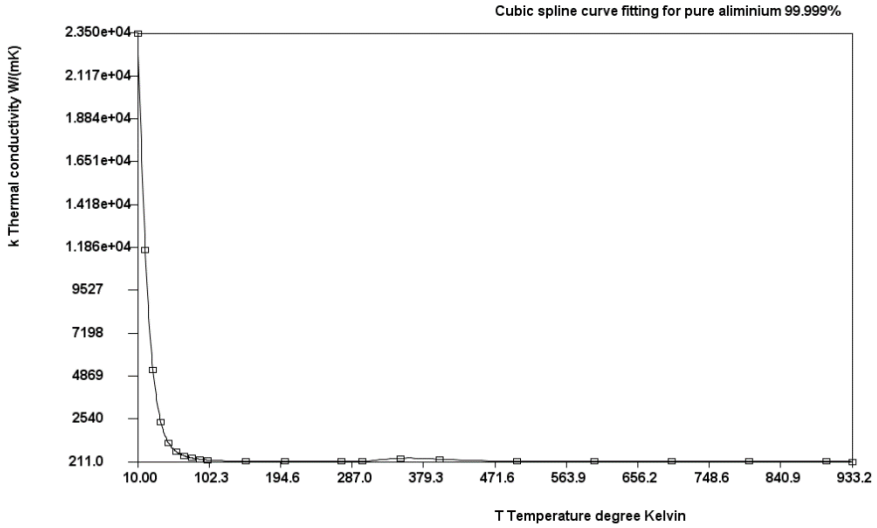
T12R Pure Aliminium 99.999%

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" SCO11B4
a0 = 29598.093432158155
a1 = -1065.8583855894676
a2 = 13.868870066959282
a3 = -0.08701054491280114
a4 = 2.987435927883477E-4
a5 = -5.909789569020431E-7
a6 = 6.713037470011706E-10
a7 = -4.064445243707968E-13
```

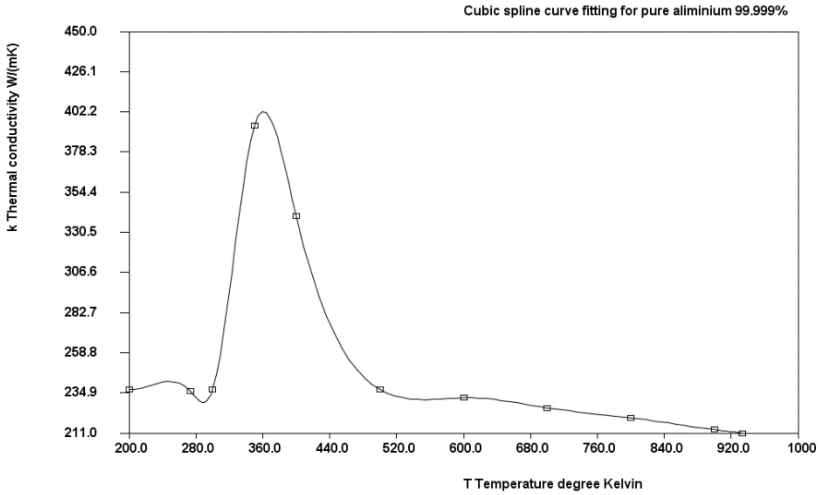
The same problem existed in least square curve fitting of aliminium



Cubic spline curve:



Even the increase temperature effect around 360 K is represented with CS curve properly.



1.4.2 SPECIFIC HEAT

specific heat of a **solid** or liquid is the amount of **heat** that raises the temperature of a unit mass of the **solid** through 1° C. Thermodynamic definition can be given as:

$C_P = T\left(\frac{\partial s}{\partial T}\right)_P$ (1.4.2-1)

$C_v = T\left(\frac{\partial s}{\partial T}\right)_v$ (1.4.2-2)

The first equation is called specific heat at constant pressure and the second one is called specific heat at constant volume. For solids, the value of both is the same. In this equations T is temperature (degree Kelvin), s is entropy (J/kgK) P is Pressure (Pa) and V is specific volume (m^3/kg). Some of the specific heat values of the solids are given in table 1.4.2-1

Table 1.4.2-1 Specific heat of some solids[116]

Name	Cp kJ/kgK	Name	Cp kJ/kgK
Agate	0.8	Iodine	0.218
Aluminum bronze	0.44	Iridium	0.13
Aluminum,(0 C)	0.87	Iron,_ 20℃	0.46
Antimony	0.21	Labradorite	0.8
Apatite	0.84	Lava	0.84
Arsenic	0.35	Limestone	0.91
Artificial_wool	1.36	Litharge	0.88
Asbestos_cement_board	0.84	Lead	0.13
Asbestos_mill_board	0.84	Leather,dry	1.5
Ashes	0.84	Lithium	3.58
Asphalt	0.92	Magnetite	0.67
Augite	0.8	Malachite	0.75
Bakelit_wood_filler	1.38	Manganese	0.46
Bakelite_asbestos filler	1.59	Magnesia(85%)	0.84
Barite	0.46	Magnesium	1.05
Barium	0.29	Marble,mica	0.88
Basalt_rock	0.84	Mercury	0.14
Beeswax	3.4	Mica	0.5
Beryl	0.84	Mineral_wool_blanket	0.84
Beryllium	1.02	Molybdenum	0.27
Bismuth	0.13	Nickel	0.46
Boile_scale	0.8	Oliglocose	0.88
Bone	0.44	Orthoclose	0.8
Borax	1	Osmium	0.13
Boron	1.3	Oxide_o_chrome	0.75
Brass	0.38	Paper	1.34
Brick,common	0.9	Paraffin_wax	2.9
Brick,hard	1	Peat	1.88
Bronze,phosphor	0.38	Phosphorbronze	0.36

Cadmium	0.25	Phosphorus	0.8
Calcite_0-38C	0.8	Pig_iron,white	0.54
Calcite_0_100C	0.84	Pinchbeck	0.38
Calcium	0.63	Pit coal	1.02
Calcium_carbonat	0.76	Plaster,_light	1
Calcium_sulfate	1.1	Plaster,_sand	0.9
Carbon,_Diamond	0.52	Plastics,_foam	1.3
Carbon,_Graphite	0.71	Plastics,_solid	1.67
Carborundum	0.67	Platinum,_℃	0.13
Cassiterite	0.38	Porcelain	1.07
Cement_dry	1.55	Potassium	0.54
Cement_powder	0.84	Pyrex_glass	0.84
Cellulose	1.6	Pyrolusite	0.67
Celluloid	1.5	Pyroxyli_plastics	1.51
Charcoal	1	Quartz_mineral 55 - 212°F	0.8
Chalk	0.9	Quartz_mineral(0 ℃)	0.71
Chalcopyrite	0.54	Red_lead	0.09
Charcoal,_wood	1	Red_metal	0.38
Chromium	0.5	Rhenium	0.14
Clay	0.92	Rhodium	0.24
Coal,_anthracite	1.26	Rock salt	0.92
Coal_bituminous	1.38	Rosin	1.3
Cobalt	0.46	Rubber	2.01
Coke	0.85	Rubidium	0.33
Concrete,stone	0.75	Salt	0.88
Concrete,light	0.96	Sand,_dry	0.8
Constantan	0.41	Sandstone	0.92
Copper	0.39	Sawdust	0.9
Cork,Corkboard	1.9	Selenium	0.33
Corundum	0.42	Serpentine	1.09
Cotton	1.34	Silica_aerogel	0.84
Diamond	0.63	Silicon	0.75
Dolomite_rock	0.92	Silicon,carbide	0.67
Duralium	0.92	Silk	1.38
Earth,_dry	1.26	Silver, 20℃	0.23
Electron	1	Slate	0.76
Emery	0.96	Sodium	1.26
Fats	1.93	Soil,dry	0.8
Fiberboard,_light	2.5	Soil,wet	1.48
Fiber_hardboard	2.1	Steatite	0.83
Fire_brick	1.05	Steel	0.49
Fluorite	0.92	Stone	0.84
Fluorspar	0.88	Stoneware	0.8
Galena	0.21	Sulphur,sulfur	0.71
Garnet	0.75	Tantalium	0.14
Glass	0.84	Tar	1.47
Glass,_crystal	0.5	Tellurium	0.21
Glass,_plate	0.5	Thorium	0.14
Glass,_Pyrex	0.75	Tile_hollow	0.63
Glass_window	0.84	Tin	0.24
Glas_wool	0.67	Titanium	0.47
Gold	0.13	Topaz	0.88
Granite	0.79	Tungsten	0.134
Graphite	0.71	Uranium	0.12
Gypsum	1.09	Vanadium	0.5
Hairfelt	2.1	Vermiculite	0.84
Hermatite	0.67	Vulcanite	1.38
Hornblende	0.84	Wax	3.43
Hypersthene	0.8	Welding_iron	0.52
Ice_-59C	1.47	White_metal	0.15
Ice_-40C	1.8	Wood, balsa	2.9
Ice_-20C	1.97	Wood,oak	2

Ice_0C	2.09	Wood,white pine	2.5
Indi_rubber_min	1.13	Wool,loose	1.26
India_rubber_max	4.1	Wool,felt	1.38
Ingo_iron	0.49	Zinc	0.38

1.4.3 DENSITY

The **density** (more precisely, the **volumetric mass density**; also known as **specific mass**), of a substance is its mass per unit volume. The symbol most often used for density is ρ. density is defined as mass divided by volume.

$$\rho = \frac{m}{V} \quad (1.4.3\text{-}1)$$

where ρ is the density, m is the mass, and V is the volume. Densities of some solids are listed in the table below.

Tablo 1.4.3 Densities of some solids[115]

Material	ρ low kg/m^3	ρ high kg/m^3	Material	ρ low kg/m^3	ρ high kg/m^3
ABS-copolymer_of_acrylonitrile,butadiene_and_styrene	1060		Hornblende	3000	
Acetals	1420		Ice	917	
Agate	2500	2700	Iron,_cast	7000	7400
Acrylic	1190		Iodine	4950	
Agate	2600		Iridium	22500	
Alabaster_carbonate	2700	2800	Ivory	1800	1900
Alabaster_sulfate	2300		Kaolin	2600	
Alum,_lumpy	881		Lead	11350	
Alum,_pulverized	752		Leather,_dry	860	
Alumina(aluminium_oxide)	3950	4100	Lime,_slaked	1350	
Aluminum	2700		Limestone	2700	2800
Aluminum_Bronze	7700		Linoleum	1200	
Albite	2600	2650	Lithium	530	
Amber	1060	1100	Magnesia	3200	3600
Amphiboles	2900	3200	Magnesium	1740	
Andesite,_solid	2770		Magnetite	4900	5200
Anorthite	2740	2760	Malachite	3700	4100
Antimony,_cast	6700		Manganese	7430	
Arsenic	4700		Marble	2600	2800
Artificial_Wool	1500		Meerschaum	1000	1300
Asbestos	2000	2800	Metals	0	
Asbestos,_shredded	350		Mica	2600	3200
Asbestos,_solid	2450		Mineral_wool_quilt	50	
Ashes	650		Molybdenum	10200	
Asphalt,_compacted	2360		Muscovite	2800	3000
Asphalt,_crushed	720		Nickel	8900	
Bakelite	1360		Nylon_6	1120	1170
Baking_powder	720		Nylon_6,6	1130	1150
Balsa_Wood	130		Oak	720	
Barite,_crushed	2890		Ochre	3500	
Barium	3780		Opal	2200	
Bark,_wood_refuse	240		Osmium	22480	
Barytes	4500		Palladium	12000	
Basalt	2400	3100	Paper	700	1150
Bauxite,crushed	1280		Paraffin	900	
Beeswax	960		Peat_blocks	850	
Beryl	2700		Phenolic_cast_resin	1240	1320
Beryllia	3000		Phosphorbronce	8800	
Beryllium	1850		Phosphorus	1820	
Biotite	2700	3100	Pinchbeck	8650	
Bismuth	9800		Pitch	1100	

Boiler_scale	2500		Pit_coal	1350	
Bone	1700	2000	Plaster_board	800	
Bone,_pulverized	880		Platinum	21500	
Borax,_fine	850		Plywood	540	
Brasses	8470	8750	Polyacrylonitrile	1160	1180
Bronzes	8740	8890	Polyamides	1150	1250
Brown_iron_ore	5100		PC-poly_carbonate	1200	
Brick	1400	2400	PBT-poly_butylene_terephthalate	1350	
Brick,_fire	2300		LDPE-low_density_poly_ethylene	910	
Brick,_hard	2000		HDPE-(PEH)-_high_density_poly_ethylene	960	
Brick,_pressed	2200		PET-poly_ethylene_terephthalate	1350	
Brickwork_in_cement	1800		PMMA-poly_methyl_methacrylate	1200	
Brickwork_in_mortar	1600		POM-poly_oxy_methylene	1400	
Butter	860	870	PP-poly_propylene	910	940
Cadmium	8640		PPO-poly_penylene_ether	1100	
Calamine	4100	4500	PS-poly_styrene	1030	
Calcium	1550		PTFE-poly_tetra_fluoro_ethylene,_Teflon	2280	2300
Calcspar	2600	2800	PU-poly_urethane_foam	30	
Camphor	1000		PVDF-poly_vinylidene_fluoride	1760	
Carbon	3510		Porcelain	2300	2500
Caoutchouc	900	1000	Porphyry	2600	2900
Cardboard	700		Potassium	860	
Cast_Iron	7200		Pressed_wood,_pulp_board	190	
Celluloid	1400		PVC-poly_vinyl_chloride	1390	1420
Cellulose,_cotton,_wood_pulp,_regenerated	1480	1530	Pyrex	2250	
Cellulose_acetat,_moulded	1220	1340	Pyrite	4900	5100
Cellulose_acetat,_sheet	1280	1320	Quartz	2650	
Cellulose_nitrate,_celluloid	1350	1400	Radium	5000	
Chlorinated_polyether	1400		Red_lead	8600	9100
Cement,_set	2700	3000	Red_metal	8800	
Cement,_Portland	1500		Resin	1070	
Cerium	6770		Rhenium	21400	
Chalk	1900	2800	Rhodium	12300	
Charcoal,_oak	600		Rock_salt	2200	
Charcoal,_pine	300	400	Rock_wool	220	390
Chromium	7100		Rosin	1070	
Chrom_oxide	5210		Rubber,_hard	1200	
Cinnabar	8100		Rubber,_soft_commercial	1100	
Clay	1800	2600	Rubber,_pure_gum	910	930
Coal,_anthracite	1400	1800	Rubber,_foam	70	
Coal,_bituminous	1200	1500	Rubidium	1520	
Cobalt	8800		Sand,_dry	1400	1600
Cocoa,_butter	900		Sandstone	2100	2400
Coke	1000	1700	Sapphire	3980	
Concrete,_lightweight	450	1000	Selenium	4400	
Concrete,_medium	1300	1700	Serpentine	2500	2650
Concrete,_dense	2000	2400	Silica,_fused_transparent	2200	
Constantan	8890		Silica,_translucent	2100	
Copal	1000	1150	Silicium_carbide	3160	
Copper	8790		Silicon	2330	
Cork	200	250	Silver	10500	
Cork,_linoleum	550		Slag	2000	3900
Corundum	4000		Slate	2600	3300
Cotton	80		Snow	100	
CPVC-Chlorinated_poly_vinyl_chloride	1600		Soapstone	2600	2800
Lead_Crystal	3100		Sodium	980	
Diamond	3000	3500	Soil	2050	
Dolomite	2800		Solder	8700	9400
Duralium	2800		Soot	1600	1700
Earth,_loose	1200		Spermaceti	950	
Earth,_rammed	1600		Starch	1500	
Ebonite	1150		Steatite	2600	2700

Emery	4000		Steel	7820	
Electron	1800		Stone	2300	2800
Epidote	3200	3500	Sulfur,_cryst.	2000	
Epoxy_cast_resin	1110	1400	Sugar	1600	
Epoxy_glass_fibre	1500		Talc	2700	2800
Expanded_polystyrene	15	30	Tallow,_beef	950	
Feldspar	2600	2800	Tallow,_mutton	950	
Fire_brick	1800	2200	Tantalum	16600	
Flint	2600		Tar	1050	
Fluorite	3200		Teflon	2200	
Galena	7300	7600	Tellurium	6250	
Gallium	5900		Thoria	4160	
Gamboge	1200		Thorium	11700	
Garnet	3200	4300	Timber	0	
Gas_carbon	1900		Tin	7280	
Gelatin	1300		Titanium	4500	
Germanium	5320		Topaz	3500	3600
Glass,_common	2400	2800	Tourmaline	3000	3200
Glass,_flint	2900	5900	Tungsten	19200	
Glass,_Pyrex	2210		Tungsten_carbide	14000	15000
Glass-wool	25		Uranium	19100	
Glue	1300		Urethane_foam_(Urea_formaldehyde_foam)	80	
Gneiss	2690		Vanadium	6100	
Gold	19290		Vermiculite	120	
Granite	2600	2800	Wax,_sealing	1800	
Graphite	2300	2700	White_metal	7500	10000
Gum_arabic	1300	1400	Wood_(seasoned)	0	
Gypsum	2300		Wood_woo_slab	500	800
Hardboard	1000		Zinc	7120	
Hematite	4900	5300			

1.4.4 THERMAL DIFFUSIVITY

Thermal diffusivity is the *rate of transfer of heat of a material from the hot side to the cold side* - a measure of how quickly a material can absorb heat from its surroundings. It can be calculated by taking the thermal conductivity divided by density and specific heat capacity at constant pressure.

Thermal diffusivity has the SI derived unit of **[m²/s]**, and is usually denoted **α** but a, κ, K, and D are also used. The formula is:

$$\alpha(T) = \frac{k(T)}{\rho(T)C_p(T)} \qquad (1.4.4\text{-}1)$$

where
k = is thermal conductivity [W/(m·K)]
ρ = density [kg/m³]
C_p = specific heat capacity [J/(kg·K)]

Eventhogh it is just a derived property, due to simplification of the differential equation, it is frequently used in heat conduction problem solving.

2.0 ONE DIMENSIONAL SOLUTIONS

As it is seen previously developed thermal conductivity conservation of energy equation

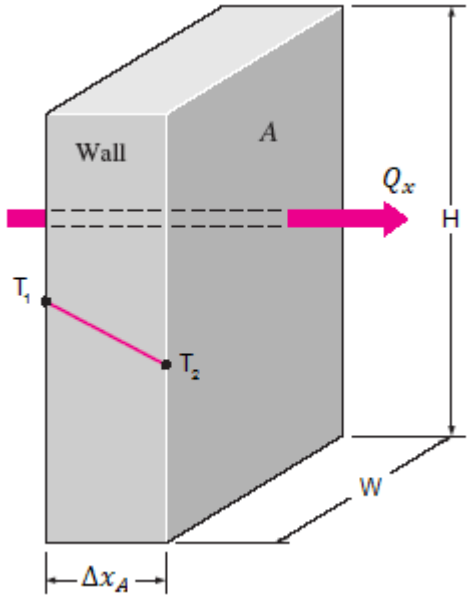
$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} + \frac{g}{k} = \frac{\rho(T)C_p(T)}{k(T)}\frac{\partial T}{\partial t} = \frac{1}{\alpha}\frac{\partial T}{\partial t}$$

It requires four dimensional treatment of the problem. Four dimensional differential equation solutions involve a high degree of mathematics. Before developing multidimensional solutions, some simplified one dimensional solutions can be investigated.

2.1 THE WALL

Let us consider a one dimensional steady state problem, general cartesian equation

$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}+\frac{\partial^2 T}{\partial z^2}+\frac{g}{k}=\frac{\rho(T)C_p(T)}{k(T)}\frac{\partial T}{\partial t}=\frac{1}{\alpha}\frac{\partial T}{\partial t}$ will reduce to



$\frac{\partial^2 T}{\partial x^2}=0$ we will consider two boundary conditions. As the first condition let us assume that temperature at both side of the wall is known. So boundary conditions for the D.E. will become:

x=0 T=T_1

$x=\Delta x_A$ T=T_2

Solving of the differential equation yields

$\frac{\partial T}{\partial x}=C_1$ (This equation indicates that heat transfer is constant $q_x=\frac{Q_x}{A}=-k_A\frac{dT}{dx}$)

$T=C_1x+C_2$ (This equation indicates that temperature profile is a linear function)

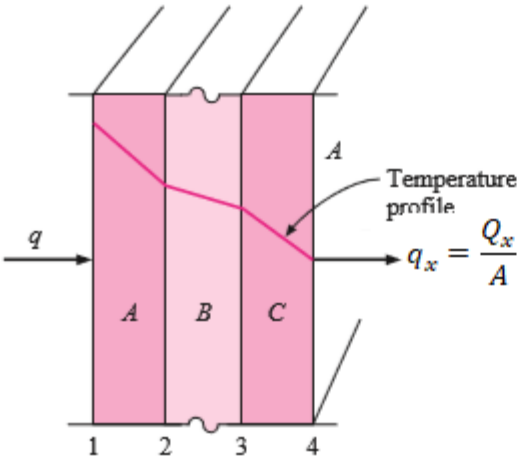
Applying boundary conditions:

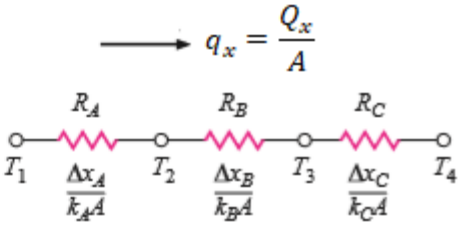
$T_1=C_2$

$T_2=C_1\Delta x_A+T_1$

$\frac{T_2-T_1}{\Delta x_A}=C_1$ Also note that $q_x=\frac{Q_x}{A}=-k_A\frac{dT}{dx}=-k_AC_1=-k_A\frac{T_2-T_1}{\Delta x_A}=\frac{T_1-T_2}{\left(\frac{\Delta x_A}{k_A}\right)}=\frac{T_1-T_2}{R_A}=constant$

Where R is called thermal resistance





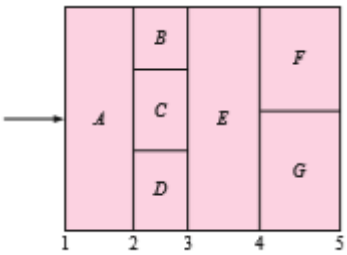
If we have more than one layer of wall like 3 layers A,B and C shown in the figure and since

$q_x = constant$ due to solution of the differential equation

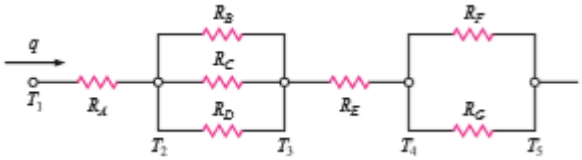
$$q_x = \frac{Q_x}{A} = \frac{T_1 - T_2}{\left(\frac{\Delta x_A}{k_A}\right)} = \frac{T_2 - T_3}{\left(\frac{\Delta x_B}{k_B}\right)} = \frac{T_3 - T_4}{\left(\frac{\Delta x_C}{k_C}\right)}$$

$$q_x = \frac{Q_x}{A} = \frac{T_1 - T_2}{R_A} = \frac{T_2 - T_3}{R_B} = \frac{T_3 - T_4}{R_C} = \frac{T_1 - T_4}{R_A + R_B + R_C} = \frac{T_1 - T_4}{R}$$

Thermal resistance concept works similar to electrical resistances, so when you have a wall in the figure below



Resistances of parallel thermal layers (like A,B and C or F and G) behave like parallel resistances



$$R = R_A + \frac{1}{\frac{1}{R_B} + \frac{1}{R_C} + \frac{1}{R_D}} + R_E + \frac{1}{\frac{1}{R_F} + \frac{1}{R_G}}$$

$$q_x = \frac{Q_x}{A} = \frac{T_1 - T_5}{R}$$

Now consider convective heat transfer outside of the first layer.

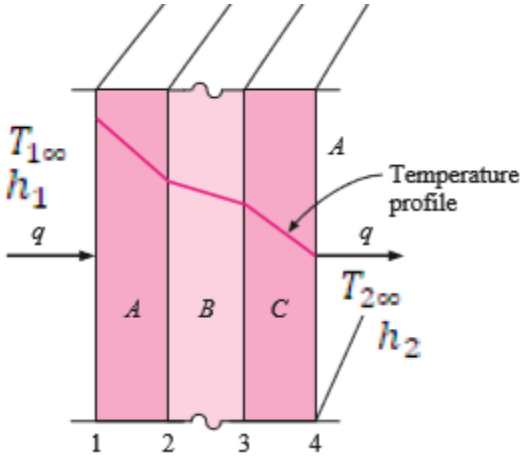
$\frac{\partial^2 T}{\partial x^2} = 0 \quad \frac{\partial T}{\partial x} = C_1 \quad T = C_1 x + C_2$

The boundary condition

x=0 $q_x = \frac{Q_x}{A} = -k_A \frac{dT}{dx} = h_1(T_1 - T_{1\infty}) = -k_A C_1 = \frac{(T_{1\infty} - T_1)}{\frac{1}{h_1}} = \frac{(T_{1\infty} - T_1)}{\frac{1}{h_1}} = \frac{(T_{1\infty} - T_1)}{R_1}$

$x = \Delta x_A$ T=T_2

Therefore for a wall with convective heat transfer in both sides



$$q_x = \frac{Q_x}{A} = \frac{(T_{1\infty} - T_1)}{\left(\frac{1}{h_1}\right)} = \frac{T_1 - T_2}{\left(\frac{\Delta x_A}{k_A}\right)} = \frac{T_2 - T_3}{\left(\frac{\Delta x_B}{k_B}\right)} = \frac{T_3 - T_4}{\left(\frac{\Delta x_C}{k_C}\right)} = \frac{(T_4 - T_{2\infty})}{\left(\frac{1}{h_2}\right)}$$

$$q_x = \frac{Q_x}{A} = \frac{(T_{1\infty} - T_1)}{R_1} = \frac{T_1 - T_2}{R_A} = \frac{T_2 - T_3}{R_B} = \frac{T_3 - T_4}{R_C} = \frac{(T_4 - T_{2\infty})}{R_2} = \frac{T_{1\infty} - T_{2\infty}}{R_1 + R_A + R_B + R_C + R_2} = \frac{T_{1\infty} - T_{2\infty}}{R}$$

or

$$Q_x = \frac{T_{1\infty} - T_{2\infty}}{\left(\frac{1}{h_1 A}\right) + \left(\frac{\Delta x_A}{k_A A}\right) + \left(\frac{\Delta x_B}{k_B A}\right) + \left(\frac{\Delta x_C}{k_C A}\right) + \left(\frac{1}{h_2 A}\right)} = UA(T_{1\infty} - T_{2\infty})$$

Where U is called overall heat transfer coefficient

Another case for one dimensional wall: Assume that thermal conductivity is changing as a function of temperature

$k(T) = k_0(1 + \beta T)$

$\frac{\partial}{\partial x}\left(k(T)\frac{\partial T(x)}{\partial x}\right) = 0$ B.C: x=0 T=T_1 $x = \Delta x$ T=T_2

$k(T)\frac{\partial T(x)}{\partial x} = C_1$ $(1 + \beta T)\frac{\partial T(x)}{\partial x} = C_1$ $\int_{T=T_1}^{T_2}(1 + \beta T)dT = C_1 \Delta x$

$(T_2 - T_1) + \frac{\beta}{2}(T_2^2 - T_1^2) = C_1 \Delta x$

$q_x = \frac{Q_x}{A} = -\frac{k_0}{2\Delta x}\left[(T_2 - T_1) + \frac{\beta}{2}(T_2^2 - T_1^2)\right]$

Another case for one dimensional wall: Assume a constant heat generation g_0 in the wall

$\frac{\partial^2 T}{\partial x^2} + \frac{g_0}{k} = 0$

$\frac{\partial T}{\partial x} = -\frac{g_0}{k}x + C_1$

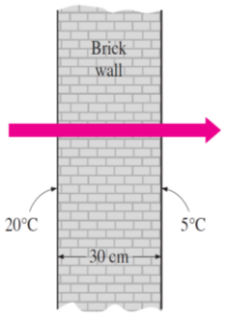
$T(x) = -\frac{g_0}{k}\frac{x^2}{2} + C_1 x + C_2$

B.C. x=0 T=T_1

$x = \Delta x$ T=T_2

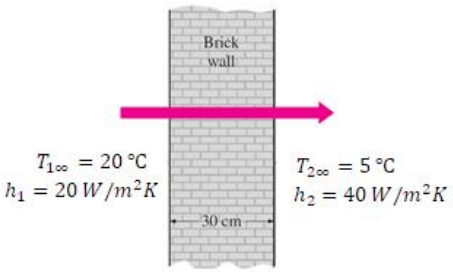
$T_1 = C_2$ $T_2 = -\frac{g_0}{k}\frac{\Delta x^2}{2} + C_1 \Delta x + T_1$ $\frac{T_2 - T_1}{\Delta x} + \frac{g_0}{k}\frac{\Delta x}{2} = C_1$ $R = \sum R_i$

PROBLEM: The inner and outer surfaces of a 5 m x 5 m brick wall of thickness 0.3 m and thermal conductivity k=0.7 W/mK are maintained at temperatures of 20 C and 5 C, respectively. Determine the heat transfer through the wall.



A	25	m^2	
k	0.7	W/mK	
T1	20	C	
T2	5	C	
L	0.3	m	
Q	875	W	k*A*ΔT/L

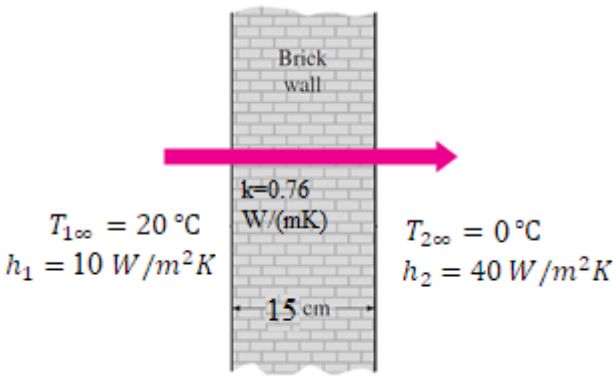
PROBLEM: Surface of a brick wall of a 5 m x 5 m of thikness 0.3 m and thermal conductivity k=0.7 W/mK exposed of air $T_{1\infty}$ =20 C and h_1=20 W/m^{2}K on one side and $T_{2\infty}$ =5 C and h_1=40 W/m^{2}K on the other side. Determine the heat transfer through the wall.



A	25	m^2	
k	0.7	W/mK	
T1inf	20	C	
h1	20	W/m^2K	
T2inf	5	C	
h2	40	W/m^2K	
L	0.3	m	
R1	0.05	m^2K/W	1/h1
R2	0.4285714	m^2K/W	L/k
R3	0.025	m^2K/W	1/h2
R=R1+R2+R3	0.5035714	m^2K/W	
Q	744.68085	W	A/R*ΔT

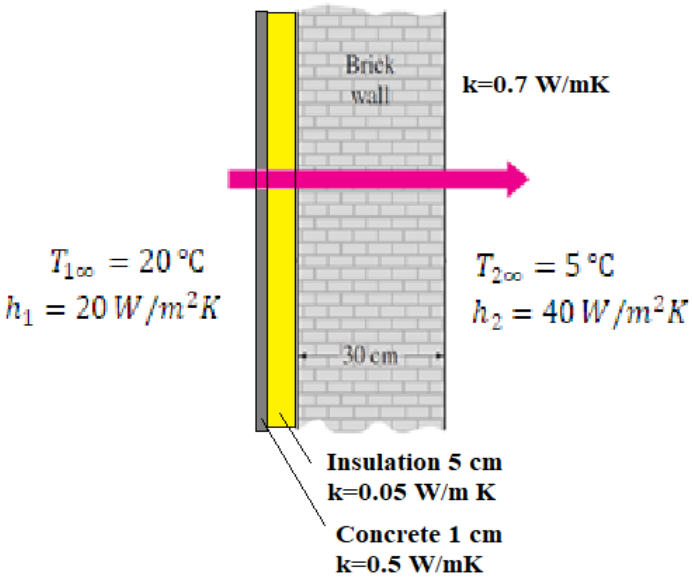
PROBLEM:

A brick wall of thickness L_1=15x10^{-2} m has thermal conductivity k_1=0.76 W/(mK). Inside surface is exposed to air at T_i=20 C, h_i=10 W/(m^{2}K) and and the outside surfaces T_0 =0 C, h_0=40 W/(m^{2}K). Determine the rate of heat loss per square meter of Wall surface



A	1	m^2	
k	0.76	W/mK	
T1inf	20	C	
h1	10	W/m^2K	
T2inf	0	C	
h2	40	W/m^2K	
L	0.15	m	
R1	0.1	m^2K/W	1/h1
R2	0.1973684	m^2K/W	L/k
R3	0.025	m^2K/W	1/h2
R=R1+R2+R	0.3223684	m^2K/W	
Q	744.68085	W	A/R*DT

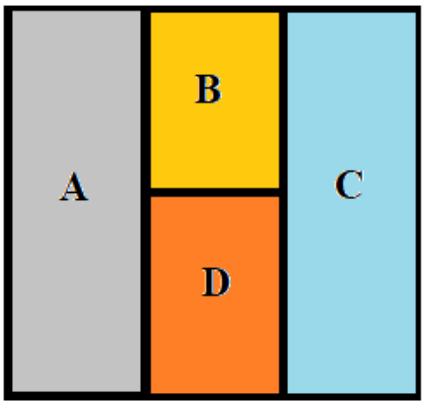
PROBLEM: Surface of a brick wall of a 5 m x 5 m has three layers. The first layer is concrete 1 cm thick and has a thermal conductivity coefficient of k=0.5 W/(mK), The second layer is an insulation of 5 cm thick and has a thermal conductivity coefficient of k=0.05 W/(mK). The thisd layer is brick wall of thikness 0.3 m and thermal conductivity k=0.7 W/mK. The composit wall is exposed of air $T_{1\infty}$ =20 C and h_1=20 W/m^{2}K on one side and $T_{2\infty}$ =5 C and h_1=40 W/m^{2}K on the other side. Determine the heat transfer through the wall.



A	25	m^2	
k1	0.5	W/mK	
k2	0.05	W/mK	
k3	0.7	W/mK	
T1inf	20	C	
h1	20	W/m^{2}K	
T2inf	5	C	
h2	40	W/m^{2}K	
L1	0.01	m	
L2	0.05	m	
L3	0.3	m	
R1	0.05	m^{2}K/W	1/h1
R2	0.02	m^{2}K/W	L1/k1
R3	1	m^{2}K/W	L2/k2
R4	0.4285714	m^{2}K/W	L3/k3
R5	0.025	m^{2}K/W	1/h2
R= R1+R2+R3+R4+	1.5235714	m^{2}K/W	
Q	246.13221	W	A/R*ΔT

LA	0.04	m
LB	0.1	m
LC	0.05	m
LD	0.1	m
H	2	m
W	1	m
kA	20	W/mK
kB	70	W/mK
kC	50	W/mK
kD	20	W/mK
RA	0.001	K/W
RB	0.0014286	K/W
RC	0.0005	K/W
RD	0.005	K/W
RE	0.0011111	K/W
Rtotal	0.0026111	K/W
ΔT	400	K
Q	153191.49	W

PROBLEM: A composite wall consisting of four different materials is shown in the side sketch. Since the upper and lower surfaces are insulated, the heat flow can be assumed to be one dimensional. Surface temperature difference ΔT=400 ℃ between the outer surfaces. Depth of the wall is W=1 m. Calculate heat transfer.



$$R_A = \frac{L_A}{k_A(2W)} \quad R_B = \frac{L_B}{k_B(W)} \quad R_D = \frac{L_D}{k_D(W)}$$

$$R_C = \frac{L_C}{k_C(2W)}$$

$$\frac{1}{R_E} = \left[\frac{1}{R_B} + \frac{1}{R_D}\right]$$

$$R_{total} = R_A + R_E + R_C$$

$$Q = \frac{\Delta T}{R_{total}}$$

LA	0.04	m
LB	0.1	m
LC	0.05	m
LD	0.1	m
H	2	m
W	1	m
kA	20	W/mK
kB	70	W/mK
kC	50	W/mK
kD	20	W/mK
RA	0.001	K/W
RB	0.0014286	K/W
RC	0.0005	K/W
RD	0.005	K/W
RE	0.0011111	K/W
Rtotal	0.0026111	K/W
ΔT	400	K
Q	153191.49	W

PROBLEM : A composite plane wall consist of two layers. First layer is sandstone(k =1.7 W/(mK)) and second layer is portland cement(k=0.29 W/(mK)). The contact conductance between layers A and B is h=3 W/(m^{2}K). Outside of sandstone layer lays air at T=200 ℃ and thermal convectivity of boundary is h=20 W/(m^{2}K) and Outside of cement layer lays air at T=20 ℃ and thermal convectivity of boundary is h=20 W/(m^{2}K). Sandstone layer thickness is 2x10^{-2} m and Portland cement layer thickness 0.08 m. Calculate heat loss per m^2 of wall area

sandstone wall

thickness	2.00E-02	m
k	1.7	W/(mK)
R	1.18E-02	m^{2}K/W

Portland cement wall

thickness	0.08	m
k	0.29	W/(mK)
R	0.275862	m^{2}K/W

contact conductance

h	3	W/(m2K)
R	0.333333	m^{2}K/W

air outside sandstone layer

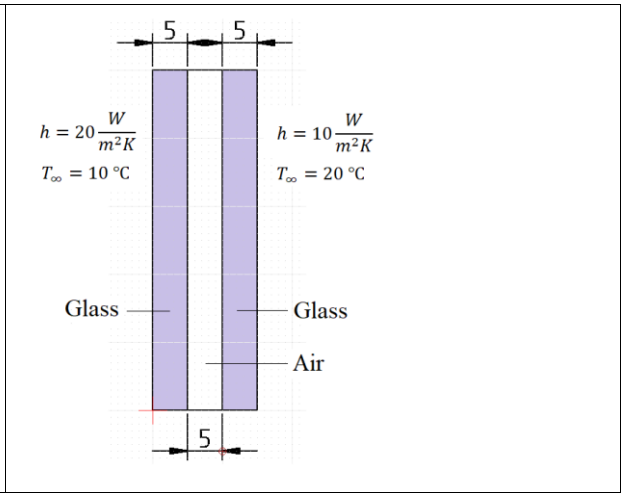
T	200	derece C
h	20	W/m^{2}K
R	0.05	m^{2}K/W

air outside portland cement layer

T	20	degree C
h	20	w/m^{2}K
R	0.05	m^{2}K/W
Total R	**7.21E-01**	**m^{2}K/W**
dT	**180**	**K**
Q	**1.30E+02**	**W/m^2**

PROBLEM : A window consist of two layer of glasses seperated by still air. The thermal conductivity of the glass is k=0.96 W/(mK). Thermal conductivity of still air can be taken as: k=0.05 W/(mK). The thickness of glasses are $5x10^{-3}$ m each and thickness of air layer is also $5x10^{-3}$ m. Convection heat transfer coefficient for outside is h=20 W/(m^{2}K) and inside h=10 W/(m^{2}K). If outside temperature is 0 ℃ and inside temperature is 20 ℃, and window area is A=4 m^2. Calculate the total heat loss from the window.

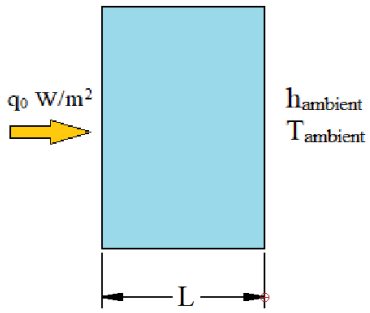
kglass	0.96	W/mK)
kair	0.05	W/(mK)
dx glass	5.00E-03	m
dx air	5.00E-03	m
h outside	20	W/(m^{2}K)
T outside	0	Degree C
h inside	10	W/(m^{2}K)
T inside	20	Degree C
A	4	m^2
R outside	0.05	(m^{2}K) /W
R glass 1	5.21E-03	(m^{2}K) /W
R air	1.00E-01	(m^{2}K) /W
R glass 2	5.21E-03	(m^{2}K) /W
R inside	0.1	(m^{2}K) /W
R total	0.260417	(m^{2}K) /W
Q	307.2	W



PROBLEM : A window consist of one layer of glass The thermal conductivity of the glass is k=0.96 W/(mKThe thickness of glas is $5x10^{-3}$ m. Convection heat transfer coefficient for outside is h=20 W/(m^{2}K) and inside h=10 W/(m^{2}K). If outside temperature is 0 ℃ and inside temperature is 20 ℃, and window area is A=4 m^2. Calculate the total heat loss from the window.

kglass	0.96	W/mK)
dx glass	5.00E-03	m
h outside	20	W/(m^{2}K)
T outside	0	0
h inside	10	W/(m^{2}K)
T inside	20	0
A	4	m^2
R outside	0.05	(m^{2}K) /W
R glass 1	5.21E-03	(m^{2}K) /W
R inside	0.1	(m^{2}K) /W
R total	0.155208	(m^{2}K) /W
Q	515.4362	W

PROBLEM: A plane Wall of thickness L is exposed to a uniform heat flux of q_0 W/m^2 on one side and dissipates heat by convection with a heat transfer coefficient $h_{ambient}$ into the ambient air temperature $T_{ambient}$ on the other side. Write the mathematical formulation of this problem for the determination of one dimensional steady state temperature distribution T(x) within the Wall



Let us consider a one dimensional steady state problem, general cartesian equation
$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}+\frac{\partial^2 T}{\partial z^2}+\frac{g}{k}=\frac{\rho(T)C_p(T)}{k(T)}\frac{\partial T}{\partial t}=\frac{1}{\alpha}\frac{\partial T}{\partial t}$ will reduce to
$\frac{\partial^2 T}{\partial x^2}=0$ we will consider two boundary conditions.
Boundary conditions:
$-k\frac{\partial T}{\partial x}=q_0$ x=0
$-k\frac{\partial T}{\partial x}=h_{ambient}(T-T_{ambient})$

PROBLEM: A pressure vessel for nuclear reactor is approximated as a large flat plate of thickness L. The inside surface of the plate at x=0 is insulated, the outside surface at x=L is maintained at a uniform temperature T_2 and the gamma ray heating of the plate can be represented as a heat generation term in the form
$g(x)=g_0e^{-\gamma x}$ W/m³
Where g_0 and γ are constants and x measured from the insulated inside surface.

a) Develop an expression for temperature distribution in the plate.
b) Develop an expression for the temperature at the insulated surface x=L

Let us consider a one dimensional steady state problem, general cartesian equation
$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}+\frac{\partial^2 T}{\partial z^2}+\frac{g}{k}=\frac{\rho(T)C_p(T)}{k(T)}\frac{\partial T}{\partial t}=\frac{1}{\alpha}\frac{\partial T}{\partial t}$ will reduce to
$\frac{\partial^2 T}{\partial x^2}+\frac{g(x)}{k}=0$
Boundaryconditions:
x=0 $\frac{\partial T}{\partial x}=0$
x=L T=T_2
solution:
$\frac{\partial^2 T}{\partial x^2}+\frac{g_0e^{-\gamma x}}{k}=0$ $\frac{\partial^2 T}{\partial x^2}=-\frac{g_0e^{-\gamma x}}{k}$
$\frac{\partial T}{\partial x}=\frac{g_0e^{-\gamma x}}{\gamma k}+C_1$
From boundary condition x=0 $\frac{\partial T}{\partial x}=0$
$C_1=-\frac{g_0}{\gamma k}$
$T(x)=-\frac{g_0e^{-\gamma x}}{\gamma^2 k}-\frac{g_0}{\gamma k}x+C_2$
From boundary condition x=L T=T_2
$T_2=-\frac{g_0e^{-\gamma L}}{\gamma^2 k}-\frac{g_0}{\gamma k}L+C_2$
$C_2=T_2+\frac{g_0e^{-\gamma L}}{\gamma^2 k}+\frac{g_0}{\gamma k}L$

PROBLEM: A pressure vessel for nuclear reactor is approximated as a large flat plate of thickness L. The inside surface of the plate at x=0 is maintained at a uniform temperature T_1, the outside surface at x=L is maintained at a uniform temperature T_2 and the gamma ray heating of the plate can be represented as a heat generation term in the form
$g(x)=g_0e^{-\gamma x}$ W/m³

Where g_0 and γ are constants and x measured from the insulated inside surface. Thermal conductivity of the rod is given as a polynomial

$$k(T) = a_0 + a_1T + a_2T^2 + a_3T^3$$

Develop an expression for temperature distribution in the plate.

Let us consider a one dimensional steady state problem, general cartesian equation

$$\left[\frac{\partial}{\partial x}\left(k_x(T)\frac{dT}{dx}\right) + \frac{\partial}{\partial y}\left(k_y(T)\frac{dT}{dy}\right) + \frac{\partial}{\partial z}\left(k_z(T)\frac{dT}{dz}\right)\right] + g = 0$$

will reduce to

$$\frac{\partial}{\partial x}\left(k_x(T)\frac{dT}{dx}\right) + g = 0$$

$$\frac{\partial}{\partial x}\left([a_0 + a_1T + a_2T^2 + a_3T^3]\frac{dT}{dx}\right) + g_0e^{-\gamma x} = 0$$

Boundary conditions:

x=0 $T=T_1$ and x=L $T=T_2$

$$\frac{\partial}{\partial x}\left([a_0 + a_1T + a_2T^2 + a_3T^3]\frac{dT}{dx}\right) = -g_0e^{-\gamma x}$$

$$d\left([a_0 + a_1T + a_2T^2 + a_3T^3]\frac{dT}{dx}\right) = -g_0e^{-\gamma x}dx$$

Integrating the equation gives

$$\int d\left([a_0 + a_1T + a_2T^2 + a_3T^3]\frac{dT}{dx}\right) = -\int g_0e^{-\gamma x}dx$$

$$\left([a_0 + a_1T + a_2T^2 + a_3T^3]\frac{dT}{dx}\right) = \frac{g_0e^{-\gamma x}}{\gamma} + C_1$$

$$[a_0 + a_1T + a_2T^2 + a_3T^3]dT = \left[\frac{g_0e^{-\gamma x}}{\gamma} + C_1\right]dx$$

$$\int [a_0 + a_1T + a_2T^2 + a_3T^3]dT = \int \left[\frac{g_0e^{-\gamma x}}{\gamma} + C_1\right]dx$$

$$\left[a_0T + \frac{a_1}{2}T^2 + \frac{a_2}{3}T^3 + \frac{a_3}{4}T^4\right] = -\frac{g_0e^{-\gamma x}}{\gamma^2} + C_1x + C_2$$

First boundary condition: x=0 $T=T_1$

$$\left[a_0T_1 + \frac{a_1}{2}T_1^2 + \frac{a_2}{3}T_1^3 + \frac{a_3}{4}T_1^4\right] = -\frac{g_0}{\gamma^2} + C_2$$

$$C_2 = \frac{g_0}{\gamma^2} + \left[a_0T_1 + \frac{a_1}{2}T_1^2 + \frac{a_2}{3}T_1^3 + \frac{a_3}{4}T_1^4\right]$$

Second boundary condition: x=L $T=T_2$

$$\left[a_0T + \frac{a_1}{2}T^2 + \frac{a_2}{3}T^3 + \frac{a_3}{4}T^4\right] = -\frac{g_0e^{-\gamma x}}{\gamma^2} + C_1x + \frac{g_0}{\gamma^2} + \left[a_0T_1 + \frac{a_1}{2}T_1^2 + \frac{a_2}{3}T_1^3 + \frac{a_3}{4}T_1^4\right]$$

$$\left[a_0T_2 + \frac{a_1}{2}T_2^2 + \frac{a_2}{3}T_2^3 + \frac{a_3}{4}T_2^4\right] = -\frac{g_0e^{-\gamma L}}{\gamma^2} + C_1L + \frac{g_0}{\gamma^2} + \left[a_0T_1 + \frac{a_1}{2}T_1^2 + \frac{a_2}{3}T_1^3 + \frac{a_3}{4}T_1^4\right]$$

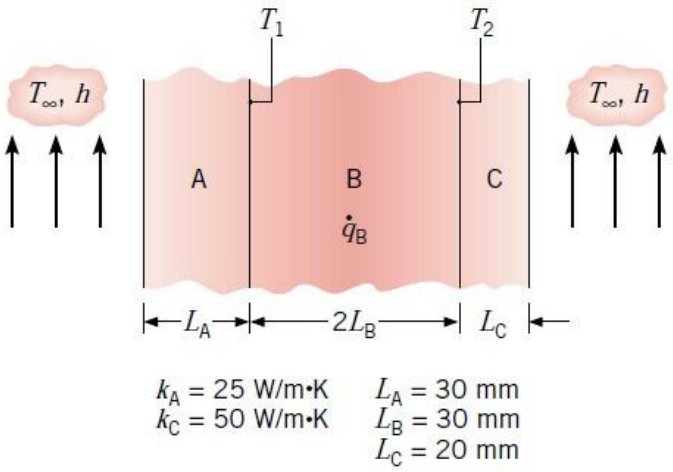
$$C_1 = \frac{1}{L}\left[a_0(T_2 - T_1) + \frac{a_1}{2}(T_2^2 - T_1^2) + \frac{a_2}{3}(T_2^3 - T_1^3) + \frac{a_3}{4}(T_2^4 - T_1^3)\right] + \frac{g_0 e^{-\gamma L}}{L\gamma^2} - \frac{g_0}{L\gamma^2}$$

So temperature profile becomes:

$$\left[a_0 T + \frac{a_1}{2}T^2 + \frac{a_2}{3}T^3 + \frac{a_3}{4}T^4\right] = -\frac{g_0 e^{-\gamma x}}{\gamma^2} + C_1 x + C_2$$

Solving of T form this equation directly is not possible. A numerical root finding method should be apply to solve this equation

PROBLEM:



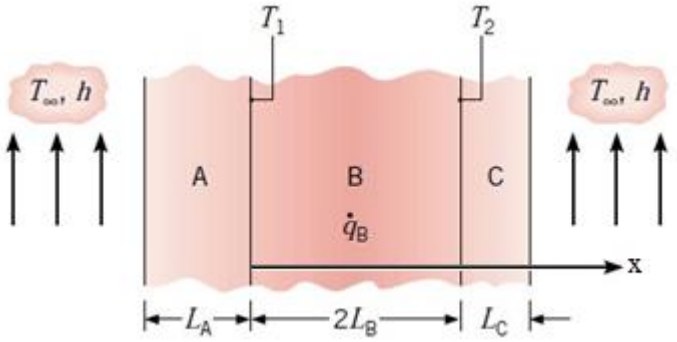
Fluid temperature and convection heat transfer coefficient are 25°C and 1000 W/m2 K. There is no generation in walls A and C. However, wall B experiences uniform heat generation. T_1= 261 °C and T_2= 211 °C. Determine the volumetric heat generation q_B and the thermal conductivity k_B.

T_∞	25.000000	degree C
h	1000.000000	W/m2K)
T_1	261.000000	degree C
T_2	211.000000	degree C
k_A	25.000000	W/(mK)
k_C	50.000000	W/(mK)
L_A	0.030000	m
L_B	0.030000	m
L_c	0.020000	m
R_{conv}=1/h	0.001000	m2K/W
R_A=L_A/k_A	0.001200	m2K/W
R_1=Rconv+RA	0.002200	m2K/W
Q1=(T_∞-T_1)/R	107272.727273	W/m2
R_C=L_C/k_C	0.000400	m2K/W
R_2=R_{conv}+R_C	0.001400	m2K/W
Q2=(T_2- T_∞)/R_2	132857.142857	W/m2
QB=Q_2+Q_1	240129.870130	W/m2
g_B=Q_B/(2*L_B)	4002164.502165	W/m3
C_2=T_1	261.000000	degree C
k_B=(-g_B*2*L_B^2+Q_1*L_B)/(T_2-T_1)	15.350649	W/(mK)
C1=Q1/kB	6988.155668	degree C/m

$$Q_1 = \frac{(T_1 - T_\infty)}{R_{conv} + R_A} = \frac{(T_1 - T_\infty)}{\frac{1}{h} + \frac{L_A}{k_A}} = 107272.727273 \ \mathrm{W/m^2}$$

$$Q_2 = \frac{(T_2 - T_\infty)}{R_{conv} + R_C} = \frac{(T_2 - T_\infty)}{\frac{1}{h} + \frac{L_C}{k_C}} = 132857.142857 \ \text{W}/m^2$$

$$g_B = \frac{Q_1 + Q_2}{2L_B} = 4002164.502165 \ \text{W}/m^3$$



Let us select x axis from the starting point of heat generation region

$$\frac{\partial^2 T(x)}{\partial x^2} + \frac{g_B}{k_B} = 0$$

$$\frac{\partial^2 T(x)}{\partial x^2} = -\frac{g_B}{k_B}$$

$$\frac{\partial T(x)}{\partial x} = -\frac{g_B}{k_B}x + C_1$$

$$T(x) = -\frac{g_B}{k_B}\frac{x^2}{2} + C_1 x + C_2$$

$$T(x = 0) = C_2 = 261 \ ℃$$

$$Q(x) = -k_B \frac{\partial T(x)}{\partial x} = -k_B\left(-\frac{g_B}{k_B}x + C_1\right)$$

$$Q(x = 0) = -k_B \frac{\partial T(x)}{\partial x} = -k_B C_1 = -Q_1$$

$$C_1 = \frac{Q_1}{k_B}$$

$$T(x) = -\frac{g_B}{k_B}\frac{x^2}{2} + C_1 x + C_2$$

$$T(x = 2L_B) = -\frac{g_B}{k_B}2L_B^2 + C_1 2L_B + C_2$$

$$T_2 = -\frac{g_B}{k_B}2L_B^2 + \frac{Q_1}{k_B}2L_B + T_1$$

$$k_B = \frac{(T_2 - T_1)}{-g_B 2L_B^2 + Q_1 2L_B} = 15.350649 \ \text{W}/(\text{mK})$$

2.2 CYLINDER – PIPE

PROBLEM: A solid bar of Radius r=b is initially at a uniform temperature T_0. For times t>0, it is cooled by convection from the surface at r=b into ambient air at temperature T_{air} with a heat transfer coefficient h_{air} . Write the mathematical formulation of this heat conduction problem for the determination of one dimensional time dependent temperature distribution T(r,t) within the rod



Let us consider a one dimensional steady state problem, general cylindrical coordinates

$$\frac{1}{r}\frac{\partial}{\partial r}\left(k_r(T)r\frac{\partial T}{\partial r}\right)+\frac{1}{r^2}\frac{\partial}{\partial \phi}\left(k_\phi(T)\frac{\partial T}{\partial \phi}\right)++\frac{\partial}{\partial z}\left(k_z(T)\frac{\partial T}{\partial z}\right)+g=\rho C_p(T)\frac{\partial T}{\partial t}$$

Boundary conditions:

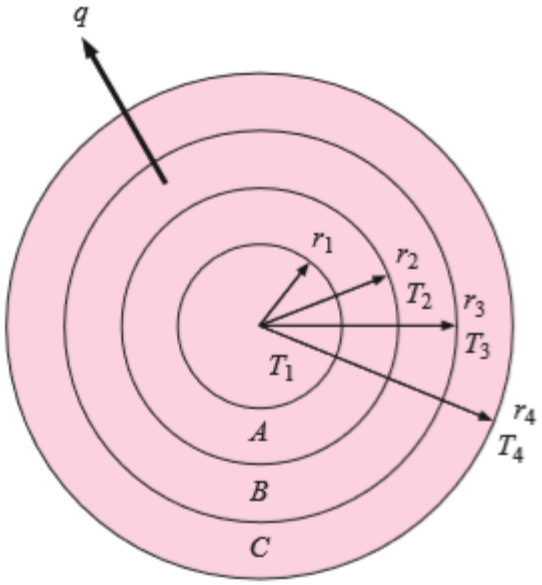
$$\frac{1}{r}\frac{\partial}{\partial r}\left(k_r(T)r\frac{\partial T}{\partial r}\right)=\rho C_p(T)\frac{\partial T}{\partial t} \quad \text{in } 0\le r\le b \quad t>0$$

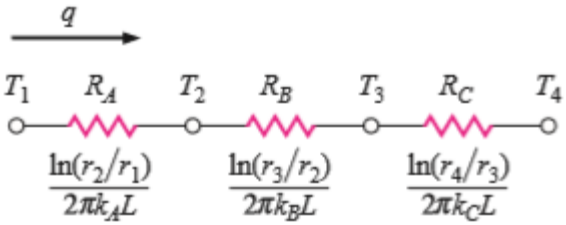
$$\frac{\partial T}{\partial r}=0 \quad \text{at r=0} \quad t>0$$

$$-k_r(T)\frac{\partial T}{\partial r}=h_{air}(T-T_{air}) \quad \text{at r=b} \quad t>0$$

$$T=T_0 \quad \text{at t=0} \quad 0\le r\le b$$

One dimensional steady state case:





Let us consider a one dimensional steady state problem, general cylindrical coordinates

$$\frac{1}{r}\frac{\partial}{\partial r}\left(k_r(T)r\frac{\partial T}{\partial r}\right)+\frac{1}{r^2}\frac{\partial}{\partial \phi}\left(k_\phi(T)\frac{\partial T}{\partial \phi}\right)++\frac{\partial}{\partial z}\left(k_z(T)\frac{\partial T}{\partial z}\right)+g=\rho C_p(T)\frac{\partial T}{\partial t}$$

$$\frac{1}{r}\frac{\partial}{\partial r}\left(k_r(T)r\frac{\partial T}{\partial r}\right)=0$$

Bounday conditions:

r=r1 T=T_1

r=r2 T=T_2

$$k_r(T)r\frac{\partial T}{\partial r}=C_1 \quad \frac{\partial T}{\partial r}=\frac{C_1}{r} \quad T=C_1\ln(r)+C_2$$

$T_1 = C_1 \ln(r_1) + C_2$

$T_2 = C_1 \ln(r_2) + C_2$

$\frac{(T_2 - T_1)}{\ln\left(\frac{r_2}{r_1}\right)} = C_1$

$$q = \frac{2\pi kL(T_2 - T_1)}{\ln\left(\frac{r_2}{r_1}\right)} = \frac{(T_2 - T_1)}{\frac{\ln\left(\frac{r_2}{r_1}\right)}{2\pi kL}} = \frac{(T_2 - T_1)}{R}$$

For the serial case of resistances as shown in the figure

$$q_r = \frac{(T_4 - T_1)}{\frac{\ln\left(\frac{r_2}{r_1}\right)}{2\pi k_A L} + \frac{\ln\left(\frac{r_3}{r_2}\right)}{2\pi k_B L} + \frac{\ln\left(\frac{r_4}{r_3}\right)}{2\pi k_C L}} = \frac{(T_4 - T_1)}{R_A + R_B + R_C}$$

Now consider convective heat transfer outside of the first layer. The boundary condition

$\frac{1}{r}\frac{\partial}{\partial r}\left(k_r(T) r \frac{\partial T}{\partial r}\right) = 0 \quad k_r(T) r \frac{\partial T}{\partial r} = C_1 \quad \frac{\partial T}{\partial r} = \frac{C_1}{r}$

$$r = r_1 \quad Q_r = -k_A(2\pi rL)\frac{dT}{dr} = h_1(2\pi rL)(T_1 - T_{1\infty}) = \frac{(T_{1\infty} - T_1)}{\frac{1}{h_1(2\pi r_1 L)}} = \frac{(T_{1\infty} - T_1)}{R_1}$$

For the serial case of serial resistances

$$Q_r = \frac{(T_{1\infty} - T_{2\infty})}{\frac{1}{h_1(2\pi r_1 L)} + \frac{\ln\left(\frac{r_2}{r_1}\right)}{2\pi k_A L} + \frac{\ln\left(\frac{r_3}{r_2}\right)}{2\pi k_B L} + \frac{\ln\left(\frac{r_4}{r_3}\right)}{2\pi k_C L} + \frac{1}{h_2(2\pi r_2 L)}} = \frac{(T_{1\infty} - T_{2\infty})}{R_1 + R_A + R_B + R_C + R_2}$$

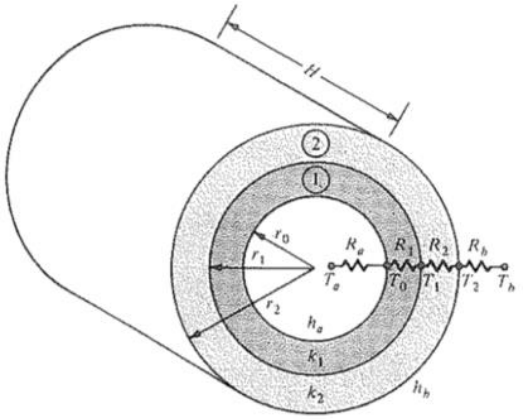
It should be note that inverse of the overall thermal resistance is called **overall heat transfer coefficient**

$$UA = \frac{1}{\frac{1}{h_1(2\pi r_1 L)} + \frac{\ln\left(\frac{r_2}{r_1}\right)}{2\pi k_A L} + \frac{\ln\left(\frac{r_3}{r_2}\right)}{2\pi k_B L} + \frac{\ln\left(\frac{r_4}{r_3}\right)}{2\pi k_C L} + \frac{1}{h_2(2\pi r_2 L)}} = \frac{1}{R_1 + R_A + R_B + R_C + R_2}$$

PROBLEM: A steel tube has 5×10^{-2} m ID, 7.6×10^{-2} m OD, and thermal conductivity coefficient k=15 W/mK. A hot gas T_a=330 C, h_a=400 W/m^{2}K flows inside the tube. The outer surface is exposed to cooler air at T_b=30 C with h_b=60 W/m^{2}K. Calculate the heat loss from the tube to the air for H=10 m of the tube.

PROBLEM		
ID	5.00E-02	m
OD	7.60E-02	m
K1	1.50E+01	W/mK
Ta	3.30E+02	C
ha	4.00E+02	W/M^2K
Tb	3.00E+01	C
hb	6.00E+01	W/m^2K
H	1.00E+01	m
r0	2.50E-02	m
r1	3.80E-02	m
A0	1.57E+00	m^2
A1	2.39E+00	m^2
R0	1.59E-03	K/W
R1	4.44E-04	K/W
R2	6.98E-03	K/W
R	9.02E-03	K/W
Q	**3.33E+04**	**W**

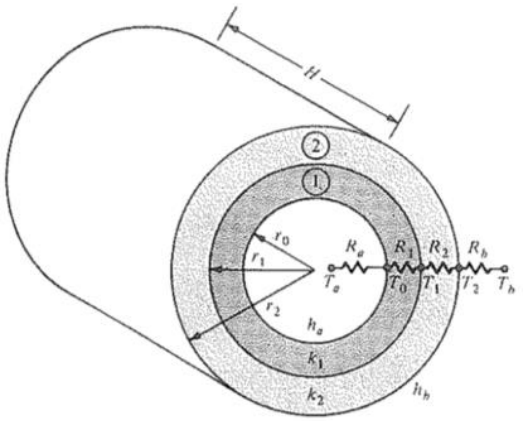
PROBLEM :



A steel tube with $5x10^{-2}$ m ID, $7.6 x10^{-2}$ m OD, and k=15 W/mK is covered with an insulative covering of thickness **$t=2 x10^{-2}$ m and k=0.2 W/mK**. A hot gas T_a=330 C, h_a=400 W/m^{2}K flows inside the tube. The outer surface of the insulation is exposed to cooler air at T_b=30 C with h_b=60 W/m^{2}K. Calculate the heat loss from the tube to the air for H=10 m of the tube.

PROBLEM		
ID	5.0000000E-02	m
OD	7.6000000E-02	m
K1	1.5000000E+01	W/mK
t	**2.0000000E-02**	m
k2	**2.0000000E-01**	W/mK
Ta	3.3000000E+02	C
ha	4.0000000E+02	W/M^2K
Tb	3.0000000E+01	C
hb	6.0000000E+01	W/m^2K
H	1.0000000E+01	m
r0	2.5000000E-02	m
r1	3.8000000E-02	m
r2	5.8000000E-02	m
A0	1.5707963E+00	m^2
A2	3.6442475E+00	m^2
R0	1.5915494E-03	K/W
R1	4.4866505E-04	K/W
R2	3.3649879E-02	K/W
R3	4.5734179E-03	K/W
R	4.0263511E-02	K/W
Q	**7.4509150E+03**	W

PROBLEM :

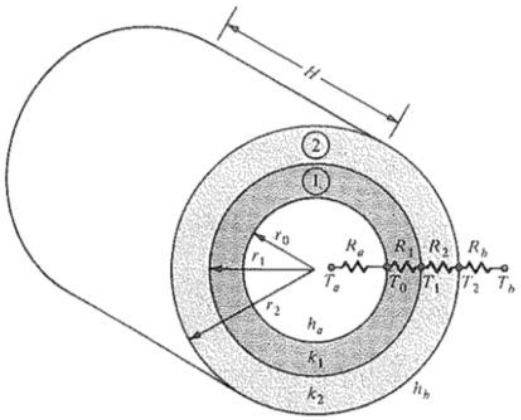


A steel tube with $5x10^{-2}$ m ID, $7.6 x10^{-2}$ m OD, and k=15 W/mK is covered with an insulative covering of thickness **$t=2 x10^{-2}$ m and k=0.05 W/mK**. A hot gas T_a=330 C, h_a=400 W/m^{2}K flows inside the tube. The outer surface of the insulation is exposed to cooler air at T_b=30 C with h_b=60 W/m^{2}K. Calculate the heat loss from the tube to the air for H=10 m of the tube.

PROBLEM		
ID	5.00E-02	m
OD	7.60E-02	m
K1	1.50E+01	W/mK
t	**2.00E-02**	**m**

k2	**5.00E-02**	**W/mK**
Ta	3.30E+02	C
ha	4.00E+02	W/M^2K
Tb	3.00E+01	C
hb	6.00E+01	W/m^2K
H	1.00E+01	m
r0	2.50E-02	m
r1	3.80E-02	m
r2	5.80E-02	m
A0	1.57E+00	m^2
A2	3.64E+00	m^2
R0	1.59E-03	K/W
R1	4.44E-04	K/W
R2	1.35E-01	K/W
R3	4.57E-03	K/W
R	1.41E-01	K/W
Q	**2.12E+03**	W

PROBLEM :



A steel tube with $5x10^{-2}$ m ID, 7.6 $x10^{-2}$ m OD, and k=15 W/mK is covered with an insulative covering of thickness **$t=5 x10^{-2}$ m and k=0.05 W/mK**. A hot gas T_a=330 C, h_a=400 W/m^2K flows inside the tube. The outer surface of the insulation is exposed to cooler air at T_b=30 C with h_b=60 W/m^2K. Calculate the heat loss from the tube to the air for H=10 m of the tube.

PROBLEM		
ID	5.00E-02	m
OD	7.60E-02	m
K1	1.50E+01	W/mK
t	**5.00E-02**	**m**
k2	**5.00E-02**	**W/mK**
Ta	3.30E+02	C
ha	4.00E+02	W/M^2K
Tb	3.00E+01	C
hb	6.00E+01	W/m^2K
H	1.00E+01	m
r0	2.50E-02	m
r1	3.80E-02	m
r2	8.80E-02	m
A0	1.57E+00	m^2
A2	5.53E+00	m^2
R0	1.59E-03	K/W
R1	4.44E-04	K/W
R2	2.67E-01	K/W
R3	3.01E-03	K/W
R	2.72E-01	K/W
Q	**1.10E+03**	W

PROBLEM :

A solid cylinder with a constant heat generation where the surface temperature at r=R is $T=T_w$ and thermal conductivity is constant. Calculate temperature profile.

$\frac{1}{r}\frac{\partial}{\partial r}\left(k_r(T)r\frac{\partial T}{\partial r}\right)+g_0=0 \quad \frac{\partial}{\partial r}\left(r\frac{\partial T}{\partial r}\right)=-\frac{g_0}{k}r$

B.C. T=T_w r=R $\frac{\partial T}{\partial r}=0$ r=0 (maximum temperature at the center due to heat generation)

$\left(r\frac{\partial T}{\partial r}\right)=-\frac{g_0}{k}\frac{r^2}{2}+C_1 \quad \frac{\partial T}{\partial r}=-\frac{g_0}{k}\frac{r}{2}+\frac{C_1}{r} \quad T=-\frac{g_0}{k}\frac{r^2}{4}+C_1\ln(r)+C_2$

B.C $\frac{\partial T}{\partial r}=0$ r=0 $\xrightarrow{yields}$ $C_1=0$

B.C. T=T_w r=R $\xrightarrow{yields}$ $T_w+\frac{g_0}{k}\frac{R^2}{4}=C_2$

$T=-\frac{g_0}{k}\frac{r^2}{4}+T_w+\frac{g_0}{k}\frac{R^2}{4} \quad T-T_w=\frac{g_0}{k}\frac{(R^2-r^2)}{4}$

PROBLEM:

A solid bar of Radius r=b is initially at a uniform temperature T_0. For times t>0, it is cooled by convection from the surface at r=b into ambient air at temperature T_{air} with a heat transfer coefficient h_{air} . Write the mathematical formulation of this heat conduction problem for the determination of one dimensional time dependent temperature distribution T(r,t) within the rod

$\frac{1}{r}\frac{\partial}{\partial r}\left(k_r(T)r\frac{\partial T}{\partial r}\right)+\frac{1}{r^2}\frac{\partial}{\partial \phi}\left(k_\phi(T)\frac{\partial T}{\partial \phi}\right)+\frac{\partial}{\partial z}\left(k_z(T)\frac{\partial T}{\partial z}\right)+g=\rho C_p(T)\frac{\partial T}{\partial t}$

$k_r(T)=k=const$ r and t dimensions only $C_p(T)=C_p=const$

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial T(r,t)}{\partial r}\right)=\frac{\rho C_p}{k}\frac{\partial T(r,t)}{\partial t}=\frac{1}{\alpha}\frac{\partial T(r,t)}{\partial t}$$

Boundary conditions:

$T(r,t)=T_0 \quad t=0 \ 0\le r\le b$

$-k\frac{\partial T(r,t)}{\partial r}=h(T(r,t)-T_\infty) \quad t>0 \ r=b$

$\frac{\partial T(r,t)}{\partial r}=0 \quad t>0 \ r=b$

PROBLEM :

A hollow cylinder (pipe), $a\le r\le b$, has its boundary surfaces at r=a and r=b maintained at uniform temperatures T_1 and T_2, respectively Termal conductivity varies with temperature in the form: $k(T)=k_0(1+\beta T)$ where k_0 and $\beta$are constants

$\frac{1}{r}\frac{\partial}{\partial r}\left(k(T)r\frac{\partial T}{\partial r}\right)=0 \quad \frac{1}{r}\frac{\partial}{\partial r}\left([k_0(1+\beta T)]r\frac{\partial T}{\partial r}\right)=0$

Boundary conditions:

$T=T_1$ at r=a

$T=T_2$ at r=b

$k(T)r\frac{\partial T}{\partial r}=C_1 \qquad [k_0(1+\beta T)]r\frac{\partial T}{\partial r}=C_1$

$Q=(2\pi rL)\left(-k(T)\frac{\partial T}{\partial r}\right)$

$Q=-2\pi LC_1$

$[k_0(1+\beta T)]r\frac{\partial T}{\partial r}=C_1$

$[k_0(1+\beta T)]dT=\frac{C_1}{r}dr$

$\left[k_0(T+\beta\frac{T^2}{2})\right]_{T_1}^{T_2}=C_1\ln(r)|_{r=a}^{b}$

$k_0(T_2-T_1)+\frac{k_0\beta}{2}(T_2^2-T_1^2))=C_1\ln(\frac{b}{a})$

$$Q = \frac{2\pi L k_0}{\ln\left(\frac{b}{a}\right)}\left[1 + \frac{\beta}{2}(T_2 + T_1)\right](T_1 - T_2)$$

PROBLEM: A long cylindrical rod of Radius b=5 x10^{-2} m , k=10 W/(mK) contains radioactive material which generates heat uniformly within the cylinder at a constant rate g_0=3x10^5 W/m^3. The rod is cooled by convection from its cylindrical surface into the ambient air at T_{inf}=50 C with a heat transfer coefficient h_{inf}=60 W/(m^2K). Determine the temperature at the centre and outer surface of this cylinder rod.

$\frac{1}{r}\frac{\partial}{\partial r}\left(k_r(T) r \frac{\partial T}{\partial r}\right) + \frac{1}{r^2}\frac{\partial}{\partial \phi}\left(k_\phi(T)\frac{\partial T}{\partial \phi}\right) + \frac{\partial}{\partial z}\left(k_z(T)\frac{\partial T}{\partial z}\right) + g = \rho C_p(T)\frac{\partial T}{\partial t}$
$k_r(T) = k = const$ r dimension only g(x)= $g_0 = const$

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial T(r)}{\partial r}\right) + \frac{g_0}{k} = 0$$

Boundary conditions:

$$-k\frac{\partial T}{\partial r} = h_{ambient}(T - T_{ambient}) \quad r = b$$

$$\frac{\partial T}{\partial r} = 0 \quad r = 0 \quad (maximum\ temperature)$$

$$d\left(r\frac{\partial T(r)}{\partial r}\right) = -\frac{g_0}{k} r dr$$

$$r\frac{\partial T(r)}{\partial r} = -\frac{g_0}{k}\frac{r^2}{2} + C_1$$

$$\frac{\partial T(r)}{\partial r} = -\frac{g_0}{k}\frac{r^2}{4} + \frac{C_1}{r}$$

$$T(r) = -\frac{g_0}{k}\frac{r^2}{4} + C_1 \ln(r) + C_2$$

$\frac{\partial T}{\partial r} = 0 \quad r = 0 \quad C_1 = 0$

$$-k\frac{\partial T}{\partial r} = h(T - T_\infty) \quad r = b$$

$$-k\left(-\frac{g_0}{k}\frac{b}{2}\right) = \frac{g_0 b}{2} = h\left(-\frac{g_0}{k}\frac{b^2}{4} + C_2 - T_\infty\right)$$

$$C_2 = \frac{g_0 b}{2h} + \frac{g_0}{k}\frac{b^2}{4} + T_\infty$$

$$C_2 = \frac{g_0 b}{2}\left(\frac{1}{h} + \frac{b}{2k}\right) + T_\infty$$

$$C_2 = \frac{3x10^5 * 5x10^{-2}}{2}\left(\frac{1}{60} + \frac{5x10^{-2}}{2 * 10}\right) + 50 = 193.75$$

$$T(r) = -7500r^2 + 193.75$$

r=0 T=193.75 ℃
r=b=5x10^{-2} m T=175 ℃

2.3 LUMPED SOLUTION OF FINS

Fins are extensive surfaces to increase heat transfer by increasing the surface area exposed to outside fluid. They are connected to base wall. Geometry of fins can change significantly depends on the applications. In the figure below some of the fin geometres are represented.

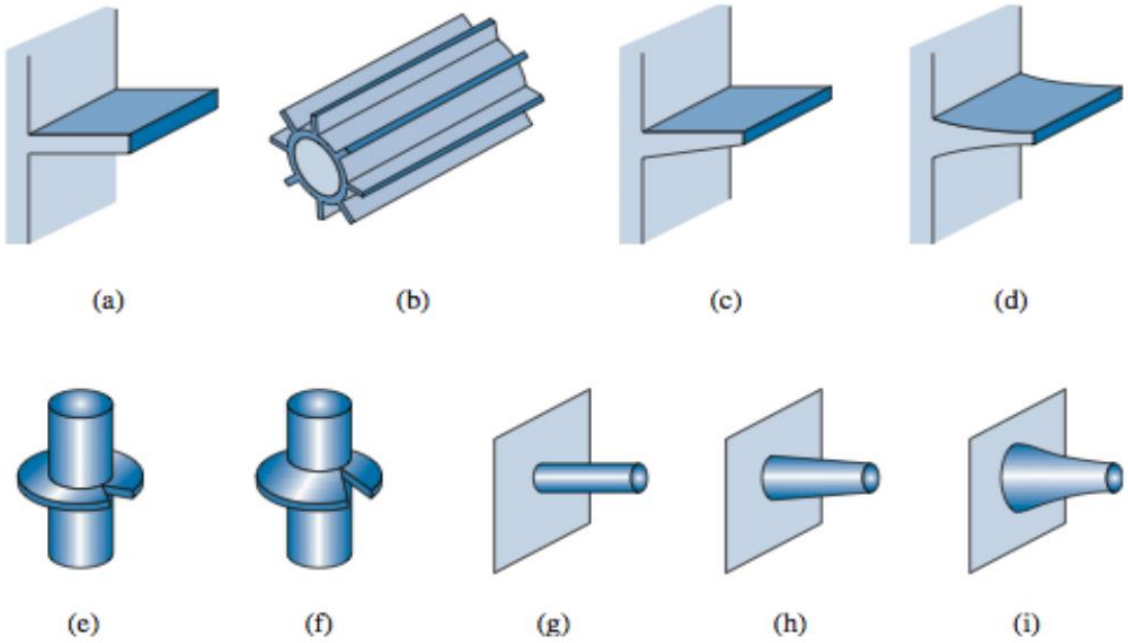
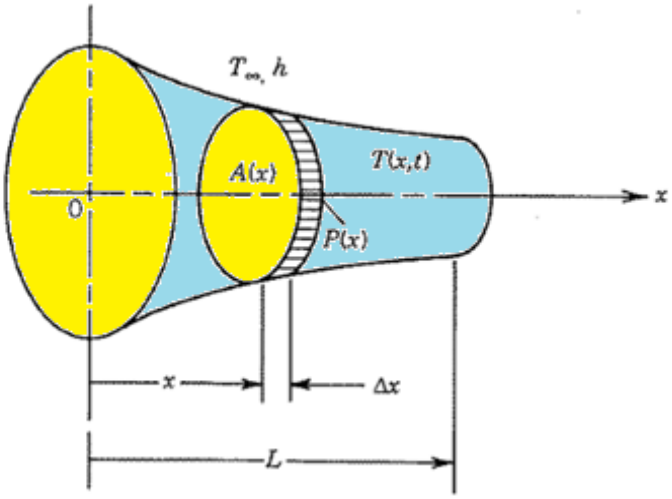


Figure 2.3-1 Fin types

(a) Longitudinal fin – Rectangular profile
(b) Longitudinal fin – Rectangular profile
(c) Longitudinal fin - Trapezoidal profile
(d) Longitudinal fin - Concave parabolic
(e) Radial fin – Rectangular profile
(f) Radial fin – Triangular profile
(g) Pin fin – Cylindrical
(h) Pin fin – Tapered profile
(i) Pin fin – Concave parabolic

Heat transfer equation for a fin can simply be written as



A(x) = cross sectional area of the disk

P(x) = perimeter of the disk
L = Length of the fin
$\Delta S(x) = P(x)\Delta x$ =lateral surface area of the disk
h = convective heat transfer coefficient
k= thermal conductivity of solid
T_∞= ambient temperature
T_0 = base temperature (assumed to be constant)
Energy balance (First law of thermodynamics)
Net rate of heat gain by conduction +rate of gain by convection from the lateral surfaces=rate of increase of internal energy

$$A(x)q_x - A(x+\Delta x)q_{x+\Delta x} + h(x)\Delta S(x)\big(T_\infty - T(x,t)\big) = \rho C_p \Delta x A(x)\frac{\partial T(x,t)}{\partial t}$$

$$A(x)q_x - A(x)q_x - \frac{\partial}{\partial x}(A(x)q_x)\Delta x + h(x)\Delta S(x)\big(T_\infty - T(x,t)\big) = \rho C_p \Delta x A(x)\frac{\partial T(x,t)}{\partial t}$$

$$-\frac{\partial}{\partial x}(A(x)q_x)\Delta x + h(x)\Delta S(x)\big(T_\infty - T(x,t)\big) = \rho C_p \Delta x A(x)\frac{\partial T(x,t)}{\partial t}$$

Fourier's law of conduction:

$$q_x = -k\frac{\partial T(x,t)}{\partial x}$$

$$\frac{\partial}{\partial x}(A(x)k\frac{\partial T(x,t)}{\partial x})\Delta x + h(x)\Delta S(x)\big(T_\infty - T(x,t)\big) = \rho C_p \Delta x A(x)\frac{\partial T(x,t)}{\partial t}$$

$$\frac{\partial}{\partial x}\left(A(x)k\frac{\partial T(x,t)}{\partial x}\right) - h(x)\frac{\Delta S(x)}{\Delta x}(T(x,t) - T_\infty) = \rho C_p A(x)\frac{\partial T(x,t)}{\partial t}$$

$$\theta(x,t) = \frac{T(x,t)-T_\infty}{T_0-T_\infty}$$

$$\frac{\partial}{\partial x}\left(A(x)k\frac{\partial \theta(x,t)}{\partial x}\right) - h(x)\frac{\Delta S(x)}{\Delta x}\theta(x,t) = \rho C_p A(x)\frac{\partial \theta(x,t)}{\partial t}$$

k≠ $f(x)$

$$\frac{1}{A(x)}\frac{\partial}{\partial x}\left(A(x)\frac{\partial \theta(x,t)}{\partial x}\right) - \frac{h(x)}{A(x)k}\frac{\Delta S(x)}{\Delta x}\theta(x,t) = \frac{\rho C_p}{k}\frac{\partial \theta(x,t)}{\partial t} = \frac{1}{\alpha}\frac{\partial \theta(x,t)}{\partial t}$$

If steady state (independent of time)

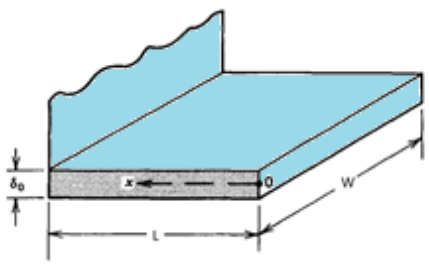
$$\frac{\partial}{\partial x}\left(A(x)\frac{\partial \theta(x)}{\partial x}\right) - \frac{h(x)}{k}\frac{\Delta S(x)}{\Delta x}\theta(x) = 0$$

$$\frac{\partial}{\partial x}\left(A(x)\frac{\partial \theta(x)}{\partial x}\right) - \frac{h(x)P(x)}{k}\theta(x) = 0$$

$$\theta(x,t) = \frac{T(x,t) - T_\infty}{T_0 - T_\infty}$$

Geometry 1: very thin plate (thickness δ_0) with constant cross section

(x starting from tips going backward)



Conduction flow area:

$$A(x) = A_0 = \delta_0 W = constant$$

Perimeter of fin (fin surface perimeter to convective heat transfer)

$$P(x) = P_0 = 2\delta_0 + 2W = 2W\left(1 + \frac{\delta_0}{W}\right) \cong 2W$$

Assume that convective heat transfer coefficient is constant $h(x) = h$

$m^2 = \frac{hP_0}{A_0 k} = \frac{2hW}{\delta_0 W k} = \frac{2h}{\delta_0 k}$ $\quad m = \sqrt{\frac{2h}{\delta_0 k}}$ M=mL

$$\frac{\partial^2 \theta(x)}{\partial x^2} - m^2\theta(x) = 0$$

Boundary conditions : x=L $\theta(x) = 1$

x=0 $\quad Q_x = kA(x)\frac{\partial\theta(x)}{\partial x}\Big|_{x=0} = 0$ therefore $\frac{\partial\theta(x)}{\partial x}\Big|_{x=0} = 0$

solution:

$$\theta(x) = C_1 \sinh(mx) + C_2 \cosh(mx)$$

$$\frac{\partial\theta(x)}{\partial x} = C_1 \,\mathrm{mcosh}(mx) + C_2 \,\mathrm{msinh}(mx)$$

If heat transfer at the edge is ignored(remeber it is very thin):

B.C. x=0 $\quad Q_x = kA(x)\frac{\partial\theta(x)}{\partial x}\Big|_{x=0}(T_0 - T_\infty) = 0$ therefore $\frac{\partial\theta(x)}{\partial x}\Big|_{x=0} = 0$

$$\frac{\partial\theta(x)}{\partial x} = C_1 \,\mathrm{mcosh}(m0) + C_2 \,\mathrm{msinh}(m0) = 0$$

$\cosh(m0) = 1$ $\sinhh(m0) = 0$ therefore $C_1 = 0$

$$\theta(x) = C_2 \cosh(mx)$$

B.C. x=L $\theta(x) = 1$

$\theta(L) = C_2 \cosh(mL) = C_2 \cosh(M)$=1

$$C_2 = \frac{1}{\cosh(mL)} = \frac{1}{\cosh(M)}$$

$$\theta(x) = \frac{\cosh(mx)}{\cosh(M)}$$

$$Q_{fin} = kA(x)\frac{\partial\theta(x)}{\partial x}\Big|_{x=L}(T_0 - T_\infty) = k(\delta_0 W)\frac{\mathrm{msinhh}(mx)}{\cosh(M)}\Big|_{x=L}$$

$$Q_{fin} = k(2WL)(T_0 - T_\infty)mtanh(mL) = k(\delta_0 W)(T_0 - T_\infty)mtanh(M)$$

Maximum possible heat transfer ocur if all the fin surface at the base temperature T_0 (or conductive heat transfer coefficient is infinitely big)

$$Q_{maximum\ fin} = hA_{fin\ surface}(T_0 - T_\infty) = h(2WL)\,(T_0 - T_\infty)$$

Fin efficiency: ratio of actual heat transfer to maximum possible heat transfer

$$\eta_{fin} = \frac{Q_{fin}}{Q_{maximum\ fin}} = \frac{k(\delta_0 W)(T_0 - T_\infty)mtanh(mL)}{h(2WL)\,(T_0 - T_\infty)} = \frac{k(\delta_0)\sqrt{\frac{2h}{\delta_0 k}}tanh(mL)}{h(2L)}$$

$$\eta_{fin} = \frac{tanh(mL)}{\sqrt{\frac{2h}{\delta_0 k}}L} = \frac{tanh(mL)}{mL} = \frac{tanh(M)}{M}$$

PROBLEM : For a longitudinal fin of rectangular cross section the following values are given: h_∞=30 W/m^{2}K k=50 W/mK fin length L=0.1 m, width W=0.4 m and thicknes=0.01 m. Calculate change of dimensionles temperature profile $\theta(X)$ for $0 \le X \le 1$ and η=$\frac{Q_{fin}}{Q_{maximum}}$ fin efficiency for $0 \le M \le 2$

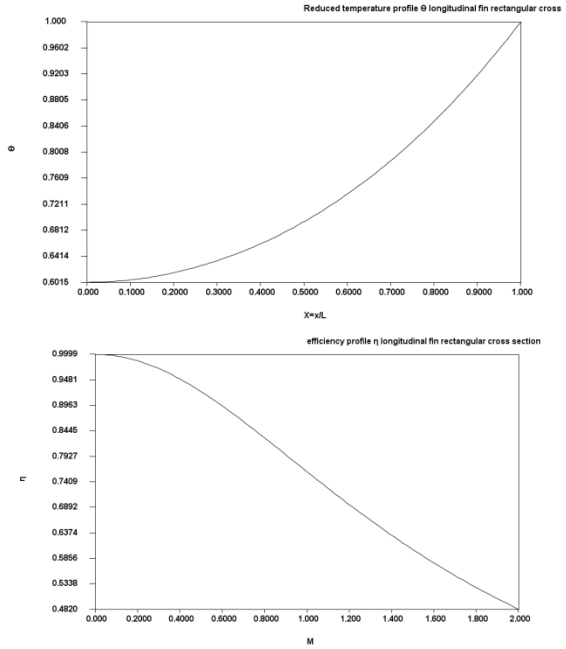
If base temperature of the fin T_b=120 ℃ and environmental temperature is $T_\infty = 20$ ℃, calculate heat transfer from the fin.

public class longitudinal_fin_rectangular_CS

```
{
public static void main(String arg[])
{   double h=30; //W/m^2K
double k=50; //W/mK
double L=0.1; //m
double d=0.01; //m
double W=0.4;
double M1=Math.sqrt(2.0*h/(k*d))*L;
if_x eta=(M)->Math.tanh(M)/M;
if_x teta=X->Math.cosh(M1*X)/Math.cosh(M1);
double teta1[]=new double[101];
Plot p1=new Plot(teta,0.0,1.0,100);
p1.setPlabel("Reduced temperature profile "+'\u03F4'+" longitudinal fin rectangular cross section");
p1.setXlabel("X=x/L");
p1.setYlabel(""+'\u03F4');
p1.plot();
Plot p2=new Plot(eta,0.0,2.0,100);
p2.setPlabel("efficiency profile "+'\u03B7'+" longitudinal fin rectangular cross section");
p2.setXlabel("M");
p2.setYlabel(""+'\u03B7');
p2.plot();
double A=2*L*W;
double T1=120;
double T2=20;
double Qmax=h*A*(T1-T2);
double eta1=eta.func(M1);
double Q=eta1*Qmax;
System.out.println("M="+M1+"eta = "+eta1+"Qmax="+Qmax+"Q="+Q);

}
}
```

```
---------- Capture Output ----------
      "E:\co\java\bin\java.exe" longitudinal_fin_rectangular_CS
M=1.0954451150103324eta = 0.7292532634501073Qmax=240.00000000000003Q=175.02078322802578
```



Hand calculation of the problem:

h_∞=30 W/m^{2}K k=50 W/mK fin length L=0.1 m, width W=0.4 m and thicknes=0.01

$m = \sqrt{\frac{2h}{\delta_0 k}} =$ M=mL

$$m = \sqrt{\frac{2h}{\delta_0 k}} = \sqrt{\frac{2 * 30}{0.01 * 50}} = 10.95445115$$

$$M = m * L = 10.95445115 * 0.1 = 1.095445115$$

$$\eta_{fin} = \frac{tanh(mL)}{mL} = \frac{tanh(M)}{M} = \frac{tanh(1.095445115)}{1.095445115} = 0.729253263$$

$$A = 2 * L * W = 2 * 0.1 * 0.4 = 0.08\, m^2$$

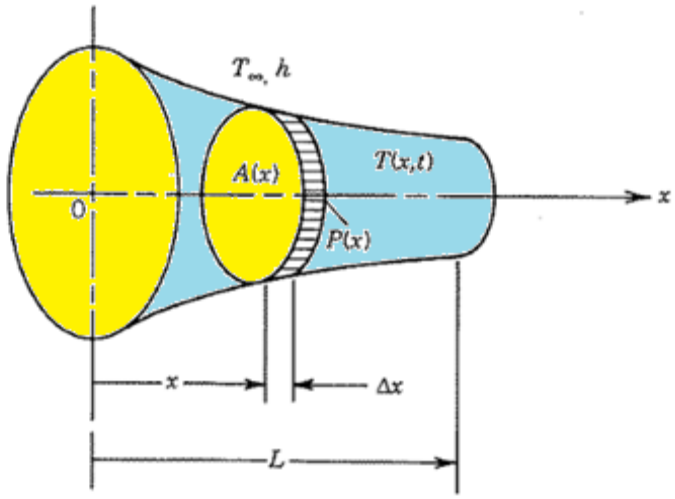
$$Q_{max} = hA(T_0 - T_\infty) = 30 * 0.08 * 100 = 240\ W$$

$$Q = \eta_{fin} * Q_{max} = 0.729253263 * 240\ W = 175.0207831\ W$$

Simplification of fin equation by assuming some general polynomial profiles

Instead of solving full differential equation by assuming some of the dimensions are not changing a simplified differential equations can be derived (again by assuming energy balance is hold for a given control volume). The following examples is assuming one dimensional steady state solution of a fin structure. Consider a heat transfer problem given as in the figure shown above. An extended surface has one dimensional conductive flow, and heat is removed by convection in each step.

Following normalisations and assumptions can be applied to this differential equation:



A(x) = cross sectional area of the disk

P(x) = perimeter of the disk

L = Length of the fin

$\Delta S(x) = P(x)\Delta x$ =lateral surface area of the disk

h = convective heat transfer coefficient

k= thermal conductivity of solid

T_∞= ambient temperature

T_0 = base temperature (assumed to be constant)

$$\frac{\partial}{\partial x}\left(A(x)\frac{\partial \theta(x)}{\partial x}\right) - \frac{h(x)P(x)}{k}\theta(x) = 0$$

Additional definitions are given as:

A_0= base cross-sectional area

P_0= base perimeter

L = Length of the fin

h_{av}=an average thermal convectivity

$X = \frac{x}{L}$ normalised length $0 \le X \le 1$

$\boldsymbol{K(X) = \frac{A(x)}{A_0}}$ **normalised cross-sectional area**

$\boldsymbol{m^2 = \frac{h_{av}P_0}{kA_0}}$

$W(X) = \frac{h(x)}{P_0 h_{av}}\frac{\Delta S(x)}{\Delta x}$ Note that $\frac{\Delta S(x)}{\Delta x} = P(x)$ also

$W(X) = \frac{h(x)}{h_{av}} \frac{P(x)}{P_0}$

$M = mL \qquad M^2 = m^2L^2 = \frac{h_{av}P_0}{kA_0}L^2$

Equation becomes

$\frac{\partial}{\partial X}\left(K(X)\frac{\partial\theta(X)}{\partial X}\right) - M^2W(X)\theta(X) = 0 \qquad 0 \le X \le 1$

Boundary conditions:

$\theta(X) = 1$ X=1 $(T(x) = T_0$ x=0)

$\frac{d\theta(X)}{dX} = 0$ at X=0 $(Q(x) = -k * A(x)\frac{dT(x)}{dx} = 0$ x=L heat transfer at the edge is negligable)

Furthermore in order to simulate different geometries, some general polynomials are assigned with 3 different contants m,c and n

$$\boldsymbol{K(X) = X^{1-2m}}$$

(change in the fin geometry)

$$\boldsymbol{W(X) = c^2n^2X^{2c-2}K(X)}$$

Then the fin equation becomes:

$$\frac{\partial^2\theta(X)}{\partial X^2} + \frac{1-2m}{X}\frac{\partial\theta(X)}{\partial X} - M^2c^2n^2X^{2c-2}\theta(X) = 0$$

Solution of this differential equation for the boundary conditions given above:

$\theta(X) = X^m \frac{I_{-\left(\frac{m}{c}\right)}(nMX^c)}{I_{-\left(\frac{m}{c}\right)}(nM)} \quad \frac{m}{c} < 1$

$\theta(X) = X^m \frac{I_{\left(\frac{m}{c}\right)}(nMX^c)}{I_{\left(\frac{m}{c}\right)}(nM)} \quad \frac{m}{c} > 1$

Values of c,m and n for different geometries for cn=1 is given in the table below:

Name	K(X)	W(X)	m	c	n	ν=m/c
Longitudinal fin of rectangular profile	1	1	1/2	1	1	1/2
Longitudinal fin of triangular profile	X	1	0	1/2	2	0
Longitudinal fin of concave parabolic profile	X^2	1	-1/2	0	∞	-∞
Longitudinal fin of convex parabolic profile	$X^{1/2}$	1	1/4	3/4	4/3	1/3
Radial fin of rectangular profile	X	X	0	1	1	0
Radial fin of hyperbolic profile	1	X	1/2	3/2	2/3	1/3
Conic spine	X^2	X	-1/2	1/2	2	-1
Spine fin of concave parabolic profile	X^4	X^2	-3/2	0	∞	-∞
Spine fin of convex parabolic profile	X	$X^{1/2}$	0	3/4	4/3	0

$I_\nu(z)$ in the equation is called exponential Bessel function. It is given with the serial solution for ν=$\frac{m}{c}$

$I_\nu(z) = \sum_{k=0}^{\infty} \frac{1}{k!\Gamma(k+\nu+1)} \left(\frac{z}{2}\right)^{2k+\nu}$ where Γ is gamma function

Where gamma function defined as $\Gamma(z) = \int_{z=0}^{\infty} t^{z-1}e^{-t}dt = \frac{e^{-\gamma t}}{t} \prod_{n=1}^{\infty}\left(1+\frac{t}{n}\right)^{-1} e^{t/n}$

Where γ=0.57721566490153286... is the Euler-Mascheroni constant. A program is given to calculate I below:

```
public class bessel_I
{   //C Euler-Mascheroni constant
   public static double C=0.57721566490153286060651209008240243l;
   public static double I(double v,double x)
{ //I Bessel fonksiyonu
double JJ=0;double a=1.0;int fact=1;double b;
int vtest=(int)(v*10)/10;
if(v==0 && v==(double)vtest) JJ=bessi0(x);
else if(v==1 && v==(double)vtest) JJ=bessi1(x);
else if (v==(double)vtest) JJ=bessi((int)v,x);
else
```

```
{
double x1,x2,x3,x4;
long x5=1;
double lna=0.0;
x1=v*Math.log(0.5*x);
double lna1=Math.log(0.25*x*x);
for(int k=0;k<100;k++)
{
    x2=lna;//Math.log(a);
    x3=lngamma(k+1)[0];
    x4=lngamma(v+k+1)[0];
            b=x1+x2-x3-x4;
            b=Math.exp(b);
            lna+=lna1;
            JJ+=b;
}
if(v==0 && x==0) JJ=1;
}
return JJ;
}
//natural logarithm of gamma function
        public static double [] lngamma(double x)
   {//logaithm of gamma function
   //xx[0] value of logarithm of gamma function
   //xx[1] sign of gamma functon
        double xx[]=new double[2];
   double result;
   double a;
   double b;
   double c;
   double p;
   double q;
   double u;
   double w;
   double z;
   int i;
   double logpi;
   double ls2pi;
   double tmp;
   double sgngam = 1;
   logpi = 1.14472988584940017414;
   ls2pi = 0.91893853320467274178;
   if( x<-34.0 )
   {
      q = -x;
      double yy[]=lngamma(q);
      w = yy[0];
      tmp=yy[1];
      p = Math.floor(q);
      i = (int)Math.round(p);
      if( i%2==0 )
      {
         sgngam = -1;
      }
      else
      {
         sgngam = 1;
      }
      z = q-p;
      if( z>0.5 )
      {
         p = p+1;
         z = p-q;
      }
      z = q*Math.sin(Math.PI*z);
      result = logpi-Math.log(z)-w;
      xx[0]=result;
      xx[1]=sgngam;
      return xx;
   }
   if( x<13 )
   {
      z = 1;
      p = 0;
      u = x;
      while(u>=3)
      {
```

```
            p = p-1;
            u = x+p;
            z = z*u;
        }
        while(u<2)
        {
            z = z/u;
            p = p+1;
            u = x+p;
        }
        if( z<0 )
        {
            sgngam = -1;
            z = -z;
        }
        else
        {
            sgngam = 1;
        }
        if( u==2 )
        {
            result = Math.log(z);
            //return result;
            xx[0]=result;
            xx[1]=sgngam;
            return xx;
        }
        p = p-2;
        x = x+p;
        b = -1378.25152569120859100;
        b = -38801.6315134637840924+x*b;
        b = -331612.992738871184744+x*b;
        b = -1162370.97492762307383+x*b;
        b = -1721737.00820839662146+x*b;
        b = -853555.664245765465627+x*b;
        c = 1;
        c = -351.815701436523470549+x*c;
        c = -17064.2106651881159223+x*c;
        c = -220528.590553854454839+x*c;
        c = -1139334.44367982507207+x*c;
        c = -2532523.07177582951285+x*c;
        c = -2018891.41433532773231+x*c;
        p = x*b/c;
        result = Math.log(z)+p;
        //return result;
        xx[0]=result;
        xx[1]=sgngam;
        return xx;
    }
    q = (x-0.5)*Math.log(x)-x+ls2pi;
    if( x>100000000 )
    {
        result = q;
        //return result;
        xx[0]=result;
        xx[1]=sgngam;
        return xx;
    }
    p = 1/(x*x);
    if( x>=1000.0 )
    {
        q = q+((7.936507936507936507936507936507936*0.0001*p-2.777777777777777777778*0.001)*p+0.0833333333333333333333)/x;
    }
    else
    {
        a = 8.11614167470508450300*0.0001;
        a = -5.950619042843014383324*0.0001+p*a;
        a = 7.93650340457716943945*0.0001+p*a;
        a = -2.77777777730099687205*0.001+p*a;
        a = 8.33333333333331927722*0.01+p*a;
        q = q+a/x;
    }
    xx[0]= q;
    xx[1]=sgngam;
    return xx;
    }
//I Bessel function of integer order 0
```

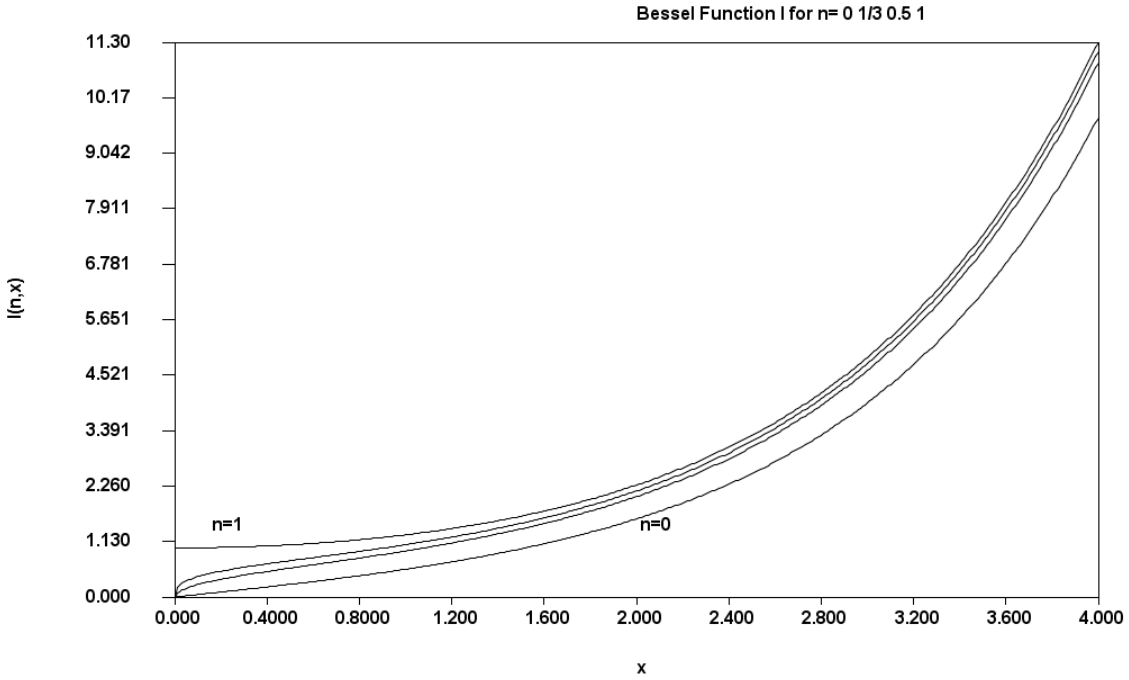
```
public static double bessi0(double x)
{
	double ax,ans;
	double y;

	if ((ax=Math.abs(x)) < 3.75) {
		y=x/3.75;
		y*=y;
		ans=1.0+y*(3.5156229+y*(3.0899424+y*(1.2067492
			+y*(0.2659732+y*(0.360768e-1+y*0.45813e-2)))));
	} else {
		y=3.75/ax;
		ans=(Math.exp(ax)/Math.sqrt(ax))*(0.39894228+y*(0.1328592e-1
			+y*(0.225319e-2+y*(-0.157565e-2+y*(0.916281e-2
			+y*(-0.2057706e-1+y*(0.2635537e-1+y*(-0.1647633e-1
			+y*0.392377e-2)))))));
	}
	return ans;
}
//I Bessel function of integer order 1
public static double bessi1(double x)
{
	double ax,ans;
	double y;

	if ((ax=Math.abs(x)) < 3.75) {
		y=x/3.75;
		y*=y;
		ans=ax*(0.5+y*(0.87890594+y*(0.51498869+y*(0.15084934
			+y*(0.2658733e-1+y*(0.301532e-2+y*0.32411e-3))))));
	} else {
		y=3.75/ax;
		ans=0.2282967e-1+y*(-0.2895312e-1+y*(0.1787654e-1
			-y*0.420059e-2));
		ans=0.39894228+y*(-0.3988024e-1+y*(-0.362018e-2
			+y*(0.163801e-2+y*(-0.1031555e-1+y*ans))));
		ans *= (Math.exp(ax)/Math.sqrt(ax));
	}
	return x < 0.0 ? -ans : ans;
}
//I Bessel function of integer order n
public static double bessi(int n, double x)
{
   double ACC=40.0;
	double BIGNO=1.0e10;
	double BIGNI=1.0e-10;
	int j;
	double bi,bim,bip,tox,ans;

	if (n < 2) System.out.println("Index n less than 2 in bessi");
	if (x == 0.0)
		return 0.0;
	else {
		tox=2.0/Math.abs(x);
		bip=ans=0.0;
		bi=1.0;
		for (j=2*(n+(int) Math.sqrt(ACC*n));j>0;j--) {
			bim=bip+j*tox*bi;
			bip=bi;
			bi=bim;
			if (Math.abs(bi) > BIGNO) {
				ans *= BIGNI;
				bi *= BIGNI;
				bip *= BIGNI;
			}
			if (j == n) ans=bip;
		}
		ans *= bessi0(x)/bi;
		if(x<0) return -ans;
		else return ans;
	}
}
public static void main(String arg[])
{   double x[]=new double[401];
   double y[][]=new double[4][401];
	for(int i=0;i<=400;i++)
	{x[i]=0.01*i;
```

```
            y[0][i]=I(0.0,x[i]);
            y[1][i]=I((1.0/3.0),x[i]);
            y[2][i]=I(0.5,x[i]);
            y[3][i]=I(1.0,x[i]);
           }
           Plot pp=new Plot(x,y[0]);
           pp.addData(x,y[1]);
           pp.addData(x,y[2]);
           pp.addData(x,y[3]);
           pp.plot();
}
}
```



Modified Bessel Functions of first and second kind

Data is created by using Mathd.java (it existed in SCO1)

```
public class Bessel_Table
{public static void main(String arg[])
 {
  double x,K0,K1,I0,I1;
  for(int i=0;i<101;i++)
  {x=0.1*i;
   I0=Mathd.I(0,x)*Math.exp(-x);
   I1=Mathd.I(1,x)*Math.exp(-x);
   K0=Mathd.K(0,x)*Math.exp(x);
   K1=Mathd.K(1,x)*Math.exp(x);
   System.out.println(x+" "+I0+" "+I1+" "+K0+" "+K1);
  }}}
```

x	$e^{-x}I_0(x)$	$e^{-x}I_1(x)$	$e^{x}K_0(x)$	$e^{x}K_1(x)$
0	1	0	∞	∞
0.1000000000000	0.9071009244482	0.0452984467891	2.6823261024353	10.8901826860671
0.2000000000000	0.8269385469884	0.0822831233904	2.1407573119304	5.8333860435406
0.3000000000000	0.7575806163207	0.1123775602390	1.8526272735686	4.1251577717766
0.4000000000000	0.6974021575741	0.1367632234988	1.6626820448337	3.2586738912011
0.5000000000000	0.6450352542264	0.1564208019100	1.5241093260797	2.7310097214144
0.6000000000000	0.5993271850232	0.1721644177836	1.4167375507951	2.3739200504625
0.7000000000000	0.5593055082344	0.1846699805981	1.3301235813854	2.1150113240440
0.8000000000000	0.5241489249832	0.1944986909392	1.2582030499339	1.9179303074833
0.9000000000000	0.4931629517575	0.2021165283827	1.1971633186895	1.7623882254095
1.0000000000000	0.4657595966611	0.2079104129914	1.1444630331685	1.6361534902582

1.1000000000000	0.4414403703010	0.2122016111145	1.0983302532758	1.5314037579606
1.2000000000000	0.4197820752806	0.2152568578028	1.0574845189621	1.4428975581328
1.3000000000000	0.4004249120754	0.2172975868126	1.0209731627618	1.3669872939768
1.4000000000000	0.3830625169034	0.2185075918921	0.9880700111091	1.3010537533648
1.5000000000000	0.3674336118153	0.2190393873980	0.9582100832152	1.2431658877838
1.6000000000000	0.3533150008735	0.2190194902505	0.9309460278780	1.1918675752615
1.7000000000000	0.3405156908463	0.2185528071420	0.9059182013118	1.1460392450530
1.8000000000000	0.3288719518620	0.2177262793860	0.8828335934588	1.1048053556836
1.9000000000000	0.3182431642312	0.2166119116992	0.8614506675320	1.0674709013680
2.0000000000000	0.3085083232249	0.2152692896043	0.8415682686449	1.0334768336463
2.1000000000000	0.2995630948688	0.2137476722503	0.8230170924649	1.0023681332472
2.2000000000000	0.2913173335077	0.2120877326314	0.8056539353977	0.9737702339885
2.3000000000000	0.2836929866294	0.2103230049131	0.7893561120531	0.9473722578087
2.4000000000000	0.2766223246944	0.2084810884009	0.7740181420420	0.9229136713920
2.5000000000000	0.2700464439386	0.2065846492634	0.7595487023967	0.9001744154108
2.6000000000000	0.2639139986236	0.2046522541312	0.7458682582271	0.8789672669202
2.7000000000000	0.2581801263093	0.2026990639048	0.7329071651074	0.8591318742419
2.8000000000000	0.2528055356399	0.2007374112966	0.7206041344830	0.8405300522413
2.9000000000000	0.2477557310754	0.1987772816520	0.7089049819469	0.8230420375707
3.0000000000000	0.2430003531247	0.1968267132844	0.6977615983054	0.8065634824532
3.1000000000000	0.2385126160824	0.1948921308173	0.6871310977794	0.7910030220530
3.2000000000000	0.2342688281507	0.1929786227490	0.6769751082409	0.7762802913037
3.3000000000000	0.2302479812404	0.1910901725629	0.6672591762199	0.7623242968745
3.4000000000000	0.2264313997578	0.1892298511365	0.6579522653110	0.7490720719541
3.5000000000000	0.2228024393801	0.1873999768991	0.6490263310890	0.7364675579211
3.6000000000000	0.2193462282331	0.1856022491005	0.6404559590813	0.7244606692892
3.7000000000000	0.2160494440777	0.1838378586560	0.6322180550095	0.7130065076679
3.8000000000000	0.2129001299773	0.1821075833710	0.6242915785906	0.7020646976223
3.9000000000000	0.2098875283762	0.1804118566460	0.6166573138262	0.6915988228336
4.0000000000000	0.2070019252180	0.1787508395700	0.6092976700000	0.6815759452344
4.1000000000000	0.2042345330188	0.1771244752911	0.6021965086390	0.6719661931467
4.2000000000000	0.2015773893518	0.1755325255304	0.5953389925201	0.6627424070815
4.3000000000000	0.1990232608252	0.1739746095179	0.5887114534727	0.6538798339478
4.4000000000000	0.1965655607441	0.1724502351448	0.5823012762672	0.6453558620842
4.5000000000000	0.1941982778769	0.1709588242056	0.5760967963239	0.6371497908587
4.6000000000000	0.1919159144466	0.1694997330083	0.5700872093342	0.6292426296587
4.7000000000000	0.1897134319114	0.1680722692664	0.5642624911879	0.6216169219642
4.8000000000000	0.1875862033948	0.1666757059572	0.5586133268420	0.6142565909060
4.9000000000000	0.1855299718356	0.1653092926808	0.5531310469731	0.6071468032904
5.0000000000000	0.1835408130863	0.1639722649451	0.5478075714251	0.6002738495505
5.1000000000000	0.1816151033044	0.1626638517245	0.5426353586059	0.5936250374766
5.2000000000000	0.1797494900810	0.1613832815814	0.5376073601051	0.5871885979026
5.3000000000000	0.1779408668296	0.1601297875883	0.5327169799092	0.5809536007993
5.4000000000000	0.1761863500230	0.1589026112525	0.5279580376703	0.5749098804508
5.5000000000000	0.1744832589237	0.1577010056097	0.5233247355615	0.5690479685793
5.6000000000000	0.1728290975013	0.1565242376305	0.5188116283123	0.5633590344444
5.7000000000000	0.1712215382689	0.1553715900548	0.5144135960669	0.5578348310768
5.8000000000000	0.1696584078093	0.1542423627570	0.5101258197584	0.5524676469216
5.9000000000000	0.1681376737887	0.1531358737219	0.5059437587258	0.5472502622619
6.0000000000000	0.1666574332843	0.1520514597043	0.5018631303368	0.5421759098768
6.1000000000000	0.1652159022726	0.1509884766275	0.4978798914073	0.5372382394577
6.2000000000000	0.1638114061469	0.1499462997719	0.4939902212316	0.5324312853684
6.3000000000000	0.1624423711466	0.1489243237925	0.4901905060633	0.5277494373848
6.4000000000000	0.1611073165976	0.1479219626000	0.4864773248995	0.5231874140961
6.5000000000000	0.1598048478756	0.1469386491331	0.4828474364431	0.5187402386866
6.6000000000000	0.1585336500125	0.1459738350456	0.4792977671273	0.5144032168521
6.7000000000000	0.1572924818794	0.1450269903272	0.4758254001040	0.5101719166322
6.8000000000000	0.1560801708839	0.1440976028745	0.4724275651026	0.5060421499665
6.9000000000000	0.1548956081306	0.1431851780243	0.4691016290815	0.5020099558037
7.0000000000000	0.1537377439951	0.1422892380615	0.4658450875983	0.4980715846122
7.1000000000000	0.1526055840729	0.1414093217086	0.4626555568358	0.4942234841565
7.2000000000000	0.1514981854641	0.1405449836057	0.4595307662239	0.4904622864209

7.3000000000000	0.1504146533617	0.1396957937854	0.4564685516065	0.4867847955719
7.4000000000000	0.1493541379150	0.1388613371489	0.4534668489069	0.4831879768639
7.5000000000000	0.1483158313416	0.1380412129449	0.4505236882475	0.4796689464023
7.6000000000000	0.1472989652652	0.1372350342568	0.4476371884875	0.4762249616880
7.7000000000000	0.1463028082578	0.1364424274979	0.4448055521430	0.4728534128719
7.8000000000000	0.1453266635686	0.1356630319193	0.4420270606585	0.4695518146602
7.9000000000000	0.1443698670226	0.1348964991287	0.4393000700010	0.4663177988117
8.0000000000000	0.1434317850746	0.1341424926241	0.4366230065519	0.4631491071788
8.1000000000000	0.1425118130032	0.1334006873407	0.4339943632719	0.4600435852445
8.2000000000000	0.1416093732361	0.1326707692129	0.4314126961192	0.4569991761165
8.3000000000000	0.1407239137921	0.1319524347511	0.4288766206998	0.4540139149388
8.4000000000000	0.1398549068328	0.1312453906334	0.4263848091337	0.4510859236878
8.5000000000000	0.1390018473135	0.1305493533123	0.4239359871202	0.4482134063231
8.6000000000000	0.1381642517261	0.1298640486366	0.4215289311870	0.4453946442625
8.7000000000000	0.1373416569260	0.1291892114864	0.4191624661108	0.4426279921590
8.8000000000000	0.1365336190365	0.1285245854244	0.4168354624955	0.4399118739537
8.9000000000000	0.1357397124256	0.1278699223596	0.4145468344983	0.4372447791848
9.0000000000000	0.1349595287480	0.1272249822250	0.4122955376916	0.4346252595340
9.1000000000000	0.1341926760484	0.1265895326684	0.4100805670524	0.4320519255905
9.2000000000000	0.1334387779221	0.1259633487562	0.4079009550691	0.4295234438192
9.3000000000000	0.1326974727265	0.1253462126887	0.4057557699595	0.4270385337161
9.4000000000000	0.1319684128426	0.1247379135283	0.4036441139904	0.4245959651378
9.5000000000000	0.1312512639803	0.1241382469379	0.4015651218928	0.4221945557936
9.6000000000000	0.1305457045257	0.1235470149312	0.3995179593673	0.4198331688882
9.7000000000000	0.1298514249281	0.1229640256330	0.3975018216717	0.4175107109032
9.8000000000000	0.1291681271218	0.1223890930494	0.3955159322877	0.4152261295105
9.9000000000000	0.1284955239821	0.1218220368487	0.3935595416590	0.4129784116057
10.0000000000000	0.1278333388132	0.1212626821498	0.3916319259996	0.4107665814554

The ratio of fin heat transfer to maximum possible heat transfer is called **fin efficiency**

$$\eta=\frac{Q_{fin}}{Q_{maximum}}$$

or

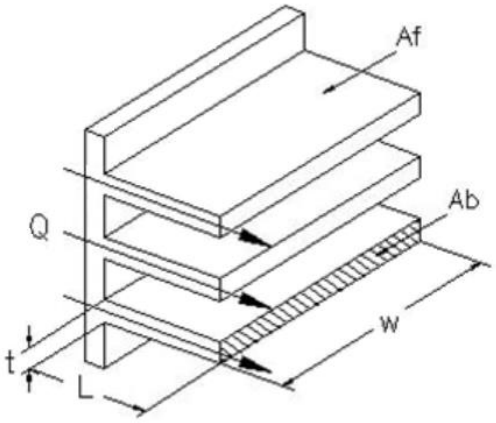
$$Q_{fin}=\eta Q_{maximum}$$

Maximum fin heat transfer is the heat transfer when fin thermal conductivity is infinitly large, or all fin surface temperatures are equal to fin base temperature

$$Q_{maximum}=hA_{fin}(T_0-T_\infty)=-kA_0\left.\frac{dT}{dx}\right|_{x=0}$$

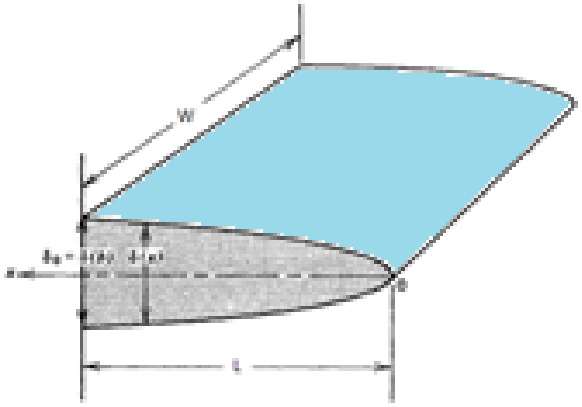
One more point before going to actual fin geometries. Fins an a surface only occupies some certain area, the remaining of the surface. For the base sırfaces fin efficiency is equal to 1 (fin surface is at base temperature) . Therefore total heat transfer is equal to

$$Q=\left[(A-A_f)1+A_f\eta\right]\Delta T=\left[A-(1-A_f)\eta\right]\Delta T$$



Let us now consider the heat transfer from the fin, Let us consider an **arbitrary Longitudinal fin** of length L and width W. For this fin $A(x) = \delta(x)W \quad A_0 = \delta_0 W \;\; P(x) = 2W\left[1 + \frac{\delta(x)}{W}\right] \cong 2W \quad P_0 = 2W \; K(x) = \frac{\delta(x)}{\delta_0} \;\; W(x) = \frac{h(x)}{h_0}$

$M = \left(\frac{2h_{av}}{k\delta_0}\right)^{1/2} L$ and dimensionless coordinates $X = \frac{x}{L}$ normalised length $0 \leq X \leq 1$ with X=0 at fin tip and X=1 at fin base. And if we further assume that heat transfer coefficient is constant $h(x) = h_0 = h_{av} = const.$



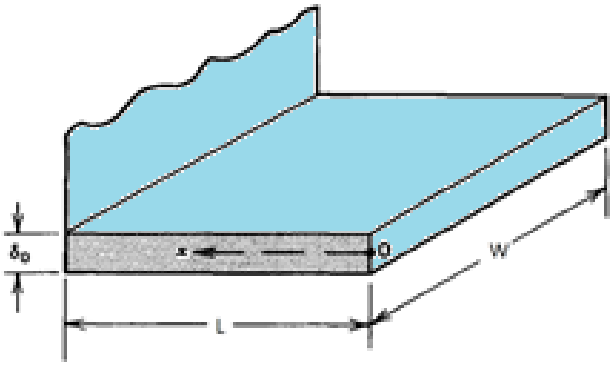
Then $W(x) = 1$ The heat transfer from the fin (at the base X=1):

$$Q_{fin} = kA(X)\frac{d\theta(X)}{dX}\Big|_{X=1} = k\delta_0 W(T_0 - T_\infty)\frac{1}{L}\frac{d\theta(X)}{dX}\Big|_{X=1}$$

The ratio of fin heat transfer to maximum possible heat transfer is called fin efficiency.

$\eta = \frac{Q_{fin}}{Q_{maximum}} = \frac{1}{M^2}\frac{d\theta(X)}{dX}\Big|_{X=1}$ or $Q_{fin} = \eta Q_{maximum}$

Now we can look at fin efficiencies as a function of actual geometries. As a first geometry let us look at **Longitudinal fin of rectangular profile**



m=1/2 c=1 n=1 ν=m/c=1/2

$\theta(X) = X^{1/2}\frac{I_{-(1/2)}(MX)}{I_{-(1/2)}(M)}$ but also $I_{-\left(\frac{1}{2}\right)}(X) = \sqrt{\frac{2}{\pi X}}\cosh(X)$ therefore: $\theta(X) = \frac{cosh(MX)}{cosh(M)} \quad \eta = \frac{tanh(M)}{M}$

where $M = \left(\frac{2h_{av}}{k\delta_0}\right)^{1/2} L$

If all fin temperature is at the base temperature of T_0 maximum heat transfer will occur

$Q_{maximum} = h_{av}2WL(T_0 - T_\infty)$

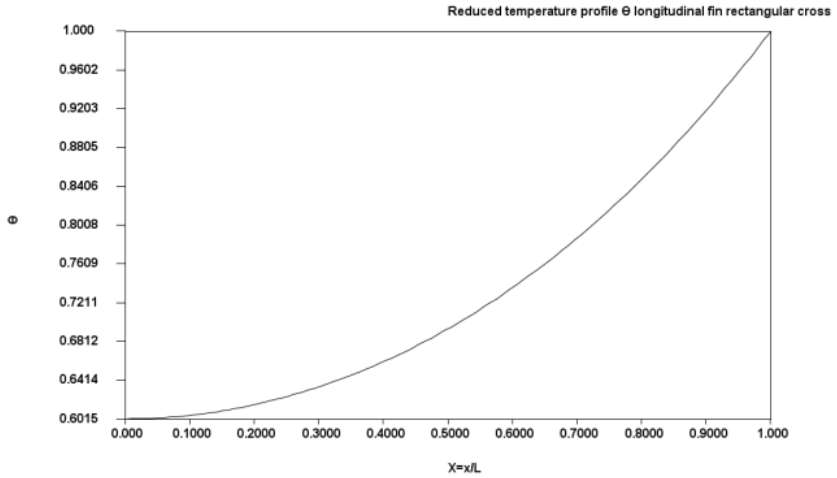
$\eta = \frac{Q_{fin}}{Q_{maximum}} = \frac{1}{M^2}\frac{d\theta(X)}{dX}\Big|_{X=1}$ or $Q_{fin} = \eta Q_{maximum}$

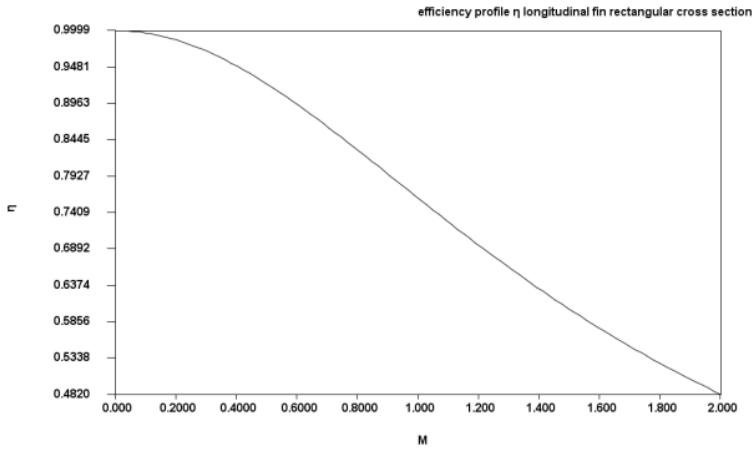
PROBLEM : For a longitudinal fin of rectangular cross section the following values are given: h_∞=30 W/m^{2}K k=50 W/mK fin length L=0.1 m, width W=0.4 m and thicknes=0.01 m. Calculate change of dimensionles temperature profile $\theta(X)$ for $0 \leq X \leq 1$ and η=$\frac{Q_{fin}}{Q_{maximum}}$ fin efficiency for $0 \leq M \leq 2$ If base temperature of the fin T_b=120 ℃ and environmental temperature is $T_\infty = 20$ ℃, calculate heat transfer from the fin.

```
 public class longitudinal_fin_rectangular_CS
{
public static void main(String arg[])
{   double h=30; //W/m^2K
    double k=50; //W/mK
    double L=0.1; //m
    double d=0.01; //m
    double W=0.4;
    double M1=Math.sqrt(2.0*h/(k*d))*L;
                if_x eta=(M)->Math.tanh(M)/M;
                if_x teta=X->Math.cosh(M1*X)/Math.cosh(M1);
    double teta1[]=new double[101];
                Plot p1=new Plot(teta,0.0,1.0,100);
                p1.setPlabel("Reduced temperature profile "+'\u03F4'+" longitudinal fin rectangular cross section");
                p1.setXlabel("X=x/L");
                p1.setYlabel(""+'\u03F4');
                p1.plot();
                Plot p2=new Plot(eta,0.0,2.0,100);
                p2.setPlabel("efficiency profile "+'\u03B7'+" longitudinal fin rectangular cross section");
    p2.setXlabel("M");
    p2.setYlabel(""+'\u03B7');
                p2.plot();
                double A=2*L*W;
                double T1=120;
                double T2=20;
                double Qmax=h*A*(T1-T2);
                double eta1=eta.func(M1);
                double Q=eta1*Qmax;
                System.out.println("M="+M1+"eta = "+eta1+"Qmax="+Qmax+"Q="+Q);

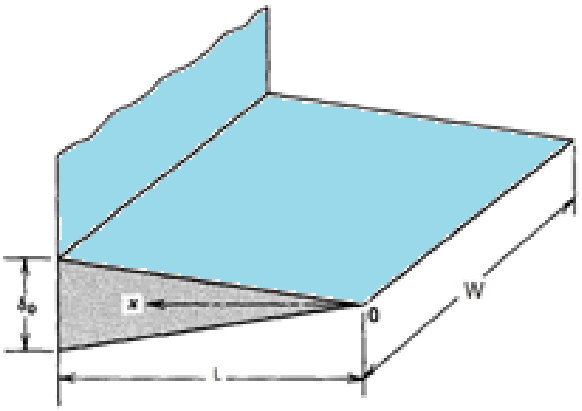
}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" longitudinal_fin_rectangular_CS
M=1.0954451150103324eta = 0.7292532634501073Qmax=240.00000000000003Q=175.02078322802578
```





Longitudinal fin of triangualar profile



$\delta(x) = \delta_0 \frac{x}{L} = \delta_0 X \qquad K(x) = \frac{\delta(x)}{\delta_0} = \frac{\delta_0 X}{\delta_0} = X \;\; W(x) = \frac{h(x)}{h_0} = 1$

m=0 c=1/2 n=2 ν=m/c=0

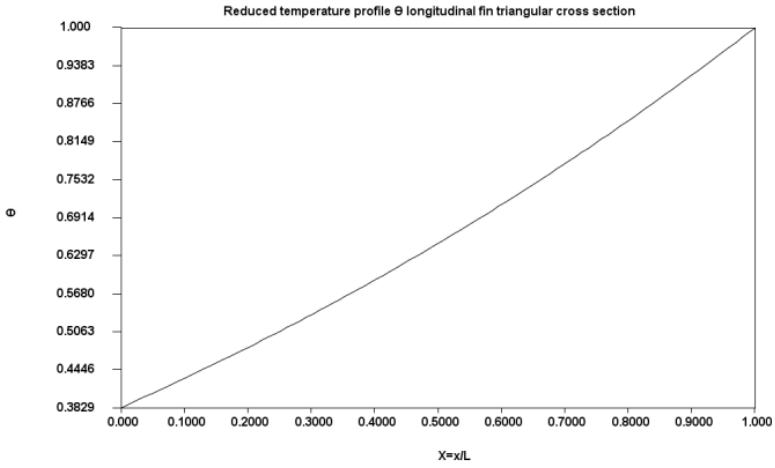
$\theta(X) = \frac{I_0(2M\sqrt{X})}{I_0(2M)} \quad \eta = \frac{I_1(2M)}{MI_0(2M)}$ where $M = \left(\frac{2h_{av}}{k\delta_0}\right)^{1/2} L$

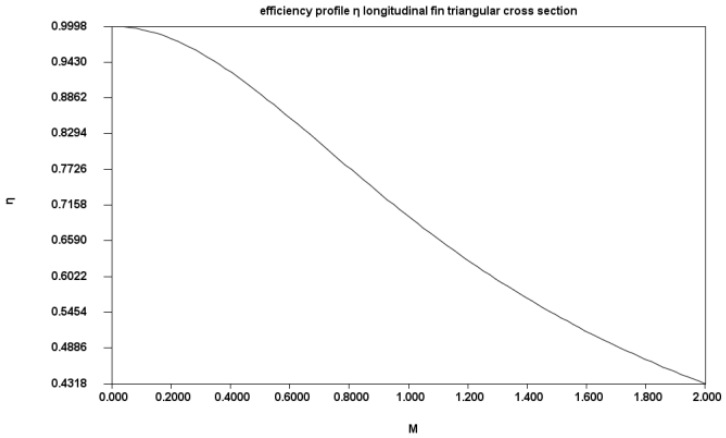
PROBLEM : For a longitudinal fin of triangular cross section the following values are given: h_∞=30 W/m^{2}K k=50 W/mK fin length L=0.1 m, width W=0.4 m and thicknes at the base δ_0=0.01 m. Calculate change of dimensionles temperature profile $\theta(X)$ for $0 \le X \le 1$ and $\eta = \frac{Q_{fin}}{Q_{maximum}}$ fin efficiency for $0 \le M \le 2$

```
 public class longitudinal_fin_tridiagonal_CS
{
public static void main(String arg[])
{   double h=30; //W/m^2K
    double k=50; //W/mK
    double L=0.1; //m
    double d0=0.01; //m
    double M1=Math.sqrt(2.0*h/(k*d0))*L;
              if_x eta=(M)->Mathd.I(1.0,2.0*M)/(M*Mathd.I(0.0,2.0*M));
              if_x teta=X->Mathd.I(0.0,2.0*M1*Math.sqrt(X))/Mathd.I(0.0,2.0*M1);
    double teta1[]=new double[101];
              Plot p1=new Plot(teta,0.0,1.0,100);
              p1.setPlabel("Reduced temperature profile "+'\u03F4'+" longitudinal fin triangular cross section");
              p1.setXlabel("X=x/L");
              p1.setYlabel(""+'\u03F4');
              p1.plot();
              Plot p2=new Plot(eta,0.0,2.0,100);
              p2.setPlabel("efficiency profile "+'\u03B7'+" longitudinal fin triangular cross section");
    p2.setXlabel("M");
    p2.setYlabel(""+'\u03B7');
              p2.plot();
```

```
        System.out.println("M="+M1+"eta = "+eta.func(M1));
}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" longitudinal_fin_tridiagonal_CS
M=1.0954451150103324eta = 0.663436620996408
```





Hand calculation of the problem:

h_∞=30 W/m^{2}K k=50 W/mK fin length L=0.1 m, width W=0.4 m and thicknes=0.01

$$M = \left(\frac{2h_{av}}{k\delta_0}\right)^{1/2} L$$

$m = \sqrt{\frac{2h}{\delta_0 k}} =$ M=mL

$$m = \sqrt{\frac{2h}{\delta_0 k}} = \sqrt{\frac{2 * 30}{0.01 * 50}} = 10.95445115$$

$$M = m * L = 10.95445115 * 0.1 = 1.095445115$$

$$2M = 2.19089023$$

$\theta(X) = \frac{I_0(2M\sqrt{X})}{I_0(2M)}$ $\eta = \frac{I_1(2M)}{MI_0(2M)}$

Linear interpolation from table:

x	$e^{-x}I_0(x)$	$e^{-x}I_1(x)$
2.1	0.299563095	0.213747672
2.2	0.291317334	0.212087733
2.19089023	0.292068503	0.212238949

$I_0(2M) = 0.292068503 * e^{2.19089023} = 2.612018581 I_1(2M) = 0.212238949 * e^{2.19089023}$ =1.898089227

$$\eta = \frac{I_1(2M)}{MI_0(2M)} = \frac{1.898089227}{1.095445115*2.612018581} = \mathbf{0.6633607}$$

$$A_{convection} = \left[2\sqrt{\left(\frac{\delta_0}{2}\right)^2 + L^2}\right]W = \left[2\sqrt{\left(\frac{0.01}{2}\right)^2 + 0.1^2}\right]0.4 = 0.080099938$$

$$Q_{max} = hA_{convection}(T_0 - T_\infty) = 30 * 0.800062498$$

$$Q_{fin} = \eta Q_{max} = 0.6633607 * 240.2998127 = 159.405 \text{ W}$$

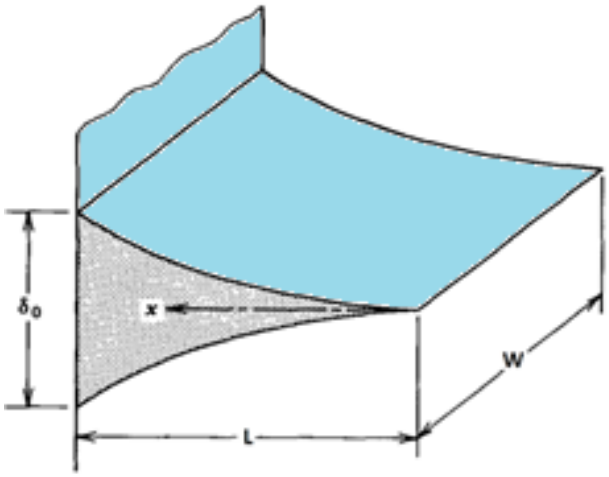
Note that computer program calculates the same efficiency as η=**0.663436620996408** . Small difference is due to linear interpolation error.

Longitudinal fin of concave parabolic profile

$\delta(x) = \delta_0\left(\frac{x}{L}\right)^2 = \delta_0 X^2$ $K(x) = \frac{\delta(x)}{\delta_0} = \frac{\delta_0 X^2}{\delta_0} = X^2$ $W(x) = \frac{h(x)}{h_0} = 1$

m=-1/2 c=0 n=∞ ν=m/c=-∞

$\theta(X) = X^{-\left(\frac{1}{2}\right)+\sqrt{\left(\frac{1}{4}\right)+M^2}}$ $\eta = \frac{2}{1+\sqrt{1+(2M)^2}}$ where $M = \left(\frac{2h_{av}}{k\delta_0}\right)^{1/2} L$



Longitudinal fin of convex parabolic profile

$\delta(x) = \delta_0\left(\frac{x}{L}\right)^{1/2} = \delta_0 X^{1/2}$ $K(x) = \frac{\delta(x)}{\delta_0} = \frac{\delta_0 X^{1/2}}{\delta_0} = X^{1/2}$ $W(x) = \frac{h(x)}{h_0} = 1$

m=1/4 c=3/4 n4/3ν=m/c=1/3

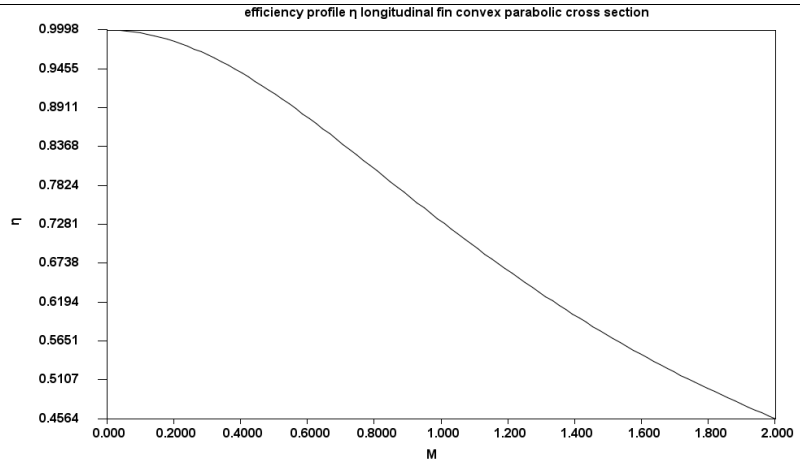
$\theta(X) = X^{1/4}\frac{I_{-\left(\frac{1}{3}\right)}(4/3MX)}{I_{-\left(\frac{1}{3}\right)}(4/3M)}$ $\eta = \frac{I_{\frac{2}{3}}(4/3M)}{MI_{-\frac{1}{3}}(4/3M)}$ where $M = \left(\frac{2h_{av}}{k\delta_0}\right)^{1/2} L$

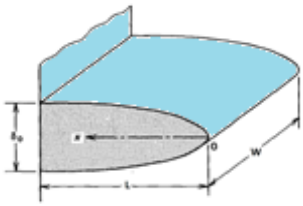
PROBLEM For a longitudinal fin of convex parabolic profile the following values are given: h_∞=30 W/m^{2}K k=50 W/mK fin length L=0.1 m, width W=0.4 m and thicknes at the base δ_0=0.01 m. Calculate change of dimensionles temperature profile $\theta(X)$ for $0 \le X \le 1$ and η=$\frac{Q_{fin}}{Q_{maximum}}$ fin efficiency for $0 \le M \le 2$

```
public class longitudinal_fin_convex_parabolic_CS
{
public static void main(String arg[])
{   double h=30; //W/m^2K
    double k=50; //W/mK
    double L=0.1; //m
    double W=0.4; //m
    double d0=0.01; //m
    double M1=Math.sqrt(2.0*h/(k*d0))*L;
            if_x eta=(M)->Mathd.I(2.0/3.0,4.0/3.0*M)/(M*Mathd.I(-1.0/3.0,4.0/3.0*M));
```

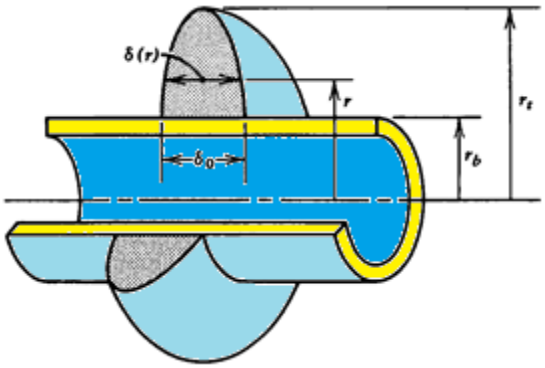
```
            if_x teta=X->Math.pow(X,0.25)*Mathd.I(-1.0/3.0,2.0*4.0/3.0*M1*X)/Mathd.I(-1.0/3.0,4.0/3.0*M1);
    double teta1[]=new double[101];
            Plot p1=new Plot(teta,0.0,1.0,100);
            p1.setPlabel("Reduced temperature profile "+'\u03F4'+" longitudinal fin convex parabolic cross section");
            p1.setXlabel("X=x/L");
            p1.setYlabel(""+'\u03F4');
            p1.plot();
            Plot p2=new Plot(eta,0.0,2.0,100);
            p2.setPlabel("efficiency profile "+'\u03B7'+" longitudinal fin convex parabolic cross section");
    p2.setXlabel("M");
    p2.setYlabel(""+'\u03B7');
            p2.plot();
            System.out.println("M="+M1+"eta = "+eta.func(M1));
}}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" longitudinal_fin_convex_parabolic_CS
M=1.0954451150103324eta = 0.6988647038726997
```





radial fin of arbitrary cross section



Let us now consider the radial fins in the form of circular disks. Thickness at the base r=r_b coss sectional area A(r) and perimeter P(r) of the pin is given as:

$A(r) = 2\pi r\delta(r)$ $A_0 = 2\pi r_b\delta_0$ $P(r) = 4\pi r_b$ where A_0 and P_0 are the cross section area and perimeter of the fin at the base of the fin.

$X = R = \frac{r}{r_t}$ $X_b = R_b = \frac{r_b}{r_t}$ $X_t = R_t = 1$ $M = \left(\frac{2h_{av}}{k\delta_0}\right)^{1/2} r_t$ $K(r) = R\frac{\delta(r)}{\delta_0}$ $W(r) = \frac{h(x)}{h_0}R = R$

The heat transfer from the fin (at the base $X_b = R_b = \frac{r_b}{r_t}$):

$$Q_{fin} = -k(2\pi r\delta(r))\frac{dT(r)}{dr}\Big|_{r=r_b} = -k2\pi\delta_0(T_b - T_\infty)R\frac{d\theta(R)}{dR}\Big|_{R=R_b}$$

If all fin temperature is at the base temperature of $T_b$maximum heat transfer will occur

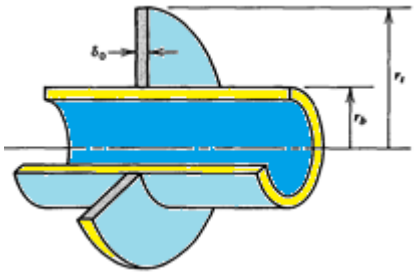
$$Q_{maximum} = 2\pi h_{av} r_t^2(1 - R_b^2)(T_b - T_\infty)$$

The ratio of fin heat transfer to maximum possible heat transfer is called fin efficiency

$$\eta = \frac{Q_{fin}}{Q_{maximum}} = -\frac{2}{M^2}\frac{R}{(1-R^2)}\frac{d\theta(R)}{dR}\Big|_{R=R_b} \quad \text{or } Q_{fin} = \eta Q_{maximum}$$

Now radial fins with known profile can be investigated

radial fin of rectangular cross section



$$K(r) = R \quad W(r) = \frac{h(x)}{h_0}R = R \quad R = \frac{r}{r_t}$$

m=0 c=1 n=1 ν=m/c=0

The temperature distribution

$$\theta(R) = \frac{K_1(M)I_0(MR) + I_1(M)K_0(MR)}{K_1(M)I_0(MR_b) + I_1(M)K_0(MR_b)}$$

Where K is hyperbolic Bessel function which defines as:

$K_\nu(z) = \frac{\pi}{2}\frac{I_{-\nu}(z) - I_\nu(z)}{\sin(\nu\pi)}$ It should be also note that $K_{-1}(z) = K_1(z) \quad I_{-1}(z) = I_1(z)$

$$\eta = \frac{Q_{fin}}{Q_{maximum}} = \frac{2}{M}\frac{R_b}{1-R_b^2}\frac{I_1(M)K_1(MR_b) - K_1(M)I_1(MR_b)}{I_1(M)K_0(MR_b) + K_1(M)I_0(MR_b)} \text{ where } M = \left(\frac{2h_{av}}{k\delta_0}\right)^{1/2} r_t$$

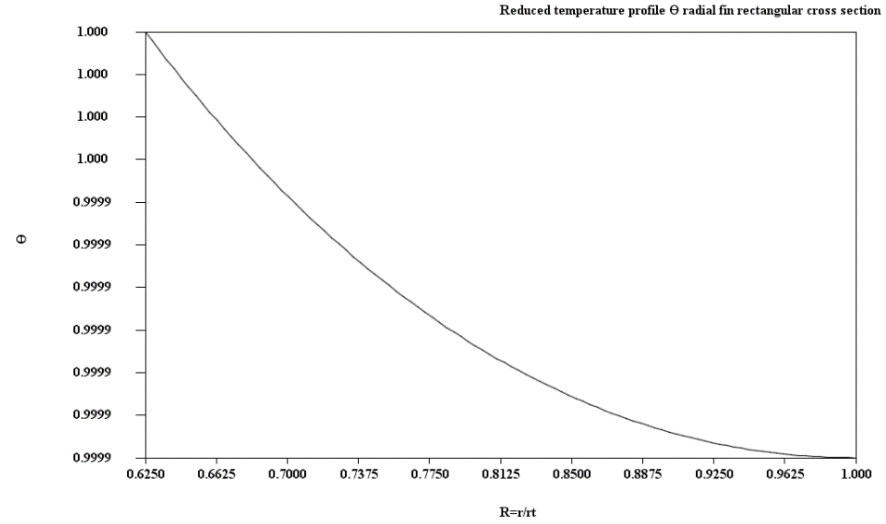
PROBLEM : For radial fin with the rectangular cross section the following values are given: h_∞=30 W/m^{2}K k=50 W/mK fin tip Radius r_t =0.1 m, fin base radius r_b=0.05 m, and fin thickness is 10^{-3} m Calculate change of dimensionles temperature profile $\theta(X)$ for $0 \le X \le 1$ and $\eta = \frac{Q_{fin}}{Q_{maximum}}$ fin efficiency for $0 \le M \le 2$

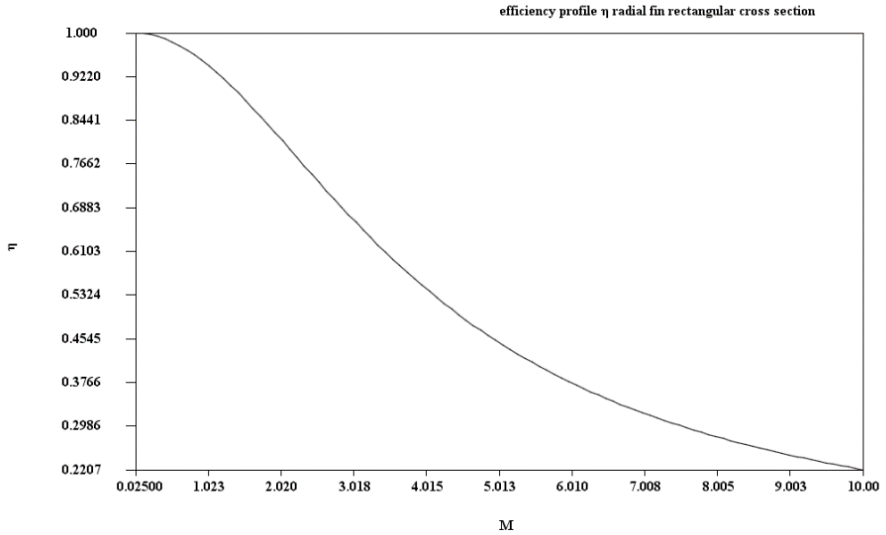
```
 public class radial_fin_rectangular_cs
{
public static void main(String arg[])
{   double h=50; //W/m^2K
    double k=200; //W/mK
    double d=5e-3; //m
    double rt=4e-3; //m
    double rb=2.5e-3; //m
    double Rb=rb/rt;
    System.out.println("Rb="+Rb);
    double Rt=1.0;
    double M1=Math.sqrt(2.0*h/(k*d))*rt;
    double M0=Math.sqrt(2.0*h/(k*d))*rb;
    if_x eta=M->2.0/M*Rb/(1-Rb*Rb)*(Mathd.I(1.0,M)*Mathd.K(1.0,M*Rb)-
Mathd.K(1.0,M)*Mathd.I(1.0,M*Rb))/(Mathd.I(1.0,M)*Mathd.K(0.0,M*Rb)+Mathd.K(1.0,M)*Mathd.I(0.0,M*Rb));
            if_x teta=R-
>(Mathd.K(1.0,M1)*Mathd.I(0.0,M1*R)+Mathd.I(1.0,M1)*Mathd.K(0.0,M1*R))/(Mathd.K(1.0,M1)*Mathd.I(0.0,M1*Rb)+Mathd.I(1.0,M
1)*Mathd.K(0.0,M1*Rb));
    double teta1[]=new double[101];
            Plot p1=new Plot(teta,Rb,Rt,100);
            p1.setPlabel("Reduced temperature profile "+'\u03F4'+" radial fin rectangular cross section");
            p1.setXlabel("R=r/rt");
            p1.setYlabel(""+'\u03F4');
            p1.plot();
```

```
            Plot p2=new Plot(eta,M0,10.0,100);
            p2.setPlabel("efficiency profile "+'\u03B7'+" radial fin rectangular cross section");
    p2.setXlabel("M");
    p2.setYlabel(""+'\u03B7');
            p2.plot();
            System.out.println("M="+M1+"eta = "+eta.func(M1));

}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" radial_fin_rectangular_cs
M=6.9282032302755105eta = 0.09703374200859885
```





PROBLEM: A radial fin of rectangular profile is exposed to surroundings at a temperature of T=35 °C and a heat transfer coefficient of h = 40 W/m^2 · K. The temperature at the fin base is 110 °C and the fin is made from a steel with k = 40 W/(mK) with outer and inner diameters of 25 and 10 cm. The fin thickness is 0.25 cm. Determine

(a) the fin efficiency,

(b) the temperature at the tip of the fin, and

(c) the dissipation of the fin.

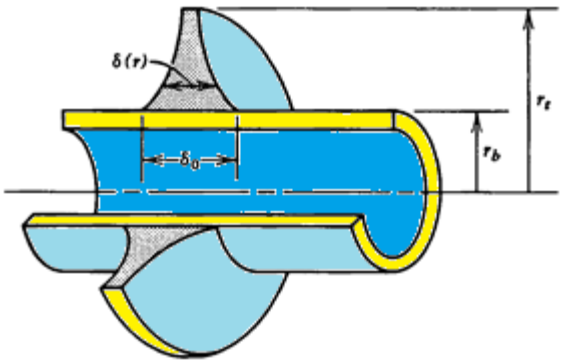
```
public class radial_fin_rectangular_cs_2
{
public static void main(String arg[])
{   double h=40; //W/m^2K
    double k=40; //W/mK
    double d=10e-2; //m
    double D=25e-2;
    double rt=D/2.0; //m
```

```
    double rb=d/2.0; //m
    double Rb=rb/rt;
    double t=0.25e-2;
    double t1=7e-3;
    System.out.println("rt="+rt+"rb="+rb+"Rb="+Rb);
    double Rt=1.0;
    double m=Math.sqrt(2.0*h/(k*t));
    double M1=m*rt;
    System.out.println("M1="+M1);
    if_x eta=M->2.0/M*Rb/(1-Rb*Rb)*(Mathd.I(1.0,M)*Mathd.K(1.0,M*Rb)-
Mathd.K(1.0,M)*Mathd.I(1.0,M*Rb))/(Mathd.I(1.0,M)*Mathd.K(0.0,M*Rb)+Mathd.K(1.0,M)*Mathd.I(0.0,M*Rb));
    double eta1=eta.func(M1);
    System.out.println("Rb="+Rb+"eta="+eta1);
    if_x teta=R-
>(Mathd.K(1.0,M1)*Mathd.I(0.0,M1*R)+Mathd.I(1.0,M1)*Mathd.K(0.0,M1*R))/(Mathd.K(1.0,M1)*Mathd.I(0.0,M1*Rb)+Mathd.I(1.0,M
1)*Mathd.K(0.0,M1*Rb));
    double Ta=110.0;
    double Tinf=35.0;
    double tetat=teta.func(Rt)*(Ta-Tinf);
    System.out.println("tetat="+tetat);
    double Tt=tetat+Tinf;
    System.out.println("Tt="+Tt+"degree C");
    double A=2.0*Math.PI*(rt*rt-rb*rb);
    System.out.println("A="+A+" m^2");
    double q=h*A*eta1*(Ta-Tinf);
    System.out.println("q="+q+" W");
}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" radial_fin_rectangular_cs_2
rt=0.125rb=0.05Rb=0.4
M1=3.5355339059327378
Rb=0.4eta=0.3430647848803193
tetat=13.546948228559222
Tt=48.54694822855922degree C
A=0.08246680715673206 m^2
q=84.87437237097318 W

> Terminated with exit code 0.
```

radial fin of hyperbolic cross section



$\delta(r) = \delta_0 \frac{r_b}{r}$ $M = \left(\frac{2h_{av}}{k\delta_0}\right)^{1/2} r_t^2$ $K(R) = R\frac{r_b}{r} = R_b$ $W(R) = R$

To maket he equation homegenious M and K will be further divided by R_b so

$M^* = \left(\frac{2h_{av}}{k\delta_0}\right)^{1/2} \frac{r_t}{\sqrt{R_b}}$ $K(R) = 1$ $W(R) = R$ for this case

m=1/2 c=3/2 n=2/3 ν=m/c=1/3

Then the solution for the temperature profile

$$\theta(R) = \left(\frac{R}{R_b}\right)^{1/2} \frac{I_{2/3}(2/3M^*)I_{1/3}(2/3M^*R^{3/2}) + I_{-2/3}(2/3M^*)I_{-1/3}(2/3M^*R^{3/2})}{I_{2/3}(2/3M^*)I_{1/3}(2/3M^*R_b^{2/3}) + I_{-2/3}(2/3M^*)I_{-1/3}(2/3M^*R_b^{2/3})}$$

And fin efficiency will be:

$$\eta = \left(\frac{2}{M^*}\right)\left(\frac{R_b}{1-R_b^2}\right)\frac{I_{2/3}(2/3M^*)I_{-2/3}(2/3M^*R_b^{3/2}) - I_{-2/3}(2/3M^*)I_{2/3}(2/3M^*R_b^{3/2})}{I_{-2/3}(2/3M^*)I_{-1/3}(2/3M^*R_b^{2/3}) - I_{2/3}(2/3M^*)I_{1/3}(2/3M^*R_b^{2/3})}$$

spine fin of arbitrary cross section

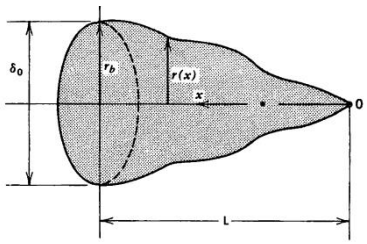
Let us now consider a spine of arbitrary cross section where L is height of spine and r_b is the base diameter

$R(x) = \frac{r(x)}{r_b} \quad X = \frac{x}{L}$

Then $X_t = 0$ (spine tip) $X_b = 1$ (spine base)

$M = \left(\frac{2h_{av}}{kr_b}\right)^{1/2} L \quad K(X) = [R(x)]^2 \quad W(X) = R(x)$

$Q_{fin} = -k(2\pi r_b^2)\frac{dT(x)}{dx}\Big|_{x=L} = -k(2\pi r_b^2)\frac{(T_b - T_\infty)}{L}\frac{d\theta(X)}{dX}\Big|_{X=1}$

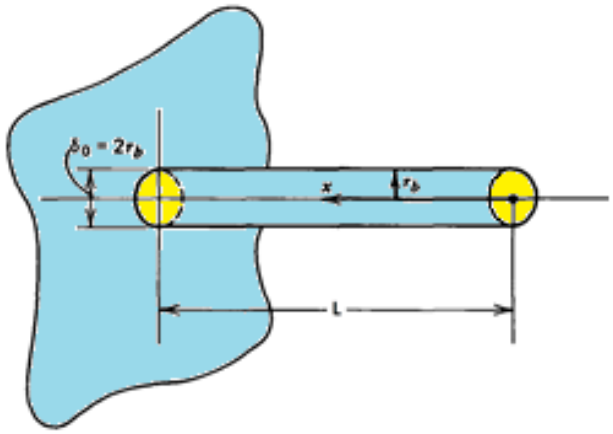


$Q_{maximum} = 2\pi r_b L \int_{X=0}^{1} R(x) h_{av}(T_b - T_\infty) dX$

$\eta = \frac{Q_{fin}}{Q_{maximum}} = \frac{1}{M^2 \int_{X=0}^{1} R(x) dX}\frac{d\theta(X)}{dX}\Big|_{X=1}$

Cyclindrical spine cross section

$R(x) = \frac{r_b}{r_b} = 1 \quad K(X) = 1 \quad W(X) = 1 \qquad 0 \le X \le 1$



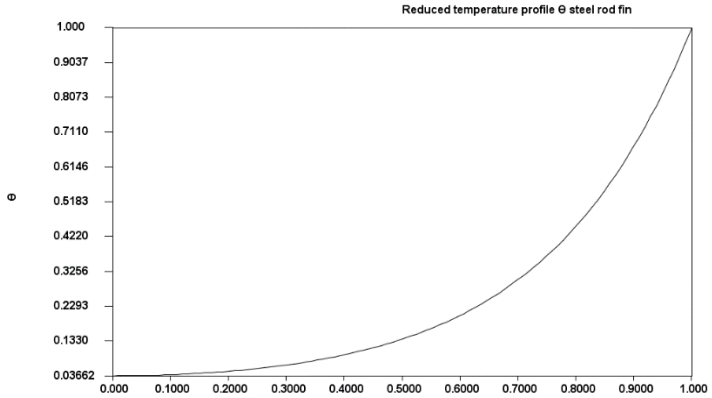
$\theta(X) = \frac{cosh(MX)}{cosh(M)} \quad \eta = \frac{tanh(M)}{M}$ where $M = \left(\frac{2h_{av}}{kr_b}\right)^{1/2} L$

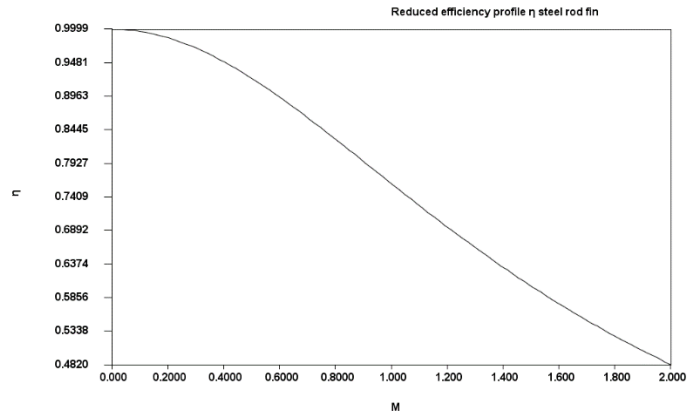
PROBLEM : For a steel rod following values are given: h_∞=64 W/m^{2}K k=50 W/mK fin length L=0.25 m, fin base diameter r_b=0.01 m. Calculate change of dimensionles temperature profile $\theta(X)$ for $0 \le X \le 1$ and $\eta = \frac{Q_{fin}}{Q_{maximum}}$ fin efficiency . Enviromental Temperature T_∞=20 °C and fin base temperature T_b=120 °C. Also calculate fin heat transfer for the given conditions.

```
public class steel_rod_fin
{
public static void main(String arg[])
{   double h=64; //W/m^2K
    double k=50; //W/mK
    double L=0.25; //m
    double rb=0.01; //m
    double T1=120;
    double T2=20;
    double M1=Math.sqrt(2.0*h/(k*rb))*L;
              if_x eta=(M)->Math.tanh(M)/M;
              if_x teta=X->Math.cosh(M1*X)/Math.cosh(M1);
             double teta1[]=new double[101];
              Plot p1=new Plot(teta,0.0,1.0,100);
              p1.setPlabel("Reduced temperature profile "+'\u03F4'+" steel rod fin");
              p1.setXlabel("X=x/L");
              p1.setYlabel(""+'\u03F4');
```

```
            p1.plot();
            Plot p2=new Plot(eta,0.0,2.0,100);
            p2.setPlabel("Reduced efficiency profile "+'\u03B7'+" steel rod fin");
            p2.setXlabel("M");
            p2.setYlabel(""+'\u03B7');
            p2.plot();
            double eta1=eta.func(M1);
            double A=Math.PI*2*rb*L;
            double Qmax=h*A*(T1-T2);
            double Q=eta1*Qmax;
            System.out.println("M = "+M1+" eta = "+eta1+" Qmax = "+Qmax+" Q = "+Q);
}
}
```

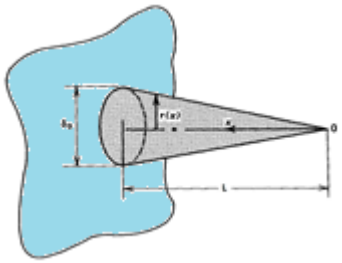
```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" steel_rod_fin
M = 4.0 eta = 0.24983232493476676 Qmax = 100.5309649148734 W Q = 25.115884692618287 W
```





Conical spine cross section

$r(x) = \frac{\delta_0}{2}\frac{x}{L} = r_b\frac{x}{L}$ $R(x) = \frac{r(x)}{r_b} = X$



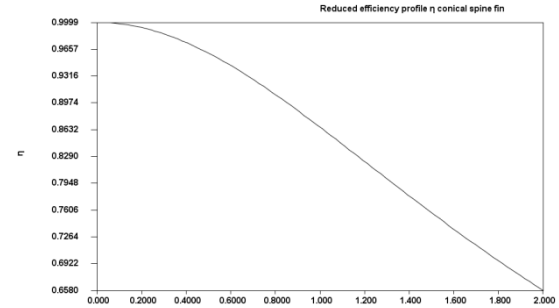
$K(X) = X^2$ $W(X) = X$ m=-1/2 c=1/2 n=2 ν=m/c=-1

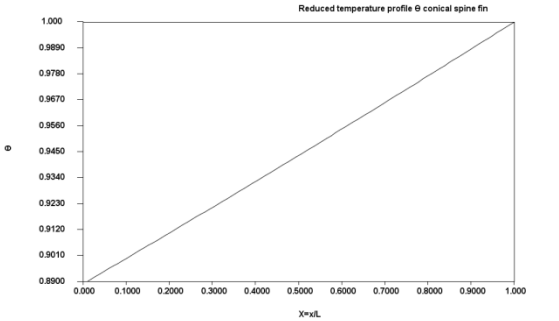
$\theta(X) = X^{-1/2}\frac{I_1(2M\sqrt{X}}{I_1(2M)}$ η=$\frac{Q_{fin}}{Q_{maximum}} = \frac{2}{M}\frac{I_2(2M)}{I_1(2M)}$ where $M = \left(\frac{2h_{av}}{kr_b}\right)^{1/2}L$

PROBLEM : For a conical spine fin the following values are given: h_∞=30 W/m^{2}K k=50 W/mK fin length L=0.1 m, fin base diameter r_b=0.05 m. Calculate change of dimensionles temperature profile $\theta(X)$ for $0 \le X \le 1$ and $\eta=\frac{Q_{fin}}{Q_{maximum}}$ fin efficiency for $0 \le M \le 2$

```
public class conical_spine_fin
{
public static void main(String arg[])
{   double h=30; //W/m^2K
    double k=50; //W/mK
    double L=0.1; //m
    double rb=0.05; //m
    double M1=Math.sqrt(2.0*h/(k*rb))*L;
              if_x eta=(M)->2/M*Mathd.I(2.0,(2.0*M))/Mathd.I(1.0,(2.0*M));
              if_x teta=X->1/Math.sqrt(X)*Mathd.I(1.0,(2.0*M1*Math.sqrt(X)))/Mathd.I(1.0,(2.0*M1));
    double teta1[]=new double[101];
              Plot p1=new Plot(teta,0.0,1.0,100);
              p1.setPlabel("Reduced temperature profile "+'\u03F4'+" conical spine fin");
              p1.setXlabel("X=x/L");
              p1.setYlabel(""+'\u03F4');
              p1.plot();
              Plot p2=new Plot(eta,0.0,2.0,100);
              p2.setPlabel("Reduced efficiency profile "+'\u03B7'+" conical spine fin");
             p2.setXlabel("M");
             p2.setYlabel(""+'\u03B7');
              p2.plot();
              System.out.println("M="+M1+"eta = "+eta.func(M1));
}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" conical_spine_fin
M=0.4898979485566356eta = 0.9622557596200761
```

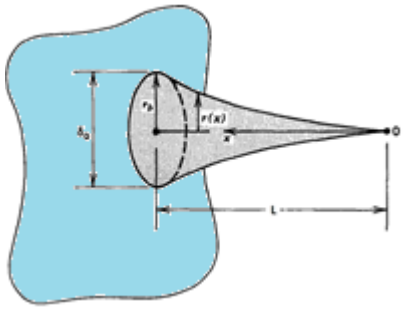




Concave parabolic spine cross section

$r(x)=\frac{\delta_0}{2}\left(\frac{x}{L}\right)^2=r_b\left(\frac{x}{L}\right)^2 \quad R(x)=X^2 \;\; K(X)=X^4 \;\; W(X)=X^2$

m=-3/2 c=0 n= ∞ ν=m/c=-∞

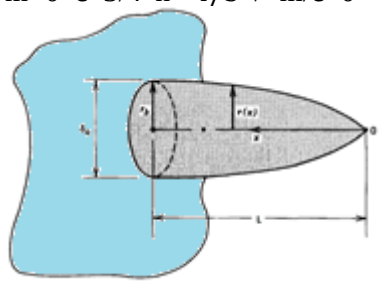


$\theta(X) = X^{-\frac{3}{2}+\sqrt{\left(\frac{9}{4}\right)+M^2}}$ $\eta=\frac{Q_{fin}}{Q_{maximum}} = \frac{2}{1+\sqrt{1+[2/3M]^2}}$ where $M = \left(\frac{2h_{av}}{kr_b}\right)^{1/2} L$

Convex parabolic spine cross section

$r(x) = \frac{\delta_0}{2}\left(\frac{x}{L}\right)^{1/2} = r_b\left(\frac{x}{L}\right)^{1/2}$ $R(x) = X^{1/2}$ $K(X) = X$ $W(X) = X^{1/2}$

m=0 c=3/4 n= 4/3 ν=m/c=0



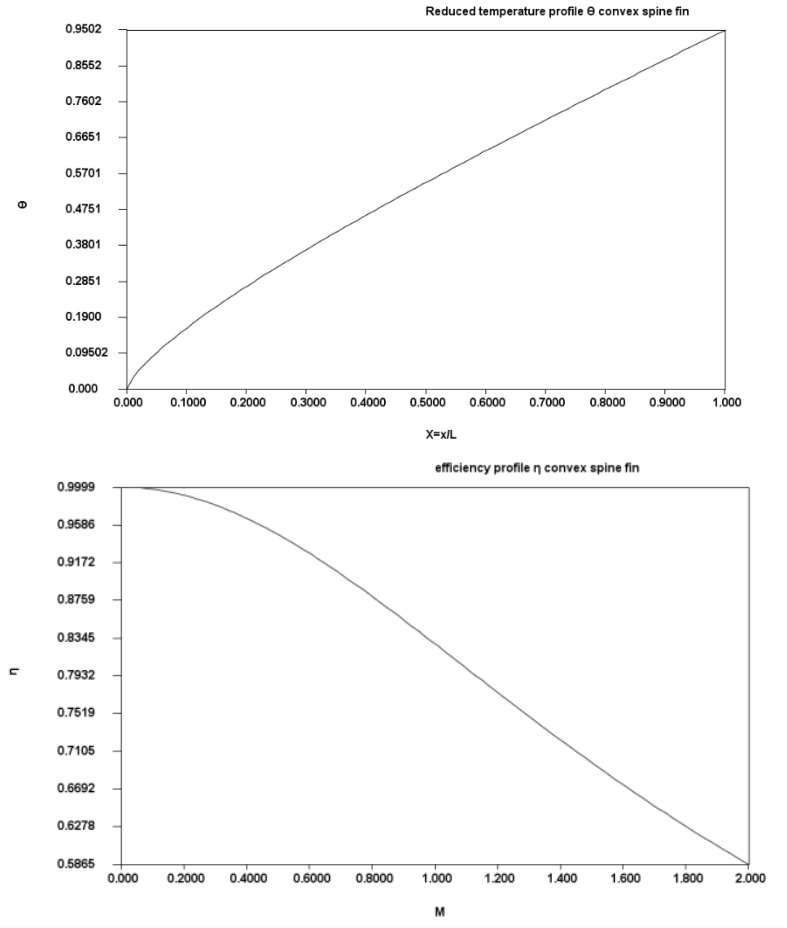
$\theta(X) = \frac{I_o(\frac{4}{3}MX^{\frac{3}{4}})}{I_o(\frac{4}{3}M)}$ $\eta=\frac{Q_{fin}}{Q_{maximum}} = \frac{3}{2M}\frac{I_1(\frac{4}{3}M)}{I_0(\frac{4}{3}M)}$ where $M = \left(\frac{2h_{av}}{kr_b}\right)^{1/2} L$

PROBLEM : For a convex spine fin the following values are given: h_∞=30 W/m^{2}K k=50 W/mK fin length L=0.1 m, fin base diameter r_b=0.05 m. Calculate change of dimensionles temperature profile $\theta(X)$ for $0 \le X \le 1$ and $\eta=\frac{Q_{fin}}{Q_{maximum}}$ fin efficiency for $0 \le M \le 2$

```
public class convex_spine_fin
{
public static void main(String arg[])
{   double h=30; //W/m^2K
    double k=50; //W/mK
    double L=0.1; //m
    double rb=0.05; //m
    double M1=Math.sqrt(2.0*h/(k*rb))*L;
            if_x eta=(M)->3.0/2.0/M*Mathd.I(1.0,(4.0/3.0*M))/Mathd.I(0.0,(4.0/3.0*M));
            if_x teta=X->3.0/2.0/M1*Mathd.I(1.0,(4.0/3.0*M1*Math.pow(X,(3.0/4.0))))/Mathd.I(0.0,(4.0/3.0*M1));
    double teta1[]=new double[101];
            Plot p1=new Plot(teta,0.0,1.0,100);
            p1.setPlabel("Reduced temperature profile "+'\u03F4'+" convex spine fin");
            p1.setXlabel("X=x/L");
            p1.setYlabel(""+'\u03F4');
            p1.plot();
            Plot p2=new Plot(eta,0.0,2.0,100);
            p2.setPlabel("Reduced efficiency profile "+'\u03B7'+" convex spine fin");
    p2.setXlabel("M");
    p2.setYlabel(""+'\u03B7');
            p2.plot();
            System.out.println("M="+M1+"eta = "+eta.func(M1));

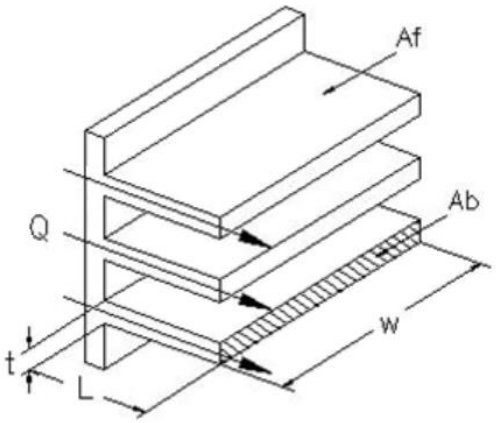
}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" convex_spine_fin
M=0.4898979485566356eta = 0.9502002443343172
```



PROBLEM: Aliminium fins of rectangular profile are attached on a plane wall with 5 mm spacing The fins have thickness t=1 mm, length L=10 mm, and thermal conductivity of the fin material is k=200 W/(mK). The plane wall is maintained at T_0=240 C, and the fins dissipate heat by convection into the ambient air at T_{air}=40 C with heat transfer coefficient h_m=50 W/(m^2K)

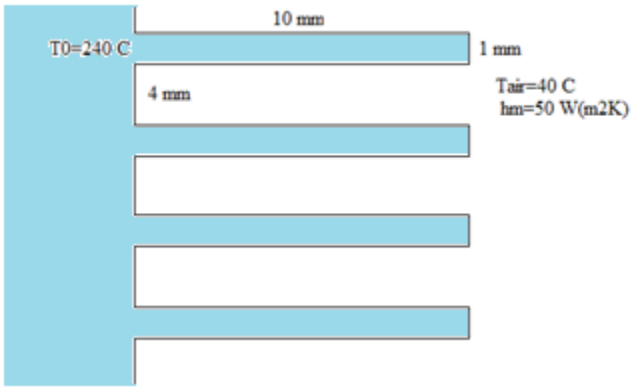
$$Q = \left[\left(A - A_f\right)1 + A_f\,\eta\right]\Delta T = \left[A - (1 - A_f)\eta\right]\Delta T$$



a) Determine the fin efficiency

b) Determine the area weighted fin efficiency

c) Determine the heat loss from the wall per square meter of wall surface



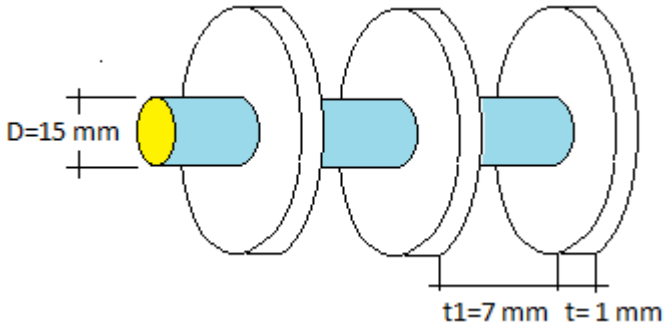
Base temperature	To	240	degree C
air temperature	Tair	40	degree C
air-to fin thermal convection coef.	hm	50	W/m^2K
thermal conductivity of fin material	k	200	W/(mK)
Fin length	L	1.00E-02	m
Fin width	W	1.00E+00	m
Fin Thickness	t	1.00E-03	m
Distance between fins	t_1	4.00E-03	m
m=√(2h/(δ_0 k))		22.36068	
M=mL		2.24E-01	
η_{fin}		9.84E-01	
Afin	2*L*W	2.00E-02	
η_{base}		1	
Abase	t_1*W	4.00E-03	
$A_{base}\eta_{base} + A_{fin}\eta_{fin}$		2.37E-02	
Number of fin+base in 1 m	N	2.00E+02	
N*(A_{base} η_base+A_{fin} η_fin)		4.73E+00	
Total heat transfer	Q	946.928	W/m height 1 m weight

PROBLEM: Circular aluminium fins of constant thickness are attached to a pipe with $D=2.5x10^{-2}$ m diameter. Fin thickness $t=10^{-3}$ m and outside diameter of $D=5.5x10^{-2}$ m (Fin height $L=1.5x10^{-2}$ m). Thermal conductivity of fins k=200 $W/(m^2K)$. The pipe surface temperature T_0=200 ℃. The space between circular fin plates are $t1=8x10^{-3}$ m. Ambient temperature is $T_\infty = 40$ ℃ and thermal convectivity is h=50 $W/(m^2K)$

a) Determine the fin efficiency

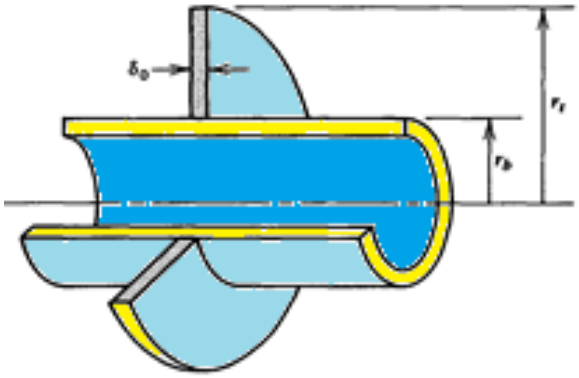
b) Area weighted fin efficiency

c) What will be the heat loss per m of the pipe(total of 125 fins)



Radial fin solution was given as:

radial fin of rectangular cross section



$K(r) = R \quad W(r) = \frac{h(x)}{h_0} R = R$

m=0 c=1 n=1 ν=m/c=0

The temperature distribution

$\theta(R) = \frac{K_1(M)I_0(MR)+I_1(M)K_0(MR)}{K_1(M)I_0(MR_b)+I_1(M)K_0(MR_b)}$

Where K is hyperbolic Bessel function which defines as:

$K_\nu(z) = \frac{\pi}{2}\frac{I_{-\nu}(z)-I_\nu(z)}{\sin(\nu\pi)}$ It should be also note that $K_{-1}(z) = K_1(z) \quad I_{-1}(z) = I_1(z)$

$\eta = \frac{Q_{fin}}{Q_{maximum}} = \frac{2}{M}\frac{R_b}{1-R_b^2}\frac{I_1(M)K_1(MR_b)-K_1(M)I_1(MR_b)}{I_1(M)K_0(MR_b)+K_1(M)I_0(MR_b)}$ where $M = \left(\frac{2h_{av}}{k\delta_0}\right)^{1/2} r_t$

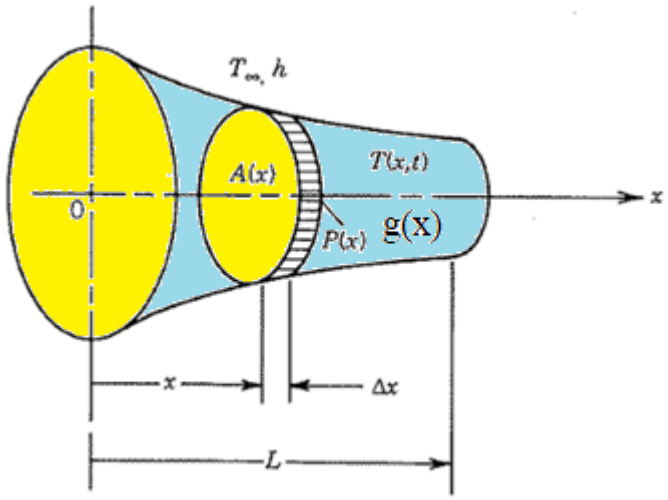
```
 public class radial_fin_rectangular_cs_1
{
public static void main(String arg[])
{   double h=80; //W/m^2K
    double k=200; //W/mK
    double d=2.5e-2; //m
    double D=5.5e-2;
    double rt=D/2.0; //m
    double rb=d/2.0; //m
    double Rb=rb/rt;
    double t=1e-3;
    double t1=7e-3;
    double Ta=200;
    double Tinf=40;
    System.out.println("rt="+rt+"rb="+rb+"Rb="+Rb);
    double Rt=1.0;
    double M1=Math.sqrt(2.0*h/(k*t))*rt;
    System.out.println("M1="+M1);
    if_x eta=M->2.0/M*Rb/(1-Rb*Rb)*(Mathd.I(1.0,M)*Mathd.K(1.0,M*Rb)-
Mathd.K(1.0,M)*Mathd.I(1.0,M*Rb))/(Mathd.I(1.0,M)*Mathd.K(0.0,M*Rb)+Mathd.K(1.0,M)*Mathd.I(0.0,M*Rb));
    double eta1=eta.func(M1);
    System.out.println("eta="+eta1);
    double Af=2.0*Math.PI*(rt*rt-rb*rb);
    double A_unfinned=Math.PI*d*t1;
    double Atotal=Af+A_unfinned;
    System.out.println("A fin="+Af+"A_unfinned="+A_unfinned+"Atotal="+Atotal);
    double beta=Af/Atotal;
    System.out.println("beta="+beta);
    double eta2=beta*eta1+(1.0-beta);
    System.out.println("eta2="+eta2);
    int N=(int)(1.0/(t+t1));
    System.out.println("N="+N);
    double Qf=eta2*Atotal*h*(Ta-Tinf)*N;
    System.out.println("Qf="+Qf+" W/m");
    double A1=Math.PI*d*1.0;
    double Q_no_fin=A1*h*(Ta-Tinf);
    System.out.println("Q_no_fin="+Q_no_fin+" W/m");

}
}
---------- Capture Output ----------
```

```
> "D:\co\java\bin\javaw.exe" radial_fin_rectangular_cs_1
rt=0.0275rb=0.0125Rb=0.4545454545454546
M1=0.7778174593052023
eta=0.9187895996858196
A fin=0.003769911843077513A_unfinned=5.497787143782138E-4Atotal=0.004319689898685965
beta=0.8727272727272728
eta2=0.9291254688167153
N=125
Qf=6421.654243615082 W/m
Q_no_fin=1005.3096491487338 W/m
> Terminated with exit code 0.
```

A case of heat generation in fin structure:

Let us look at a fin with constant heat generation inside.



A(x) = cross sectional area of the disk

P(x) = perimeter of the disk

L = Length of the fin

$\Delta S(x) = P(x)\Delta x$ =lateral surface area of the disk $g(x)$

h = convective heat transfer coefficient

k= thermal conductivity of solid

g(x) = heat generation at point x

T_∞= ambient temperature

T_0 = base temperature (assumed to be constant)

Energy balance (First law of thermodynamics)

Net rate of heat gain by conduction +rate of gain by convection from the lateral surfaces=rate of increase of internal energy

$$A(x)q_x - A(x+\Delta x)q_{x+\Delta x} + h(x)\Delta S(x)\big(T_\infty - T(x,t)\big) + \Delta x A(x)g(x) = \rho C_p \Delta x A(x)\frac{\partial T(x,t)}{\partial t}$$

$$A(x)q_x - A(x)q_x - \frac{\partial}{\partial x}(A(x)q_x)\Delta x + h(x)\Delta S(x)\big(T_\infty - T(x,t)\big) + \Delta x A(x)g(x) = \rho C_p \Delta x A(x)\frac{\partial T(x,t)}{\partial t}$$

$$-\frac{\partial}{\partial x}(A(x)q_x)\Delta x + h(x)\Delta S(x)\big(T_\infty - T(x,t)\big) + \Delta x A(x)g(x) = \rho C_p \Delta x A(x)\frac{\partial T(x,t)}{\partial t}$$

Fourier's law of conduction:

$$q_x = -k\frac{\partial T(x,t)}{\partial x}$$

$$\frac{\partial}{\partial x}\left(A(x)k\frac{\partial T(x,t)}{\partial x}\right)\Delta x + h(x)\Delta S(x)\big(T_\infty - T(x,t)\big) + A(x)g(x) = \rho C_p \Delta x A(x)\frac{\partial T(x,t)}{\partial t}$$

$$\frac{\partial}{\partial x}\left(A(x)k\frac{\partial T(x,t)}{\partial x}\right) - h(x)\frac{\Delta S(x)}{\Delta x}(T(x,t) - T_\infty) + A(x)g(x) = \rho C_p A(x)\frac{\partial T(x,t)}{\partial t}$$

$$\theta(x,t) = \frac{T(x,t) - T_\infty}{T_0 - T_\infty}$$

$\frac{\partial}{\partial x}\left(A(x)k\frac{\partial\theta(x,t)}{\partial x}\right)-h(x)\frac{\Delta S(x)}{\Delta x}\theta(x,t)+A(x)g(x)=\rho C_pA(x)\frac{\partial\theta(x,t)}{\partial t}$

k≠ $f(x)$

$\frac{1}{A(x)}\frac{\partial}{\partial x}\left(A(x)\frac{\partial\theta(x,t)}{\partial x}\right)-\frac{h(x)}{A(x)k}\frac{\Delta S(x)}{\Delta x}\theta(x,t)+\frac{1}{k}g(x)=\frac{\rho C_p}{k}\frac{\partial\theta(x,t)}{\partial t}=\frac{1}{\alpha}\frac{\partial\theta(x,t)}{\partial t}$

If steady state (independent of time)

$\frac{\partial}{\partial x}\left(A(x)\frac{\partial\theta(x)}{\partial x}\right)-\frac{h(x)}{k}\frac{\Delta S(x)}{\Delta x}\theta(x)+\frac{1}{k}g(x)=0$

$\frac{\partial}{\partial x}\left(A(x)\frac{\partial\theta(x)}{\partial x}\right)-\frac{h(x)P(x)}{k}\theta(x)+\frac{1}{k}g(x)=0$

$$\theta(x,t)=\frac{T(x,t)-T_\infty}{T_0-T_\infty}$$

Now let us consider a solid cylider of diameter D and length L with constant heat generation g and constant convective heat transfer,h the equation will become:

$\frac{\partial^2\theta(x)}{\partial x^2}-\frac{hP}{Ak}\theta(x)+\frac{1}{k}g=0\quad\frac{P}{A}=\frac{\pi D}{\left(\frac{\pi D^2}{4}\right)}=\frac{4}{D}\quad\frac{\partial^2\theta(x)}{\partial x^2}-\frac{4h}{Dk}\theta(x)+\frac{1}{k}g=0\quad m^2=\frac{4h}{Dk}$

$\frac{\partial^2\theta(x)}{\partial x^2}-m^2\theta(x)+\frac{1}{k}g=0$

With boundary conditions:

$\theta(x,t)=1$ at x=0

$\theta(x,t)=0$ at x=0 (very long length so temperature reach to the air temperature at the end of the rod)

$\theta(x)=C_1\cosh(mx)+C_2\sinh(mx)+\frac{g}{km^2}\qquad\frac{g}{km^2}=\frac{g}{k\frac{4h}{Dk}}=\frac{gD}{4h}$

$\theta(x)=C_1\cosh(mx)+C_2\sinh(mx)+\frac{gD}{4h}$

$\theta(x,t)=1$ at x=0

$1=C_1\cosh(m0)+C_2\sinh(m0)+\frac{gD}{4h}$

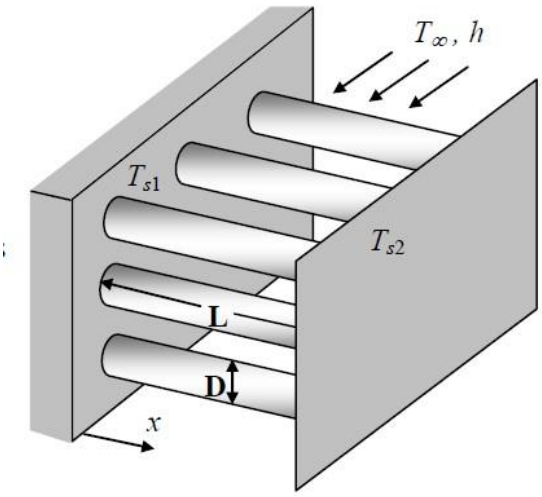
$C_1=1-\frac{gD}{4h}$

$\theta(x,t)=0$ at x=0

$0=\left(1-\frac{gD}{4h}\right)\cosh(mL)+C_2\sinh(mL)+\frac{gD}{4h}$

$C_2=-\left(1-\frac{gD}{4h}\right)\frac{\cosh(mL)}{\sinh(mL)}-\frac{1}{\sinh(mL)}\frac{gD}{4h}$

PROBLEM:



Circular cooling fins of diameter D= 1 mm and length L =25.4 mm, made of copper (k =400 W/m · K), are used to enhance heat transfer from a surface that is maintained at temperature T_{s1}= 132°C. Each rod has one end attached to this surface (x =0), while the opposite end (x =L) is joined to a second surface, which is maintained at T_{s2}= 0°C. The air flowing between the surfaces and the rods is also at T_∞=0°C, and the convection coefficient is h=100 W/m^2 · K.

a) Express the function $\theta(x) = T(x) - T_\infty$ along a fin, and calculate the temperature at $x = L/2$.
b) Determine the rate of heat transferred from the hot surface through each fin and the fin effectiveness
c) What is the total rate of heat transfer from a 10-cm by 10-cm section of the wall, which has 625 uniformly distributed fins? Assume the same convection coefficient for the fin and for the unfinned wall surface

$$\theta = T(x) - T_\infty$$

$$\frac{\partial^2\theta(x)}{\partial x^2} - m^2\theta(x) = 0$$

$$m^2 = \frac{hP}{Ak} = \frac{h2\pi r}{\pi r^2 k} = \frac{2h}{kr}$$

$$\theta(x) = C_1 \sinh(mx) + C_2 \cosh(mx)$$

x=0

$$\theta_{s1} = T_{s1} - T_\infty = C_1 \sinh(m0) + C_2 \cosh(m0)$$

$$C_2 = \theta_{s1} = T_{s1} - T_\infty$$

x=L

$$\theta_{s2} = T_{s2} - T_\infty = C_1 \sinh(mL) + \theta_{s1} \cosh(mL)$$

$$C_1 = \frac{\theta_{s2} - \theta_{s1}\cosh(mL)}{\sinh(mL)}$$

$$\theta(x) = C_1 \sinh(mx) + C_2 \cosh(mx) = \frac{\theta_{s2} - \theta_{s1}\cosh(mL)}{\sinh(mL)} \sinh(mx) + \theta_{s1} \cosh(mx)$$

$$\frac{\partial\theta(x)}{\partial x} = C_1 m \cosh(mx) + C_2 \, \mathrm{m} \sinh(mx)$$

$$q = kA\frac{\partial\theta(x)}{\partial x}\bigg|_{x=L} - kA\frac{\partial\theta(x)}{\partial x}\bigg|_{x=0} = k(\pi r^2)m[C_1(\cosh(mL) - 1) + C_2 \sinh(mL)]$$

$$q = k(\pi r^2)m[C_1(\cosh(mL) - 1) + C_2 \sinh(mL)]$$

$$km(\pi r^2)C_1 m * (\cosh(ml) - 1)$$

$$q_{max} = hA(T_{s1} - T_\infty) = h(2\pi rL)\theta_{s1}$$

$$\eta = \frac{q}{q_{max}} = \frac{k(\pi r^2)C_1 \, \mathrm{m}(\cosh(mL) - 1)}{h(2\pi rL)\theta_{s1}}$$

Temperature at x=L/2

$$T(L/2) = T_\infty + \frac{\theta_{s2} - \theta_{s1}\cosh(mL)}{\sinh(mL)}\sinh(m/2) + \theta_{s1}\cosh(m/2)$$

PROBLEM

D	0.001000	m
r	0.000500	m
L	0.025400	m
k	400.000000	W/(mK)
Ts1	132.000000	degree C
Ts2	0.000000	degree c
T∞	0.000000	degree C
θs1	132.000000	degree K
θs2	0.000000	degree K
h	100.000000	W/(m2K)
m2	1000.000000	
m	31.6227766016838	
x=L/2	0.012700	m
C2	132.000000	
C1	-198.248129	
θ(x)	61.013107	degree K
T(x)	61.013107	degree C
qmax	1.053313	W
q	0.500057	W
□	0.474746	
N	625.000000	
q*N	312.535372	W

2.4 ONE DIMENSIONAL TRANSIENT (TIME DEPENDENT) LUMPED SOLUTION

Consider a solid body of any arbitrary shape (with a volume V and surface area A) and a constant temperature of T_0 (it is assumed that all the internal volume has the same temperature) is suddenly immersed at a time t=0 in a well stirred fluid with a uniform temperature of T_∞ and a convective heat transfer coefficient of h. The energy balance can be expressed as:

(rate of heat flow by convection into solid volume through the boundary surface A)=
(rate of internal energy increase of solid volume).

$$Q_{conv} = \dot{E}_{st}$$

$Ah[T_\infty - T(t)] = m\frac{dU(T(t))}{dt}$ where m is the mass of solid U is the internal energy. Considering that $dU(T) = C_p(T(t))dT$ and $m = \rho V$ where ρ is the density of solid and V is the volume of solid

$Ah[T_\infty - T(t)] = \rho V\frac{C_p(T(t))dT(t)\,]}{dt}$ if we futher assume that $C_p = constant$ equation becomes

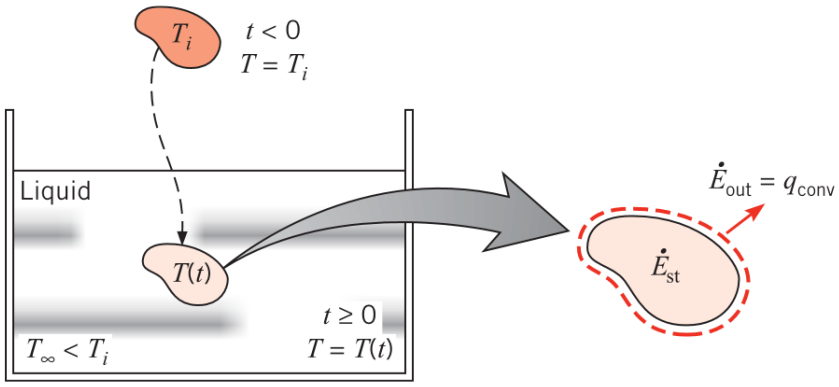
$Ah[T_\infty - T(t)] = \rho V C_p\frac{dT(t)}{dt}$ by arranging equation

$\frac{dT(t)}{dt} + \frac{Ah}{\rho V C_p}[T(t) - T_\infty] = 0$ defining $m = \frac{Ah}{\rho V C_p}$ and $\theta(t) = T(t) - T_\infty$

$\frac{d\theta(t)}{dt} + m\theta(t) = 0$ subject to initial conditions

$t = 0 \quad \theta(t) = T_0 - T_\infty = \theta_0$ solution of eqaution becomes

$\frac{\theta(t)}{\theta_0} = \frac{T(t)-T_\infty}{T_0-T_\infty} = e^{-mt}$



In this lumped solution, it is assumed that temperature of the solid remains constant. In a real solid temperature profile will exist due to dimensional effects. To ignore dimensional effects, it should be assumed that solid body is relatively small so dimensional temperature changes can be ignored. In order to measure this smallness assumption of an artificial test is applied. Charecteristic length: $L_s = \frac{V}{A}$ Biot number $Bi = \frac{hV}{kA} = \frac{hL_s}{k}$ It will be assumed that temperature variation inside of the solid is less than 5% if $Bi = \frac{hV}{kA} = \frac{hL_s}{k} < 0.1$ so that this solution can be used to evaluate time dependent temperature variation and heat transfer for the solid body.

PROBLEM: An aliminium plate [k=160 W/mK, ρ=2790 kg/m^3, C_p=880 J/kgK of thickness L=3x10^{-2} m and at a uniform temperature T_0=225 C is suddenly immersed at time t=0 seconds in a well stirrred fluid maintained at a constant temperature T_∞=25 C. The heat transfer coefficent between the plate and fluid is h=300 W/(m^{2}K). Determine the time required for the center of the plate reach 50 C.

k	1.6000000E+02	W/mK
r	2.7900000E+03	kg/m^3
Cp	8.8000000E+02	J/kgK
L	3.0000000E-02	m
T0	2.2500000E+02	degree C
Tinf	2.5000000E+01	degree C
h	3.0000000E+02	W/m^2K
T	5.0000000E+01	degree C
dT=(T-Tinf)/(T0-Tinf)	1.2500000E-01	
Ls	1.5000000E-02	
Bi	2.8125000E-02	<0.1 lumped
m	8.1459759E-03	
ln(dT)	-2.0794415E+00	
t	2.5527224E+02	s

PROBLEM: A solid copper sphere of 8x10^{-2} m diameter [ρ=8954 kg/m^3, C_p=383 J/(kgK), k=386 W/(mK)] initially at a uniform temperature Ti=200 ℃, is suddenly immersed in a well-stirred fluid which is maintained at a uniform temperature of $T_\infty = 50$℃. The heat transfer coefficint between the sphere and fluid is: h=200 W/m^{2}K

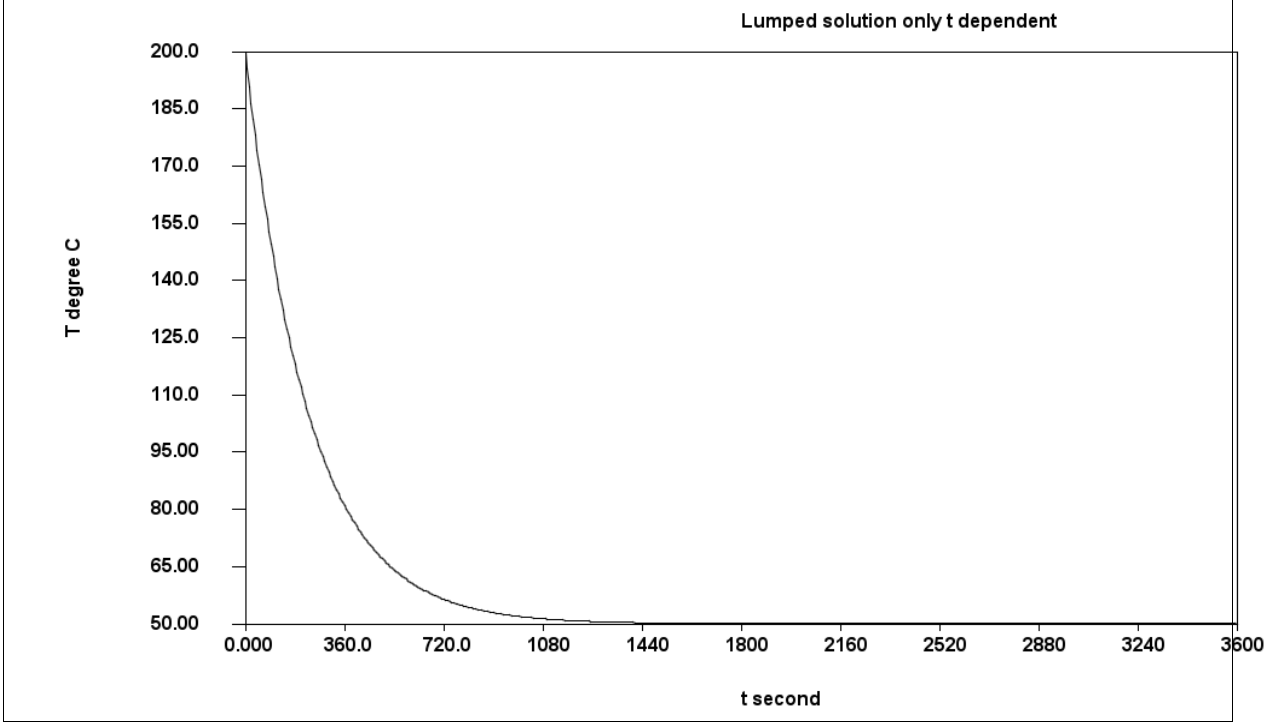
A) Check if lumped system analysis is suitable
B) Determine the temperature of copper block at t=300,600,1200 and 3600 seconds.

D	8.00E-02	m
r	4.00E-02	m
V	0.000268	m^3
A	2.01E-02	m^2
Ls	1.33E-02	
Ti	200	deg C
Tinf	50	deg C
h	200	W/(m^2K)
k	386	W/(mK)
ρ	8954	kg/m^3
Cp	383	J/(kgK)
Bi	6.91E-03	<0.1
m	0.004374	1/sec

t sec	T deg C	
300	90.38448	
600	60.87271	
1200	50.78811	
3600	50.00002	

```
public class HT__lumped_t
{ public static void main(String arg[])
  {   double D=8e-2;
      double r=D/2.0;
      double V=4.0/3.0*Math.PI*r*r*r;
      double A=4.0*Math.PI*r*r;
      double Ls=V/A;
      double Ti=200.0;  //degree C
      double Tinf=50.0; //degree C
      double h=200.0;  //W/(m^2K)
      double k=386.0;  //W/(mK)
      double ro=8954.0; //kg/m^3
      double Cp=383.0; //J/(kgK)
      double Bi=h*Ls/k;
      double m=A*h/(ro*Cp*V);
                if_x T=t->Tinf+(Ti-Tinf)*Math.exp(-m*t);
                double t[]={300,600,1200,3600};
                String s="";
                for(int i=0;i<t.length;i++)
                        {s+="t = "+t[i]+"T="+T.func(t[i])+" degree C\n";}
                System.out.println(s);
                Plot p=new Plot(T,0,3600.0);
                p.setPlabel("Lumped solution only t dependent");
                p.setXlabel("t second");
                p.setYlabel("T degree C");
                p.plot();
}
}
```

```
---------- Capture Output ----------
> "F:\co\java\bin\javaw.exe" HT__lumped_t
t = 300.0T=90.38447858192835 degree C
t = 600.0T=60.8727074022282 degree C
t = 1200.0T=50.78810510836312 degree C
t = 3600.0T=50.00002175554211 degree C
```



PROBLEM: An aliminium plate L= 0.1 m thick [ρ=2707 kg/m^3, C_p=896 J/(kgK), k=204 W/(mK)] initially at a uniform temperature Ti=200 ℃, is suddenly immersed in a well-stirred fluid which is

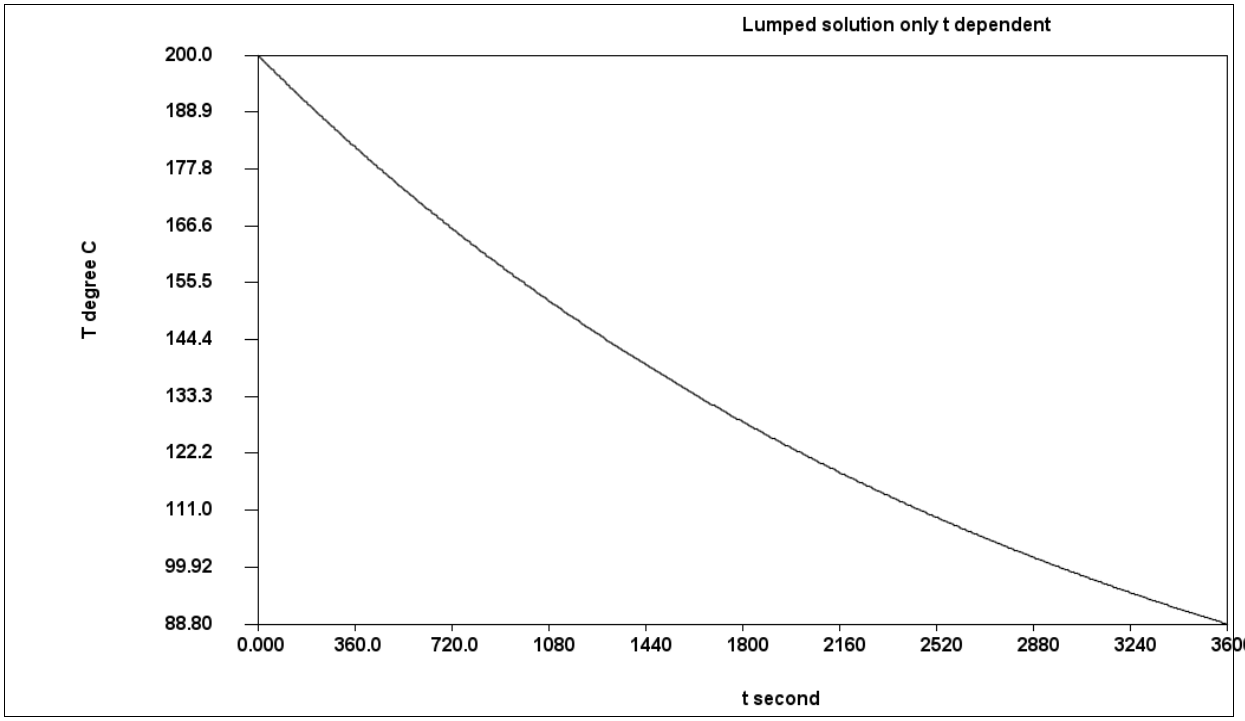
maintained at a uniform temperature of $T_\infty = 40℃$. The heat transfer coefficint between the spher and fluid is: h=80 W/m^{2}K

A) Check if lumped system analysis is suitable
B) Determine the temperature of copper block at t=5,10,30 and 60 minutes.

L	0.100000	m
Ls	0.100000	
Ti	200.000000	deg C
Tinf	40.000000	deg C
h	80.000000	W/(m^2K)
k	204.000000	W/(mK)
ρ	2707.000000	kg/m^3
Cp	896.000000	J/(kgK)
Bi	0.039216	<0.1
m	0.000330	1/sec
t sec	T deg C	
300	184.926107	
600	171.272352	
1800	128.364909	
3600	88.802233	

```
public class HT__lumped_t1
{ public static void main(String arg[])
  {   double Ls=0.1;
      double Ti=200.0;  //degree C
      double Tinf=40.0; //degree C
      double h=80.0;  //W/(m^2K)
      double k=204.0;  //W/(mK)
      double ro=2707.0; //kg/m^3
      double Cp=896.0; //J/(kgK)
      double Bi=h*Ls/k;
      double m=h/(ro*Cp*Ls);
            if_x T=t->Tinf+(Ti-Tinf)*Math.exp(-m*t);
            double t[]={5*60,10*60,30*60,60*60};
            String s="";
            for(int i=0;i<t.length;i++)
                  {s+="t = "+t[i]+"T="+T.func(t[i])+" degree C\n";}
            System.out.println(s);
            Plot p=new Plot(T,0,60*60);
            p.setPlabel("Lumped solution only t dependent");
            p.setXlabel("t second");
            p.setYlabel("T degree C");
            p.plot();
}
}
```

```
---------- Capture Output ----------
> "F:\co\java\bin\javaw.exe" HT__lumped_t1
t = 300.0T=184.9261065577593 degree C
t = 600.0T=171.27235226244377 degree C
t = 1800.0T=128.36490948441354 degree C
t = 3600.0T=88.80223267617875 degree C
```



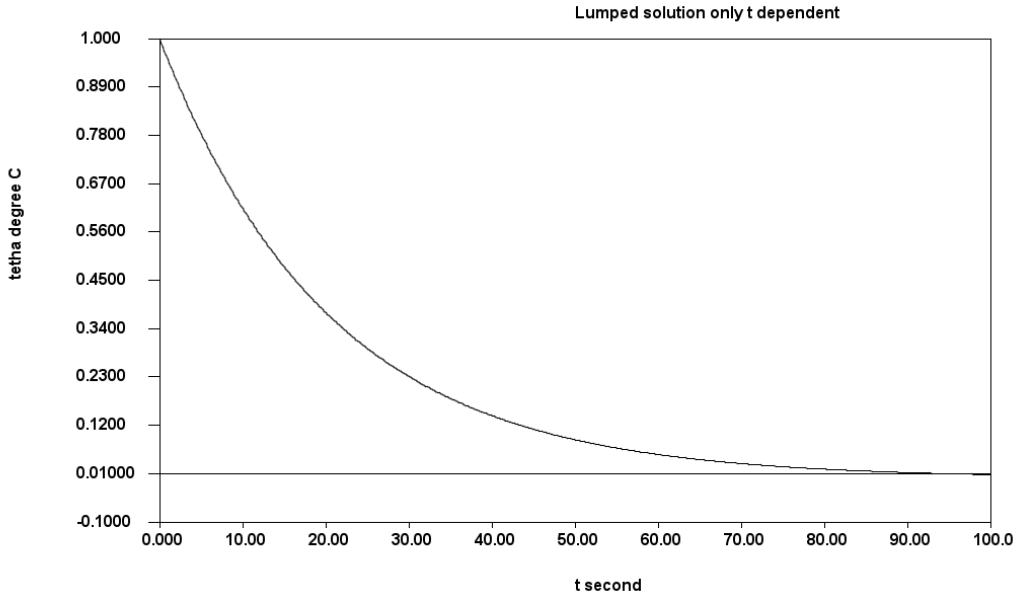
PROBLEM: The steel bar with diameter of D=0.05 m[ρ=7800 kg/m^3, C_p=500 J/(kgK), k=50 W/(mK)] is to be cooled from T_i=800 ℃ to T_f=100 ℃ in atmosphere with ambient temperature of $T_\infty = 30$ ℃. Thermal convectivity coefficient for ambient air is h=40 W(m^{2}K). Detemine the time requiremnts for the cooling process

L	1	m
D	5.00E-02	m
r	2.50E-02	
V	1.96E-03	
A	1.57E-01	m
Ls	1.25E-02	m
Ti	800	deg C
To	100	deg C
Tinf	30	deg C
h	40	W/(m^2K)
10k	50	W/(mK)
ρ	7800	kg/m^3
Cp	500	J/(kgK)
Bi	1.00E-02	<0.1
m	8.21E-04	1/sec
t sec	2922.435	sec
t min	4.87E+01	min

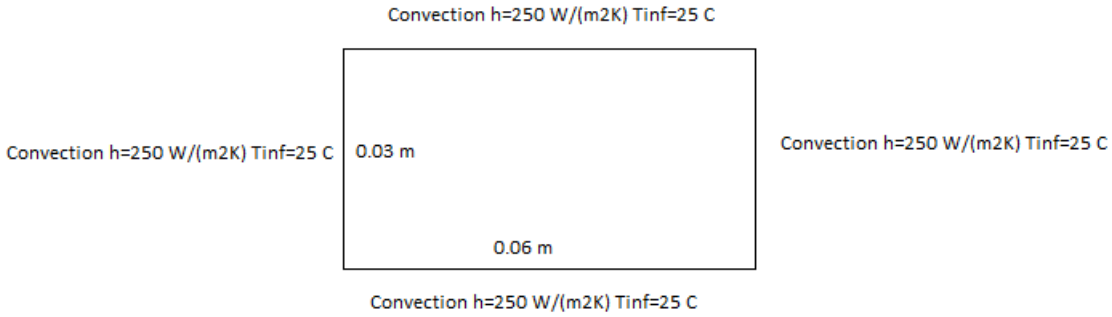
PROBLEM: PT1000 resistance thermometer reads temperature from side of it is tip about L=10^{-2} m long with a diameter of D=2x10^{-3}m . Temperature of an air stream to be measured with PT1000 resistant thermometer. Assuming sensing side of thermometer as a cylinder of L=10^{-2} m and D=2x10^{-3}m [k=43 W/(mK), ρ=7800 kg/m^3 , Cp=420 J/(kgK), and the heat transfer coefficient between sensing edge of thermometer and air is h=80 W/(m^{2}K) calculate how long it will take thermometer to record 99% of actual temperature

D	2.00E-03	
r	1.00E-03	
L	1.00E-02	
V	3.14E-08	
A	0.000063	m
Ls	0.000500	
teta=(t-tinf)/(ti-tinf)	**0.010000**	**deg C**
h	80.000000	W/(m^2K)
k	43.000000	W/(mK)
ρ	7800.000000	kg/m^3

Cp	420.000000	J/(kgK)
Bi	0.000930	<0.1
m	0.048840	1/sec
t sec	**94.290860**	**sec**



PROBLEM: A rectangular aliminium bar 0.06 m by 0.03 m [k=200 W/(mK), C_p=890 J/(kgK), ρ=2700 kg/m^3,α=8.4x10^{-5}m^2/s] is initially at a uniform temperature T_i= 200 ℃, suddenly all four surfaces of the road are subjected to convective cooling with a heat transfer coefficient h=250 W/(m^{2}K) into an anbient at $T_\infty = 27$ ℃. Determine the center temperature T_0 of the bar at t=120 s after the start of the cooling



$$Bi = \frac{hV}{kA} = \frac{hL_c}{k}$$

$$V = (0.06 * 0.03)z$$

$$A = (2 * 0.06 + 2 * 0.03)z = 6 * 0.03z$$

$$L_c = \frac{V}{A} = \frac{0.06 * 0.03}{6 * 0.03} = 0.01$$

$$Bi = \frac{hV}{kA} = \frac{250 * 0.01}{200} = 0.025 < 0.1$$

Lumped solution is valid

$$m = \frac{Ah}{\rho V C_p}$$

$$\frac{\theta(t)}{\theta_0} = \frac{T(t) - T_\infty}{T_0 - T_\infty} = e^{-mt}$$

k	2.00E+02	W/mK
ρ	2.70E+03	kg/m^3
Cp	8.90E+02	J/kgK
V	1.80E-03	m
A	1.80E-01	
T0	2.00E+02	degree C
Tinf	2.70E+01	degree C
h	2.50E+02	W/m^2K
Ls	1.00E-02	
Bi	1.25E-02	<0.1
m	1.04E-02	
t	2.55E+05	s
T0-Tinf	1.73E+02	degree K
T	2.70E+01	degree C

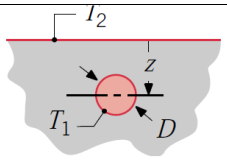
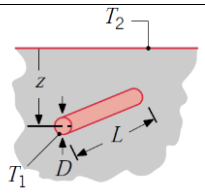
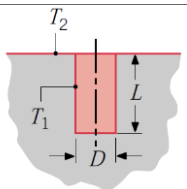
2.5 2-3 DIMENSIONAL STEADY STATE SHAPE FACTOR SOLUTIONS

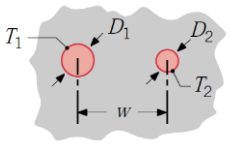
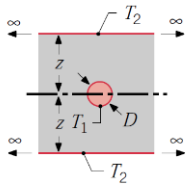
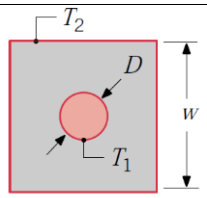
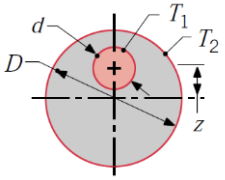
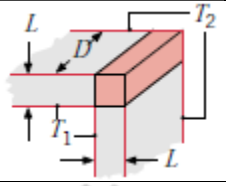
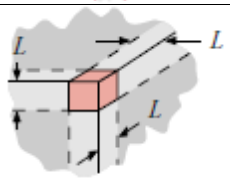
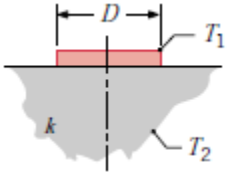
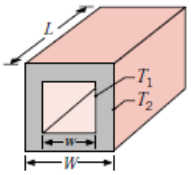
Two or three dimensional steady state problems will usually involve quite complex mathematical solutions. For this type of problems some simplified approximate solutions existed to calculate heat transfer. This approach is called shape factor. Shape factor equation is as follows:

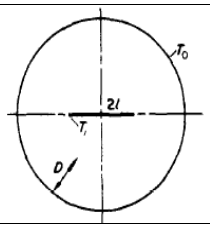
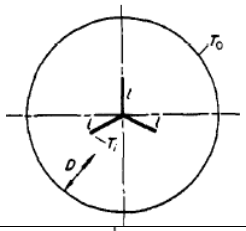
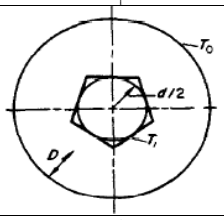
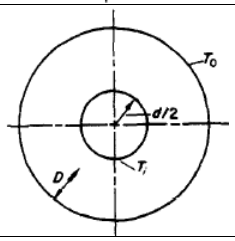
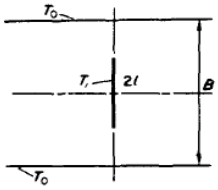
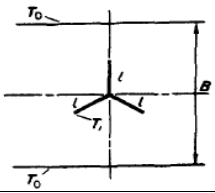
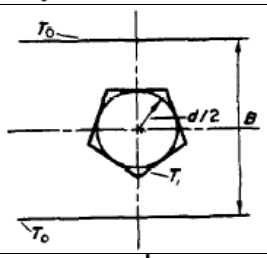
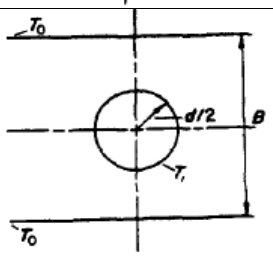
$q = Sk(T_1 - T_2) = \frac{(T_1 - T_2)}{\frac{1}{Sk}} = \frac{(T_1 - T_2)}{R}$

As it is seen from the equation, this is simply a convertion of the problem into one dimensional one. S is called shape factor, R is called two or three dimensional thermal resistance. In the table below, some of the geometries and related shape factors are defined

Incropera et al.[35], Lewis[81]

System	Schematic	Restrictions	Shape Factor, S
Case 1 Isothermal sphere buried in a semi-infinite medium		$z > D/2$	$\frac{2\pi D}{1 - D/4z}$
Case 2 Horizontal isothermal L D cylinder of length L buried in a semi-infinite medium		$L \gg D$	$\frac{2\pi L}{\cosh^{-1}(\frac{2z}{D})}$
		$L \gg D$ $z > D/2$	$\frac{2\pi L}{\ln(\frac{4z}{D})}$
Case 3 Vertical cylinder in a semi-infinite medium		$L \gg D$	$\frac{2\pi L}{\ln(\frac{4L}{D})}$

Case 4 Conduction between two cylinders of length L in L $D1$, $D2$ infinite medium		$L \gg D_1, D_2$ $L \gg w$	$\frac{2\pi L}{\cosh^{-1}\left(\frac{4w^2 - D_1^2 - D_2^2}{2D_1D_2}\right)}$
Case 5 Horizontal circular cylinder of length L midway between parallel planes of equal length and infinite width		$z \gg D/2$ $L \gg z$	$\frac{2\pi L}{\ln\left(\frac{8z}{\pi D}\right)}$
Case 6 Circular cylinder of length L centered in a square solid of equal length		$w > D$ $L \gg D$	$\frac{2\pi L}{\ln\left(\frac{1.08w}{D}\right)}$
Case 7 Eccentric circular cylinder of length L in a cylinder of equal length		$D > d$ $L \gg D$	$\frac{2\pi L}{\ln\left(\frac{1.08w}{D}\right)}$
Case 8 Conduction through the edge of adjoining walls		$D > 5L$	$\frac{2\pi L}{\ln\left(\frac{1.08w}{D}\right)}$
Case 9 Conduction through corner of three walls with a temperature L _ length and difference $T1_2$ across width of wall $0.15L$ the walls		$L \ll$length and width of wall	$\frac{2\pi L}{\ln\left(\frac{1.08w}{D}\right)}$
Case 10 Disk of diameter D and temperature $T1$ on a semi-infinite medium of thermal conductivity k and temperature $T2$			$\frac{2\pi L}{\ln\left(\frac{1.08w}{D}\right)}$
Case 11 Square channel of length L		$\frac{W}{w} < 1.4$	$\frac{2\pi L}{0.785 \ln\left(\frac{W}{w}\right)}$
		$\frac{W}{w} < 1.4$ $L \gg W$	$\frac{2\pi L}{0.93 \ln\left(\frac{W}{w}\right) - 0.05}$

Case 12 Single central slit			$\frac{2\pi}{\ln(\frac{D}{l})}$
Case 13 N equi-pitched radial slits		Values of A_1 form table below	$\frac{2\pi}{\ln(A_1\frac{D}{l})}$
Case 14 n-sided regular polygon hole		Values of A_2 form table below	$\frac{2\pi}{\ln(A_2\frac{D}{d})}$
Case 15 Central circular hole			$\frac{2\pi}{\ln\left(\frac{D}{d}\right)}$
Case 16 **Single central slit**			$\frac{2\pi}{\ln\left(\frac{4B}{\pi l}\right)}$
Case 17 **N equi-pitched radial slits**			$\frac{2\pi}{\ln\left(\frac{2^{(2+n)/n}B}{\pi l}\right)}$
Case 18			$\frac{2\pi}{\ln\left(\frac{4B}{\pi d}\left[\frac{2}{\sqrt{\left(\frac{n}{n-2}\right)+1}}\right]^{2/n}\right)}$
Case 19			$\frac{2\pi}{\ln\left(\frac{4B}{\pi d}\right)}$

n	N equi-pitched radial slits coefficients A_1 **Case 13**	Polygon hole coefficients A_2 **Case 14**

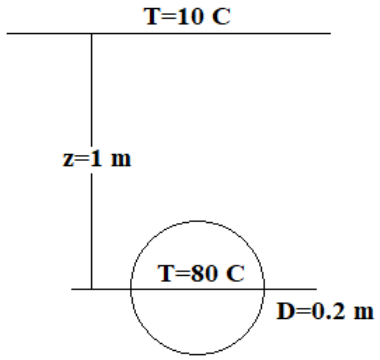
2	1	
3	0.79521	0.81226
4	0.70705	0.91018
5	0.66033	0.94711
6	0.62901	0.96512
7	0.61001	0.97527
8	0.59516	0.98155
9	0.58291	0.98570
10	0.57414	0.98860
∞	0.5	1

PROBLEM: An sphere of diameter D=0.2 m is buried underground. The sphere is maintained at a uniform temperature of T=80 C. The surface of the earth z=1 m above the sphere is 10 C. Calculate the heat transfer from the sphere. Thermal conductivity of earth k=1.2 W/mK

$$S = \frac{2\pi D}{1 - D/4z}$$

$$\boldsymbol{q = Sk\Delta T}$$

T1	80	°C
T2	10	°C
D	0.2	m
z	1	m
S	1.322776	
ΔT	70	°C
k	1.2	W/mK
Q	111.1132	W



PROBLEM: A pipe contains hot fluid of T_f=150 ℃ is burried underground.The distance from the center of the pipe to earth surface is H=1 m.. The pipe diameter is D=0.2 m, and it is covered with a dx=0.05 m thick insulation layer with thermal conductivity of k=0.04 W/(mK). The graound layer has a thermal conductivity of k=1.3 W/(mK). Ground surface temperature T=0 ℃. Determine heat loss per meter of pipe.

Heat loss across the insulation layer =
Heat loss from pipe to earth

$$Q = \frac{T_f - T_1}{R} = Sk(T_1 - T_2)$$

$$S = \frac{2\pi L}{\text{acosh}\left(\frac{z}{r0 + dx}\right)} = \frac{2\pi 1}{\text{acosh}\left(\frac{z}{r0 + dx}\right)}$$

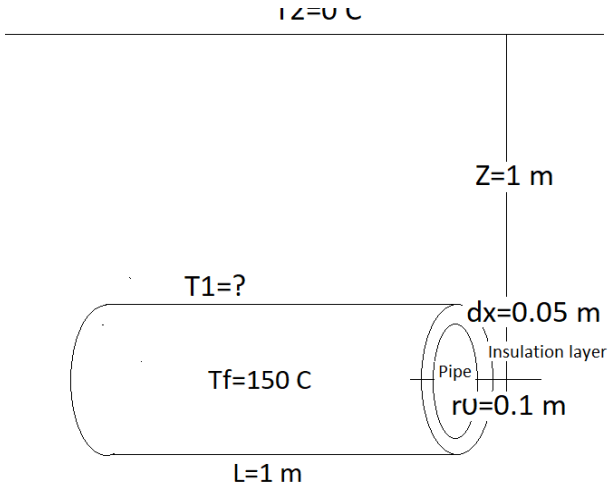
S=24.31

$$R = ln\left(\frac{r0 + dx}{2\pi Lk}\right) = \frac{ln\frac{0.15}{0.1}}{2\pi * 1 * 0.04} = 0.16133$$

$$\frac{150 - T_1}{0.16131} = 24.31 * 1.3 * (T_1 - 0)$$

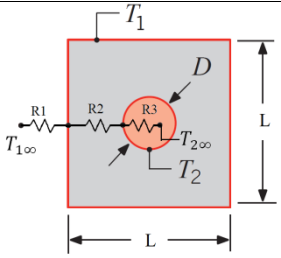
$$T_1 = 24.6$$

$$Q = 24.31 * 1.3 * 24.6 = 777.3\ W$$



PROBLEM: In the center of a solid block a hole is existed. The hole diameter D=0.2 m and L=1 m. The block has a thermal conductivity coefficient of k=0.1 W/mK. In outer surface atmospheric air is existed with temperature $T_{1\infty} = 20$ ℃ and thermal convectivity coefficient of h_1=10 W/(m^2K). Inside of the hole a high temperature fluid is flowing with temperature $T_{2\infty} = 300$ ℃ and thermal

convectivity coefficient of h_1=50 W/(m^2K). Determine the surface temperatures and heat loss per Z=20 meter of solid block.



$$Q = \frac{T_{2\infty} - T_{1\infty}}{R} = \frac{T_{2\infty} - T_{1\infty}}{R_1 + R_2 + R_2} = \frac{T_2 - T_1}{R_2} = Sk(T_2 - T_1)$$

$$S = \frac{2\pi Z}{\ln\left(\frac{1.08L}{D}\right)}$$

$T_{1\infty}$	20	°C
h_1	10	W/m²K
$T_{2\infty}$	300	°C
h_2	50	W/m²K
k	0.1	W/mK
D	0.2	m
L	1.2	m
Z	20	m
S	1632.822325	K/W
A_1	96	m²
A_2	12.56637061	m²
R_2=1/(Sk)	0.006124365	K/W
R_1=1/(h_1*A_1)	0.001041667	K/W
R_3=1/(h_2*A_2)	0.001591549	K/W
R	0.008757581	K/W
$Q = (T_{1\infty} - T_{2\infty})/R$	31972.29862	W
$T_1 = T_{1\infty} - QR_2$	53.30447773	°C
$T_2 = T_{2\infty} - QR_3$	249.1145063	°C

3.0 SEPARATION OF VARIABLE METHODS FOR HOMOGENEOUS AND NON-HOMOGENEOUS, STEADY STATE AND TIME DEPENDENT HEAT CONDUCTION

3.1 SEPARATION OF VARIABLES FOR UNIFORM MEDIUM

Conduction equation has four independent variable and one dependent variable for cartesian coordinate system $T(x, y, z, t)$. This type of partial differential equations are relatively hard to solve by using analytical solutions. One approach to solve these type of equations (assuming equation is homogeneous, i.e equation or boundary conditions does not have a non-homogeneous condition) is assuming that equation can be separated into variables with lesser independent variables, ideally only one variables each and original function (in this case $T(x, y, z, t)$) can be expressed as multiplication of this simplified functions.

Let us consider cartesian coordinate system heat conduction equation

$$\frac{\partial^2 T(x,y,z,t)}{\partial x^2} + \frac{\partial^2 T(x,y,z,t)}{\partial y^2} + \frac{\partial^2 T(x,y,z,t)}{\partial z^2} + \frac{g(x,y,z,t)}{k} = \frac{1}{\alpha}\frac{\partial T(x,y,z,t)}{\partial t}$$

Assume a separation of variables in the form:

$T(x, y, z, t) = \varphi(x, y, z)\Gamma(t)$ in this case above equation becomes:

$$\Gamma(t)\left[\frac{\partial^2 \varphi(x,y,z)}{\partial x^2} + \frac{\partial^2 \varphi(x,y,z)}{\partial y^2} + \frac{\partial^2 \varphi(x,y,z)}{\partial z^2}\right] = \varphi(x, y, z)\frac{1}{\alpha}\frac{\partial \Gamma(t)}{\partial t}$$

$$\frac{1}{\varphi(x,y,z)}\left[\frac{\partial^2 \varphi(x,y,z)}{\partial x^2} + \frac{\partial^2 \varphi(x,y,z)}{\partial y^2} + \frac{\partial^2 \varphi(x,y,z)}{\partial z^2}\right] = \frac{1}{\alpha\Gamma(t)}\frac{\partial \Gamma(t)}{\partial t} = -\lambda^2$$

Then separated functions becomes:

$$\frac{\partial^2 \varphi(x,y,z)}{\partial x^2} + \frac{\partial^2 \varphi(x,y,z)}{\partial y^2} + \frac{\partial^2 \varphi(x,y,z)}{\partial z^2} + \lambda^2 \varphi(x, y, z) = 0$$

$$\frac{\partial \Gamma(t)}{\partial t} + \alpha\lambda^2\Gamma(t) = 0$$

The other side can also be separated the similar way such as:

$$\varphi(x, y, z) = X(x)Y(y)Z(z)$$

Substitution of this equation into above differential equation yields:

$$Y(y)Z(z)\frac{\partial^2 X(x)}{\partial x^2}+X(x)Z(z)\frac{\partial^2 Y(y)}{\partial y^2}+X(x)Y(y)\frac{\partial^2 Z(z)}{\partial z^2}+\lambda^2 X(x)Y(y)Z(z)=0$$

$$\frac{1}{X(x)}\frac{\partial^2 X(x)}{\partial x^2}=-\beta^2 \qquad \frac{1}{Y(y)}\frac{\partial^2 Y(y)}{\partial y^2}=-\gamma^2 \quad \frac{1}{Z(z)}\frac{\partial^2 Z(z)}{\partial z^2}=-\eta^2$$

Where

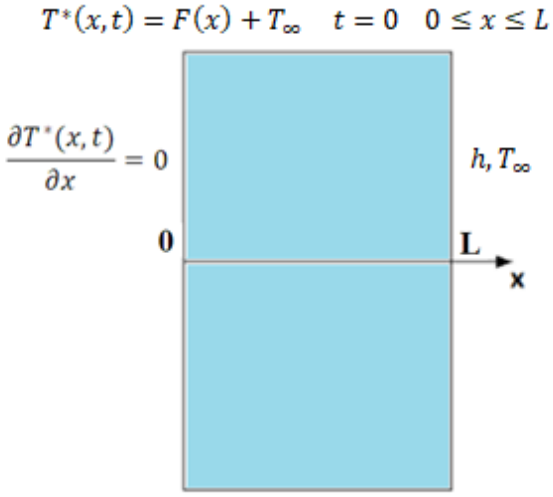
$\beta^2+\gamma^2+\eta^2=\lambda^2$

General solution of the equation becomes:

$$T(x,y,z,t)=[C_1\sin(\beta x)+C_2\cos(\beta x)][C_3\sin(\gamma y)+C_4\cos(\gamma y)][C_5\sin(\eta z)+C_6\cos(\eta z)]\left[\exp\left(-\alpha\left(\beta^2+\gamma^2+\eta^2\right)t\right)\right]$$

PROBLEM 3.1 SEPARATION OF VARIABLE HOMOGENEOUS SOLUTION, TIME DEPENDENT; CARTESIAN COORDINATES

In this problem one side of the wall (x=0) is insulated and other side is exposed to the convective heat transfer. At initial time, the wall has a temperature profile of F(x). It should be also note that by taking $T=T^*-T_\infty^*$ equation becomes homegeneous. [$q_{convection}=hT=h(T^*-T_\infty^*)$]



$\frac{\partial^2 T^*(x,t)}{\partial x^2}=\frac{1}{\alpha}\frac{\partial T^*(x,t)}{\partial t}$ Boundary conditions $\frac{\partial T^*(x,t)}{\partial x}=0$ x=0 t >0 $-k\frac{\partial T^*(x,t)}{\partial x}=h(T^*(x,t)-T_\infty)$ by arranging $\frac{\partial T^*(x,t)}{\partial x}+H(T^*(x,t)-T_\infty)$ x=L t>0 $H=\frac{h}{k}$ $T^*(x,t)=F(x)+T_\infty$ t=0 $0\le x\le L$	$T(x,t)=T^*(x,t)-T_\infty$ $\frac{\partial^2 T(x,t)}{\partial x^2}=\frac{1}{\alpha}\frac{\partial T(x,t)}{\partial t}$ Boundary conditions $\frac{\partial T(x,t)}{\partial x}=0$ x=0 t >0 $\frac{\partial T(x,t)}{\partial x}+HT(x,t)=0$ x=L t>0 $H=\frac{h}{k}$ $T(x,t)=F(x)$ t=0 $0\le x\le L$

Separation of variables:

$T(x,t)=X(x)\Gamma(t)$

$\Gamma(t)\frac{\partial^2 X(x)}{\partial x^2}=\frac{X(x)}{\alpha}\frac{\partial \Gamma(t)}{\partial t}$

$$\frac{1}{X(x)}\frac{\partial^2 X(x)}{\partial x^2}=\frac{1}{\alpha\Gamma(t)}\frac{\partial\Gamma(t)}{\partial t}=-\beta^2$$

$\frac{\partial^2 X(x)}{\partial x^2} + \beta^2 X(x) = 0$

$\frac{\partial \Gamma(t)}{\partial t} + \beta^2 \alpha \Gamma(t) = 0$

Boundary conditions

$\frac{\partial X(x)}{\partial x} = 0$ x=0 t >0

$\frac{\partial X(x)}{\partial x} + HX(x) = 0$ x=L t>0 $H = \frac{h}{k}$

$T(x,t) = F(x)$ t=0 $0 \le x \le L$

Solution set:

$X(x) = C_1 \cos(\beta x) + C_2 \sin(\beta x)$

$\Gamma(t) = C_3 \exp(-\beta^2 \alpha t)$

Application of boundary conditions:

$\frac{\partial X(x)}{\partial x} = -C_1 \beta \sin(\beta x) + C_2 \beta \cos(\beta x)$

$\frac{\partial X(x)}{\partial x} = 0$ x=0 t >0

$\frac{\partial X(x)}{\partial x} = -C_1 \beta \sin(\beta 0) + C_2 \beta \cos(\beta 0) = -C_1 * 0 + C_2 \beta * 1 = 0 \xrightarrow{yields} C_2 = 0$

$\frac{\partial X(x)}{\partial x} + HX(x) = 0$ x=L t>0

$-C_1 \beta \sin(\beta L) + C_1 H \cos(\beta L) = 0 \xrightarrow{because} C_1 \neq 0$

$\beta_m \sin(\beta_m L) - H \cos(\beta_m L) = 0$ or

$\beta_m \tan(\beta_m L) = H$

$\beta_m L \sin(\beta_m L) - HL \cos(\beta_m L) = 0$

$\beta_m \mathrm{L} \tan(\beta_m L) = HL$ $\xi_m = \beta_m \mathrm{L}$ $Bi = HL = \frac{hL}{k} = Biot\ number$

$\xi_m \sin(\xi_m) + Bi \cos(\xi_m) = 0$ or $\xi_m \tan(\xi_m) = Bi$

These equations has more than one solution. It can be solved only by numerical root finding methods. If $\xi_m \tan(\xi_m) = Bi$ form is taken into account, a caution should be recmmended due to fact that $\tan(\xi_n)$ is not a continious function. Temperature profile becomes.

$$T^*(x,t) - T_\infty = T(x,t) = \sum_{m=1}^{\infty} C_m \cos(\beta_m x) \exp(-\beta_m^2 \alpha t)$$

We still do not know the C_n coefficients we will use the remaining boundary conditions to find it.

$T(x,t) = F(x)$ t=0 $0 \le x \le L$

$$F(x) = \sum_{m=1}^{\infty} C_m \cos(\beta_n x)$$

This equation has n coefficients. In order to solve these coefficients, we will use properties of orthogonal functions.

$$\int_0^L X(\beta_n x) X(\beta_m x) dx = \begin{cases} 0 & m \neq n \\ N(\beta_m) & m = n \end{cases}$$

$$\int_0^L F(x) \cos(\beta_m x) = C_m \int_0^L \cos^2(\beta_m x) dx$$

$$N(\beta_m) = \int_0^L \cos^2(\beta_m x) dx = \int_0^L \frac{1 + \cos(2\beta_m x)}{2} dx = \left[\frac{x}{2} + \frac{\sin(2\beta_m x)}{4\beta_m}\right]_{x=0}^{L} = \frac{L}{2} + \frac{\sin(2\beta_m L)}{4\beta_m}$$

To get an alternative formula for $N(\beta_m)$ eigenvalue function

$\beta_m \sin(\beta_n L) = H \cos(\beta_m L)$ can be utilised

$\sin(\beta_m L) = \frac{H}{\beta_m}\cos(\beta_m L)$ $\sin(2\beta_m L) = 2\sin(\beta_m L)\cos(\beta_m L)$ $\sin(2\beta_m L) = 2\frac{H}{\beta_m}\cos^2(\beta_m L)$

On the other hand taking square of eigenvalue function:

$\sin^2(\beta_m L) = \left[\frac{H}{\beta_m}\right]^2 \cos^2(\beta_m L)$ $\sin^2(\beta_m L) + \cos^2(\beta_m L) = 1$ $\sin^2(\beta_m L) = 1 - \cos^2(\beta_m L)$

$1 - \cos^2(\beta_m L) = \frac{H^2}{\beta_m^2}\cos^2(\beta_m L)$ $1 = \left[\frac{H^2}{\beta_m^2} + 1\right]\cos^2(\beta_m L)$ $\cos^2(\beta_m L) = \frac{\beta_m^2}{H^2+\beta_m^2}$

Substituting into previous equation

$$\sin(2\beta_m L) = 2\frac{H}{\beta_m}\cos^2(\beta_m L) = 2\frac{H}{\beta_m}\frac{\beta_m^2}{H^2+\beta_m^2}$$

$$N(\beta_m) = \int_0^L cos^2(\beta_m x)dx = \frac{L}{2} + \frac{\sin(2\beta_m L)}{4\beta_m} = \frac{L}{2} + \frac{2\frac{H}{\beta_m}\frac{\beta_m^2}{H^2+\beta_m^2}}{4\beta_m} = \frac{L(\beta_m^2+H^2)+H}{2(\beta_m^2+H^2)}$$

$$C_m = \frac{1}{N(\beta_m)}\int_0^L F(x)\cos(\beta_m x)dx$$

Depends on value of F(x), this integration can be solved by using numerical or analytical methods.

$T(x,t) = T^*(x,t) - T_\infty^* = \sum_{m=1}^{\infty} C_m \cos(\beta_m x)\exp(-\beta_m^2 \alpha t) = \sum_{m=1}^{\infty}\frac{1}{N(\beta_m)}\left[\int_0^L F(x)\cos(\beta_m x)dx\right]\cos(\beta_m x)\exp(-\beta_m^2\alpha t)$

PROBLEM 3.1 MATLAB SOLUTION

```
function  [T] = T1(x,t,L,H,alpha)
        T=0;
        for m=1:20
        beta=beta1(m,L,H);
        T=T+CN1(beta,L,H)*cos(beta*x)*exp(-beta*beta*alpha*t);
        end
```

```
function [C] = CN1(beta,L,H)
        y=@(x) F1(x).*cos(beta*x);
        %matlab integration formula is used
        C1=quad(y,0,L);
      N=(L*(beta*beta+H*H)+H)/(2*(beta*beta+H*H));
        C=C1/N;
end
```

```
 function [beta] = beta1(m,L,H)
        PI=4.0*atan(1.0);
        x1=((m-1)*PI+0.0000001)/L;
        x2= x1+(PI/2.0-0.0000001)/L;
        beta=bisection(@(x) func1(x,L,H),x1,x2);
```

```
function xr=bisection(f,xl,xu)
% bisection root finding method
maxit=100;
iter=0;
es=0.0000001;
ea=1.1*es;
while((ea>es)&&(iter<maxit))
   xr=(xl+xu)/2.0;
   iter=iter+1;
   if xr~=0 ea=abs((xu-xl)/(xu+xl))*100;end
   fxl= f(xl);
   fxr= f(xr);
   test= fxl*fxr;
   if test == 0.0      ea=0;
   elseif test < 0.0  xu=xr;
   else               xl=xr;
   end
end
if(iter>=maxit) fprintf('Maximum number of iteration is exceeded result might not be valid',"MAKSİMUM NUMBER OF ITERATION
WARNING');
end
end
```

```
function [F] = F1(x)
```

```
        F=1.0;
end
```

```
function [f] = func1(beta,L,H)
        f=(beta*L)*tan(beta*L)-H*L;
end
```

```
function [N] = N1(m,L,H)
        beta=beta1(m,L,H);
        N=(L*(beta*beta+H*H)+H)/(2*(beta*beta+H*H));
end
```

PROBLEM 3.1 JAVA SOLUTION

In order to simplify java codes a root class (interface created) to define functions. Some basic numerical processes such as derivatives, integral and root finding processes are embedded in this class so that it should not be repeated in every program. Interface if_x.java is listed first, and in the Wall problem this class is utilised as the root of all functions (defined as Lambda variable functions)

```
import static java.lang.Math.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import static java.lang.Math.*;
  // single function single independent variable
  // example f=x*x
  // includes full set of derivatives
  // Reference :"Generation of Finite Difference Formulas on Arbitrary Spaced Grids",
  // Bength Fornberg, Mathematics of Computation, Volume 51, Number 184, October 1988
  // pages 699-706
  //interface (empty function) version
@FunctionalInterface
interface if_x
{  public double func(double x);
//first order derivative
default double dfunc(double x)
{double h=1.0e-3;
 int n=1;
 int M=10;
 return dfunc(x,n,M,h);
}
//second order derivative
default double dfunc2(double x)
{double h=1.0e-3;
 int n=2;
 int M=10;
 return dfunc(x,n,M,h);
}
default double dfunc(double x,int N,int Mi,double hi)
{// order of the maximum derivative
 // N order of derivative
 // M degree of difference formula
double c[][][];
double alpha[];
double h;
int M;
double a[]=new double[0];
h=0.01;
double x0=0;
M=20;
double alphai[]={0,1,-1,2,-2,3,-3,4,-4,5,-5,6,-6,7,-7,8,-8,9,-9,10,-10,11,-11,12,-12,13,-13,14,-14,15,
-15,16,-16,17,-17,18,-18,19,-19,20,-20,21,-21,22,-22,23,-23,24,-24,25,-25,26,-26,27,-27,28,-28,29,-29,30,-30,
31,-31,32,-32,33,-33,34,-34,35,-35,36,-36,37,-37,38,-38,39,-39,40,-40,41,-41,42,-42,43,-43,44,-44,45,-45,46,
-46,47,-47,48,-48,49,-49,50,-50,51,-51,52,-52,53,-53,54,-54,55,-55,56,-56,57,-57,58,-58,59,-59,60,-60,
-61,61,-62,62,-63,63,-64,64,-65,65,-66,66,-67,67,-68,68,-70,70,-71,71,-72,72,-73,73,-74,74,-75,75,
-76,76,-77,77,-78,78,-79,79,-80,80,-81,81,-82,82,-83,83,-84,84,-85,85,-86,86,-87,87,
-88,88,-89,89,-90,90,-91,91,-92,92,-93,93,-94,94,-95,95,-96,96,-97,97,-98,98,-99,99,-100,100};
alpha=alphai;
int N1=alpha.length-1;
// M   degree of highest derivative
// N+1 number of coefficients
double delta[][][]=new double[N1+1][N1+1][M+1];
double c1,c2,c3;
delta[0][0][0]=1.0;
c1=1.0;
```

```
for(int n=1;n<=N1;n++)
{  c2=1;
   for(int nu=0;nu<=(n-1);nu++)
   {c3=alpha[n]-alpha[nu];
    c2=c2*c3;
    if(n<=M) delta[n-1][nu][n]=0.0;
    for(int m=0;m<=Math.min(n,M);m++)
    {
       if(m==0)
       { delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m])/c3;}
       else
       {delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m]-m*delta[n-1][nu][m-1])/c3;}
    }//next m
   }//next nu
   for(int m=0;m<=Math.min(n,M);m++)
   {   if(m==0)
                {delta[n][n][m]=c1/c2*(-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
                else
                {delta[n][n][m]=c1/c2*(m*delta[n-1][n-1][m-1]-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
   }//next m
   c1=c2;
}//next n
c=delta;
  if(Mi<N) M=N;
  else M=Mi;
  h=hi;
  double deriv=0;
  double h1=1/h;
  double h2=1;
  for(int j=0;j<N;j++)
  {h2*=h1;}
  for(int i=0;i<c[0].length;i++)
  {   deriv+=c[M][i][N]*func(x+alpha[i]*h);}
  return deriv*h2;}
default double dfunc(double x,int N)
{int M=30;double h=0.05*N;return dfunc(x,N,M,h);}
default double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{
              //calculates legendre gauss-coefficients as coefficients of the integral
              //for n terms
              double EPS=3.0e-15;
              int m,j,i;
              double z1,z,xm,xl,pp,p3,p2,p1;
    //double x[]=new double[n];
    //double w[]=new double[n];
    double a[][]=new double[2][n];//a[0][i]=x[i]   a[1][i]=w[i]
              m=(n+1)/2;
              xm=0.5*(x2+x1);
              xl=0.5*(x2-x1);
              for (i=1;i<=m;i++)
              { z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                        do {
                                  p1=1.0;
                                  p2=0.0;
                                  for (j=1;j<=n;j++) {
                                            p3=p2;
                                            p2=p1;
                                            p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                                  }
                                  pp=n*(z*p1-p2)/(z*z-1.0);
                                  z1=z;
                                  z=z1-p1/pp;
                        } while (Math.abs(z-z1) > EPS);
                        a[0][i-1]=xm-xl*z;
                        a[0][n-i]=xm+xl*z;
                        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                        a[1][n-i]=a[1][i-1];
              }
return a;
}

default double integral(double x1,double x2,int n)
{
//n : number of integral coefficients
// this routine first generates gauss legendre coefficients
// for [x1,x2] band
// then calculates gauss legendre integral
```

```
double a[][]=new double[2][n];
a=gauss_legendre_coefficients(x1,x2,n);
double z=0;
for(int i=0;i<n;i++)
{ z+=a[1][i]*func(a[0][i]);
}
return z;
}
default double bisection(double a,double b)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   double eps=1.0e-6;
   int nmax=100;
   int i=1;
   while(Math.abs(func(r))>eps && i<nmax)
   {if(func(a)*func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;;
    i++;
   }
   if(i>=nmax) r=bisection(a,b1);
   return r;
  }
default double secant(double x)
{ double eps=1.0e-10;
  double y=1.0;
  double dy=0.0;
  int miter=100;
  int i=0;
  while(abs(y)>eps && i<miter)
  {y=func(x);
   dy=dfunc(x);
   x=x-y/dy;
   i++;
  }
  if(i>=miter)
    System.out.println("i="+i+"results may not be valid");
  return x;
}
default double illinois(double xa,double xu)
{
//Modified False position root finding: Illinois method
double test;
double p=0;
double es,ea;
double fa,fp,fu;
int maxit=100,iter=0;
es=0.000001;
ea=1.1*es;
double xold;
int ia=0,iu=0;
fa=func(xa);
fu=func(xu);
p=xu;
double xpold;
while(iter<maxit && ea>es)
{
xpold=p;
p=xu-fu*(xa-xu)/(fa-fu);
fp=func(p);
if(signum(fp)!=signum(fu))
{xa=xu;fa=fu;}
else
{fa=fa/2;}
xu=p;fu=fp;
iter++;
if(p!=0) ea=abs((p-xpold)/p)*100.0;
}
if(iter>=maxit) System.out.println("Maximum number of iteration is exceeded ");
return p;
}
default double newton_bisection(double x1, double x2)
{
double xacc=1.0e-10; //accuracy
double h=0.0001; //derivative function step
int MAXIT=100;  //Maximum number of iterations
int j;
```

```
double dfr,dx,dxold,fr,fh,fl;
double temp,b,a,r;
fl=func(x1);
fh=func(x2);
//Check if a root is existed in the given region
if ((fl > 0.0 && fh > 0.0) || (fl < 0.0 && fh < 0.0))
System.out.println("There are no root in the given region or double root?");
if (fl == 0.0) return x1;
if (fh == 0.0) return x2;
if (fl < 0.0)   { a=x1;b=x2;}
else           { b=x1;a=x2;}
r=0.5*(x1+x2); //midpoint value
dxold=Math.abs(x2-x1);
dx=dxold;
fr=func(r);   //function value at midpoint
dfr=dfunc(r); //derivative of function value at midpoint
for (j=1;j<=MAXIT;j++)
{
if ((((r-b)*dfr-fr)*((r-a)*dfr-fr) > 0.0) || (Math.abs(2.0*fr) > Math.abs(dxold*dfr)))
   { //bisection step
                dxold=dx;
     dx=0.5*(b-a);
     r=a+dx;
     if (a == r) return r; //çözüm!!!!
   }
else{ //Newton-Raphson (Secant) step
                dxold=dx;
     dx=fr/dfr;
     temp=r;
     r -= dx;
     if (temp == r) return r;  //solution!!!!!
   }
if (Math.abs(dx) < xacc) return r; //solution!!!!!
fr=func(r);
dfr=dfunc(r);
if (fr < 0.0)
a=r;
else
b=r;
}
System.out.println("Maximum number of iterations are exceeded");
return 0.0; //program should never reach here
          //dummy return
}
default double brent(double a,double b)
{
double test;
double p=0;
double es,ea;
double f1,f2,f3,fp;
int maxit=500,iter=0;
double tol=1.0e-15;
es=0.0000001;
double x1=a;f1=func(x1);
double x2=b;f2=func(x2);
double x3=(x1+x2)/2.0;f3=func(x3);
if(f1==0) return x1;
else if(f2==0) return x2;
else if(f3==0) return x3;
if(f1*f2>0) System.out.println("No root is existed in the given region");
p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
fp=func(p);
ea=Math.abs(x3-p);
while((ea>es)&&(iter<maxit))
{             if(Math.abs(f3)<tol) return x3;
   if((p<x1) && (p>x2))
   {p=(x1+x2)/2.0;
    if(p>x3) {x1=x3;f1=f3;x3=p;f3=fp;}
    else if(p<x3) {x2=x3;f2=f3;x3=p;f3=fp;}
   }
   else
   {
   if(p>x3) {x1=x3;f1=f3;x3=p;f3=fp;}
   else if(p<x3) {x2=x3;f2=f3;x3=p;f3=fp;}
   p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
   fp=func(p);
   ea=Math.abs(x3-p);
```

```
    }
    iter++;
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Warning : MAximum number of iteration is exceeded \n"+
   " result may not be valid","MAXIMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return p;
}
default double enlarge(double x0,double dx)
{  //enlarge region untill a root is existed
                double x1=x0;
                double x2=x1+dx;
                int NTRY=200;
                double a[]=new double[2];
                double FACTOR=1.001;
                int j;
                double f1,f2;
                if (x1 == x2) System.out.println("Giriş değerleri yanlış verildi");
                f1=func(x1);
                f2=func(x2);
                for (j=1;j<=NTRY;j++)
                { if (f1*f2 < 0.0) {break;}
                  else {x2+=dx;f2=func(x2);}
                }
                return x2;
}
}
```

```
import java.io.*;
import javax.swing.*;
import java.awt.*;
import static java.lang.Math.*;
public class HT_plane_wall
{
public if_x F;
public double h,L,k,alpha,Bi,Ti,Tinf,H;
public  HT_plane_wall(double hi,double Li,double ki,double alphai,double Tii,double Tinfi)
{h=hi;k=ki;L=Li;alpha=alphai;Bi=h*L/k;Ti=Tii;Tinf=Tinfi;H=h/k;
 //initial temperature profile is defined as constant change
 //if you desired to do so
 F=xx->Ti-Tinf;
}

  public static double ksi(int n,double Bi)
  {double x1=n*PI+0.0000001;
   double x2=x1+(PI/2-0.000001);
   if_x f1=x->x*tan(x)-Bi;
   double ksi=f1.bisection(x1,x2);
   return ksi;
  }
  public double teta(double x,double t)
  {double ksi=0;
   double beta=0;
   double C=0;
   double N=0,N1=0,C1=0,C2=0;
   double teta=0;
   //coefficient of equation
   for(int n=0;n<20;n++)
   {ksi=ksi(n,Bi);
    beta=ksi/L;
    final double beta1=beta;
    N=L/2.0+sin(2*ksi)/(4.0*beta);
    N1=(L*(beta*beta+H*H)+H)/(2*(beta*beta+H*H));
    if_x Cn=xx->F.func(xx)*cos(beta1*xx);
    C=1.0/N1*Cn.integral(0,L,20);
    teta+=C*cos(beta*x)*exp(-beta*beta*alpha*t);
    }
    return teta;
  }
  public double dteta(double x,double t)
  {double ksi=0;
   double beta=0;
   double C=0;
   double N=0,N1=0,C1=0,C2=0;
   double dteta=0;
   double x1,x2,x3;
```

```
  //coefficient of equation
  for(int n=0;n<20;n++)
  {ksi=ksi(n,Bi);
   beta=ksi/L;
   final double beta1=beta;
   x1=tan(ksi);
   x2=(H/beta);
   x3=Bi/ksi;
   N=L/2.0+sin(2*ksi)/(4.0*beta);
   N1=(L*(beta*beta+H*H)+H)/(2*(beta*beta+H*H));
   if_x Cn=xx->F.func(xx)*cos(beta1*xx);
   C=1.0/N*Cn.integral(0,L,20);
   dteta+=-C*beta*sin(beta*x)*exp(-beta*beta*alpha*t);
   }
   return dteta;
  }
  public double T(double x,double t)
  {double teta=teta(x,t);
   double T=teta+Tinf;
   return T;
  }
  public double q(double x,double t)
  {double q=-k*dteta(x,t);
   return q;
  }
}
```

When a numerical code is given it will be a good idea to check the code outputs if possible. The first three eigenvalues of this solution is listed in "Fundamentals of Heat and Mass Transfer" by Incropera, Dewitt, Bergman, Lavine as:

$Bi = \frac{hL}{k}$	ξ_1	ξ_2	ξ_3	ξ_4
0	0	3.1416	6.2832	9.4248
0.001	0.0316	3.1419	6.2833	9.4249
0.002	0.0447	3.1422	6.2835	9.4250
0.004	0.0632	3.1429	6.2838	9.4252
0.006	0.0774	3.1435	6.2841	9.4254
0.008	0.0893	3.1441	6.2845	9.4256
0.01	0.0998	3.1448	6.2848	9.4258
0.02	0.1410	3.1479	6.2864	9.4269
0.04	0.1987	3.1543	6.2895	9.4290
0.06	0.2425	3.1606	6.2927	9.4311
0.08	0.2791	3.1668	6.2959	9.4333
0.1	0.3111	3.1731	6.2991	9.4354
0.2	0.4328	3.2039	6.3148	9.4459
0.3	0.5218	3.2341	6.3305	9.4565
0.4	0.5932	3.2636	6.3461	9.4670
0.5	0.6533	3.2923	6.3616	9.4775
0.6	0.7051	3.3204	6.3770	9.4879
0.7	0.7506	3.3477	6.3923	9.4983
0.8	0.7910	3.3744	6.4074	9.5087
0.9	0.8274	3.4003	6.4224	9.5190
1.0	0.8603	3.4256	6.4373	9.5293
1.5	0.9882	3.5422	6.5097	9.5801
2.0	1.0769	3.6436	6.5783	9.6296
3.0	1.1925	3.8088	6.7040	9.7240
4.0	1.2646	3.9352	6.8140	9.8119
5.0	1.3138	4.0336	6.9096	9.8928
6.0	1.3496	4.1116	6.9924	9.9667
7.0	1.3766	4.1746	7.0640	10.0339
8.0	1.3978	4.2264	7.1263	10.0949
9.0	1.4149	4.2694	7.1806	10.1502
10.0	1.4289	4.3058	7.2281	10.2003
15.0	1.4729	4.4255	7.3959	10.3898
20.0	1.4961	4.4915	7.4954	10.5117
30.0	1.5202	4.5615	7.6057	10.6543
40.0	1.5325	4.5979	7.6647	10.7334
50.0	1.5400	4.6202	7.7012	10.7832
60.0	1.5451	4.6353	7.7259	10.8172
80.0	1.5514	4.6543	7.7573	10.8606
100.0	1.5552	4.6658	7.7764	10.8871
∞	1.5708	4.7124	7.8540	10.9956

Now let us check the eigenvalues by using our own root finding routine. The actual root finding is located at if_x class (bisection) . Eigenvalues are calculated at HT_plane_wall class, public static double ksi(int n,double Bi) method.

Java program results for eigenvalues:

```
 public class Bi_eps
{public static void main(String arg[])
   {
     double
Bi[]={0.001,0.002,0.004,0.006,0.008,0.01,0.02,0.04,0.06,0.08,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.5,2,3,4,5,6,7,8,9,10,15,20,30,40,50,60,8
0,100,1000};
     int n=Bi.length;
     int n1=8;
     double ksi[][]=new double[n][n1];
     String
s[]={"Bi=hl/k",""+'\u03BE'+"1",""+'\u03BE'+"2",""+'\u03BE'+"3",""+'\u03BE'+"4",""+'\u03BE'+"5",""+'\u03BE'+"6",""+'\u03BE'+"7"};
     double A1[]=new double[n1];
     for(int i=0;i<n;i++)
     { ksi[i][0]=Bi[i];
       for(int j=0;j<n1-1;j++){ksi[i][j+1]=HT_plane_wall.ksi(j,Bi[i]);}
     }
     Text.print(ksi,s,"eigenvalues");
}}
```

eigenvalues

Bi=hl/k	ξ1	ξ2	ξ3	ξ4	ξ5	ξ6	
0.001	0.03161446517398786	3.141911084361186	6.283344572565282	9.424884046190915	12.566450203425324	15.708026940476545	18
0.002	0.04470123043933842	3.1422294151325785	6.28350373795098	9.424990218865844	12.566529692491475	15.708090606630824	18
0.004	0.0632048619134238	3.1428653276617844	6.2838218814689775	9.425202376962304	12.566688857877168	15.708217938939383	18
0.006	0.07738219095088689	3.1435012401909903	6.284140024986975	9.42541453505877	12.566848023262864	15.708345177762124	18
0.008	0.0893184713577451	3.1441371527201953	6.2844583557583675	9.425626693155234	12.567007188648564	15.708472509929798	18
0.01	0.0998346220174018	3.144772316235822	6.28477631202297	9.425838851251694	12.56716635403426	15.708599842238357	18
0.02	0.14095247149122755	3.147945886773212	6.286366842359566	9.42689964173401	12.567961993709343	15.709236410154444	18
0.04	0.19867445400089873	3.15427317898813	6.289544906978435	9.429020099178278	12.569552898552729	15.710509358733225	18
0.06	0.24252470501364384	3.1605742557277576	6.292719975542983	9.431139807608968	12.571143428889325	15.711782026431912	18
0.08	0.2791275006287509	3.1668487424853033	6.295891299039635	9.433258392519281	12.572733491092434	15.713054506877208	18
0.1	0.31105345545191376	3.1730970137675585	6.2990592519751765	9.435376041162623	12.574323178788749	15.71432680006911	18
0.2	0.43284006746455905	3.203935026366138	6.314846211195519	9.445947806080575	12.582264595268022	15.7206846145874	18
0.3	0.5217914221682278	3.234089938579987	6.330539169211603	9.456491482989286	12.590194214783413	15.727036436997054	18
0.4	0.5932413255675135	3.263550140698621	6.3461325084215785	9.467004824848022	12.59811072656116	15.733381424657795	18
0.5	0.6532710189163873	3.29231001512019	6.36162042397020 5	9.477485771869429	12.606013381587683	15.739719296689522	18
0.6	0.7050653079559356	3.320366565790377	6.376997298255633	9.487932077012776	12.613901430849403	15.746049584958762	18
0.7	0.7505581457490673	3.3477205417227602	6.392258262689596	9.498342054997504	12.62177356357255	15.75237182133202	18
0.8	0.7910333415714695	3.374375687985239	6.407398074177035	9.50871364603627	12.629629217996943	15.758685350422406	18
0.9	0.8274009469225359	3.400338745700031	6.422412800396655	9.519044977595122	12.637467457855607	15.764989891349842	18
1.0	0.8603335759968795	3.4256187030300937	6.4372983217737705	9.529334364393511	12.645287159628172	15.771284882354133	18
1.5	0.9882407524764324	3.542166339559194	6.5096590256860285	9.580091644131901	12.684082317991205	15.802600859243292	18
2.0	1.0768739636621034	3.6435970095150374	6.57833383377235	9.629560246006331	12.722298769737177	15.833611425845303	18
3.0	1.1924587423244387	3.808762181238582	6.703955769820094	9.724027523964654	12.796648385583758	15.894513064627699	19
4.0	1.264591559765059	3.9351615933821034	6.814010394879755	9.811878203212885	12.867755708897006	15.953625778101568	19
5.0	1.313837704606306	4.033567746502927	6.909595762859249	9.892752569979343	12.93522132818552	16.010658574491167	19
6.0	1.3495528255027147	4.111617770313552	6.99235183899143	9.966671409390873	12.998813142866279	16.065394988871972	19
7.0	1.3766150606457863	4.174637806752871	7.064030753292723	10.033914290766113	13.058441926696595	16.117689244475997	19
8.0	1.39781560914045	4.226362250276107	7.126280991025457	10.094915548354615	13.114131741737713	16.16745923068958	19
9.0	1.4148652882228738	4.269444387862823	7.180559196348842	10.150186852058049	13.165990820451912	16.214677421263666	19
10.0	1.4288700164435535	4.3058014133970755	7.228109761698937	10.200262623065564	13.214185724733982	16.25936123076397	19
15.0	1.4729157365939387	4.425495258061233	7.395901349539864	10.389796202592619	13.407759670763587	16.447368788748985	19
20.0	1.496128954797408	4.491480054095912	7.495412087030749	10.511669981060606	13.541976795112355	16.5863946973131	19
30.0	1.5201673840831922	4.5614949176964235	7.605689964950525	10.654324087998347	13.708547139718178	16.76905682625955	19
40.0	1.5325024937404261	4.597942714313872	7.664659973782444	10.733414632111291	13.804845147305453	16.879444504888784	19
50.0	1.540005942083949	4.620245731800399	7.701159370413534	10.783164231111495	13.866633360692852	16.95188394885361	20
60.0	1.5450511637002098	4.635287337659696	7.725920951829726	10.817203633079954	13.909368026198585	17.002622122894046	20
80.0	1.551406180023728	4.654276035887111	7.757317371607012	10.86064118638776	13.964353645589702	17.06855443823896	20
100.0	1.5552451291675182	4.665765142020605	7.7763740779268	10.887130101374716	13.998089735094208	17.109307260626032	20
1000.0	1.5692271009805032	4.707681333828946	7.846135659321186	10.984590139208333	14.123044835195774	17.261499809046036	20

If the same values are checked in mathlab(octave):

```
>> L=1;H=0.1;alfa=1e-5;x=0.05;t=1000;
>> x=[beta1(1,L,H),beta1(2,L,H),beta1(3,L,H),beta1(4,L,H),beta1(5,L,H)]
x =0.311052848268378   3.173097175184144   6.299059360548651   9.435375971163087   12.574323171641831
```

```
>> L=1;H=0.5;alfa=1e-5;x=0.05;t=1000;
>> x=[beta1(1,L,H),beta1(2,L,H),beta1(3,L,H),beta1(4,L,H),beta1(5,L,H)]
x =0.653271187307584   3.292310022477304   6.361620392838449   9.477485702590318 12.606013439411305
```

```
>> L=1;H=1;alfa=1e-5;x=0.05;t=1000;
>> x=[beta1(1,L,H),beta1(2,L,H),beta1(3,L,H),beta1(4,L,H),beta1(5,L,H)]
x =0.860333588363406   3.425618459073878   6.437298181858815   9.529334406744880 12.645287216547336
```

```
>> L=1;H=2;alfa=1e-5;x=0.05;t=1000;
>> x=[beta1(1,L,H),beta1(2,L,H),beta1(3,L,H),beta1(4,L,H),beta1(5,L,H)]
x =1.076873985989610   3.643597165479853   6.578333727851468   9.629560345782938 12.722298777154021
```

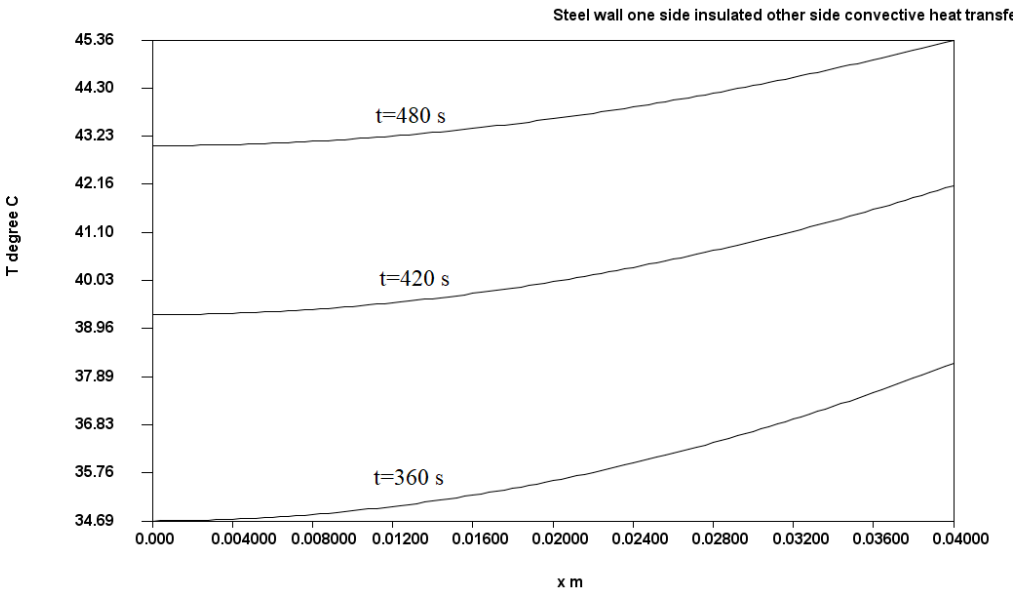
As it is seen from the examples, as long as algorithms and codes entered correctly the results are independent of the language used for the code.

Numerical example:

A steel wall (AISI 1010, k=63.9 W7mK) has Wall thickness of 0.04 m. Initially the pipe temperature is -20 ℃. (constant) The pipe is insulated on one surface. The other wall is exposed to 60 ℃ fluid with thermal convectivity coefficient of h=500 W/m^{2}K. Find the temperature profile of the Wall at a) t=360 s, b)t=420 s and c) t=480 s F(x)=-20-60=-80 ℃,

```
 public class HT_plane_wall_test
{
public static void main (String args[])
{ double h=500; //W/m^2K
  double k=63.9;
  double alpha=18.8e-6;
  double L=40e-3;
  double t[]={6*60,7*80,8*60};  //s
  double Tinf=60;
```

```
double Ti=-20;
int n1=t.length;
double x[]=new double[101];
double T[][]=new double[n1][101];
double q[][]=new double[n1][101];
HT_plane_wall htpw=new HT_plane_wall(h,L,k,alpha,Ti,Tinf);
//initial temperature variation t=0
htpw.F=xx->(Ti-Tinf);
for(int j=0;j<n1;j++)
{
for(int i=0;i<=100;i++)
    {   x[i]=i*L/100;
                T[j][i]=htpw.T(x[i],t[j]);
                q[j][i]=htpw.q(x[i],t[j]);
    }
 }
    Plot pp=new Plot(x,T[0]);
    for(int j=1;j<n1;j++)
    {pp.addData(x,T[j]);}
    pp.setPlabel("Steel wall one side insulated other side convective heat transfer");
    pp.setXlabel(" x m");
    pp.setYlabel(" T degree C");
    pp.plot();
    Plot pp1=new Plot(x,q[0]);
    for(int j=1;j<n1;j++)
    {pp1.addData(x,q[j]);}
    pp1.setPlabel("Steel wall one side insulated other side convective heat transfer");
    pp1.setXlabel(" x m");
    pp1.setYlabel(" q W/m^2");
    pp1.plot();
}
}
```

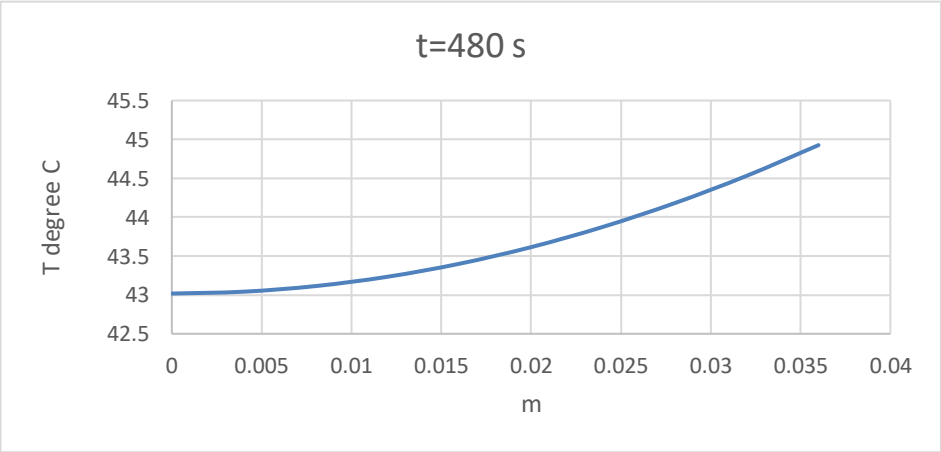


t=480 s , x=0-0.04 solution in excel:

	H	7.824	m									
	L	0.04	m									
h		500	W/m									
k		63.9	W/m									
Bi	H*L	0.312										
Fo	at/L2	5.64										
α		1.88E	m2/s									
t		480	s									
x	m	0	0.004	0.008	0.012	0.016	0.02	0.024	0.028	0.032	0.036	0.04
Tinf		60	degre									
Ti		-20	degre									

m	β	β*L=	C_n	Σ								
1	13.29	0.5318	1.0467	0.21228	0.21198	0.21108	0.20958	0.20749	0.20481	0.20156	0.19773	0.1933	0.18842
2	80.94	3.2379	-	-1.2E-	-1.1E-	-9.6E-	-6.8E-	-3.3E-	5.8E-29	4.38E-	7.72E-	1.03E-	1.17E-
3	158.3	6.3325	0.0154	9.2E-	7.4E-	2.8E-	-3E-101	-8E-101	-9E-101	-7E-101	-3E-101	3.2E-	7.7E-
4	236.4	9.4578	-	-5E-222	-3E-222	1.7E-	5.2E-	4.4E-	-9E-224	-5E-222	-5E-222	-2E-	3.4E-
5	314.7	12.591	0.0039	0	0	0	0	0	0	0	0	0	0

	0.21228	0.21198	0.21108	0.20958	0.20749	0.20481	0.20156	0.19773	0.1933	0.18842
T	43.01	43.04	43.11	43.23	43.40	43.61	43.87	44.18	44.53	44.92
	degre									



Root finding process (Bisection method):

H	7.825
L	0.04
Bi=H*L	0.313
	root_findin
m	1

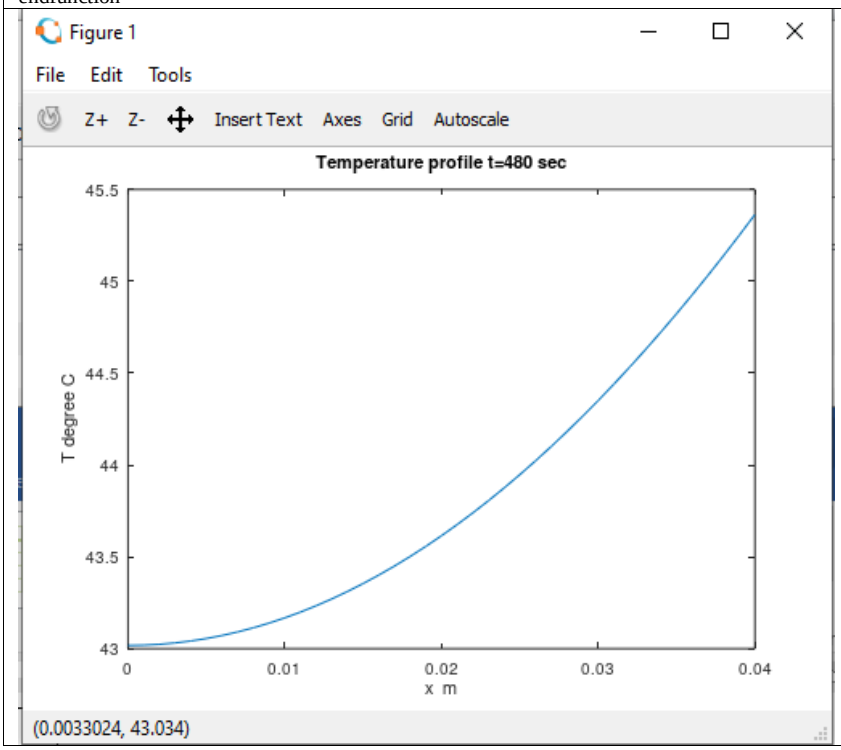
xl	xu	xr	fxl	fxu	fxr	fxr*fxl	
3E-06	39.27	19.635	-0.313	3E+16	0.47241	-0.1479	0.7854
3E-06	19.635	9.81748	-0.313	0.472	-0.1503	0.04705	0.3927
9.8175	19.635	14.7262	-0.15	0.472	0.0806	-0.0121	0.58905
9.8175	14.726	12.2718	-0.15	0.081	-0.0506	0.00761	0.49087
12.272	14.726	13.499	-0.051	0.081	0.01065	-0.0005	0.53996
12.272	13.499	12.8854	-0.051	0.011	-0.021	0.00106	0.51542
12.885	13.499	13.1922	-0.021	0.011	-0.0054	0.00011	0.52769
13.192	13.499	13.3456	-0.005	0.011	0.00254	-1E-05	0.53383
13.192	13.346	13.2689	-0.005	0.003	-0.0015	8E-06	0.53076
13.269	13.346	13.3073	-0.001	0.003	0.00053	-8E-07	0.53229
13.269	13.307	13.2881	-0.001	5E-04	-0.0005	6.9E-07	0.53152
13.288	13.307	13.2977	-5E-04	5E-04	3E-05	-1E-08	0.53191
13.288	13.298	13.2929	-5E-04	3E-05	-0.0002	1E-07	0.53172
13.293	13.298	13.2953	-2E-04	3E-05	-1E-04	2.1E-08	0.53181
13.295	13.298	13.2965	-1E-04	3E-05	-3E-05	3.1E-09	0.53186
13.296	13.298	13.2971	-3E-05	3E-05	-2E-06	5.5E-11	0.53188
13.297	13.298	13.2974	-2E-06	3E-05	1.4E-05	-2E-11	0.5319
13.297	13.297	13.2972	-2E-06	1E-05	6.1E-06	-1E-11	0.53189
13.297	13.297	13.2972	-2E-06	6E-06	2.2E-06	-4E-12	0.53189
13.297	13.297	13.2971	-2E-06	2E-06	2.8E-07	-5E-13	0.53189

Octave environment solution:

```
function [F] = F1(x)
        F=-80.0;
end
```

```
function  [T] = plot_T1()
  t=480.0;
  Tinf=60.0;
```

```
  x=0:0.001:0.04;
  h=500.0;
  k=63.9;
  H=h/k;
  L=0.04;
  alpha=18.8e-6;
  T=T1(x,t,L,H,alpha)+Tinf;
  plot(x,T);
  title('Temperature profile t=480 sec')
  xlabel('x  m')
ylabel('T degree C')
endfunction
```



Python version of the problem solution:

```
 from math import *
from f_x import *;
import matplotlib.pyplot as plt

def bisection(f,xl,xu):
  # bisection root finding method
  maxit=100
  iter=0
  es=0.0000001
  ea=1.1*es
  s=""
  fxl= f.func(xl)
  fxu= f.func(xu)
  xr=xl
  while ea>es and iter<maxit:
    xold=xr
    xr=(xl+xu)/2.0;
    fxr= f.func(xr);
    iter=iter+1;
    if xr!=0:
      ea=abs((xr-xold)/xr)*100
    test= fxl*fxr;
    if test==0.0 :      ea=0;
    elif test<0.0: xu=xr;fxu=fxr;
    elif test>0:    xl=xr;fxl=fxr;
    else :          ea=0;
    if iter>=maxit: print("Maximum number of iteration is exceeded \n result might not be valid")
  return xr

def gauss_legendre_coefficients(x1,x2,n):
   #calculates legendre gauss-coefficients as coefficients of the integral
```

```
    #for n terms
        eps=3e-15
        m=int((n+1.0)/2.0)
        xm=0.5*(x2+x1)
        xl=0.5*(x2-x1)
        nmax=100
        a=[[0.0 for i in range(n)] for j in range(2)]
        for i in range(1,m+1):
            z=cos(pi*((i-0.25)/(n+0.5)))
            for ii in range(nmax):
                #while abs(z-z1) > eps:
                p1=1.0;
                p2=0.0;
                for j in range(1,n+1):
                    p3=p2
                    p2=p1
                    p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
                pp=(float)(n*(z*p1-p2)/(z*z-1.0))
                z1=z;
                z=z1-p1/pp
                if abs(z-z1) < eps: break
            a[0][i-1]=xm-xl*z
            a[0][n-i]=xm+xl*z
            a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
            a[1][n-i]=a[1][i-1]
        return a

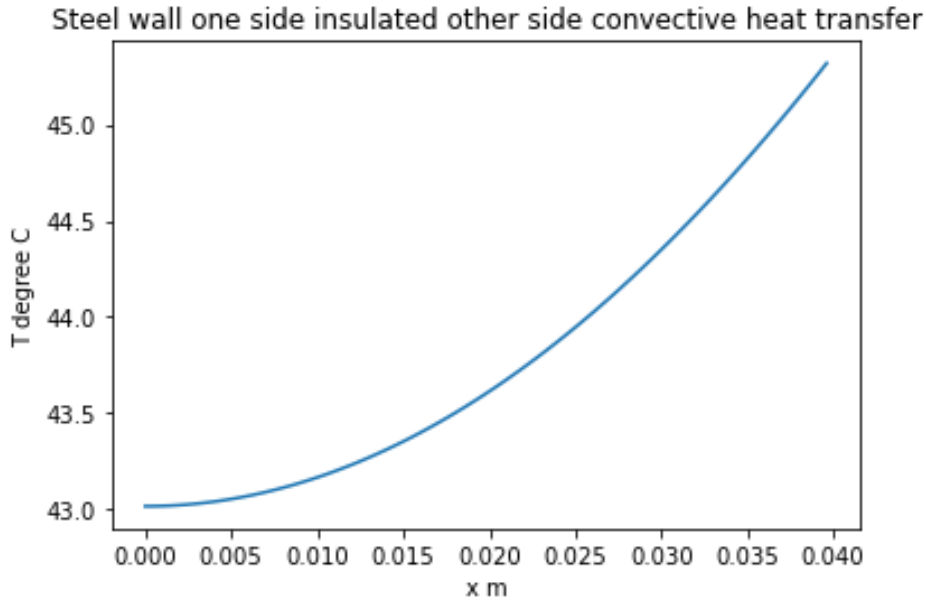

def integral(f,x1,x2,n):
    #integral func(x)dx
    #integral of a function by using gauss-legendre quadrature
    #between x1 and x2
    a=gauss_legendre_coefficients(x1,x2,n)
    z=0
    for i in range(n):
        z=z+a[1][i]*f.func(a[0][i])
    return z


def ksi(n,Bi):
    # eigenvalue function to be solved
    x1=n*pi+0.00000001
    x2=x1+(pi/2.0-0.00000001)
    class f1(f_x):func=lambda self,x: x*tan(x)-Bi
    f = f1()
    y= bisection(f,x1,x2)
    return y

def tetha(F,H,alpha,L,x,t):
    beta=0.0
    C=0.0
    N=0.0
    teta=0.0
    Bi=H*L
    for n in range(20):
        k1=ksi(n,Bi)
        beta=k1/L
        #N=L/2.0+sin(2.0*ksi)/(4.0*beta)
        N=(L*(beta*beta+H*H)+H)/(2*(beta*beta+H*H))
        class Cn(f_x):func=lambda self,x:F.func(x)*cos(beta*x)
        fCn=Cn()
        C=1.0/N*integral(fCn,0.0,L,20)
        teta=teta+C*cos(beta*x)*exp(-beta*beta*alpha*t)
    return teta

h=500
k=63.9
H=h/k
alpha=18.8e-6
L=40e-3
t=8.0*60
Tinf=60
Ti=-20
class F(f_x):func=lambda self,x: Ti-Tinf
F1 = F()
n=100
T=[0.0 for i in range(n)]
```

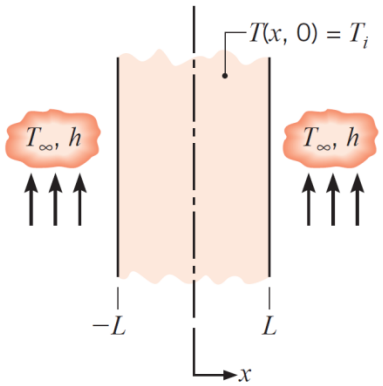
```
x=[0.0 for i in range(n)]
for i in range(n):
   x[i]=i*L/n
   T[i]=tetha(F1,H,alpha,L,x[i],t)+Tinf
plt.plot(x,T)
plt.xlabel("x m")
```



Numerical example:

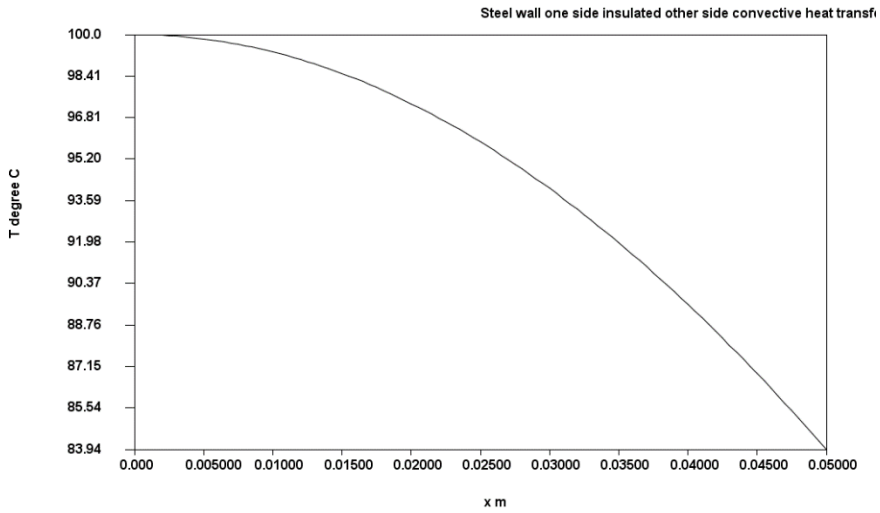
A steel plate [α=1.2e-5 and k=43 W/(mK)] of thickness 2L=0.1 m, initially at a uniform temperature of Ti=240 ℃, is suddenly immersed in an oil bath at $T_\infty = 40$ ℃. The convective heat transfer coefficient is h=600 W/(m^{2}K) calculate temperature profile for t=478.8 seconds.

F(x)=240-40=200 ℃,




```
 public class HT_plane_wall_test1
{
public static void main (String args[])
{ double h=600; //W/m^2K
  double k=43;
  double alpha=1.2e-5;
  double L=0.05;
  double t[]={478.8};  //s
  double Tinf=40.0;
  double Ti=240.0;
  int n1=t.length;
  double x[]=new double[101];
  double T[][]=new double[n1][101];
  double q[][]=new double[n1][101];
  HT_plane_wall htpw=new HT_plane_wall(h,L,k,alpha,Ti,Tinf);
  //initial temperature variation t=0
  htpw.F=xx->(Ti-Tinf);
  for(int j=0;j<n1;j++)
  {
```

```
  for(int i=0;i<=100;i++)
      {   x[i]=i*L/100;
                  T[j][i]=htpw.T(x[i],t[j]);
                  q[j][i]=htpw.q(x[i],t[j]);
      }
   }
      Plot pp=new Plot(x,T[0]);
      for(int j=1;j<n1;j++)
      {pp.addData(x,T[j]);}
      pp.setPlabel("Steel wall one side insulated other side convective heat transfer");
      pp.setXlabel(" x m");
      pp.setYlabel(" T degree C");
      pp.plot();
      Plot pp1=new Plot(x,q[0]);
      for(int j=1;j<n1;j++)
      {pp1.addData(x,q[j]);}
      pp1.setPlabel("Steel wall one side insulated other side convective heat transfer");
      pp1.setXlabel(" x m");
      pp1.setYlabel(" q W/m^2");
      pp1.plot();
}
}
```

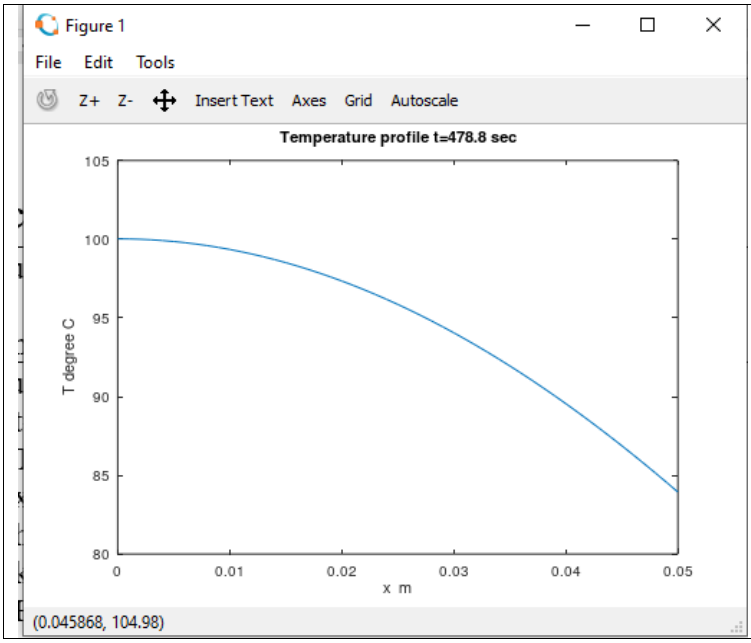


Octave(matlab) solution of the problem:

```
function [F] = F1(x)
        F=240-40;
end
```

```
function  [T] = plot_T2()
  t=478.8;
  Tinf=40.0;
  x=0:0.001:0.05;
  h=600.0;
  k=43.0;
  H=h/k;
  L=0.05;
  alpha=1.2e-5;
  T=T1(x,t,L,H,alpha)+Tinf;
  plot(x,T);
  title('Temperature profile t=478.8 sec')
  xlabel('x  m')
  ylabel('T degree C')
endfunction
```

```
>>
>> plot_T2();
>>
```



Python solution of the problem:

```
from math import *
from f_x import *;
import matplotlib.pyplot as plt

def bisection(f,xl,xu):
  # bisection root finding method
  maxit=100
  iter=0
  es=0.0000001
  ea=1.1*es
  s=""
  fxl= f.func(xl)
  fxu= f.func(xu)
  xr=xl
  while ea>es and iter<maxit:
    xold=xr
    xr=(xl+xu)/2.0;
    fxr= f.func(xr);
    iter=iter+1;
    if xr!=0:
      ea=abs((xr-xold)/xr)*100
    test= fxl*fxr;
    if test==0.0 :      ea=0;
    elif test<0.0: xu=xr;fxu=fxr;
    elif test>0:    xl=xr;fxl=fxr;
    else :          ea=0;
    if iter>=maxit: print("Maximum number of iteration is exceeded \n result might not be valid")
  return xr

def gauss_legendre_coefficients(x1,x2,n):
   #calculates legendre gauss-coefficients as coefficients of the integral
   #for n terms
      eps=3e-15
      m=int((n+1.0)/2.0)
      xm=0.5*(x2+x1)
      xl=0.5*(x2-x1)
      nmax=100
      a=[[0.0 for i in range(n)] for j in range(2)]
      for i in range(1,m+1):
         z=cos(pi*((i-0.25)/(n+0.5)))
         for ii in range(nmax):
            #while abs(z-z1) > eps:
            p1=1.0;
            p2=0.0;
            for j in range(1,n+1):
               p3=p2
               p2=p1
               p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
```

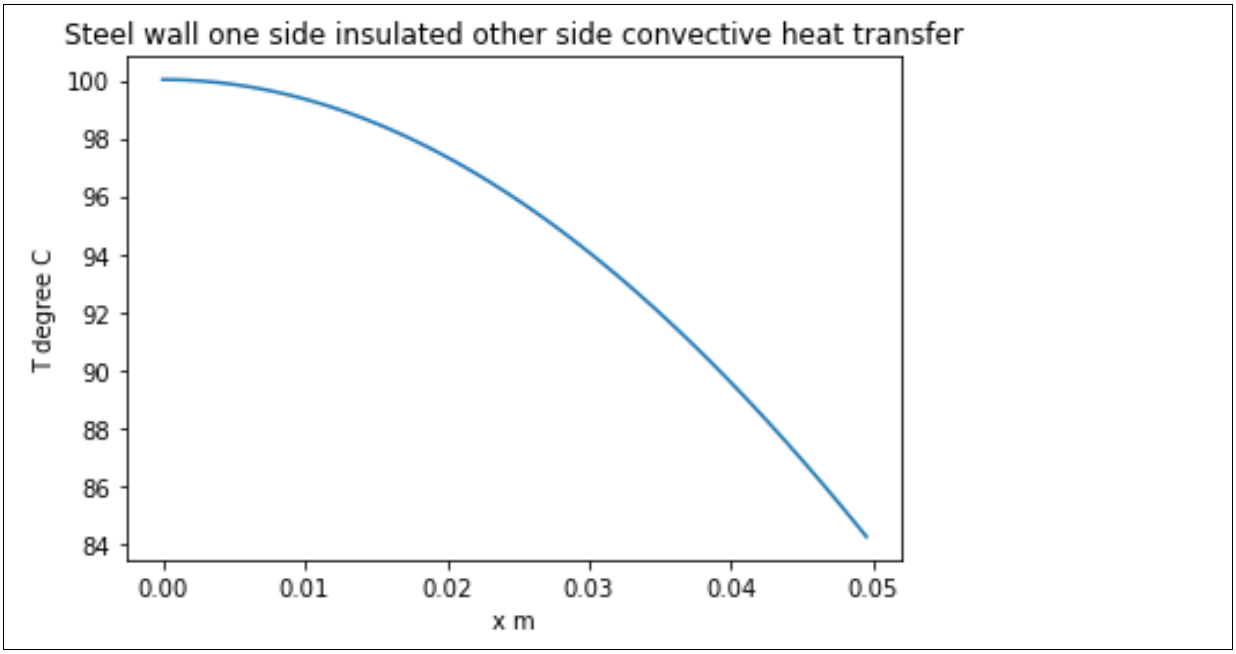
```
            pp=(float)(n*(z*p1-p2)/(z*z-1.0))
            z1=z;
            z=z1-p1/pp
            if abs(z-z1) < eps: break
        a[0][i-1]=xm-xl*z
        a[0][n-i]=xm+xl*z
        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
        a[1][n-i]=a[1][i-1]
    return a


def integral(f,x1,x2,n):
    #integral func(x)dx
    #integral of a function by using gauss-legendre quadrature
    #between x1 and x2
    a=gauss_legendre_coefficients(x1,x2,n)
    z=0
    for i in range(n):
        z=z+a[1][i]*f.func(a[0][i])
    return z


def ksi(n,Bi):
    # eigenvalue function to be solved
    x1=n*pi+0.00000001
    x2=x1+(pi/2.0-0.00000001)
    class f1(f_x):func=lambda self,x: x*tan(x)-Bi
    f = f1()
    y= bisection(f,x1,x2)
    return y

def tetha(F,H,alpha,L,x,t):
    beta=0.0
    C=0.0
    N=0.0
    teta=0.0
    Bi=H*L
    for n in range(20):
        k1=ksi(n,Bi)
        beta=k1/L
        #N=L/2.0+sin(2.0*ksi)/(4.0*beta)
        N=(L*(beta*beta+H*H)+H)/(2*(beta*beta+H*H))
        class Cn(f_x):func=lambda self,x:F.func(x)*cos(beta*x)
        fCn=Cn()
        C=1.0/N*integral(fCn,0.0,L,20)
        teta=teta+C*cos(beta*x)*exp(-beta*beta*alpha*t)
    return teta

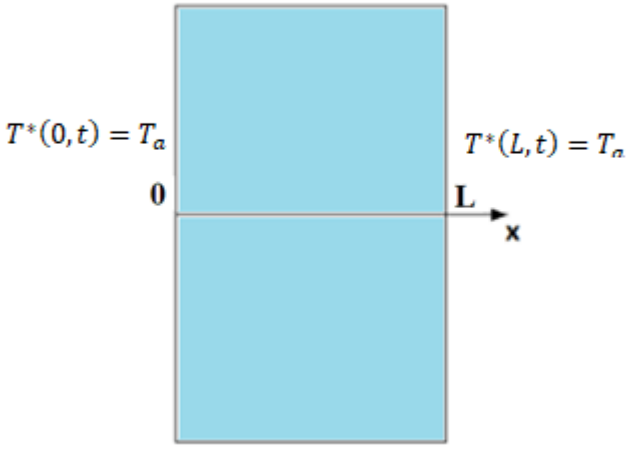
h=600.0
k=43.0
H=h/k
alpha=1.2e-5
L=0.05
t=478.8
Tinf=40
Ti=240
class F(f_x):func=lambda self,x: Ti-Tinf
F1 = F()
n=100
T=[0.0 for i in range(n)]
x=[0.0 for i in range(n)]
for i in range(n):
    x[i]=i*L/n
    T[i]=tetha(F1,H,alpha,L,x[i],t)+Tinf
plt.plot(x,T)
plt.xlabel("x m")
plt.ylabel("T degree C")
plt.title("Steel wall one side insulated other side convective heat transfer")
plt.show()
```



PROBLEM 3.2 SEPARATION OF VARIABLE HOMOGENEOUS SOLUTION, TIME DEPENDENT; CARTESIAN COORDINATES

In this problem both side of the wall of (x=0 and x=L) is exposed to a constant temperature Ta. At initial time, the wall has a temperature profile of F(x).

$$T^*(x,t) = F(x) + T_a \quad t = 0 \quad 0 \le x \le L$$



$\frac{\partial^2 T^*(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T^*(x,t)}{\partial t}$ Boundary conditions $T^*(x,t) = T_a$ x=0 t >0 $T^*(x,t) = T_a$ x=L t >0 $T^*(x,t) = F(x) + T_a$ t=0 $0 \le x \le L$	$T(x,t) = T^*(x,t) - T_a$ $\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T(x,t)}{\partial t}$ Boundary conditions $T(x,t) = 0$ x=0 t >0 $T(x,t) = 0$ x=L t >0 $T(x,t) = F(x)$ t=0 $0 \le x \le L$

Separation of variables:

$T(x,t) = X(x)\Gamma(t)$

$\Gamma(t)\frac{\partial^2 X(x)}{\partial x^2} = \frac{X(x)}{\alpha}\frac{\partial \Gamma(t)}{\partial t}$

$$\frac{1}{X(x)}\frac{\partial^2 X(x)}{\partial x^2} = \frac{1}{\alpha\Gamma(t)}\frac{\partial \Gamma(t)}{\partial t} = -\beta^2$$

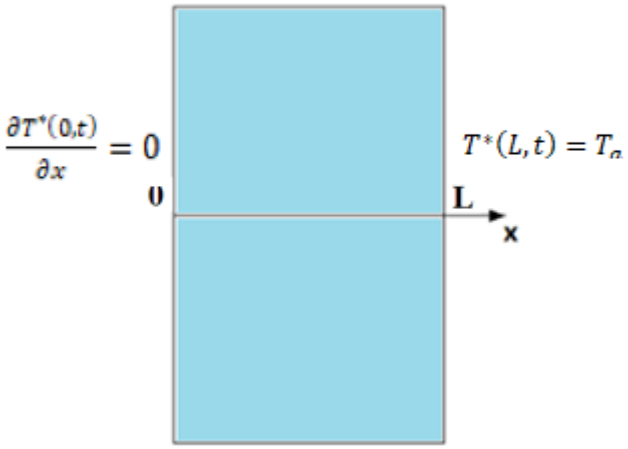
$$\frac{\partial^2 X(x)}{\partial x^2} + \beta^2 X(x) = 0$$
$$\frac{\partial \Gamma(t)}{\partial t} + \beta^2 \alpha \Gamma(t) = 0$$
Boundary conditions
$X(x) = 0$ x=0 t >0
$X(x) = 0$ x=L t >0
$T(x,t) = F(x)$ t=0 $0 \le x \le L$
Solution set:
$X(x) = C_1 \cos(\beta x) + C_2 \sin(\beta x)$
$\Gamma(t) = C_3 \exp(-\beta^2 \alpha t)$
Application of boundary conditions:
x=0 t >0
$T(x) = C_1 \cos(\beta 0) + C_2 \sin(\beta 0) = 0 \quad C_1 = 0$
x=L t >0
$T(x) = C_2 \sin(\beta L) = 0 \quad \beta_m L = m\pi \quad \beta_m = \frac{m\pi}{L}$
$$T(x,t) = T^*(x,t) - T_a = \sum_{m=1}^{\infty} C_m \sin(\beta_m x) \exp(-\beta_m^2 \alpha t) = \sum_{m=1}^{\infty} C_m \sin\left(\frac{m\pi}{L}x\right) \exp\left(-\frac{m^2\pi^2}{L^2}\alpha t\right)$$
We still do not know the C_n coefficients we will use the remaining boundary conditions to find it.
$T(x,t) = F(x)$ t=0 $0 \le x \le L$
$$F(x) = \sum_{m=1}^{\infty} C_m \sin(\beta_n x)$$
This equation has n coefficients. In order to solve these coefficients, we will use properties of orthogonal functions.
$$\int_0^L X(\beta_n x) X(\beta_m x) dx = \begin{cases} 0 & m \neq n \\ N(\beta_m) & m = n \end{cases}$$
$$\int_0^L F(x) \sin(\beta_m x) = C_m \int_0^L \sin^2(\beta_m x) dx$$
$$N(\beta_m) = \int_0^L \sin^2(\beta_m x) dx = \int_0^L \frac{1 - \cos(2\beta_m x)}{2} dx = \left[\frac{x}{2} - \frac{\sin(2\beta_m x)}{4\beta_m}\right]_{x=0}^{L} = \frac{L}{2} - \frac{\sin(2\beta_m L)}{4\beta_m}$$
$$N\left(\frac{m\pi}{L}\right) = \int_0^L \sin^2\left(\frac{m\pi}{L}x\right) dx = \frac{L}{2}$$
$$C_m = \frac{2}{L} \int_0^L F(x) \sin\left(\frac{m\pi}{L}x\right) dx$$
Depends on value of F(x), this integration can be solved by using numerical or analytical methods.
As an analytical solution example let us take $F(x) = 50 \sin\left(\frac{\pi x}{L}\right)$
$C_m = \frac{2}{L}\int_0^L 50 \sin\left(\frac{\pi x}{L}\right) \sin\left(\frac{m\pi}{L}x\right) dx$ sin is an orthogonal function so that
$$\int_0^L X(\beta_n x) X(\beta_m x) dx = \begin{cases} 0 & m \neq n \\ N(\beta_m) & m = n \end{cases}$$
Integral has only one remaining term for m=1
$$C_m = \frac{100}{L} \int_0^L \sin^2\left(\frac{m\pi}{L}x\right) dx = \frac{100}{L} \int_0^L \frac{1 - \cos(2\frac{m\pi}{L}x)}{2} dx = \frac{100}{L}\left[\frac{x}{2} - \frac{\sin(22\frac{m\pi}{L}xx)}{42\frac{m\pi}{L}x}\right]_{x=0}^{L} = 50$$

$$T(x,t) = T^*(x,t) - T_a = 50\sin\left(\frac{\pi}{L}x\right)\exp\left(-\frac{\pi^2}{L^2}\alpha t\right)$$

PROBLEM 3.3 SEPARATION OF VARIABLE HOMOGENEOUS SOLUTION, TIME DEPENDENT; CARTESIAN COORDINATES

In this problem one side of the wall of (x=0) is insulated and the other side is exposed to a constant temperature Ta. At initial time, the wall has a temperature profile of F(x).

$$T^*(x,t) = F(x) + T_a \quad t = 0 \quad 0 \le x \le L$$



$\frac{\partial^2 T^*(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T^*(x,t)}{\partial t}$ Boundary conditions $\frac{\partial T^*(x,t)}{\partial x} = 0$ x=0 t >0 $T^*(x,t) = T_a$ x=L t >0 $T^*(x,t) = F(x) + T_a$ t=0 $0 \le x \le L$	$T(x,t) = T^*(x,t) - T_a$ $\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T(x,t)}{\partial t}$ Boundary conditions $\frac{\partial T(x,t)}{\partial x} = 0$ x=0 t >0 $T(x,t) = 0$ x=L t >0 $T(x,t) = F(x)$ t=0 $0 \le x \le L$

Separation of variables:

$T(x,t) = X(x)\Gamma(t)$

$\Gamma(t)\frac{\partial^2 X(x)}{\partial x^2} = \frac{X(x)}{\alpha}\frac{\partial \Gamma(t)}{\partial t}$

$$\frac{1}{X(x)}\frac{\partial^2 X(x)}{\partial x^2} = \frac{1}{\alpha\Gamma(t)}\frac{\partial \Gamma(t)}{\partial t} = -\beta^2$$

$$\frac{\partial^2 X(x)}{\partial x^2} + \beta^2 X(x) = 0$$

$$\frac{\partial \Gamma(t)}{\partial t} + \beta^2\alpha\Gamma(t) = 0$$

Boundary conditions

$\frac{\partial X(x)}{\partial x} = 0$ x=0 t >0

$X(x) = 0$ x=L t >0

$T(x,t) = F(x)$ t=0 $0 \le x \le L$

Solution set:

$X(x) = C_1\cos(\beta x) + C_2\sin(\beta x)$

$\Gamma(t) = C_3\exp(-\beta^2\alpha t)$

Application of boundary conditions:

x=0 t >0

$\frac{\partial X(x)}{\partial x} = 0$ x=0 t >0

$$\frac{\partial X(x)}{\partial x} = -C_1\beta\sin(\beta 0) + C_2\beta\cos(\beta 0) = -C_1 * 0 + C_2\beta * 1 = 0 \xrightarrow{yields} C_2 = 0$$

x=L t >0

$T(x) = C_1 \cos(\beta L) = 0 \quad \beta_m L = (2m+1)\frac{\pi}{2} \quad \beta_m = (2m+1)\frac{\pi}{2L}$

$$T(x,t) = T^*(x,t) - T_a = \sum_{m=1}^{\infty} C_m \cos(\beta_m x) \exp(-\beta_m^2 \alpha t)\, T(x,t) = T^*(x,t) - T_a$$

$$T(x,t) = T^*(x,t) - T_a = \sum_{m=1}^{\infty} C_m \cos\left((2m+1)\frac{\pi}{2L}x\right) \exp\left(-(2m+1)^2 \frac{\pi^2}{4L^2}\alpha t\right)$$

We still do not know the C_n coefficients we will use the remaining boundary conditions to find it.

$T(x,t) = F(x)$ t=0 $0 \le x \le L$

$$F(x) = \sum_{m=1}^{\infty} C_m \cos(\beta_m x)$$

This equation has m coefficients. In order to solve these coefficients, we will use properties of orthogonal functions.

$$\int_0^L X(\beta_n x) X(\beta_m x)\, dx = \begin{cases} 0 & m \neq n \\ N(\beta_m) & m = n \end{cases}$$

$$\int_0^L F(x)\cos(\beta_m x) = C_m \int_0^L \cos^2(\beta_m x)\, dx$$

$$N(\beta_m) = \int_0^L \cos^2(\beta_m x)\, dx = \int_0^L \frac{1+\cos(2\beta_m x)}{2}\, dx = \left[\frac{x}{2} + \frac{\sin(2\beta_m x)}{4\beta_m}\right]_{x=0}^{L} = \frac{L}{2} - \frac{\sin(2\beta_m L)}{4\beta_m}$$

$$N\left((2m+1)\frac{\pi}{2L}\right) = \int_0^L \cos^2\left((2m+1)\frac{\pi}{2L}x\right) dx = \frac{L}{2}$$

$$C_m = \frac{2}{L}\int_0^L F(x)\cos\left((2m+1)\frac{\pi}{2L}x\right) dx$$

$$T(x,t) = T^*(x,t) - T_a$$

$$= \sum_{m=0}^{\infty} \frac{2}{L}\cos\left((2m+1)\frac{\pi}{2L}x\right) \exp\left(-(2m+1)^2\frac{\pi^2}{4L^2}\alpha t\right) \int_0^L F(x)\cos\left((2m+1)\frac{\pi}{2L}x\right) dx$$

A special case: if initial temperature is constant F(x)=T_i=const

$$\int_0^L F(x)\cos\left((2m+1)\frac{\pi}{2L}x\right) dx = \int_0^L T_i \cos\left((2m+1)\frac{\pi}{2L}x\right) dx$$

$$= T_i \frac{2L}{\pi(2m+1)}\left[\sin\left((2m+1)\frac{\pi}{2L}x\right)\right]_0^L$$

$$\int_0^L F(x)\cos\left((2m+1)\frac{\pi}{2L}x\right) dx = \frac{2LT_i}{\pi(2m+1)}\sin\left((2m+1)\frac{\pi}{2}\right) = \frac{2LT_i(-1)^m}{\pi(2m+1)}$$

$$T(x,t) = T^*(x,t) - T_a = \sum_{m=0}^{\infty} \frac{4T_i(-1)^m}{\pi(2m+1)}\cos\left((2m+1)\frac{\pi}{2L}x\right) \exp\left(-(2m+1)^2\frac{\pi^2}{4L^2}\alpha t\right)$$

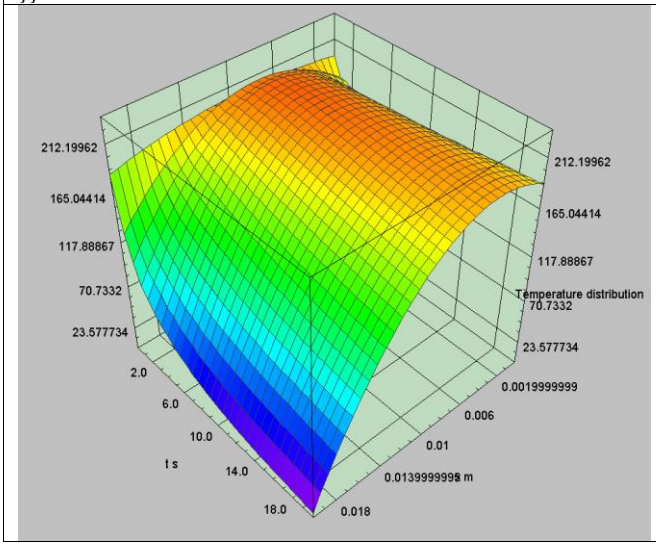
Numerical example:

L=2x10^{-2} m thick marble plate[k=2 W/(mK), α=1x10^{-6} m^2/s} is initially at a uniform temperature of T_i=200 ℃. One of the plate surface is exposed to a temperature of T_a=0 ℃ while he other side is kept insulated. Calculate the temperature profile. Assume the plate has temperature profile as a function of time and direction(x) only

```
 public class HT_xy_T_dT_PR3_3
{
public static void main(String arg[])
{   double alpha=1.0e-6;  //m^2/s
```

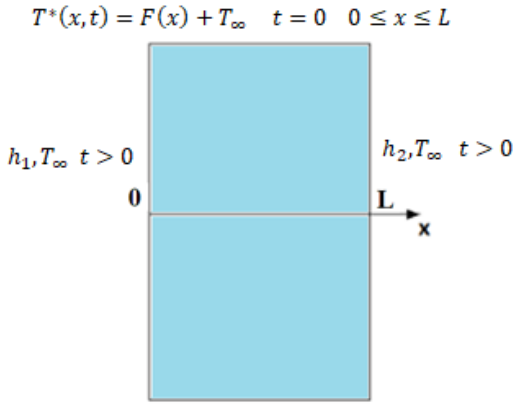
```
    double k=2; //W/mK
    double L=2.0e-2; //m
    double Ti=200.0; //degree C initial temperature
    double Ta=0.0;//degree C
    double pi=Math.PI;
              if_xj Tx_t=(double y[])->{
                        double x=y[0];
                        double t=y[1];
                        double T=Ta;
                        int plusminus=1;
                        double kk=0.0;
                        double x1=0.0;
                        double x2=0.0;
                        double x3=0.0;
                        for(int m=0;m<100 ;m++)
                        {  kk=(2.0*m+1.0);
                                  T+=4.0*(Ti-Ta)*plusminus/(pi*kk)*Math.cos(pi*kk/2.0*x/L)*Math.exp(-
pi*pi*kk*kk/4.0/(L*L)*alpha*t);
                                  plusminus*=-1;
                          }
                          return T;
                        };
                        double a[][]=new double[100][20];
                        double dx=L/99;
                        double dt=1;
                        double x=0;
                        double t=0;
                        double T=0.0;
    for(int i=0;i<100;i++)
    { x=dx*i;
                for(int j=0;j<20;j++)
     { t=dt*j;
                double y[]={x,t};
      T=Tx_t.func(y);
      System.out.println(x+" "+t+" "+T);
      a[i][j]=T;
     }

  }
        double bb[][]=surfacePlot.dataformat(0.0,L,0,20.0,a);
        Text.print(bb);
        surfacePlot p=new surfacePlot(bb,"x m","t s","Temperature distribution");
        p.plot();
}}
```



PROBLEM 3.4 SEPARATION OF VARIABLE HOMOGENEOUS SOLUTION, TIME DEPENDENT; CARTESIAN COORDINATES

This examples has two different convective heat transfer coefficient in each side of the wall, but outside fluid temperatures are the same



$q_{convection_0} = h_1(T(0,t) = h_1 T(0,t) = h_1(T^*(0,t) - T_\infty)$
$q_{convection_1} = h_2(T(L,t) = h_2 T(L,t) = h_2(T^*(L,t) - T_\infty)$
By defining temperature T(x,t) differential equation becomes homogeneous.

$\frac{\partial^2 T^*(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T^*(x,t)}{\partial t}$ Boundary conditions $-\frac{\partial T^*(x,t)}{\partial x} + H_1(T^*(x,t) - T_\infty) = 0$ x=0 t >0 $H_1 = \frac{h_1}{k}$ $\frac{\partial T^*(x,t)}{\partial x} + H_2(T^*(x,t) - T_\infty) = 0$ x=L t>0 $H_2 = \frac{h_2}{k}$ $T^*(x,t) = F(x) + T_\infty$ t=0 $0 \le x \le L$	$T(x,t) = T^*(x,t) - T_\infty$ $\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T(x,t)}{\partial t}$ Boundary conditions $-\frac{\partial T(x,t)}{\partial x} + H_1 T(x,t) = 0$ x=0 t >0 $H_1 = \frac{h_1}{k}$ $\frac{\partial T(x,t)}{\partial x} + H_2 T(x,t) = 0$ x=L t>0 $H_2 = \frac{h_2}{k}$ $T(x,t) = F(x)$ t=0 $0 \le x \le L$

Separation of variables:
$T(x,t) = X(x)\Gamma(t)$
$\Gamma(t)\frac{\partial^2 X(x)}{\partial x^2} = \frac{X(x)}{\alpha}\frac{\partial \Gamma(t)}{\partial t}$
$\frac{1}{X(x)}\frac{\partial^2 X(x)}{\partial x^2} = \frac{1}{\alpha\Gamma(t)}\frac{\partial \Gamma(t)}{\partial t} = -\beta^2$
$\frac{\partial^2 X(x)}{\partial x^2} + \beta^2 X(x) = 0$
$\frac{\partial \Gamma(t)}{\partial t} + \beta^2 \alpha \Gamma(t) = 0$
Boundary conditions
$-\frac{\partial X(x)}{\partial x} + H_1 X(x) = 0$ x=0 t >0 $H_1 = \frac{h_1}{k}$
$\frac{\partial T(x)}{\partial x} + H_2 X(x) = 0$ x=L t>0 $H_2 = \frac{h_2}{k}$
$T(x,t) = F(x)$ t=0 $0 \le x \le L$
Solution:
$X(x) = C_1 \cos(\beta x) + C_2 \sin(\beta x)$
$\Gamma(t) = C_3 \exp(-\beta^2 \alpha t)$
$T(x,t) = [C_1 \cos(\beta x) + C_2 \sin(\beta x)] \exp(-\beta^2 \alpha t)$
Applications of boundary conditions:

$$\frac{\partial X(x)}{\partial x} = -C_1 \beta \sin(\beta x) + C_2 \beta \cos(\beta x)$$

$$-C_1 \beta \sin(\beta 0) + C_2 \beta \cos((\beta 0) + H_1[C_1 \cos((\beta 0) + C_2 \sin((\beta 0)] = 0$$

$$-C_1 0 + C_2 \beta 1 + H_1[C_1 1 + C_2 0] = 0$$
$$C_2 \beta + C_1 H_1 = 0$$

$C_2 = \frac{C_1 H_1}{\beta}$

$X(x) = C_1[\beta\cos(\beta x) + H_1 \sin(\beta x)]$

$\frac{\partial X(x)}{\partial x} = C_1[-\beta^2 \sin(\beta x) + H_1 \beta \cos(\beta x)]$

$\frac{\partial T(x)}{\partial x} + H_2 X(x) = 0$ x=L t>0

$C_1[-\beta^2 \sin(\beta L) + H_1 \beta \cos(\beta L)] + H_2 C_1[\beta\cos(\beta L) + H_1 \sin(\beta L)] = 0 \rightarrow C_1 \neq 0$

$[-\beta^2 \sin(\beta L) + H_1 \beta \cos(\beta L)] + H_2[\beta\cos(\beta L) + H_1 \sin(\beta L)] = 0$

$\tan(\beta_m L) = \frac{\beta_m (H_1 + H_2)}{\beta_m^2 - H_1 H_2}$ $\xi_m = \beta_m L$ $Bi_1 = H_1 L = \frac{h_1 L}{k}$ $Bi_2 = H_2 L = \frac{h_2 L}{k}$

$\tan(\xi_m) = \frac{\xi_m (Bi_1 + Bi_2)}{\xi_m^2 - Bi_1 Bi_2}$ or

$\sin(\beta_m L) - \frac{\beta_m (H_1 + H_2)}{\beta_m^2 - H_1 H_2}\cos(\beta_m L) = 0$ or $\sin(\xi_m) - \frac{\xi_m (Bi_1 + Bi_2)}{\xi_m^2 - Bi_1 Bi_2}\cos(\xi_m) = 0$

This equation has multiple roots.

$T(x,t) = \sum_{m=1}^{\infty} C_m[\beta_m \cos(\beta_m x) + H_1 \sin(\beta x)]\exp(-\beta_m^2 \alpha t)$

$T(x,t) = F(x)$ t=0 $0 \leq x \leq L$

$F(x) = \sum_{m=1}^{\infty} C_m[\beta_m \cos(\beta_m x) + H_1 \sin(\beta x)]$

This equation has n coefficients. In order to solve these coefficients, we will use properties of orthogonal functions.

$$\int_0^L X(\beta_n x) X(\beta_m x) dx = \begin{cases} 0 & m \neq n \\ N(\beta_m) & m = n \end{cases}$$

$$\int_0^L F(x)[\beta_m \cos(\beta_m x) + H_1 \sin(\beta x)]dx = C_m \int_0^L [\beta_m \cos(\beta_m x) + H_1 \sin(\beta_m x)]^2 dx$$

$$\int_0^L [\beta_m \cos(\beta_m x) + H_1 \sin(\beta_m x)]^2 dx$$

$$= \int_0^L [\beta_m^2 \cos^2(\beta_m x) + 2\beta_m H_1 \sin(\beta_m x)\cos(\beta_m x) + H_1^2 \sin(\beta_m x)]dx$$

$$= \int_0^L \left[\beta_m^2 \frac{1+\cos(2\beta_m x)}{2} + \beta_m H_1 \sin(2\beta_m x) + H_1^2 \frac{1-\cos(2\beta_m x)}{2}\right] dx$$

$$= \int_0^L \left[\frac{\beta_m^2 + H_1^2}{2} + \frac{\beta_m^2 - H_1^2}{2}\cos(2\beta_m x) + \beta_m H_1 \sin(2\beta_m x)\right] dx$$

$$= \left[\frac{\beta_m^2 + H_1^2}{2} x + \frac{\beta_m^2 - H_1^2}{4\beta_m}\sin(2\beta_m x) - \beta_m H_1 \frac{\cos(2\beta_m x)}{2\beta_m}\right]_{x=0}^{L}$$

$N(\beta_m) = \frac{(\beta_m^2 + H_1^2)}{2} + \frac{(\beta_m^2 - H_1^2)}{4\beta_m}\sin(2\beta_m L) + \frac{H_1}{2}(1 - \cos(2\beta_m L))$

Substituting $\sin(\beta_m L) = \frac{\beta_m (H_1 + H_2)}{\beta_m^2 - H_1 H_2}\cos(\beta_m L)$ eigenvalue function

$$N(\beta_m) = 0.5\left[(\beta_m^2 + H_1^2)\left(L + \frac{H_2}{(\beta_m^2 + H_2^2)}\right) + H_1\right]$$

$$C_m = \frac{1}{N(\beta_m)} \int_0^L F(x)[\beta_m \cos(\beta_m x) + H_1 \sin(\beta x)]dx$$

Depends on value of F(x), this integration can be solved by using numerical or analytical methods.

$T(x,t) = \sum_{m=1}^{\infty}\left[\frac{1}{N(\beta_m)}\int_0^L F(x)[\beta_m \cos(\beta_m x) + H_1 \sin(\beta_m x)]dx\right][\beta_m \cos(\beta_m x) + H_1 \sin(\beta x)]\exp(-\beta_m^2 \alpha t)$

For constant initial temperature: $F(x) = T_i$

$$C_m = \frac{T_i}{N(\beta_m)} \int_0^L [\beta_m \cos(\beta_m x) + H_1 \sin(\beta x)] dx = \frac{T_i}{N(\beta_m)} \left[\sin(\beta_m x)\Big|_0^L - \frac{H_1}{\beta_m} \cos(\beta_m x)\Big|_0^L \right]$$

$$C_m = \frac{T_i}{N(\beta_m)} \left[\sin(\beta_m L) + \frac{H_1}{\beta_m}(1 - \cos(\beta_m L)) \right]$$

PROBLEM 3.4 MATLAB SOLUTION

```
function  [T] = T2(x,t,L,H1,H2,alpha)
        T=0;
        for m=1:20
        beta=beta2(m,L,H1,H2)
        C2=CN2(beta,L,H1,H2)
        T=T+C2*(beta*cos(beta*x)+H1*sin(beta*x))*exp(-beta*beta*alpha*t);
        End
```

```
function [C] = CN2(beta,L,H1,H2)
        y=@(x) F2(x).*(beta*cos(beta*x)+H1*sin(beta*x));
        C1=quad(y,0,L);
        N=0.5*(L*(beta*beta+H1*H1)*(L+H2/(beta*beta+H2*H2)+H1);
        C=C1/N;
End
```

```
 function [beta] = beta2(m,L,H1,H2)
        PI=4*atan(1);
        x1=((m-1)*PI+0.0000001)/L;
        x2= x1+(PI/2-0.000001)/L;
        beta=bisection(@(x) func2(x,L,H1,H2),x1,x2);
```

```
function [F] = F2(x)
        F=1.0;
 end
```

```
function [f] = func2(beta,L,H1,H2)
        f=tan(beta*L)-beta*(H1+H2)/(beta*beta-H1*H2);
```

PROBLEM 3.4 JAVA SOLUTION

```
 import java.io.*;
import javax.swing.*;
import java.awt.*;
import static java.lang.Math.*;
public class HT_plane_wall2
{
public if_x F;
public double h1,h2,L,k,alpha,Bi1,Bi2,Ti,Tinf,H1,H2;
public  HT_plane_wall2(double h1i,double h2i,double Li,double ki,double alphai,double Tii,double Tinfi)
{h1=h1i;h2=h2i;k=ki;L=Li;alpha=alphai;Bi1=h1*L/k;Bi2=h2*L/k;Ti=Tii;Tinf=Tinfi;H1=h1/k;H2=h2/k;
 //initial temperature profile is defined as constant change
 //if you desired to do so
 F=xx->Ti-Tinf;
}

  public static double ksi(int n,double Bi1,double Bi2)
  {double x1=n*PI+0.0000001;
   double x2=x1+(PI/2-0.000001);
   if_x f1=x->(x*x-Bi1*Bi2)*Math.tan(x)-x*(Bi1+Bi2);
   double ksi=f1.illinois(x1,x2);
   return ksi;
  }
  public double teta(double x,double t)
  {double ksi=0;
   double beta=0;
   double C=0;
   double N=0,N1=0,N2=0,C1=0,C2=0;
   double teta=0;
   //coefficient of equation
   for(int n=0;n<10;n++)
   {ksi=ksi(n,Bi1,Bi2);
    beta=ksi/L;
```

```
    final double beta1=beta;
    //N=0.5*((beta*beta+H1*H1)*(L+H2/(beta*beta+H2*H2))+H1);
    //if_x intN=(xx)->(beta1*cos(beta1*xx)+H1*sin(beta1*xx))*(beta1*cos(beta1*xx)+H1*sin(beta1*xx));
    //N1=intN.integral(0,L,40);
    N2=L/2.0*(beta*beta+H1*H1)+sin(2.0*beta*L)/(4*beta)*(beta*beta-H1*H1)+H1/2.0*(1.0-cos(2.0*beta*L));
    if_x Cn=xx->F.func(xx)*(beta1*cos(beta1*xx)+H1*sin(beta1*xx));
    C=1.0/N2*Cn.integral(0,L,20);
    System.out.println("ksi="+ksi+"N2="+N2+"C?"+C)
    teta+=C*(beta*cos(beta*x)+H1*sin(beta*x))*exp(-beta*beta*alpha*t);
    }
    return teta;
  }
  public double dteta(double x,double t)
  {double ksi=0;
   double beta=0;
   double C=0;
   double N=0,N2=0,C1=0,C2=0;
   double dteta=0;
   //coefficient of equation
   for(int n=0;n<10;n++)
   {ksi=ksi(n,Bi1,Bi2);
    beta=ksi/L;
    final double beta1=beta;
    N2=L/2.0*(beta*beta+H1*H1)+sin(2.0*beta*L)/(4*beta)*(beta*beta-H1*H1)+H1/2.0*(1.0-cos(2.0*beta*L));    if_x Cn=xx-
>F.func(xx)*(beta1*cos(beta1*xx)+H1*sin(beta1*xx));
    C=1.0/N2*Cn.integral(0,L,20);
    dteta+=C*(-beta*beta*sin(beta*x)+H1*beta*cos(beta*x))*exp(-beta*beta*alpha*t);
    }
    return dteta;
  }
  public double T(double x,double t)
  {double T=teta(x,t)+Tinf;
   return T;
  }
  public double q(double x,double t)
  {double q=-k*dteta(x,t);
   return q;
  }

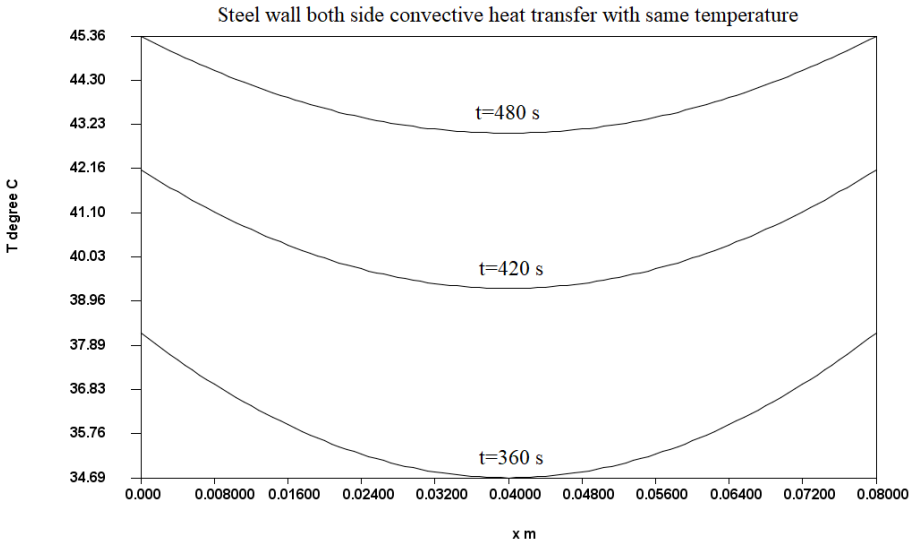
 public static void main (String args[])
 { double h1=500; //W/m^2K
   double h2=500; //W/m^2K
   double k=63.9;
   double alpha=18.8e-6;
   double L=2.0*40e-3;
   double t[]={8*60};  //s
   double Tinf=60;
   double Ti=-20;
   int n1=t.length;
   double x[]=new double[101];
   double T[][]=new double[n1][101];
   double q[][]=new double[n1][101];
   HT_plane_wall2 htpw=new HT_plane_wall2(h1,h2,L,k,alpha,Ti,Tinf);
   //initial temperature variation t=0
   htpw.F=xx->(Ti-Tinf);
   for(int j=0;j<n1;j++)
   {
   for(int i=0;i<=100;i++)
       {   x[i]=i*L/100;
                   T[j][i]=htpw.T(x[i],t[j]);
                   q[j][i]=htpw.q(x[i],t[j]);
       }
    }
       Plot pp=new Plot(x,T[0]);
       for(int j=1;j<n1;j++)
       {pp.addData(x,T[j]);}
       pp.setPlabel("Steel wall one side insulated other side convective heat transfer");
       pp.setXlabel(" x m");
       pp.setYlabel(" T degree C");
       pp.plot();
       Plot pp1=new Plot(x,q[0]);
       for(int j=1;j<n1;j++)
       {pp1.addData(x,q[j]);}
       pp1.setPlabel("Steel wall one side insulated other side convective heat transfer");
       pp1.setXlabel(" x m");
       pp1.setYlabel(" q W/m^2");
       pp1.plot();
```

```
}}
```

Numerical example:

A steel wall (AISI 1010, k=63.9 W7mK) has Wall thickness of 0.04 m. Initially the pipe temperature is -20 ℃. (constant) The wall is insulated on one surface. The other wall is exposed to 60 ℃ fluid with thermal convectivity coefficient of h=500 W/m^2K. Find the temperature profile of the Wall at a) t=360 s, b)t=420 s and c) t=480 s F(x)=-20℃ -60 ℃= -80 K,

```
public class HT_plane_wall2_test
{
public static void main (String args[])
{ double h1=500; //W/m^2K
  double h2=500; //W/m^2K
  double k=63.9;
  double alpha=18.8e-6;
  double L=2.0*40e-3;
  double t[]={8*60};  //s
  double Tinf=60;
  double Ti=-20;
  int n1=t.length;
  double x[]=new double[101];
  double T[][]=new double[n1][101];
  double q[][]=new double[n1][101];
  HT_plane_wall2 htpw=new HT_plane_wall2(h1,h2,L,k,alpha,Ti,Tinf);
  //initial temperature variation t=0
  htpw.F=xx->(Ti-Tinf);
  for(int j=0;j<n1;j++)
  {
  for(int i=0;i<=100;i++)
      {   x[i]=i*L/100;
                  T[j][i]=htpw.T(x[i],t[j]);
                  q[j][i]=htpw.q(x[i],t[j]);
      }
   }
      Plot pp=new Plot(x,T[0]);
      for(int j=1;j<n1;j++)
      {pp.addData(x,T[j]);}
      pp.setPlabel("Steel wall one side insulated other side convective heat transfer");
      pp.setXlabel(" x m");
      pp.setYlabel(" T degree C");
      pp.plot();
      Plot pp1=new Plot(x,q[0]);
      for(int j=1;j<n1;j++)
      {pp1.addData(x,q[j]);}
      pp1.setPlabel("Steel wall one side insulated other side convective heat transfer");
      pp1.setXlabel(" x m");
      pp1.setYlabel(" q W/m^2");
      pp1.plot();
}
}
```



Python version:

```
 from math import *
from f_x import *;
import matplotlib.pyplot as plt

def bisection(f,a,b):
    b1=1.1*b
    r=(a+b)/2.0
    fr=f.func(r)
    fa=f.func(a)
    eps=1.0e-8;
    nmax=100;
    i=1;
    while abs(fr)>eps and i<nmax:
        if fa*fr<0:
            b=r
        else:
            a=r
            fa=fr
        r=(a+b)/2.0;
        fr=f.func(r)
        i=i+1
    if i>=nmax:
      r=bisection(f,a,b1)
    return r

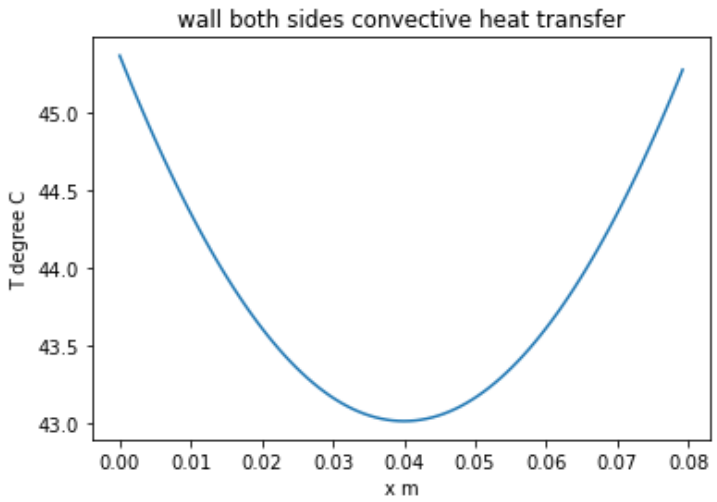
def gauss_legendre_coefficients(x1,x2,n):
    #calculates legendre gauss-coefficients as coefficients of the integral
    #for n terms
        eps=3e-15
        m=int((n+1.0)/2.0)
        xm=0.5*(x2+x1)
        xl=0.5*(x2-x1)
        nmax=100
        a=[[0.0 for i in range(n)] for j in range(2)]
        for i in range(1,m+1):
            z=cos(pi*((i-0.25)/(n+0.5)))
            for ii in range(nmax):
                #while abs(z-z1) > eps:
                p1=1.0;
                p2=0.0;
                for j in range(1,n+1):
                    p3=p2
                    p2=p1
                    p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
                pp=(float)(n*(z*p1-p2)/(z*z-1.0))
                z1=z;
                z=z1-p1/pp
                if abs(z-z1) < eps: break
            a[0][i-1]=xm-xl*z
            a[0][n-i]=xm+xl*z
            a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
            a[1][n-i]=a[1][i-1]
        return a


def integral(f,x1,x2,n):
    #integral func(x)dx
    #integral of a function by using gauss-legendre quadrature
    #between x1 and x2
    a=gauss_legendre_coefficients(x1,x2,n)
    z=0
    for i in range(n):
        z=z+a[1][i]*f.func(a[0][i])
    return z


def ksi(n,Bi1,Bi2):
    # eigenvalue function to be solved
    x1=n*pi+0.00000001
    x2=x1+(pi/2.0-0.00000001)
    class f1(f_x):func=lambda self,x: (x*x-Bi1*Bi2)*tan(x)-x*(Bi1+Bi2)
    f = f1()
    y= bisection(f,x1,x2)
    return y
```

```
def tetha(F,h1,h2,k,alpha,L,x,t):
    beta=0.0
    C=0.0
    N=0.0
    teta=0.0
    H1=h1/k
    H2=h2/k
    Bi1=H1*L
    Bi2=H2*L
    for n in range(20):
        k1=ksi(n,Bi1,Bi2)
        beta=k1/L
        N=L/2.0*(beta*beta+H1*H1)+sin(2.0*beta*L)/(4*beta)*(beta*beta-H1*H1)+H1/2.0*(1.0-cos(2.0*beta*L))
        class Cn(f_x):func=lambda self,x:F.func(x)*(beta*cos(beta*x)+H1*sin(beta*x))
        fCn=Cn()
        C=1.0/N*integral(fCn,0.0,L,20)
        teta=teta+C*(beta*cos(beta*x)+H1*sin(beta*x))*exp(-beta*beta*alpha*t)
    return teta

h1=500
h2=500
k=63.9
alpha=18.8e-6
L=2.0*40e-3
t=8*60
Ti=-20
Tinf=60
class F(f_x):func=lambda self,x: Ti-Tinf
F1 = F()
n=100
T=[0.0 for i in range(n)]
x=[0.0 for i in range(n)]
for i in range(n):
    x[i]=i*L/n
    T[i]=tetha(F1,h1,h2,k,alpha,L,x[i],t)+Tinf
plt.plot(x,T)
plt.xlabel("x m")
plt.ylabel("T degree C")
plt.title("wall both sides convective heat transfer")
plt.show()
```



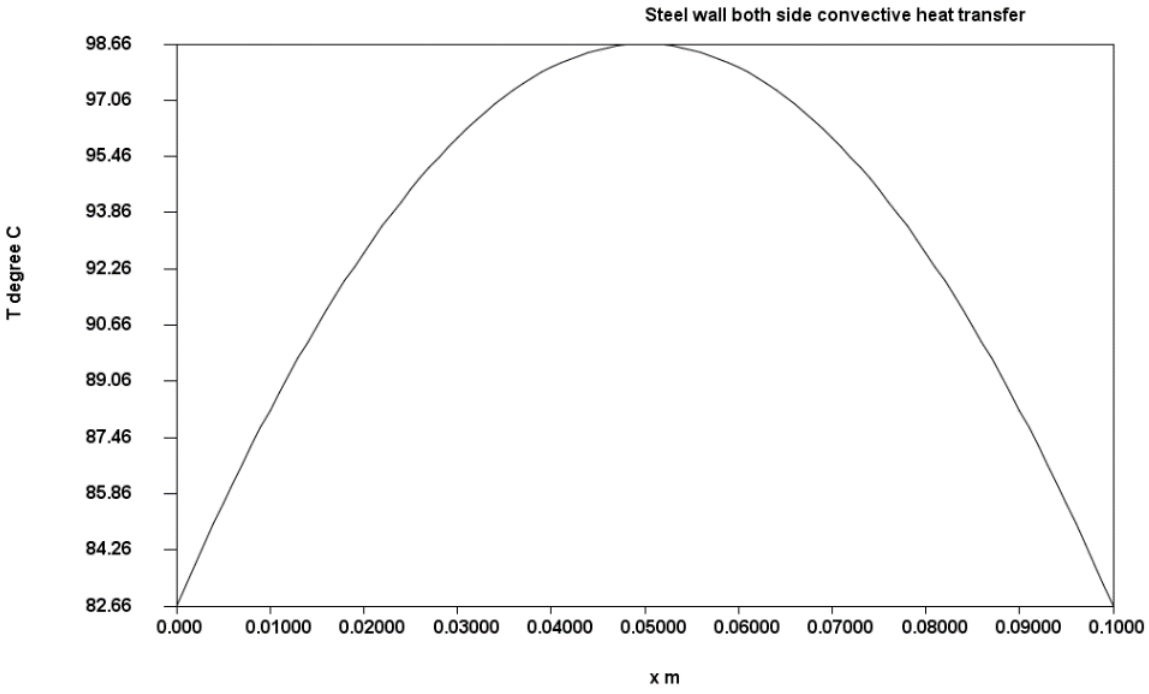
Numerical example:

A steel plate [α=1.2e-5 and k=43 W/(mK)] of thickness L=0.1 m, initially at a uniform temperature of 240 ℃, is suddenly immersed in an oil bath at $T_\infty = 40$ ℃. The convective heat transfer coefficients are h_1=600 W/(m^{2}K), h_2=600 W/(m^{2}K) calculate temperature profile for t=478.8 seconds.

F(x)=240-40=200 K

```
 public class HT_plane_wall2_test1
{
public static void main (String args[])
{ double h1=600; //W/m^2K
  double h2=600; //W/m^2K
```

```
  double k=42;
  double alpha=1.2e-5;
  double L=0.1;
  double t[]={478.8};  //s
  double Tinf=40;
  double Ti=240;
  int n1=t.length;
  double x[]=new double[101];
  double T[][]=new double[n1][101];
  double q[][]=new double[n1][101];
  HT_plane_wall2 htpw=new HT_plane_wall2(h1,h2,L,k,alpha,Ti,Tinf);
  //initial temperature variation t=0
  htpw.F=xx->(Ti-Tinf);
  for(int j=0;j<n1;j++)
  {
  for(int i=0;i<=100;i++)
      {   x[i]=i*L/100;
                  T[j][i]=htpw.T(x[i],t[j]);
                  q[j][i]=htpw.q(x[i],t[j]);
      }
   }
      Plot pp=new Plot(x,T[0]);
      for(int j=1;j<n1;j++)
      {pp.addData(x,T[j]);}
      pp.setPlabel("Steel wall both side convective heat transfer");
      pp.setXlabel(" x m");
      pp.setYlabel(" T degree C");
      pp.plot();
      Plot pp1=new Plot(x,q[0]);
      for(int j=1;j<n1;j++)
      {pp1.addData(x,q[j]);}
      pp1.setPlabel("Steel wall both side convective heat transfer");
      pp1.setXlabel(" x m");
      pp1.setYlabel(" q W/m^2");
      pp1.plot();
 }
 }
```

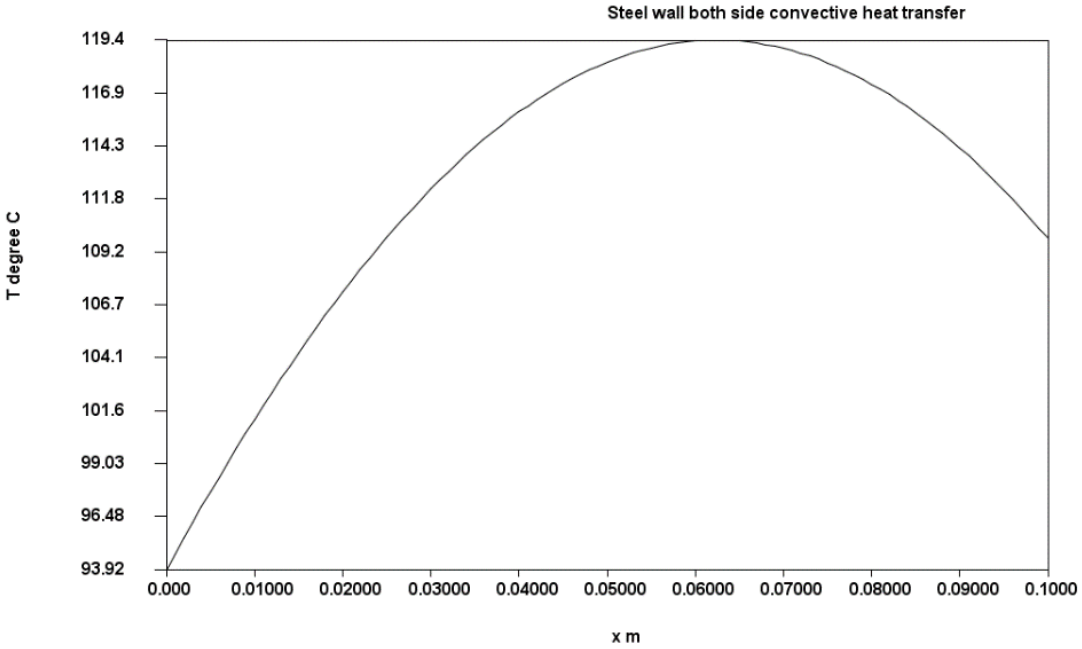


Numerical example:

A steel plate [α=1.2e-5 and k=43 W/(mK)] of thickness L=0.1 m, initially at a uniform temperature of 240 ℃, is suddenly immersed in an oil bath at $T_\infty = 40$ ℃. The convective heat transfer coefficients are h_1=600 W/(m^{2}K) in one side of the plate and, h_2=300 W/(m^{2}K) on the other side. Calculate temperature profile for t=478.8 seconds.

F(x)=240-40=200 K

```
  public static void main (String args[])
   { double L=0.1;
     double h1=600.0; //W/m^2K
     double h2=300.0; //W/m^2K
     double k=43;  //W/mK
     double tinf=40;
     double H2=h2/k;
     double alfa=1.2e-5;
     double t=478.8;
     double H1=h1/k;;
     double x[]=new double[101];
     double T[]=new double[101];
     HT_duvar2 ht=new HT_duvar2();
     for(int i=0;i<=100;i++)
     {   x[i]=i*L/100;
         T[i]=ht.teta(L,H1,H2,alfa,x[i],t,tinf);
         / / search start from initial value again
          ht.x1=0.00001; ht.ix=0;
     }
     Plot pp=new Plot(x,T);
     pp.setPlabel("Steel wall both side convective heat transfer t="+t+" seconds");
     pp.setXlabel(" x m");
     pp.setYlabel(" T degree C");
     pp.plot();
   }
```



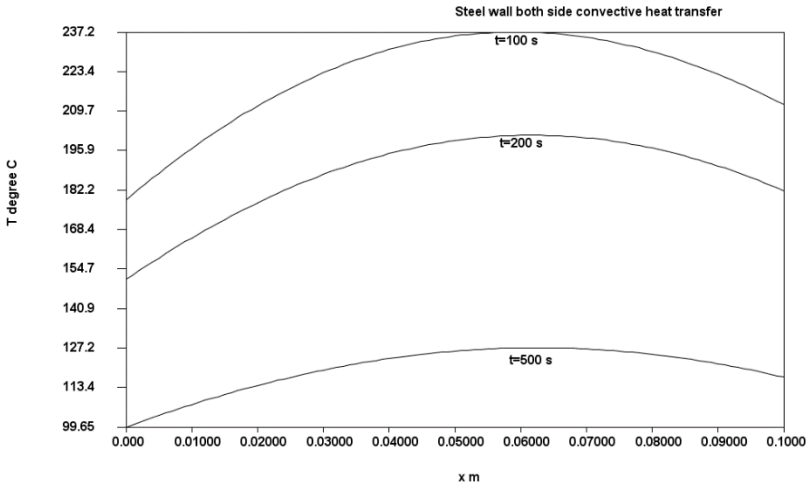
Numerical example:

A steel plate [α=1.2e-5 and k=43 W/(mK)] of thickness L=0.1 m, initially at a uniform temperature of 240 ℃, is suddenly immersed in an oil bath at T_∞ = 40 ℃. The convective heat transfer coefficients are h_1=600 W/(m^2K) at one side of the plate and, h_2=300 W/(m^2K) at the other side of the plate. Calculate temperature profile for t=100, 200 and 500 seconds.

F(x)=240-40=200 K

```
 public class HT_plane_wall2_test1
{
public static void main (String args[])
{ double h1=600; //W/m^2K
  double h2=300; //W/m^2K
  double k=42;
  double alpha=1.2e-5;
  double L=0.1;
  double t[]={478.8};  //s
  double Tinf=40;
  double Ti=240;
  int n1=t.length;
```

```
  double x[]=new double[101];
  double T[][]=new double[n1][101];
  double q[][]=new double[n1][101];
  HT_plane_wall2 htpw=new HT_plane_wall2(h1,h2,L,k,alpha,Ti,Tinf);
  //initial temperature variation t=0
  htpw.F=xx->(Ti-Tinf);
  for(int j=0;j<n1;j++)
  {
  for(int i=0;i<=100;i++)
      {   x[i]=i*L/100;
                  T[j][i]=htpw.T(x[i],t[j]);
                  q[j][i]=htpw.q(x[i],t[j]);
      }
   }
      Plot pp=new Plot(x,T[0]);
      for(int j=1;j<n1;j++)
      {pp.addData(x,T[j]);}
      pp.setPlabel("Steel wall both side convective heat transfer");
      pp.setXlabel(" x m");
      pp.setYlabel(" T degree C");
      pp.plot();
      Plot pp1=new Plot(x,q[0]);
      for(int j=1;j<n1;j++)
      {pp1.addData(x,q[j]);}
      pp1.setPlabel("Steel wall both side convective heat transfer");
      pp1.setXlabel(" x m");
      pp1.setYlabel(" q W/m^2");
      pp1.plot();
  }
  }
```



PROBLEM:

A steel plate [α=1.2e-5 and k=43 W/(mK)] of thickness L=0.1 m, initially has a temperature profile as;

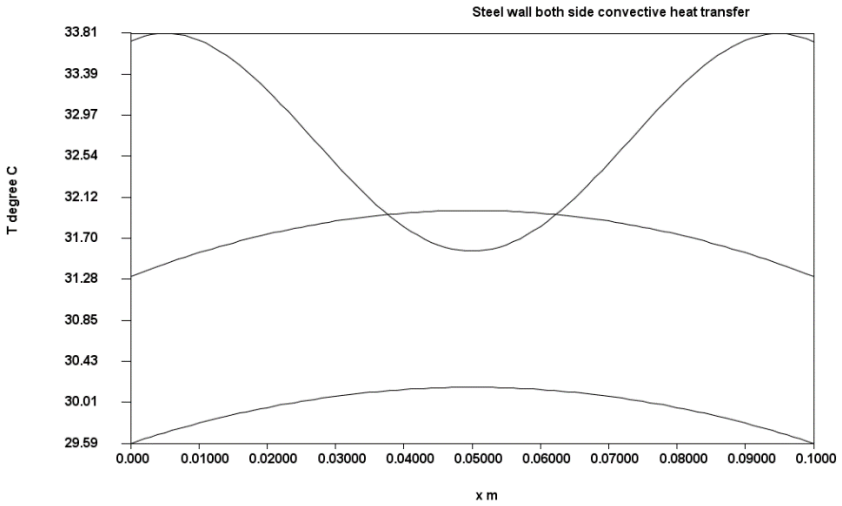
0	60
0.01	45.6
0.02	34.4
0.03	26.4
0.04	21.6
0.05	20
0.06	21.6
0.07	26.4
0.08	34.4
0.09	45.6
0.1	60

is suddenly immersed in an oil bath at $T_\infty = 20$ ℃. The convective heat transfer coefficients are h_1=100 W/(m^2K), h_2=100 W/(m^2K) calculate temperature profile for t=50, 200 and 500 seconds.

F(x)=?

```
public class HT_plane_wall2_test2
{
public static void main (String args[])
{ double h1=100; //W/m^2K
  double h2=100; //W/m^2K
  double k=42;
  double alpha=1.2e-5;
```

```
  double L=0.1;
  double t[]={50,200,500};  //s
  double Tinf=20;
  double Ti=60;
  int n1=t.length;
  double x[]=new double[101];
  double T[][]=new double[n1][101];
  double q[][]=new double[n1][101];
  HT_plane_wall2 htpw=new HT_plane_wall2(h1,h2,L,k,alpha,Ti,Tinf);
  //initial temperature variation t=0
  htpw.F=xx->16000*xx*xx-1600*xx+60-Tinf;
  for(int j=0;j<n1;j++)
  {
  for(int i=0;i<=100;i++)
      {   x[i]=i*L/100;
                  T[j][i]=htpw.T(x[i],t[j]);
                  q[j][i]=htpw.q(x[i],t[j]);
      }
   }
      Plot pp=new Plot(x,T[0]);
      for(int j=1;j<n1;j++)
      {pp.addData(x,T[j]);}
      pp.setPlabel("Steel wall both side convective heat transfer");
      pp.setXlabel(" x m");
      pp.setYlabel(" T degree C");
      pp.plot();
      Plot pp1=new Plot(x,q[0]);
      for(int j=1;j<n1;j++)
      {pp1.addData(x,q[j]);}
      pp1.setPlabel("Steel wall both side convective heat transfer");
      pp1.setXlabel(" x m");
      pp1.setYlabel(" q W/m^2");
      pp1.plot();
}
}
```



Python version:

```
from math import *
from f_x import *;
import matplotlib.pyplot as plt

def bisection(f,a,b):
    b1=1.1*b
    r=(a+b)/2.0
    fr=f.func(r)
    fa=f.func(a)
    eps=1.0e-8;
    nmax=100;
    i=1;
    while abs(fr)>eps and i<nmax:
        if fa*fr<0:
            b=r
        else:
            a=r
```

```
            fa=fr
        r=(a+b)/2.0;
        fr=f.func(r)
        i=i+1
    if i>=nmax:
      r=bisection(f,a,b1)
    return r

def gauss_legendre_coefficients(x1,x2,n):
    #calculates legendre gauss-coefficients as coefficients of the integral
    #for n terms
        eps=3e-15
        m=int((n+1.0)/2.0)
        xm=0.5*(x2+x1)
        xl=0.5*(x2-x1)
        nmax=100
        a=[[0.0 for i in range(n)] for j in range(2)]
        for i in range(1,m+1):
            z=cos(pi*((i-0.25)/(n+0.5)))
            for ii in range(nmax):
                #while abs(z-z1) > eps:
                p1=1.0;
                p2=0.0;
                for j in range(1,n+1):
                    p3=p2
                    p2=p1
                    p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
                pp=(float)(n*(z*p1-p2)/(z*z-1.0))
                z1=z;
                z=z1-p1/pp
                if abs(z-z1) < eps: break
            a[0][i-1]=xm-xl*z
            a[0][n-i]=xm+xl*z
            a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
            a[1][n-i]=a[1][i-1]
        return a

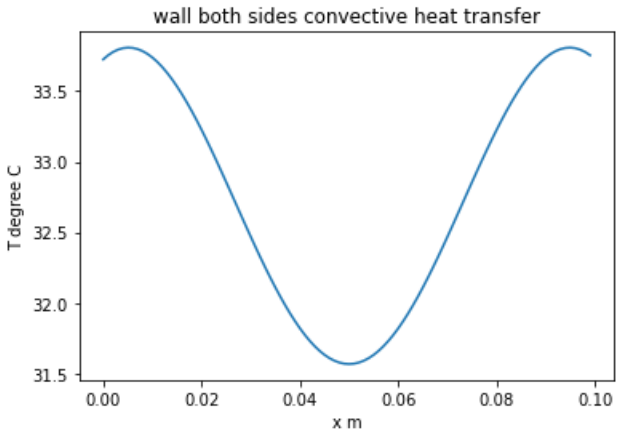

def integral(f,x1,x2,n):
    #integral func(x)dx
    #integral of a function by using gauss-legendre quadrature
    #between x1 and x2
    a=gauss_legendre_coefficients(x1,x2,n)
    z=0
    for i in range(n):
        z=z+a[1][i]*f.func(a[0][i])
    return z


def ksi(n,Bi1,Bi2):
    # eigenvalue function to be solved
    x1=n*pi+0.00000001
    x2=x1+(pi/2.0-0.00000001)
    class f1(f_x):func=lambda self,x: (x*x-Bi1*Bi2)*tan(x)-x*(Bi1+Bi2)
    f = f1()
    y= bisection(f,x1,x2)
    return y

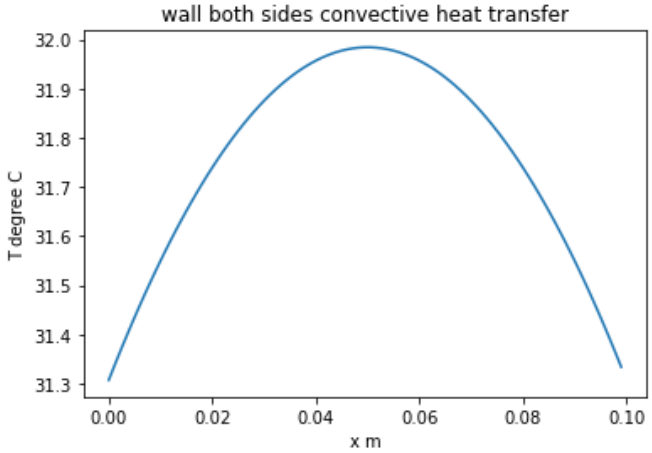
def tetha(F,h1,h2,k,alpha,L,x,t):
    beta=0.0
    C=0.0
    N=0.0
    teta=0.0
    H1=h1/k
    H2=h2/k
    Bi1=H1*L
    Bi2=H2*L
    for n in range(20):
        k1=ksi(n,Bi1,Bi2)
        beta=k1/L
        N=L/2.0*(beta*beta+H1*H1)+sin(2.0*beta*L)/(4*beta)*(beta*beta-H1*H1)+H1/2.0*(1.0-cos(2.0*beta*L))
        class Cn(f_x):func=lambda self,x:F.func(x)*(beta*cos(beta*x)+H1*sin(beta*x))
        fCn=Cn()
        C=1.0/N*integral(fCn,0.0,L,20)
        teta=teta+C*(beta*cos(beta*x)+H1*sin(beta*x))*exp(-beta*beta*alpha*t)
    return teta
```

```
h1=100.0
h2=100.0
k=42
alpha=1.2e-5
L=0.1
t=50
Ti=60
Tinf=20
class F(f_x):func=lambda self,x: 16000.0*x*x-1600.0*x+60.0-Tinf
F1 = F()
n=100
T=[0.0 for i in range(n)]
x=[0.0 for i in range(n)]
for i in range(n):
    x[i]=i*L/n
    T[i]=tetha(F1,h1,h2,k,alpha,L,x[i],t)+Tinf
plt.plot(x,T)
plt.xlabel("x m")
plt.ylabel("T degree C")
plt.title("wall both sides convective heat transfer")
plt.show()
```

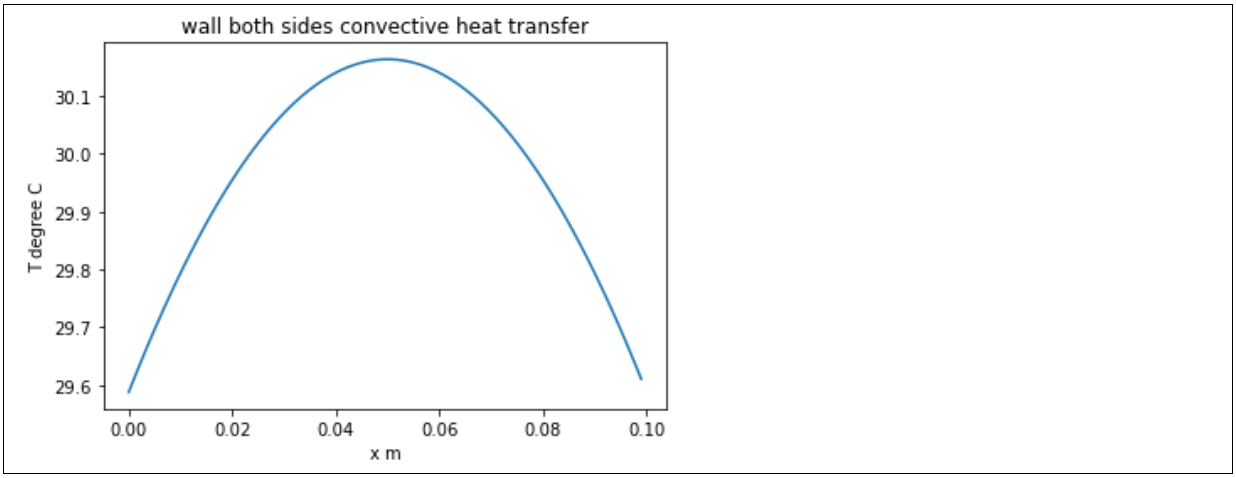
T=50 seconds



T=200 seconds

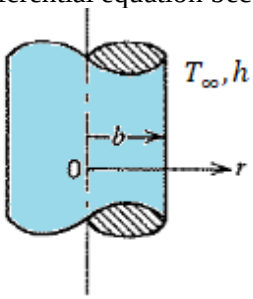


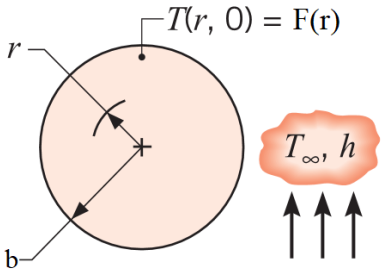
T=500 seconds



PROBLEM 3.5 SEPARATION OF VARIABLES METHOD,HOMOGENEOUS SOLUTION, TIME DEPENDENT; CYLINDRICAL COORDINATES

Problems defines a solid rod of radius r, outside exposed to convective heat transfer of $q_{convection_b} = h_1 T(b,t) = h_1(T^*(b,t) - T_\infty^*)$. By defining temperature $T(r,t) = T^*(r,t) - T_\infty^*$ differential equation becomes homogeneous.





$\frac{\partial^2 T^*(r,t)}{\partial r^2} + \frac{1}{r}\frac{\partial T^*(r,t)}{\partial r} = \frac{1}{\alpha}\frac{\partial T^*(r,t)}{\partial t}$ Boundary conditions: $\frac{\partial T^*(r,t)}{\partial r} = 0$ r=0 t >0 $H = \frac{h}{k}$ $\frac{\partial T^*(r,t)}{\partial r} + H(T^*(r,t) - T_\infty) = 0$ r = b t > 0 $T^*(r,t) = F(r) + T_\infty$ t=0 $0 \le x \le b$	$\frac{\partial^2 T(r,t)}{\partial r^2} + \frac{1}{r}\frac{\partial T(r,t)}{\partial r} = \frac{1}{\alpha}\frac{\partial T(r,t)}{\partial t}$ $T(r,t) = T^*(r,t) - T_\infty$ Boundary conditions: $\frac{\partial T(r,t)}{\partial r} = 0$ r=0 t >0 $H = \frac{h}{k}$ $\frac{\partial T(r,t)}{\partial r} + HT(r,t) = 0$ r = b t > 0 $T(r,t) = F(r)$ t=0 $0 \le x \le b$

Separation of variables:

$T(r,t) = R(r)\Gamma(t)$

$\Gamma(t)\left[\frac{\partial^2 R(r)}{\partial r^2} + \frac{1}{r}\frac{\partial R(r)}{\partial r}\right] = \frac{R(r)}{\alpha}\frac{\partial \Gamma(t)}{\partial t}$

$\frac{1}{R(r)}\left[\frac{\partial^2 R(r)}{\partial r^2} + \frac{1}{r}\frac{\partial R(r)}{\partial r}\right] = \frac{1}{\alpha\Gamma(t)}\frac{\partial \Gamma(t)}{\partial t} = -\beta^2$

$\frac{\partial^2 R(r)}{\partial r^2} + \frac{1}{r}\frac{\partial R(r)}{\partial r} + \beta^2 R(r) = 0$

$\frac{\partial \Gamma(t)}{\partial t} + \beta^2 \alpha \Gamma(t) = 0$

Boundary conditions:

$\frac{\partial R(r)}{\partial r} = 0$ r=0 t >0 $H = \frac{h}{k}$

$\frac{\partial R(r)}{\partial r} + HR(r) = 0$ r = b t > 0

$T(r,t) = F(r)$ t=0 $\quad 0 \le r \le b$

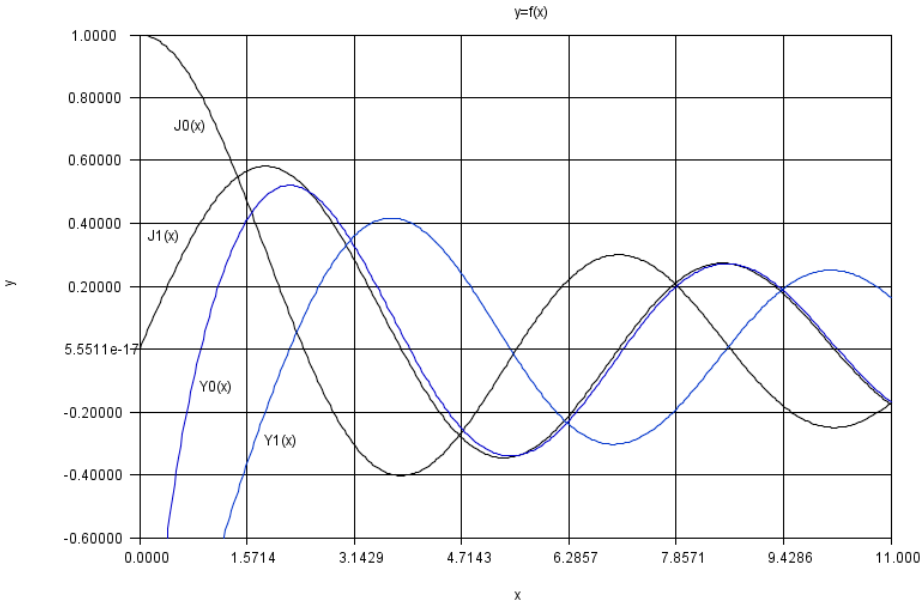
Solution:

$R(r) = C_1 J_0(\beta r) + C_2 Y_0(\beta r)$

$\Gamma(t) = C_3 \exp(-\beta^2 \alpha t)$

$T(r,t) = [C_1 J_0(\beta r) + C_2 Y_0(\beta r)] \exp(-\beta^2 \alpha t)$

J_0 and Y_0 in this equation called Bessel functions. In the figure, behavior of bessel functions are given as a plot.



As it is seen from the plot, $J_0(0) = 1$, $J_1(0) = 0$, $Y_0(0) = -\infty$, $Y_1(0) = -\infty$

Before going any further, let us look at some properties of bessel functions. Bessel functions can be defined as series. Bessel function $J_\nu(z)$ can be defined by using the following series:

$$J_\nu(z) = \left(\frac{z}{2}\right)^\nu \sum_{k=0}^{\infty} (-1)^k \frac{\left(\frac{z}{2}\right)^{2\kappa}}{k!\,\Gamma(\nu+k+1)}$$

Where gamma function defined as

$$\Gamma(z) = \int_{z=0}^{\infty} t^{z-1} e^{-t} dt = \frac{e^{-\gamma t}}{t} \prod_{n=1}^{\infty} \left(1 + \frac{t}{n}\right)^{-1} e^{t/n}$$

Where γ=0.57721566490153286... is the Euler-Mascheroni constant. The Bessel function $Y_\nu(z)$ is given by:

$$Y_\nu(z) = \frac{J_\nu(z)\cos(\nu\pi) - J_{-\nu}(z)}{\sin(\nu\pi)}$$

When ν=n is a positive integer, the solution $J_n(z)$ and $J_{-n}(z)$ are not independent: they are related by:

$J_n(z) = (-1)^n J_{-n}(z)$

Derivatives of the J or Y bessel functions can be given as:

$$\frac{d}{dz}[z^\nu J_\nu(\beta z)] = \beta z^\nu J_{\nu-1}(\beta z)$$

$$\frac{d}{dz}[J_0(\beta z)] = -\beta J_1(\beta z)$$

Integrals of the J or Y bessel functions can be given as:

$$\int [z^\nu J_{\nu-1}(\beta z)]\, dz = \frac{1}{\beta} z^\nu J_\nu(\beta z)$$

$$\int [z J_0(\beta z)]\, dz = \frac{1}{\beta} z J_1(\beta z)$$

$\int [z J_0^2(z)] dz = \frac{z^2}{2} z \left(J_0^2(z) + J_1^2(z)\right)$ After this introduction to bessel functions, boundary conditions can be applied now:

$\frac{\partial R(r)}{\partial r} = -C_1\beta J_1(\beta r) - C_2\beta Y_1(\beta r)$

$\frac{\partial R(r)}{\partial r} = 0$ r=0 t>0

$\frac{\partial R(r)}{\partial r} = -C_1\beta J_1(0) - C_2\beta Y_1(0) = 0$

$\frac{\partial R(r)}{\partial r} = -C_1\beta 0 - C_2\beta Y_1(\infty) = 0 \quad \rightarrow C_2 = 0$

$\frac{\partial R(r)}{\partial r} + HR(r) = 0$ r=b t>0

$-C_1\beta J_1(\beta b) + HC_1 J_0(\beta b) = 0 \quad \rightarrow C_1 \neq 0$

$-\beta_m J_1(\beta_m b) + HJ_0(\beta_m b) = 0$ This equation has multiple roots.

Equation can also be written as:

$-\beta_m b J_1(\beta_m b) + HbJ_0(\beta_m b) = 0$ or $-\xi_m J_1(\xi_m) + BiJ_0(\xi_m)$

where $\xi_m = \beta_m b$ and $Bi = Hb = \frac{hb}{k}$ is Biot number

$$T(r,t) = \sum_{m=1}^{\infty} C_m J_0(\beta_m r) \exp(-\beta_m^2 \alpha t)$$

$T(r,t) = F(r)$ t=0 $0 \leq r \leq b$

$F(r) = \sum_{m=1}^{\infty} C_m J_0(\beta_m r)$

This equation has n coefficients. In order to solve these coefficients, we will use properties of orthogonal functions.

$$\int_0^b r J_0(\beta_m r) J_0(\beta_n r) dr = \begin{cases} 0 & m \neq n \\ N(\beta_m) & m = n \end{cases}$$

$\int_0^b r F(r) J_0(\beta_m r) dr = C_m \int_0^b r J_0^2(\beta_m r) dr$

$N(\beta_m) = \int_{r=0}^b r J_0^2(\beta_m r) dr \qquad z = \beta_m r \qquad dz = \beta_m dr$

$$N(\beta_m) = \frac{1}{\beta_m^2} \int_{z=0}^{b\beta_m} z J_0^2(z) dz = \frac{1}{\beta_m^2} \frac{z^2}{2} [J_0^2(z) + J_1^2(z)] \Big|_{z=0}^{\beta_m b} = \frac{b^2}{2} [J_0^2(\beta_m b) + J_1^2(\beta_m b)]$$

Or alternatively due to boundary condition $-\beta_m b J_1(\beta_m b) + HbJ_0(\beta_m b) = 0 \rightarrow J_1(\beta_m b) = \frac{HbJ_0(\beta_m b)}{\beta_m b}$

$$N(\beta_m) = \frac{b^2}{2}\left[J_0^2(\beta_m b) + \left(\frac{HbJ_0(\beta_m b)}{\beta_m b}\right)^2\right] = \frac{b^2}{2}\left[J_0^2(\beta_m b) + \frac{H^2 J_0^2(\beta_m b)}{\beta_m^2}\right] = \frac{J_0^2(\beta_m b) b^2}{2}\left[\frac{\beta_m^2 + H^2}{\beta_m^2}\right]$$

$\frac{1}{N(\beta_m)} = \frac{2}{J_0^2(\beta_m b)} \frac{\beta_m^2}{b^2(\beta_m^2 + H^2)}$

$C_m = \frac{1}{N(\beta_m)} \int_0^b r F(r) J_0(\beta_m r) dr$

If $F(r) = T_i = const$

$\int_0^b r T_i J_0(\beta_m r) dr = \frac{T_i}{\beta_m} r J_1(\beta_m r)|_{r=0}^b = \frac{T_i b}{\beta_m} J_1(\beta_m b)$

$\frac{T_i b}{\beta_m} J_1(\beta_m b) = C_m N(\beta_m) = C_m \frac{b^2}{2}[J_0^2(\beta_m b) + J_1^2(\beta_m b)]$

$C_m = \frac{2T_i}{\beta_m b} \frac{J_1(\beta_m b)}{J_0^2(\beta_m b) + J_1^2(\beta_m b)} \qquad \xi_m = \beta_m b \quad C_m = \frac{2T_i}{\xi_m} \frac{J_1(\xi_m)}{J_0^2(\xi_m) + J_1^2(\xi_m)}$

Depends on value of F(r), this integration can be solved by using numerical or analytical methods.

$$T^*(r,t) - T_\infty = T(r,t) = \sum_{m=1}^{\infty} C_m J_0(\beta_m r) \exp(-\beta_m^2 \alpha t)$$

$$= \sum_{m=1}^{\infty} \left[\frac{1}{N(\beta_m)} \int_0^b r F(r) J_0(\beta_m r) dr\right] J_0(\beta_m r) \exp(-\beta_m^2 \alpha t)$$

In this case heat transfer will be:

$$q = -k\frac{dT(r,t)}{dr} = \sum_{m=1}^{\infty} kC_m\beta_m J_1(\beta_m r)\exp(-\beta_m^2\alpha t)$$

Before solving the entire problem eigenvalues will be solved by a small program and listed.

```
 import java.io.*;
import javax.swing.*;
import java.awt.*;
import static java.lang.Math.*;
class bisection
{
//J Bessel function of integer order 0
public static double J0(double x)
{   double ax,z;
        double xx,y,ans,ans1,ans2;
        if ((ax=Math.abs(x)) < 8.0) {
                y=x*x;
                ans1=57568490574.0+y*(-13362590354.0+y*(651619640.7
                        +y*(-11214424.18+y*(77392.33017+y*(-184.9052456)))));
                ans2=57568490411.0+y*(1029532985.0+y*(9494680.718
                        +y*(59272.64853+y*(267.8532712+y*1.0))));
                ans=ans1/ans2;
        } else {
                z=8.0/ax;
                y=z*z;
                xx=ax-0.785398164;
                ans1=1.0+y*(-0.1098628627e-2+y*(0.2734510407e-4
                        +y*(-0.2073370639e-5+y*0.2093887211e-6)));
                ans2 = -0.1562499995e-1+y*(0.1430488765e-3
                        +y*(-0.6911147651e-5+y*(0.7621095161e-6
                        -y*0.934935152e-7)));
                ans=sqrt(0.636619772/ax)*(cos(xx)*ans1-z*sin(xx)*ans2);
        }
        return ans;
}
//J Bessel function of integer order 1
public static double J1(double x)
{
        double ax,z;
        double xx,y,ans,ans1,ans2;

        if ((ax=Math.abs(x)) < 8.0) {
                y=x*x;
                ans1=x*(72362614232.0+y*(-7895059235.0+y*(242396853.1
                        +y*(-2972611.439+y*(15704.48260+y*(-30.16036606))))));
                ans2=144725228442.0+y*(2300535178.0+y*(18583304.74
                        +y*(99447.43394+y*(376.9991397+y*1.0))));
                ans=ans1/ans2;
        } else {
                z=8.0/ax;
                y=z*z;
                xx=ax-2.356194491;
                ans1=1.0+y*(0.183105e-2+y*(-0.3516396496e-4
                        +y*(0.2457520174e-5+y*(-0.240337019e-6))));
                ans2=0.04687499995+y*(-0.2002690873e-3
                        +y*(0.8449199096e-5+y*(-0.88228987e-6
                        +y*0.105787412e-6)));
                ans=sqrt(0.636619772/ax)*(cos(xx)*ans1-z*sin(xx)*ans2);
                if (x < 0.0) ans = -ans;
        }
        return ans;
}
public static void main (String args[]) throws java.io.IOException
{    double r;
    char beta='\u03B2';
    char ksi='\u03B6';
    double Bi[]={0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,2,3,4,5,6,7,8,9,10,20,30,40,50,100};
    String name="Eigenvalues "+ksi+"n * J1("+ksi+"n)=Bi* J0("+ksi+"n)  or "+beta+"nb * J1("+beta+"nb)=Bi* J0("+beta+"nb)";
    String sBi[]=new String[Bi.length];
    String sC[]=new String[11];
    double A[][]=new double[Bi.length][11];
    for(int i=0;i<Bi.length;i++)
    {A[i][0]=Bi[i];}
    sC[0]="Bi=Hb=hb/k";
```

```
    for(int i=1;i<11;i++)
    {sC[i]=""+ksi+i+" = "+beta+i+"*b";}
    for(int i=0;i<Bi.length;i++)
    { final double bi=Bi[i];
      if_x f1=x->x*J1(x)-bi*J0(x);
    double x0=0;
    double dx=0.0001;
    double x1=0;
    for(int n=1;n<11;n++)
    {
    x1=f1.enlarge(x0,dx);
    r= f1.bisection(x0,x1);
    A[i][n]=r;
    x0=r+dx;
    }
   }
   Text.print(A,sC,name);
}}
```

Eigenvalues ζn * J1(ζn)=Bi* J0(ζn) or βnb * J1(βnb)=Bi* J0(βnb)

Bi=Hb=hb/k	ζ1 = β1*b	ζ2 = β2*b	ζ3 = β3*b	ζ4 = β4*b	ζ5 = β5*b	ζ6 = β6*b	ζ7 = β7*b	ζ8 = β8*b	ζ9 = β9*b	ζ10 = β10*b
0.01	0.14124438702202424	3.834314632249809	7.017011746616954	10.174451080675095	13.324442509503786	16.471237133801893	19.616368435731 69	22.76052392161146	25.904058346767 14	29.047173004683046
0.02	0.19949734169608846	3.8369215589515364	7.018436719724432	10.17543382511955	13.325192819188775	16.471844460811 37	19.616878043619 37	22.760963191495527	25.904444325067637	29.0475170919872
0.03	0.24403531415247603	3.839527182895581	7.019861355482003	10.17641653927 11	13.32594357463093	16.472451532114572	19.617387764218 15	22.761402584161385	25.904830438604883	29.04786132818 63
0.04	0.281434255337335	3.84213113225548 85	7.021286299591365	10.17739926328 6501	13.326693651736012	16.473058500324736	19.617897779111 38	22.76184179504383	25.905216341059 21	29.048205665204684
0.05	0.3142624617593016	3.8447315601894303	7.022709522378595	10.17838174570 4364	13.327443898858538	16.473665642199645	19.618407119830138	22.76228124671 34	25.905602141844 98	29.048549966740115
0.06	0.34382824132569034	3.847332205774882	7.024133921668673	10.17936370559209	13.328194536767967	16.474272461040684	19.6189169258261	22.762720199722125	25.905988113407666	29.048893954925077
0.07	0.37091554534753834	3.84992938895705	7.025556750076 28	10.18034653773546	13.328944446537767	16.474879500276536	19.619426978365887	22.763159945666878	25.906374312377626	29.04923827632775
0.08	0.396035425390744	3.8525254516938 27	7.026979667131505	10.18132816997 1463	13.329695039887277	16.475486577102366	19.619936194780095	22.763599229730698	25.906760280527394	29.04958267645783
0.09	0.41953607673331894	3.8551178865234057	7.028402724359 99	10.18231046556 6292	13.330445220256248	16.476093493513392	19.620446069997868	22.764038239346 89	25.907146231004894	29.049926714179698
0.1	0.44168202620040453	3.857709635251578	7.029825389657756	10.18329265640 9002	13.331195161066617	16.476700078451117	19.620955535371 81	22.764477796276317	25.907532201894497	29.050271112131163
0.2	0.6169732894317739	3.8835055007230928	7.044029547609666	10.19310581382953	13.338693403125603	16.482767914022116	19.626051260913655	22.768869539903505	25.911391801908593	29.053713125252173
0.3	0.7464605051130706	3.9090647380738073	7.058193606434175	10.20290528108 1176	13.346185368251872	16.488832142282234	19.631145033802245	22.773260812013948	25.915250129835183	29.057154470776887
0.4	0.8515795607756881	3.9343610269278377	7.072312220369719	10.21269036662 2734	13.353670828488404	16.494893271160542	19.636236597390 25	22.777650286641197	25.919108069656975	29.060595459357657
0.5	0.9407713612644059	3.959371588745104	7.086380652101916	10.22245862657 9524	13.361149046452297	16.500949638182663	19.641326146180262	22.782038706512918	25.922964972197 19	29.06403536391538
0.6	1.0184417444598162	3.984074799987787	7.100394187771341	10.23220770337 0841	13.368618819567672	16.507002096280 63	19.646412967852573	22.786424732598903	25.926820212925 41	29.067474505801 01
0.7	1.0872539108246193	4.008451836704331	7.114347144726919	10.24193661705 2357	13.376078975928529	16.513049996609258	19.651496730092937	22.790809529454116	25.930674817760 38	29.070913014170667
0.8	1.1489720079369725	4.032487346381805	7.128235449432799	10.25164405044 2756	13.383529683891506	16.519092024798432	19.656577178107007	22.795192288962 12	25.934527569800 43	29.074350206763604
0.9	1.2048409334855388	4.056166745757099	7.142053980375918	10.26132723058 5215	13.390969224814475	16.525128406637 55	19.661654154235 8	22.799572315070876	25.938379174870 05	29.07778653446435
1.0	1.2557842296241377	4.079477840457206	7.155799268134112	10.27098534424 9315	13.398397255818805	16.531158746854643	19.666727630255487	22.803950610100742	25.942228678948 19	29.081221950405133
2.0	1.5994493658393016	4.2909581916677855	7.288388727533503	10.36583103180004	13.471882165561022	16.591032725101 79	19.717206438221552	22.847565608317 44	25.980614675016536	29.11549357459593
3.0	1.7886574563090574	4.463371406392698	7.410269783972261	10.45659874317 9503	13.543436480953545	16.649853594595 43	19.767050899543033	22.890769853153 9	26.018719861406836	29.14956541642165
4.0	1.9080785113777559	4.601845463753248	7.520070369540813	10.54228749400 3851	13.612476461498979	16.707268619496688	19.816036638734104	22.933413959068154	26.056440344388374	29.183361422384 03
5.0	1.9898144198944885	4.713141949805399	7.617707511018007	10.62230025803 9939	13.678558027742419	16.762983561435 78	19.863966428948 92	22.975361123498956	26.093678193179006	29.216810846555575
6.0	2.049011194318341	4.803311949438918	7.703886379154738	10.69638772433 4514	13.741377962936724	16.816763785434148	19.910670967521696	23.016489171618133	26.130344192026975	29.249846149496207
7.0	2.093731248705719	4.877167834704968	7.779697711800363	10.76457097414 4624	13.800759377195828	16.868434856637148	19.956011362655 56	23.056693528453238	26.166358959573 57	29.282406577069004
8.0	2.1286387336464525	4.938378965497794	7.846358034587544	10.82705962647 3815	13.856637010900307	16.917880358105823	19.999879844039064	23.095884762491 6	26.201651816285683	29.314437177859705
9.0	2.1566066766378382	4.989702944777896	7.905056641637977	10.88418157124 7845	13.909031211994224	16.965035156303212	20.042197105920 06	23.133991179458704	26.236162676281175	29.345888452837144
10.0	2.1794966093614576	5.033212088689643	7.956883498494168	10.93633008985 2158	13.958030525050916	17.009878339800 04	20.082910423361 22	23.170956891689592	26.269841545434 46	29.376717447808787
20.0	2.28804847501551 57	5.256810685530388	8.25341301893258	11.26766396621 4656	14.298318054966355	17.344204004844848	20.403686243663863	23.474927587898666	26.556135968262748	29.645700949015996
30.0	2.32614343637798 05	5.34098431678947	8.377067428121602	11.42214919217 5402	14.474796583703014	17.534811013737116	20.602043126426743	23.676198980927374	26.756844083677 9	29.843460942561123
40.0	2.3455180724465268	5.384601268782619	8.44315668909114	11.50805240861 8068	14.577392198857684	17.650845918890 6	20.728390988965764	23.810028323893206	26.895707183897 77	29.985315912283 68
50.0	2.3572420695716 94	5.411197026755691	8.483989094100505	11.56207840934 719	14.643305104074038	17.727179736751552	20.813614898802815	23.902625058614404	26.994235641036063	30.088450638808816
100.0	2.3809016587414 39	5.465206977161854	8.567831649098185	11.67473539791 0997	14.783420900872102	17.893136601237245	21.003600975493104	24.114699347668132	27.226387203083604	30.338652546149298

PROBLEM 3.5 MATLAB SOLUTION

```
function [f] = func3(beta,H,b)
        f=(beta*b)*besselj(1,beta*b)-H*b*besselj(0,beta*b);
 end
```

```
function [beta] = genislet(x0,H,b)
        x1=x0+0.01;
        x2= x1+0.00001;
        f1=func3(x1,H,b);
        f2=func3(x2,H,b);
        j=0;
        NTRY=200;
```

```
        FACTOR=1.01;
        while ((f1*f2>0) && (j<NTRY))
          x2=x2+FACTOR*(x2-x1);
          f2=func3(x2,H,b);
          j=j+1;
        end
        beta=bisection(@(x) func3(x,H,b),x1,x2);
 end
```

```
function xr=bisection(f,xl,xu)
% bisection root finding method
maxit=100;
iter=0;
es=0.000000001;
ea=1.1*es;
while((ea>es)&&(iter<maxit))
   xr=(xl+xu)/2.0;
   iter=iter+1;
   if xr~=0 ea=abs((xu-xl)/(xu+xl))*100;end
   fxl= f(xl);
   fxr= f(xr);
   test= fxl*fxr;
   if test == 0.0      ea=0;
   elseif test < 0.0  xu=xr;
   else               xl=xr;
   end
end
if(iter>=maxit) fprintf('Maximum number of iteration is exceeded result might not be valid","MAKSİMUM NUMBER OF ITERATION
WARNING');
end
end
```

```
function [beta] = beta3(m,H,b)
        x0=0;
        for i=1:m
        x0=genislet(x0,H,b);
        beta(i)=x0;
        end
 end
```

```
function [C] = CN3(beta,H,b)
        y=@(r) r.*F3(r,b).*besselj(0,beta*r);
        C1=quad(y,0,b);
        J0=besselj(0,beta*b);
        N=J0*J0/2*b*b*(H*H+beta*beta)/(beta*beta);
        C=C1/N;
 end
```

```
function [F] = F3(r,b)
        F=1.0;
 end
```

```
function  [T] = T3(r,t,H,b,alpha)
        T=0;
        m=2;
        beta=beta3(m,H,b);
        for i=1:m
        C1=CN3(beta(i),H,b)
        T=T+C1*besselj(0,(beta(i)*r))*exp(-beta(i)*beta(i)*alpha*t);
        end
end
```

PROBLEM 3.5 JAVA SOLUTION

```
 import java.io.*;
import javax.swing.*;
import java.awt.*;
import static java.lang.Math.*;
public class HT_cylinder
{public if_x F;
public double h,b,k,alpha,Bi,Ti,Tinf,H;
public  HT_cylinder(double hi,double bi,double ki,double alphai,double Tii,double Tinfi)
```

```
{h=hi;k=ki;b=bi;alpha=alphai;Bi=h*b/k;Ti=Tii;Tinf=Tinfi;H=h/k;
F=xx->Ti-Tinf;}
//J Bessel function of integer order 0
public static double J0(double x)
{   double ax,z;
        double xx,y,ans,ans1,ans2;
        if ((ax=Math.abs(x)) < 8.0) {
                y=x*x;
                ans1=57568490574.0+y*(-13362590354.0+y*(651619640.7
                        +y*(-11214424.18+y*(77392.33017+y*(-184.9052456)))));
                ans2=57568490411.0+y*(1029532985.0+y*(9494680.718
                        +y*(59272.64853+y*(267.8532712+y*1.0))));
                ans=ans1/ans2;
        } else {
                z=8.0/ax;
                y=z*z;
                xx=ax-0.785398164;
                ans1=1.0+y*(-0.1098628627e-2+y*(0.2734510407e-4
                        +y*(-0.2073370639e-5+y*0.2093887211e-6)));
                ans2 = -0.1562499995e-1+y*(0.1430488765e-3
                        +y*(-0.6911147651e-5+y*(0.7621095161e-6
                        -y*0.934935152e-7)));
                ans=sqrt(0.636619772/ax)*(cos(xx)*ans1-z*sin(xx)*ans2);
        }
        return ans;
}
//J Bessel function of integer order 1
public static double J1(double x)
{   double ax,z;
        double xx,y,ans,ans1,ans2;
        if ((ax=Math.abs(x)) < 8.0) {
                y=x*x;
                ans1=x*(72362614232.0+y*(-7895059235.0+y*(242396853.1
                        +y*(-2972611.439+y*(15704.48260+y*(-30.16036606))))));
                ans2=144725228442.0+y*(2300535178.0+y*(18583304.74
                        +y*(99447.43394+y*(376.9991397+y*1.0))));
                ans=ans1/ans2;
        } else {
                z=8.0/ax;
                y=z*z;
                xx=ax-2.356194491;
                ans1=1.0+y*(0.183105e-2+y*(-0.3516396496e-4
                        +y*(0.2457520174e-5+y*(-0.240337019e-6))));
                ans2=0.04687499995+y*(-0.2002690873e-3
                        +y*(0.8449199096e-5+y*(-0.88228987e-6
                        +y*0.105787412e-6)));
                ans=sqrt(0.636619772/ax)*(cos(xx)*ans1-z*sin(xx)*ans2);
                if (x < 0.0) ans = -ans;
        }
        return ans;
}

  public double teta(double r,double t)
  {double ksi=0;
   double beta=0;
   double C=0;
   double teta=0;
   if_x f1=xx->xx*J1(xx)-Bi*J0(xx);
   //search for eigenvalues starting from 0
   double x0=0;
   double dx=0.0001;
   double x1=0;
   for(int n=1;n<11;n++)
     {x1=f1.enlarge(x0,dx);
      ksi= f1.bisection(x0,x1);
      beta=ksi/b;
      double j0=J0(ksi);
      double j1=J1(ksi);
      double nn=2.0/(j0*j0)/(b*b)*(beta*beta)/(beta*beta+H*H);
      double nn1=b*b/2*(j0*j0+j1*j1);
      double NN=1.0/nn1;
      final double beta1=beta;
      if_x y=rr->rr*Ti*J0(beta1*rr);
      C=y.integral(0.0,b,20)/nn1;
      teta+=C*J0(beta*r)*Math.exp(-beta*beta*alpha*t);
      x0=ksi+dx;
     }
```

```
    return teta;
  }
  //derivative of tetha
  public double dteta(double r,double t)
  {double ksi=0;
   double beta=0;
   double C=0;
   double teta=0;
   if_x f1=xx->xx*J1(xx)-Bi*J0(xx);
   //search for eigenvalues starting from 0
   double x0=0;
   double dx=0.0001;
   double x1=0;
   for(int n=1;n<11;n++)
     {x1=f1.enlarge(x0,dx);
      ksi= f1.bisection(x0,x1);
      beta=ksi/b;
      double j0=J0(ksi);
      double j1=J1(ksi);
      double nn=2.0/(j0*j0)/(b*b)*(beta*beta)/(beta*beta+H*H);
      double nn1=b*b/2*(j0*j0+j1*j1);
      double NN=1.0/nn1;
      final double beta1=beta;
      if_x y=rr->rr*Ti*J0(beta1*rr);
      C=y.integral(0.0,b,20)*nn1;
      teta+=-C*beta*J1(beta*r)*Math.exp(-beta*beta*alpha*t);
      x0=ksi+dx;
     }
    return teta;
  }
  public double T(double r,double t)
  {double teta=teta(r,t);
   double T=teta+Tinf;
   return T;
  }
  public double q(double x,double t)
  {double q=-k*dteta(x,t);
   return q;
  }
}}
```

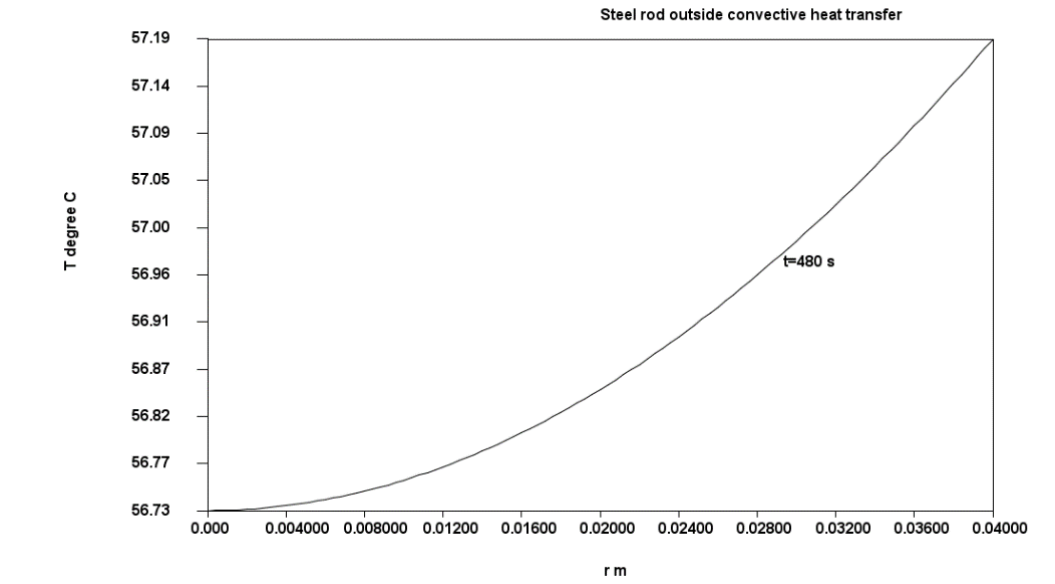
Numerical example:

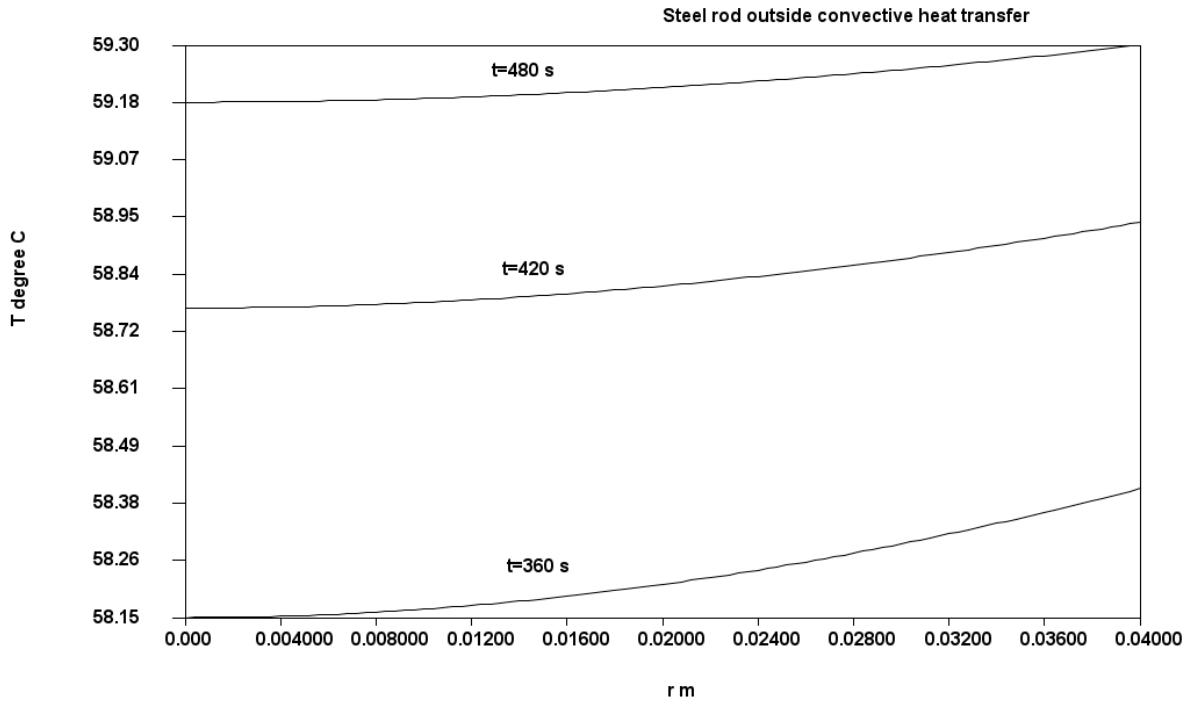
A steel rod (AISI 1010, k=63.9 W7mK) has Radius of 0.04 m. Initially the rod temperature is -20 ℃. (constant) The outside of the rod is exposed to 60 ℃ fluid with thermal convectivity coefficient of h=500 W/m^{2}K. Find the temperature profile of the rod at

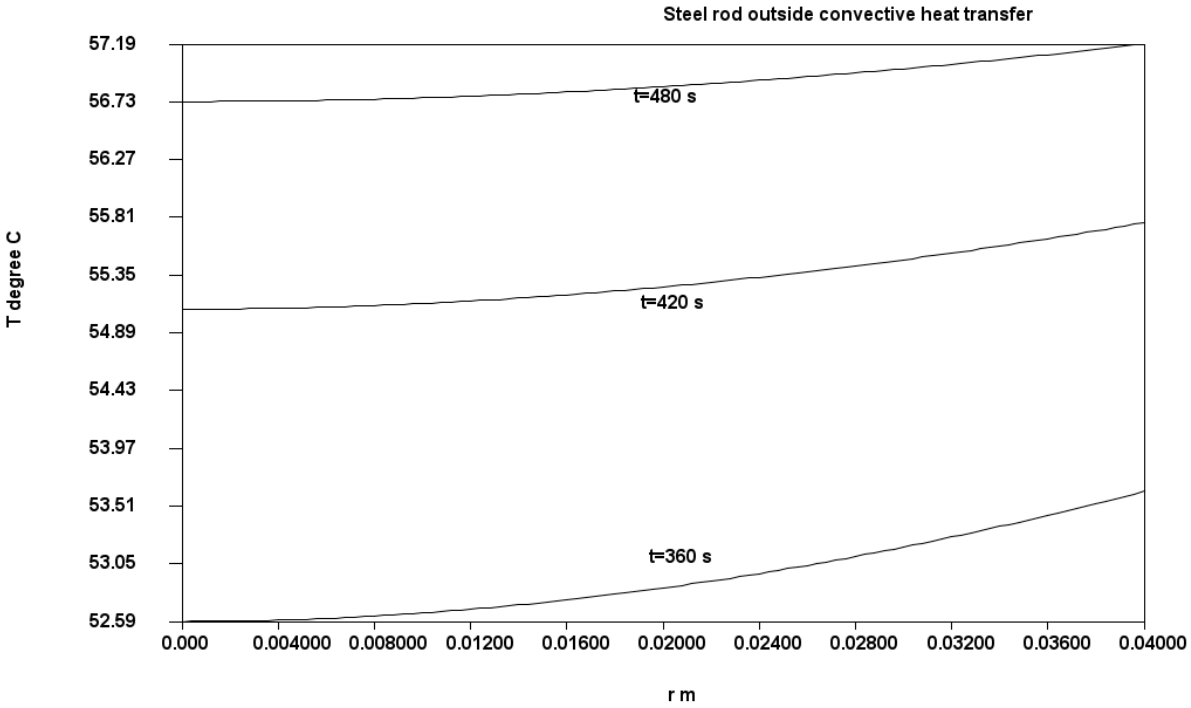
a) t=360 s, b)t=420 s and c) t=480 s F(x)=-20-60=-80 ℃,

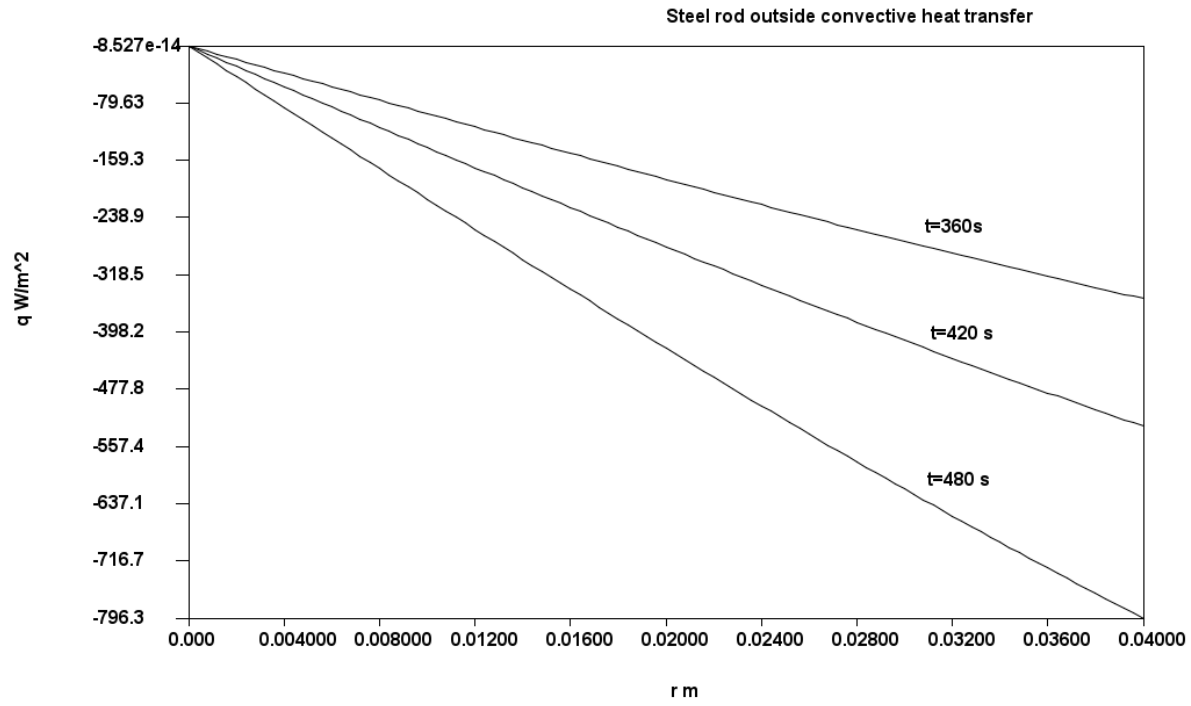
```
public class HT_cylinder_test
{
public static void main (String args[])
{ double h=500; //W/m^2K
  double k=63.9;
  double alpha=18.8e-6;
  double b=40e-3;
  double t[]={360,420,480};  //s
  double Tinf=60;
  double Ti=-20;
  int n1=t.length;
  double r[]=new double[101];
  double T[][]=new double[n1][101];
  double q[][]=new double[n1][101];
  HT_cylinder htc=new HT_cylinder(h,b,k,alpha,Ti,Tinf);
    //initial temperature variation t=0
  htc.F=xx->Ti-Tinf;
  for(int j=0;j<n1;j++)
  {
  for(int i=0;i<=100;i++)
      {   r[i]=i*b/100;
                  T[j][i]=htc.T(r[i],t[j]);
                  q[j][i]=htc.q(r[i],t[j]);
      }
   }
  Plot pp=new Plot(r,T[0]);
      for(int j=1;j<n1;j++)
```

```
    {pp.addData(r,T[j]);}
    pp.setPlabel("Steel rod outside convective heat transfer");
    pp.setXlabel(" r m");
    pp.setYlabel(" T degree C");
    pp.plot();
    Plot pp1=new Plot(r,q[0]);
    for(int j=1;j<n1;j++)
    {pp1.addData(r,q[j]);}
    pp1.setPlabel("Steel rod outside convective heat transfer");
    pp1.setXlabel(" r m");
    pp1.setYlabel(" q W/m^2");
    pp1.plot();
}
}
```









Python version of the problem:

```
 from math import *
from f_x import *;
import matplotlib.pyplot as plt


# Bessel polynomial J
def J(v,x):
    if x==0.0: x=1.0e-10
    JJ=0.0
    a=1.0
    fact=1
    b=0.0
    x1=0.0
    if x <=20:
        x5=1.0
        lna=0.0;
```

```
        x1=log(0.5*x)
        x1=x1*v
        lna1=log(0.25*x*x)
        for k in range(0,100):
            x2=lna
            # lgamma=ln(gamma) is a math library function
            x3=lgamma(k+1)
            x4=lgamma(v+k+1)
            b=x1+x2-x3-x4
            b=x5*exp(b)
            lna=lna+lna1
            x5=-1*x5
            JJ=JJ+b
    elif (v==0 and x==0):
        JJ=1.0
    else:
        JJ=sqrt(2.0/pi/x)*cos(x-0.5*v*pi-0.25*pi)
    return JJ

def bisection(f,a,b):
    b1=1.1*b
    r=(a+b)/2.0
    fr=f.func(r)
    fa=f.func(a)
    eps=1.0e-6;
    nmax=100;
    i=1;
    while abs(fr)>eps and i<nmax:
        if fa*fr<0:
            b=r
        else:
            a=r
            fa=fr
        r=(a+b)/2.0;
        fr=f.func(r)
        i=i+1
    if i>=nmax:
      r=bisection(f,a,b1)
    return r

def gauss_legendre_coefficients(x1,x2,n):
    #calculates legendre gauss-coefficients as coefficients of the integral
    #for n terms
        eps=3e-15
        m=int((n+1.0)/2.0)
        xm=0.5*(x2+x1)
        xl=0.5*(x2-x1)
        nmax=100
        a=[[0.0 for i in range(n)] for j in range(2)]
        for i in range(1,m+1):
            z=cos(pi*((i-0.25)/(n+0.5)))
            for ii in range(nmax):
                #while abs(z-z1) > eps:
                p1=1.0;
                p2=0.0;
                for j in range(1,n+1):
                    p3=p2
                    p2=p1
                    p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
                pp=(float)(n*(z*p1-p2)/(z*z-1.0))
                z1=z;
                z=z1-p1/pp
                if abs(z-z1) < eps: break
            a[0][i-1]=xm-xl*z
            a[0][n-i]=xm+xl*z
            a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
            a[1][n-i]=a[1][i-1]
        return a


def integral(f,x1,x2,n):
    #integral func(x)dx
    #integral of a function by using gauss-legendre quadrature
    #between x1 and x2
    a=gauss_legendre_coefficients(x1,x2,n)
    z=0
    for i in range(n):
```

```
        z=z+a[1][i]*f.func(a[0][i])
    return z

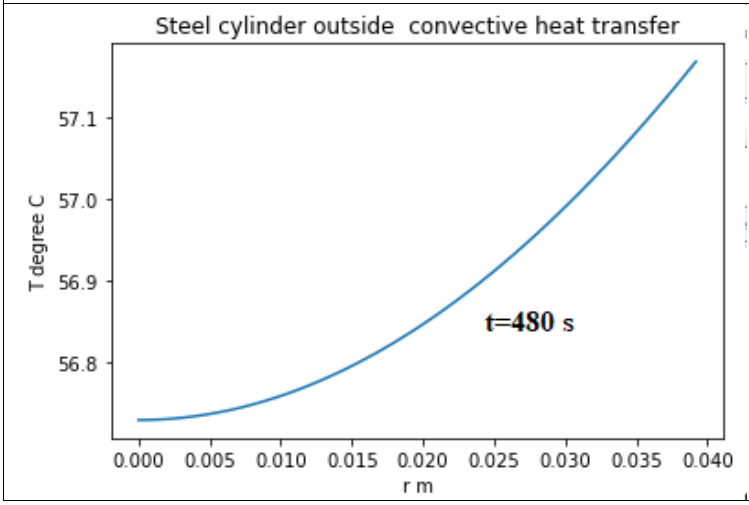
def enlarge(f,x0,dx):
    # enlarge region untill a root is existed
    x1=x0;
    x2=x1+dx
    NTRY=200
    a=[0.0 for i in range(n)]
    FACTOR=1.001
    j=0
    if x1 == x2: print("input variables are wrong")
    f1=f.func(x1)
    f2=f.func(x2)
    for j in range(NTRY+1):
        if f1*f2 < 0.0:
            break
        else:
            x2=x2+dx
            f2=f.func(x2)
    return x2

def tetha(F,H,alpha,b,r,t):
    beta=0.0
    C=0.0
    N=0.0
    teta=0.0
    Bi=H*b
    class f1(f_x):func=lambda self,xx: xx*J(1,xx)-Bi*J(0,xx)
    f = f1()
    x0=0.0
    dx=0.0001
    x1=0.0
    for n in range(1,11):
        x1=enlarge(f,x0,dx)
        ksi=bisection(f,x0,x1)
        beta=ksi/b
        J0=J(0,ksi)
        J1=J(1,ksi)
        nn=2.0/(J0*J0)/(b*b)*(beta*beta)/(beta*beta+H*H)
        NN=1.0/nn
        class Cn(f_x):func=lambda self,r:r*F.func(r)*J(0,beta*r)
        fCn=Cn()
        C=nn*integral(fCn,0.0,b,20)
        teta=teta+C*J(0,beta*r)*exp(-beta*beta*alpha*t)
        x0=ksi+dx
    return teta

h=500
k=63.9
H=h/k
alpha=18.8e-6
b=40e-3
t=480.0
Tinf=60
Ti=-20
class F(f_x):func=lambda self,x: Ti-Tinf
F1 = F()
n=50
T=[0.0 for i in range(n)]
r=[0.0 for i in range(n)]
for i in range(n):
    r[i]=i*b/n
    T[i]=tetha(F1,H,alpha,b,r[i],t)+Tinf
    print("r=",r[i],"T=",T[i])
plt.plot(r,T)
plt.xlabel("r m")
plt.ylabel("T degree C")
plt.title("Steel cylinder outside  convective heat transfer")
plt.show()
```

```
runfile('E:/okul/SCO1/HT_cylinder.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x
r= 0.0 T= 56.728936374977444
r= 0.0008 T= 56.729125929358986
r= 0.0016 T= 56.72969455955034
r= 0.0024 T= 56.73064216669806
r= 0.0032 T= 56.73196858606762
```

```
r= 0.004 T= 56.73367358707524
r= 0.0048 T= 56.7357568733324
r= 0.005600000000000001 T= 56.738218082703135
r= 0.0064 T= 56.74105678737393
r= 0.0072 T= 56.74427249393639
r= 0.008 T= 56.74786464348253
r= 0.0088 T= 56.751832611712715
r= 0.0096 T= 56.75617570905622
r= 0.0104 T= 56.76089318080446
r= 0.011200000000000002 T= 56.765984207256665
r= 0.012 T= 56.77144790387824
r= 0.0128 T= 56.77728332147152
r= 0.013600000000000001 T= 56.78348944635912
r= 0.0144 T= 56.79006520057958
r= 0.0152 T= 56.797009442095586
r= 0.016 T= 56.80432096501438
r= 0.0168 T= 56.811998499820604
r= 0.0176 T= 56.82004071362138
r= 0.0184 T= 56.828446210403584
r= 0.0192 T= 56.83721353130327
r= 0.02 T= 56.846341154887305
r= 0.0208 T= 56.855827497446924
r= 0.0216 T= 56.86567091330338
r= 0.022400000000000003 T= 56.875869695125495
r= 0.0232 T= 56.88642207425902
r= 0.024 T= 56.897326221067935
r= 0.0248 T= 56.90858024528733
r= 0.0256 T= 56.920182196388055
r= 0.0264 T= 56.93213006395287
r= 0.027200000000000002 T= 56.94442177806421
r= 0.028000000000000004 T= 56.95705520970328
r= 0.0288 T= 56.97002817116055
r= 0.0296 T= 56.98333841645753
r= 0.0304 T= 56.99698364177969
r= 0.031200000000000002 T= 57.01096148592053
r= 0.032 T= 57.02526953073656
r= 0.0328 T= 57.039905301613295
r= 0.0336 T= 57.054866267942025
r= 0.0344 T= 57.0701498436073
r= 0.0352 T= 57.08575338748502
r= 0.036000000000000004 T= 57.101674203951134
r= 0.0368 T= 57.11790954340062
r= 0.0376 T= 57.134456602776936
r= 0.0384 T= 57.15131252611151
r= 0.0392 T= 57.16847440507351
```



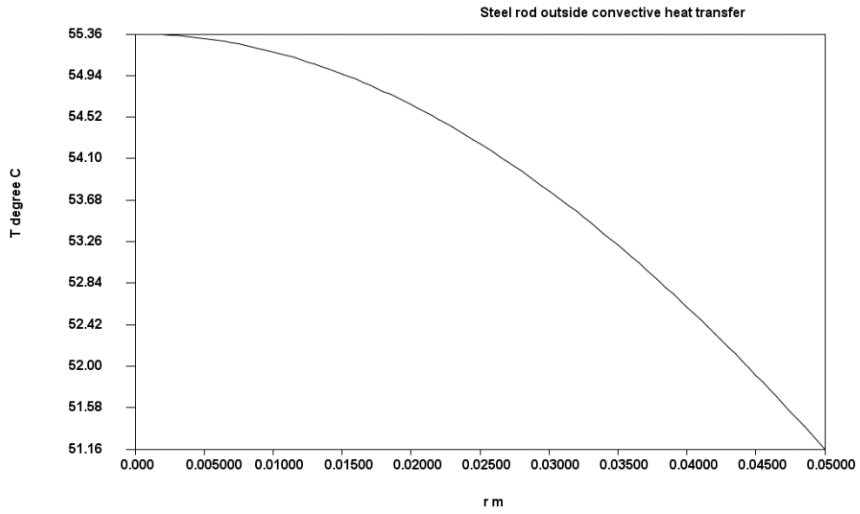
Numerical example:

A steel rod [α=1.2e-5 and k=43 W/(mK) of radius b=0.05 m, initially at a uniform temperature of 240 ℃, is suddenly immersed in an oil bath at $T_\infty = 40$ ℃. The convective heat transfer coefficient is h=600 W/(m^{2}K) calculate temperature profile for t=478.8 seconds.

F(r)=240-40=200 K (constant)

```
public class HT_cylinder_test
{
public static void main (String args[])
{ double h=600; //W/m^2K
  double k=43;
  double alpha=1.2e-5;
  double b=0.05;
  double t[]={478.8};  //s
  double Tinf=40;
  double Ti=240;
  int n1=t.length;
  double r[]=new double[101];
  double T[][]=new double[n1][101];
  double q[][]=new double[n1][101];
  HT_cylinder htc=new HT_cylinder(h,b,k,alpha,Ti,Tinf);
    //initial temperature variation t=0
  htc.F=xx->Ti-Tinf;
  for(int j=0;j<n1;j++)
  {
  for(int i=0;i<=100;i++)
      {   r[i]=i*b/100;
                  T[j][i]=htc.T(r[i],t[j]);

                  q[j][i]=htc.q(r[i],t[j]);
      }
   }
  Plot pp=new Plot(r,T[0]);
      for(int j=1;j<n1;j++)
      {pp.addData(r,T[j]);}
      pp.setPlabel("Steel rod outside convective heat transfer");
      pp.setXlabel(" r m");
      pp.setYlabel(" T degree C");
      pp.plot();
      Plot pp1=new Plot(r,q[0]);
      for(int j=1;j<n1;j++)
      {pp1.addData(r,q[j]);}
      pp1.setPlabel("Steel rod outside convective heat transfer");
      pp1.setXlabel(" r m");
      pp1.setYlabel(" q W/m^2");
      pp1.plot();
}
}
```



Python version of program:

```
from math import *
from f_x import *;
import matplotlib.pyplot as plt


# Bessel polynomial J
def J(v,x):
   if x==0.0: x=1.0e-10
   JJ=0.0
   a=1.0
```

```
    fact=1
    b=0.0
    x1=0.0
    if x <=20:
       x5=1.0
       lna=0.0;
       x1=log(0.5*x)
       x1=x1*v
       lna1=log(0.25*x*x)
       for k in range(0,100):
          x2=lna
          # lgamma=ln(gamma) is a math library function
          x3=lgamma(k+1)
          x4=lgamma(v+k+1)
          b=x1+x2-x3-x4
          b=x5*exp(b)
          lna=lna+lna1
          x5=-1*x5
          JJ=JJ+b
    elif (v==0 and x==0):
       JJ=1.0
    else:
       JJ=sqrt(2.0/pi/x)*cos(x-0.5*v*pi-0.25*pi)
    return JJ

def bisection(f,a,b):
    b1=1.1*b
    r=(a+b)/2.0
    fr=f.func(r)
    fa=f.func(a)
    eps=1.0e-6;
    nmax=100;
    i=1;
    while abs(fr)>eps and i<nmax:
       if fa*fr<0:
          b=r
       else:
          a=r
          fa=fr
       r=(a+b)/2.0;
       fr=f.func(r)
       i=i+1
    if i>=nmax:
      r=bisection(f,a,b1)
    return r

def gauss_legendre_coefficients(x1,x2,n):
    #calculates legendre gauss-coefficients as coefficients of the integral
    #for n terms
       eps=3e-15
       m=int((n+1.0)/2.0)
       xm=0.5*(x2+x1)
       xl=0.5*(x2-x1)
       nmax=100
       a=[[0.0 for i in range(n)] for j in range(2)]
       for i in range(1,m+1):
          z=cos(pi*((i-0.25)/(n+0.5)))
          for ii in range(nmax):
             #while abs(z-z1) > eps:
             p1=1.0;
             p2=0.0;
             for j in range(1,n+1):
                p3=p2
                p2=p1
                p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
             pp=(float)(n*(z*p1-p2)/(z*z-1.0))
             z1=z;
             z=z1-p1/pp
             if abs(z-z1) < eps: break
          a[0][i-1]=xm-xl*z
          a[0][n-i]=xm+xl*z
          a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
          a[1][n-i]=a[1][i-1]
       return a


def integral(f,x1,x2,n):
```

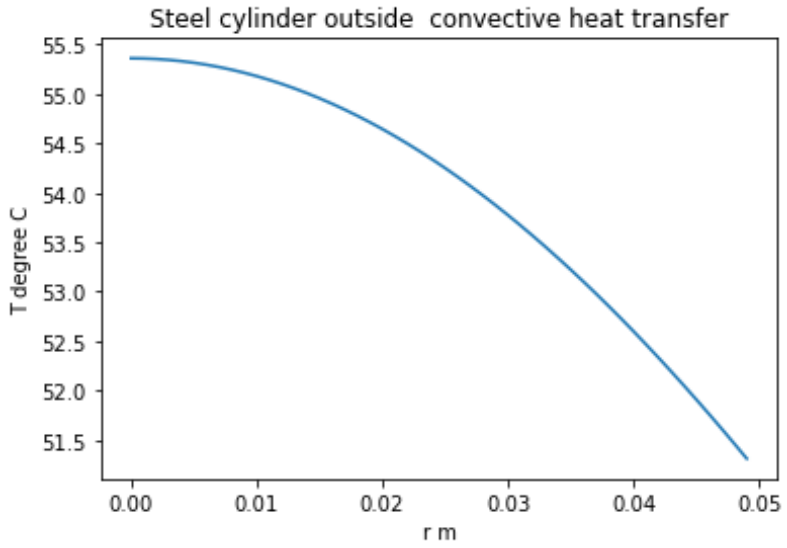
```
    #integral func(x)dx
    #integral of a function by using gauss-legendre quadrature
    #between x1 and x2
    a=gauss_legendre_coefficients(x1,x2,n)
    z=0
    for i in range(n):
        z=z+a[1][i]*f.func(a[0][i])
    return z

def enlarge(f,x0,dx):
    # enlarge region untill a root is existed
    x1=x0;
    x2=x1+dx
    NTRY=200
    a=[0.0 for i in range(n)]
    FACTOR=1.001
    j=0
    if x1 == x2: print("input variables are wrong")
    f1=f.func(x1)
    f2=f.func(x2)
    for j in range(NTRY+1):
        if f1*f2 < 0.0:
            break
        else:
            x2=x2+dx
            f2=f.func(x2)
    return x2

def tetha(F,H,alpha,b,r,t):
    beta=0.0
    C=0.0
    N=0.0
    teta=0.0
    Bi=H*b
    class f1(f_x):func=lambda self,xx: xx*J(1,xx)-Bi*J(0,xx)
    f = f1()
    x0=0.0
    dx=0.0001
    x1=0.0
    for n in range(1,11):
        x1=enlarge(f,x0,dx)
        ksi=bisection(f,x0,x1)
        beta=ksi/b
        J0=J(0,ksi)
        J1=J(1,ksi)
        nn=2.0/(J0*J0)/(b*b)*(beta*beta)/(beta*beta+H*H)
        NN=1.0/nn
        class Cn(f_x):func=lambda self,r:r*F.func(r)*J(0,beta*r)
        fCn=Cn()
        C=nn*integral(fCn,0.0,b,20)
        teta=teta+C*J(0,beta*r)*exp(-beta*beta*alpha*t)
        x0=ksi+dx
    return teta

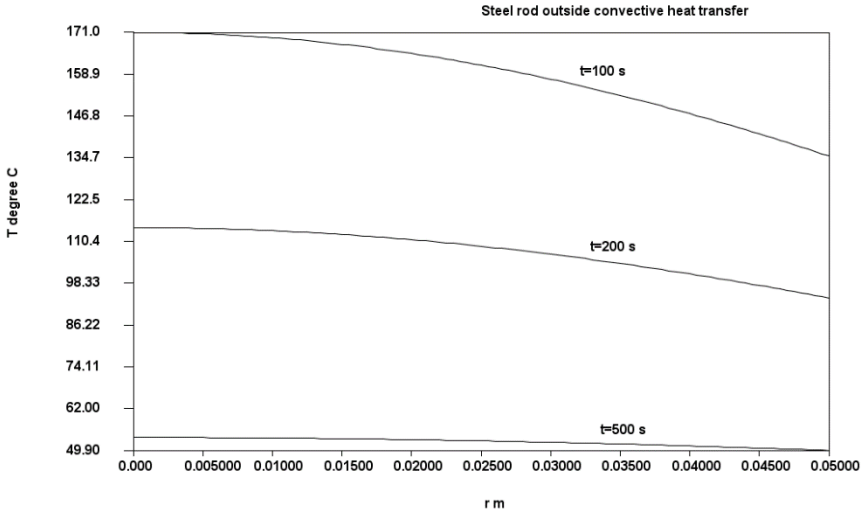
h=600.0
k=43.0
H=h/k
alpha=1.2e-5
b=0.05
t=478.8
Tinf=40.0
Ti=240.0
class F(f_x):func=lambda self,x: Ti-Tinf
F1 = F()
n=50
T=[0.0 for i in range(n)]
r=[0.0 for i in range(n)]
for i in range(n):
    r[i]=i*b/n
    T[i]=tetha(F1,H,alpha,b,r[i],t)+Tinf
    print("r=",r[i],"T=",T[i])
plt.plot(r,T)
plt.xlabel("r m")
plt.ylabel("T degree C")
plt.title("Steel cylinder outside  convective heat transfer")
plt.show()
runfile('E:/okul/SCO1/HT_cylinder.py', wdir='E:/okul/SCO1')
```

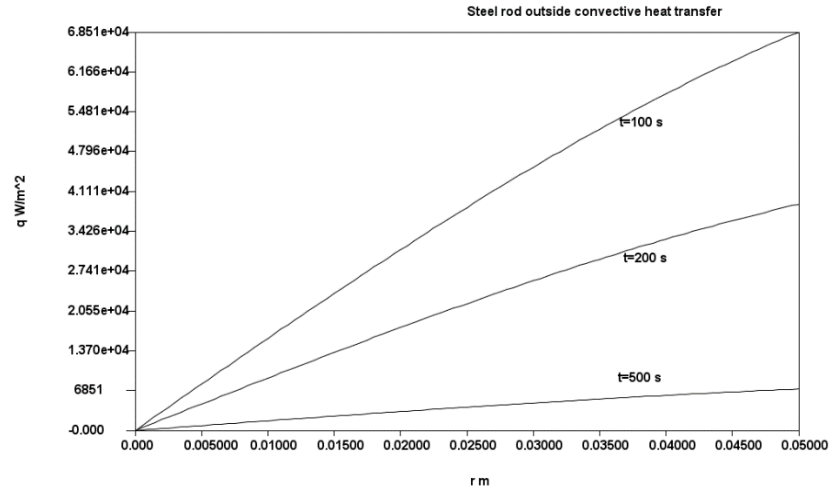
```
Reloaded modules: Mathd
r= 0.0 T= 55.36095449219902
r= 0.001 T= 55.35914374499075
r= 0.002 T= 55.35371214371365
r= 0.0030000000000000005 T= 55.34466160915922
r= 0.004 T= 55.331995341808025
r= 0.005 T= 55.31571782057215
r= 0.006000000000000001 T= 55.29583480103538
r= 0.007000000000000001 T= 55.27235331319183
r= 0.008 T= 55.24528165868386
r= 0.009000000000000001 T= 55.21462940754063
r= 0.01 T= 55.1804073944184
r= 0.011000000000000001 T= 55.142627714344314
r= 0.012000000000000002 T= 55.101303717965266
r= 0.013000000000000001 T= 55.056450006303976
r= 0.014000000000000002 T= 55.008082425024256
r= 0.015 T= 54.95621805820803
r= 0.016 T= 54.90087522164648
r= 0.017 T= 54.84207345564821
r= 0.018000000000000002 T= 54.77983351736739
r= 0.019000000000000003 T= 54.71417737265497
r= 0.02 T= 54.64512818743635
r= 0.021 T= 54.57271031861915
r= 0.022000000000000002 T= 54.496949304534716
r= 0.023000000000000003 T= 54.4178718549173
r= 0.024000000000000004 T= 54.33550584042521
r= 0.025 T= 54.24988028170796
r= 0.026000000000000002 T= 54.16102533802427
r= 0.027000000000000003 T= 54.068972295415264
r= 0.028000000000000004 T= 53.973753554438076
r= 0.029000000000000005 T= 53.87540261746457
r= 0.03 T= 53.77395407555078
r= 0.031 T= 53.66944359488208
r= 0.032 T= 53.56190790280002
r= 0.033 T= 53.4513847734164
r= 0.034 T= 53.33791301282051
r= 0.035 T= 53.22153244388565
r= 0.036000000000000004 T= 53.102283890681285
r= 0.037000000000000005 T= 52.98020916249703
r= 0.038000000000000006 T= 52.85535103748526
r= 0.03900000000000001 T= 52.72775324592894
r= 0.04 T= 52.59746045314165
r= 0.04100000000000001 T= 52.46451824200672
r= 0.042 T= 52.32897309516284
r= 0.043 T= 52.190872376843224
r= 0.044000000000000004 T= 52.050264314376086
r= 0.045 T= 51.90719797935373
r= 0.046000000000000006 T= 51.76172326847829
r= 0.047 T= 51.6138908840917
r= 0.04800000000000001 T= 51.463752314398164
r= 0.049 T= 51.31135981338707
```



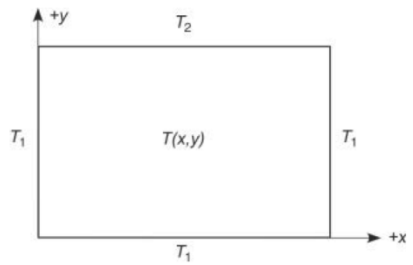
Numerical example:

A steel rod [α=1.2e-5 and k=43 W/(mK) of radius b=0.05 m, initially at a uniform temperature of 240 ℃, is suddenly immersed in an oil bath at $T_\infty = 40$ ℃. The convective heat transfer coefficient is h=600 W/(m^2K) calculate temperature profile for t=100,200 and 500 seconds.
F(r)=240-40=200 K (constant)





PROBLEM 3.6 SEPARATION OF VARIABLES, HOMOGENEOUS, STEADY STATE SOLUTION; CARTESIAN COORDINATES



$$\frac{\partial^2 T^*(x,y)}{\partial x^2} + \frac{\partial^2 T^*(x,y)}{\partial y^2} = 0$$

$T^*(x,y) = T_1$ x=0 $0 \le y \le W$
$T^*(x,y) = T_1$ x=L $0 \le y \le W$
$T^*(x,y) = T_1$ y=0 $0 \le x \le L$
$T^*(x,y) = T_2$ y=W $0 \le x \le L$

$T(x,y) = T^*(x,y) - T_1$

$\frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial y^2} = 0$	
$T(x,y) = 0$	x=0 $0 \le y \le W$
$T(x,y) = 0$	x=L $0 \le y \le W$
$T(x,y) = 0$	y=0 $0 \le x \le L$
$T(x,y) = T_2 - T_1 = T_3$	y=W $0 \le x \le L$

Separation of variables:

$T(x,y) = X(x) * Y(y)$

$$\frac{1}{X(x)}\frac{\partial^2 X(x)}{\partial x^2} = -\frac{1}{Y(y)}\frac{\partial^2 Y(y)}{\partial y^2} = -\lambda^2$$

$\frac{\partial^2 X(x)}{\partial x^2} + \lambda^2 X(x) = 0$

$\frac{\partial^2 Y(y)}{\partial y^2} - \lambda^2 Y(y) = 0$

$X(x) = 0 \quad x = 0$

$X(x) = 0 \quad x = L$

$Y(y) = 0 \quad y = 0$

$T(x,y) = T_2 - T_1 = T_3 \quad y = W$

$X(x) = C_1 \sin(\lambda x) + C_2 \cos(\lambda x)$

$Y(y) = C_3 \sinh(\lambda y) + C_4 \cosh(\lambda y)$

Boundary values:

$X(0) = C_1 \sin(\lambda 0) + C_2 \cos(\lambda 0) = 0 \xrightarrow{yields} C_2 = 0$

$X(L) = C_1 \sin(\lambda L) = 0 \xrightarrow{yields} \lambda_n = \frac{n\pi}{L}$

$Y(0) = C_3 \sinh(\lambda 0) + C_4 \cosh(\lambda 0) = 0 \xrightarrow{yields} C_4 = 0$

$$T(x,y) = \sum_{n=1}^{\infty} C_n \sin\left(\frac{n\pi x}{L}\right) \sinh\left(\frac{n\pi y}{L}\right)$$

$$T(x,W) = \sum_{n=1}^{\infty} C_n \sin\left(\frac{n\pi x}{L}\right) \sin h\left(\frac{n\pi W}{L}\right) = T_3 = (T_2 - T_1)$$

This equation has n coefficients. In order to solve these coefficients, we will use properties of orthogonal functions.

$$\int_0^L X(\lambda_n x)\, X(\lambda_m x) dx = \begin{cases} 0 & m \neq n \\ N(\lambda_m) & m = n \end{cases}$$

$\sin(\lambda_n x)$ is an orthogonal function. Therefore:

$$\int_0^L \frac{T_3}{\sin h\left(\frac{n\pi W}{L}\right)} \sin\left(\frac{n\pi x}{L}\right) dx = C_n \int_0^L \sin^2\left(\frac{n\pi x}{L}\right) dx$$

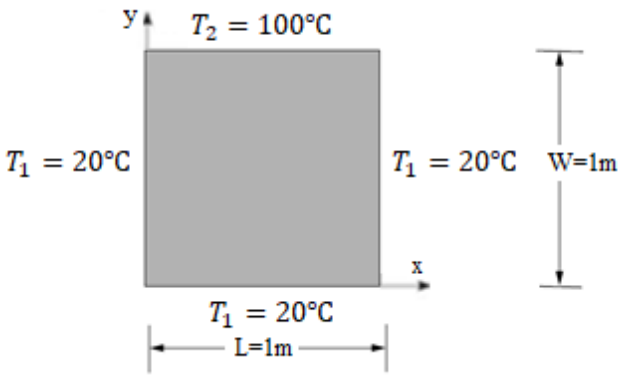
$$\int_0^L \sin^2\left(\frac{n\pi x}{L}\right) dx = \int_0^L \frac{1}{2}\left[1 - \cos\left(\frac{2n\pi x}{L}\right)\right] dx = \left[\frac{x}{2} - \frac{L}{2n\pi} sin\left(\frac{2n\pi x}{L}\right)\right]_0^L = \frac{L}{2}$$

$$C_n = \frac{T_3 \frac{L}{\pi}}{\sin h\left(\frac{n\pi W}{L}\right)}\left[\frac{(-1)^{n+1} + 1}{n}\right]$$

$$T(x,y) = T^*(x,y) - T_1 = \sum_{n=1}^{\infty} \frac{T_3 \frac{L}{\pi}}{\sin h\left(\frac{n\pi W}{L}\right)} \left[\frac{(-1)^{n+1}+1}{n}\right] \sin\left(\frac{n\pi x}{L}\right) \sinh\left(\frac{n\pi y}{L}\right)$$

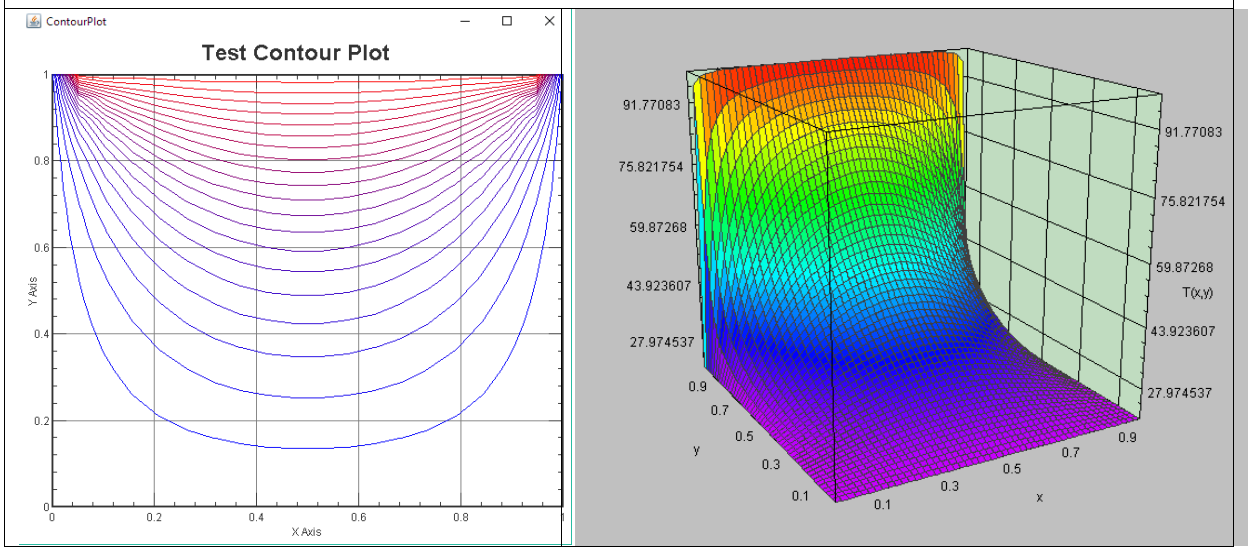
$$T^*(x,y) = T(x,y) + T_1$$

Numerical problem: A plate has a length of L=1 m and width of W=1 m. Three side of the plate has a temperature of T_1=20 ℃ and top side of the plate has a temperature of T_1=100 ℃. Calculate the temperature distribution and heat transfer distribution in the plate.



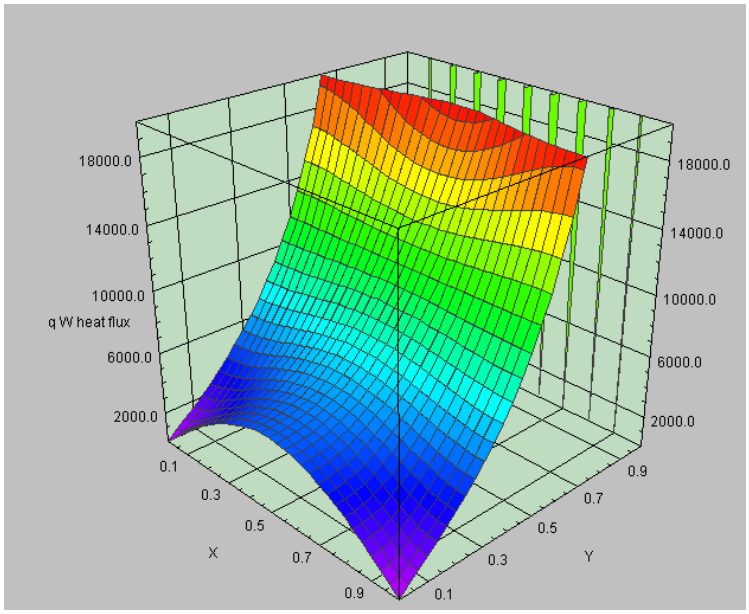

```
 import java.rmi.RemoteException;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
class HTrectangle implements if_xj
{
double L,W,T1,T2;
public HTrectangle(double Li,double Wi,double T1i,double T2i)
{L=Li;W=Wi;T1=T1i;T2=T2i;}
public double func(double x[])
{ double C=0;
int plusminus=1;
double teta=0;
for(int n=1;n<100;n++)
{C=2.0*(plusminus+1)/(Math.PI*n*Math.sinh(n*Math.PI*W/L));
 plusminus*=-1;
 teta+=C*Math.sin(n*Math.PI*x[0]/L)*Math.sinh(n*Math.PI*x[1]/L);
}
return teta*(T2-T1)+T1;
}}
public class HT_rectangular1A
{ public static void main(String arg[])
 {
 double W=1.0;
 double L=1.0;
 double k=50.0;
 double T1=20;//derece C
 double T2=100;//derece C
 double dfx,dfy;
 HTrectangle f=new HTrectangle(W,L,T1,T2);
 double teta[][]=new double[21][21];
 double q[][]=new double[21][21];
 double x[]=new double[2];
 for(int i=0;i<=20;i++)
 {for(int j=0;j<=20;j++)
   {  double x1[]={0.05*i,0.05*j};
      x=x1;
                teta[i][j]=f.func(x);
   }
 }
 Text.print(teta);
 ContourPlot p=new ContourPlot(teta,0.0,L,0.0,W,20,true,"x","y","z");
 p.plot();
 surfacePlot pp=new surfacePlot(f,0.0,L,0.0,W,T1,T2,"X","Y","Z",30);
 pp.plot();
```

```
}
}
```



Temperature profile

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0
20.0	20.21884...	20.43205...	20.63421...	20.82023...	20.98557...	21.12627...	21.23908...	21.32147...	21.37162...	21.38846...	21.37162...	21.32147...	21.23908...	21.12627...	20.98557...	20.82023...	20.63421...	20.43205...	20.21884...	20.0
20.0	20.44334...	20.87523...	21.28464...	21.66126...	21.99589...	22.28056...	22.50872...	22.67530...	22.77668...	22.81071...	22.77668...	22.67530...	22.50872...	22.28056...	21.99589...	21.66126...	21.28464...	20.87523...	20.44334...	20.0
20.0	20.67935...	21.34103...	21.96802...	22.54451...	23.05640...	23.49159...	23.84020...	24.09457...	24.24933...	24.30127...	24.24933...	24.09457...	23.84020...	23.49159...	23.05640...	22.54451...	21.96802...	21.34103...	20.67935...	20.0
20.0	20.93314...	21.84174...	22.70219...	23.49270...	24.19398...	24.78960...	25.26627...	25.61383...	25.82516...	25.89606...	25.82516...	25.61383...	25.26627...	24.78960...	24.19398...	23.49270...	22.70219...	21.84174...	20.93314...	20.0
20.0	21.21165...	22.39090...	23.50671...	24.53063...	25.43773...	26.20708...	26.82196...	27.26979...	27.54187...	27.63312...	27.54187...	27.26979...	26.82196...	26.20708...	25.43773...	24.53063...	23.50671...	22.39090...	21.21165...	20.0
20.0	21.52281...	23.00395...	24.40370...	25.68606...	26.81994...	27.77971...	28.54536...	29.10211...	29.43999...	29.55324...	29.43999...	29.10211...	28.54536...	27.77971...	26.81994...	25.68606...	24.40370...	23.00395...	21.52281...	20.0
20.0	21.87600...	23.69900...	25.41890...	26.99092...	28.37719...	29.54736...	30.47843...	31.15402...	31.56337...	31.70046...	31.56337...	31.15402...	30.47843...	29.54736...	28.37719...	26.99092...	25.41890...	23.69900...	21.87600...	20.0
20.0	22.28269...	24.49807...	26.58316...	28.48282...	30.15179...	31.55521...	32.66793...	33.47293...	33.95967...	34.12251...	33.95967...	33.47293...	32.66793...	31.55521...	30.15179...	28.48282...	26.58316...	24.49807...	22.28269...	20.00000...
20.0	22.75744...	25.42877...	27.93454...	30.20718...	32.19355...	33.85515...	35.16620...	36.11092...	36.68052...	36.87079...	36.68052...	36.11092...	35.16620...	33.85515...	32.19355...	30.20718...	27.93454...	25.42877...	22.75744...	20.00000...
20.0	23.31947...	26.52706...	29.52156...	32.22028...	34.56226...	36.50729...	38.03197...	39.12476...	39.78115...	40.0	39.78115...	39.12476...	38.03197...	36.50729...	34.56226...	32.22028...	29.52156...	26.52706...	23.31947...	20.00000...
20.0	23.99527...	27.84164...	31.40811...	34.59365...	37.33076...	39.58173...	41.33077...	42.57550...	43.31947...	43.56688...	43.31947...	42.57550...	41.33077...	39.58173...	37.33076...	34.59365...	31.40811...	27.84164...	23.99527...	20.00000...
20.0	24.82315...	29.44139...	33.68139...	37.42052...	40.58910...	43.16026...	45.13483...	46.52706...	47.35388...	47.62796...	47.35388...	46.52706...	45.13483...	43.16026...	40.58910...	37.42052...	33.68139...	29.44139...	24.82315...	20.00000...
20.0	25.86160...	31.42834...	36.46474...	40.82518...	44.44942...	47.33754...	49.52156...	51.04321...	51.93964...	52.23558...	51.93964...	51.04321...	49.52156...	47.33754...	44.44942...	40.82518...	36.46474...	31.42834...	25.86160...	20.00000...
20.0	27.20579...	33.96169...	39.93884...	44.97614...	49.05098...	52.22028...	54.56973...	56.18239...	57.12312...	57.43217...	57.12312...	56.18239...	54.56973...	52.22028...	49.05098...	44.97614...	39.93884...	33.96169...	27.20579...	20.00000...
20.0	29.02258...	37.30415...	44.37658...	50.10245...	54.56226...	57.92198...	60.35141...	61.98930...	62.93379...	63.24233...	62.93379...	61.98930...	60.35141...	57.92198...	54.56226...	50.10245...	44.37658...	37.30415...	29.02258...	20.00000...
20.0	31.63385...	41.91599...	50.19710...	56.50729...	61.17292...	64.54817...	66.91761...	68.48282...	69.37400...	69.66336...	69.37400...	68.48282...	66.91761...	64.54817...	61.17292...	56.50729...	50.19710...	41.91599...	31.63385...	20.00000...
20.0	35.74338...	48.65079...	58.03197...	64.55618...	69.06029...	72.16585...	74.27615...	75.64086...	76.40799...	76.65559...	76.40799...	75.64086...	74.27615...	72.16585...	69.06029...	64.55618...	58.03197...	48.65079...	35.74338...	20.00000...
20.0	43.17906...	59.12476...	68.72352...	74.58099...	78.30905...	80.75378...	82.36392...	83.38522...	83.95289...	84.13515...	83.95289...	83.38522...	82.36392...	80.75378...	78.30905...	74.58099...	68.72352...	59.12476...	43.17906...	20.00000...
20.0	59.78115...	75.94553...	82.94305...	86.61275...	88.78017...	90.14511...	91.02329...	91.57267...	91.87564...	91.97258...	91.87564...	91.57267...	91.02329...	90.14511...	88.78017...	86.61275...	82.94305...	75.94553...	59.78115...	20.00000...
20.0	98.37542...	99.17635...	99.43921...	99.56681...	99.63990...	99.68525...	99.71421...	99.73225...	99.74218...	99.74535...	99.74218...	99.73225...	99.71421...	99.68525...	99.63990...	99.56681...	99.43921...	99.17635...	98.37542...	20.00000...



Q heat flux, flat plate problem

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0.0	769.250...	1558.42...	2388.12...	3280.45...	4259.88...	5354.42...	6597.28...	8029.22...	9702.11...	11684.7...	14072.5...	17004.6...	20695.4...	25499.2...	32051.1...	41620.3...	57171.9...	87610.4...	177489....	280000...
762.668...	1083.23...	1734.96...	2506.73...	3367.38...	4326.39...	5406.04...	6636.71...	8057.26...	9717.78...	11684.7...	14049.9...	16945.4...	20572.5...	25255.7...	31558.4...	40546.0...	54466.9...	78695.4...	126043....	358010....
1505.75...	1690.81...	2166.70...	2822.22...	3607.17...	4513.19...	5552.44...	6749.10...	8137.35...	9762.54...	11684.7...	13985.4...	16778.6...	20229.9...	24589.8...	30252.1...	37850.6...	48378.1...	63014.6...	80750.5...	64.7535...
2210.34...	2340.19...	2703.63...	3251.61...	3950.06...	4787.81...	5771.06...	6918.36...	8258.43...	9830.19...	11684.7...	13889.2...	16532.6...	19734.9...	23658.3...	28516.9...	34564.9...	41989.6...	50500.1...	58342.9...	123445....
2858.77...	2960.07...	3254.22...	3720.36...	4341.09...	5109.93...	6032.00...	7122.41...	8405.01...	9912.06...	11684.7...	13774.5...	16244.3...	19169.6...	22634.9...	26719.3...	31451.5...	36699.0...	41955.8...	46125.1...	123.168...
3435.14...	3519.56...	3769.22...	4176.17...	4733.81...	5441.20...	6304.65...	7337.64...	8560.27...	9998.71...	11684.7...	13655.1...	15949.7...	18607.3...	21656.0...	25092.5...	28842.2...	32693.5...	36223.3...	38792.8...	79344.1...
3925.71...	3999.39...	4219.49...	4584.24...	5093.31...	5749.93...	6562.04...	7542.44...	8708.48...	10081.3...	11684.7...	13543.2...	15678.2...	18102.0...	20806.6...	23745.2...	26806.5...	29783.3...	32357.9...	34138.6...	169.525...
4319.07...	4385.74...	4586.06...	4921.31...	5394.66...	6012.04...	6782.65...	7719.03...	8836.58...	10152.6...	11684.7...	13448.1...	15450.9...	17687.9...	20129.8...	22710.0...	25309.0...	27743.3...	29771.4...	31131.2...	63036.9...
4606.37...	4668.59...	4856.14...	5171.86...	5620.77...	6210.36...	6950.67...	7854.09...	8934.71...	10207.1...	11684.7...	13376.2...	15281.2...	17383.7...	19643.2...	21984.8...	24290.9...	26399.3...	28115.7...	29245.2...	199.288...
4781.27...	4841.00...	5021.36...	5325.91...	5760.56...	6333.59...	7055.50...	7938.57...	8996.14...	10241.3...	11684.7...	13331.7...	15176.9...	17198.7...	19351.4...	21557.0...	23701.5...	25635.7...	27190.5...	28203.6...	56905.0...
4839.99...	4898.92...	5076.96...	5377.86...	5807.82...	6375.36...	7091.10...	7967.29...	9017.03...	10252.8...	11684.7...	13316.6...	15141.7...	17136.7...	19254.2...	21415.8...	23508.6...	25388.1...	26892.8...	27870.2...	209.544...
4781.27...	4841.00...	5021.36...	5325.91...	5760.56...	6333.59...	7055.50...	7938.57...	8996.14...	10241.3...	11684.7...	13331.7...	15176.9...	17198.7...	19351.4...	21557.0...	23701.5...	25635.7...	27190.5...	28203.6...	56905.0...
4606.37...	4668.59...	4856.14...	5171.86...	5620.77...	6210.36...	6950.67...	7854.09...	8934.71...	10207.1...	11684.7...	13376.2...	15281.2...	17383.7...	19643.2...	21984.8...	24290.9...	26399.3...	28115.7...	29245.2...	199.288...
4319.07...	4385.74...	4586.06...	4921.31...	5394.66...	6012.04...	6782.65...	7719.03...	8836.58...	10152.6...	11684.7...	13448.1...	15450.9...	17687.9...	20129.8...	22710.0...	25309.0...	27743.3...	29771.4...	31131.2...	63036.9...
3925.71...	3999.39...	4219.49...	4584.24...	5093.31...	5749.93...	6562.04...	7542.44...	8708.48...	10081.3...	11684.7...	13543.2...	15678.2...	18102.0...	20806.6...	23745.2...	26806.5...	29783.3...	32357.9...	34138.6...	169.525...
3435.14...	3519.56...	3769.22...	4176.17...	4733.81...	5441.20...	6304.65...	7337.64...	8560.27...	9998.71...	11684.7...	13655.1...	15949.7...	18607.3...	21656.0...	25092.5...	28842.2...	32693.5...	36223.3...	38792.8...	79344.1...
2858.77...	2960.07...	3254.22...	3720.36...	4341.09...	5109.93...	6032.00...	7122.41...	8405.01...	9912.06...	11684.7...	13774.5...	16244.3...	19169.6...	22634.9...	26719.3...	31451.5...	36699.0...	41955.8...	46125.1...	123.168...
2210.34...	2340.19...	2703.63...	3251.61...	3950.06...	4787.81...	5771.06...	6918.36...	8258.43...	9830.19...	11684.7...	13889.2...	16532.6...	19734.9...	23658.3...	28516.9...	34564.9...	41989.6...	50500.1...	58342.9...	123445....
1505.75...	1690.81...	2166.70...	2822.22...	3607.17...	4513.19...	5552.44...	6749.10...	8137.35...	9762.54...	11684.7...	13985.4...	16778.6...	20229.9...	24589.8...	30252.1...	37850.6...	48378.1...	63014.6...	80750.5...	64.7535...
762.668...	1083.23...	1734.96...	2506.73...	3367.38...	4326.39...	5406.04...	6636.71...	8057.26...	9717.78...	11684.7...	14049.9...	16945.4...	20572.5...	25255.7...	31558.4...	40546.0...	54466.9...	78695.4...	126043....	358010....
0.0	769.250...	1558.42...	2388.12...	3280.45...	4259.88...	5354.42...	6597.28...	8029.22...	9702.11...	11684.7...	14072.5...	17004.6...	20695.4...	25499.2...	32051.1...	41620.3...	57171.9...	87610.4...	177489....	279999...

In this program functions are derived from interface if_xj, list of this interface is given below:

```
@FunctionalInterface
interface if_xj
{
  // single function  multi independent variable
  // a single value is returned indiced to equation_ref
public double func(double x[]);
default double func(double x1,double y1)
{
double x[]=new double[2];
x[0]=x1;
x[1]=y1;
return func(x);
}
default double[] dfunc(double x[])
{int n=x.length;
 double c[]=new double[n];
 for(int i=0;i<n;i++)
 {c[i]=dfunc(x,i);}
 return c;
}

default double[][] d2func(double x[])
```

```
{int n=x.length;
 double c[][]=new double[n][n];
 for(int i=0;i<n;i++)
 {for(int j=0;j<n;j++)
   {   int xref[]={i,j};
                c[i][j]=d2func(x,xref);
   }
 }
 return c;
}

default double d2func(double x[],int x_ref[])
{// derivative of the function with respect to x_ref
double h0=0.0256;
int i,m;
int n=7;
double f1,f2;
double x1[];
x1=new double[x.length];
double x2[];
x2=new double[x.length];
for(i=0;i<x.length;i++)
{
x1[i]=x[i];
x2[i]=x[i];
}
//derivative of a simple function
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
//vector<double> h(n,0);
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
h[i]=h0*Math.pow(r,i);
}

for(i=0;i<n;i++)
{
x1[x_ref[1]]+=h[i];
x2[x_ref[1]]-=h[i];
f1=dfunc(x1,x_ref[0]);
f2=dfunc(x2,x_ref[0]);
T[i][0]=( f1 - f2)/(2.0*h[i]);
x1[x_ref[1]]=x[x_ref[1]];
x2[x_ref[1]]=x[x_ref[1]];
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i]
  - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}
default double dfunc(double x[],int x_ref)
{ // derivative of the function with respect to x_ref
double h0=0.0256;
int i,m;
int n=7;
double f1,f2;
double x1[];
x1=new double[x.length];
double x2[];
x2=new double[x.length];
for(i=0;i<x.length;i++)
```

```
{
x1[i]=x[i];
x2[i]=x[i];
}
//derivative of a simple function
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
//vector<double> h(n,0);
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
h[i]=h0*Math.pow(r,i);
}

for(i=0;i<n;i++)
{
x1[x_ref]+=h[i];
x2[x_ref]-=h[i];
f1=func(x1);
f2=func(x2);
T[i][0]=( f1 - f2)/(2.0*h[i]);
x1[x_ref]=x[x_ref];
x2[x_ref]=x[x_ref];
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i]
  - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}}
```

Python version:

```
from math import *
from f_xj import *;
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt


class f_rectangle(f_xj):
  def __init__(self, Li,Wi,T1i,T2i,ki):
    self.L=Li
    self.W=Wi
    self.T1=T1i
    self.T2=T2i
    self.k=ki

  def func(self,x):
      C=0
      plusminus=1
      tetha=0.0
      for n in range(1,100):
          C=2.0*(plusminus+1)/(pi*n*sinh(n*pi*W/L))
          plusminus*=-1
          tetha=tetha+C*sin(n*pi*x[0]/L)*sinh(n*pi*x[1]/L)
      return tetha*(self.T2-self.T1)+self.T1

def plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLAbel,YLable,ZLabel):
    #f=f2()
    x=[0.0 for j in range(Nx)]
    y=[0.0 for j in range(Ny)]
```

```
    z=[[0.0 for j in range(Ny)] for i in range(Nx)]
    dx=(xmax-xmin)/(Nx-1)
    dy=(ymax-ymin)/(Ny-1)
    for i in range(Nx):
        for j in range(Ny):
            x[i]=xmin+dx*i
            y[j] = ymin+dy*j
            z[i][j]=f.funcxy(x[i],y[j])
    xi=np.array(x)
    yi=np.array(y)
    xi, yi = np.meshgrid(xi, yi)
    zi=np.array(z)
    #print(x[i][j]," ",y[i][j]," ",z[i][j])
    fig = plt.figure(figsize =(14, 9))
    ax = plt.axes(projection ='3d')
    # Creating color map
    my_cmap = plt.get_cmap('terrain')
    # Creating plot
    surf = ax.plot_surface(xi, yi, zi,
                           cmap = my_cmap,
                           edgecolor ='none')
    fig.colorbar(surf, ax = ax,
                 shrink = 0.5,
                 aspect = 5)

    ax.set_title(PLabel)
    ax.set_xlabel(XLabel)
    ax.set_xlim(xmin, xmax)
    ax.set_ylabel(YLabel)
    ax.set_xlim(ymin, ymax)
    ax.set_zlabel(ZLabel)
    ax.set_zlim(np.min(zi), np.max(zi))
    ax.view_init(30, 130)
    plt.show()

# Creating dataset
def contour_plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLable,ZLabel):
    #f=f2()
    x=[0.0 for j in range(Nx)]
    y=[0.0 for j in range(Ny)]
    z=[[0.0 for j in range(Ny)] for i in range(Nx)]
    dx=(xmax-xmin)/(Nx-1)
    dy=(ymax-ymin)/(Ny-1)
    for i in range(Nx):
        for j in range(Ny):
            x[i]=xmin+dx*i
            y[j] = ymin+dy*j
            z[i][j]=f.funcxy(x[i],y[j])
    xi=np.array(x)
    yi=np.array(y)
    xi, yi = np.meshgrid(xi, yi)
    zi=np.array(z)
    fig, ax = plt.subplots(1, 1)
    # plots contour lines
    ax.contour(xi, yi, zi,linewidths=0.5)
    ax.set_title(PLabel)
    ax.set_xlabel(XLabel)
    ax.set_xlim(xmin, xmax)
    ax.set_ylabel(YLabel)
    ax.set_xlim(ymin, ymax)
    plt.show()


W=1.0
L=1.0
k=50.0
T1=20
T2=100
f=f_rectangle(W,L,T1,T2,k)
xmin=0.0
xmax=L
Nx=100
ymin=0.0
ymax=W
Ny=100
PLabel="Plane surface temperature variation"
XLabel="x m"
```

```
YLabel="y m"
ZLabel="T degree C"
plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLabel,ZLabel)
contour_plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLabel,ZLabel)
```

```
public abstract class f_xj extends Mathd
{
  // single function  multi independent variable
  // a single value is returned indiced to equation_ref
abstract public double func(double x[]);
public double func(double x1,double y1)
{
double x[]=new double[2];
x[0]=x1;
x[1]=y1;
return func(x);
}
public double[] dfunc(double x[])
{int n=x.length;
 double c[]=new double[n];
 for(int i=0;i<n;i++)
 {c[i]=dfunc(x,i);}
 return c;
}

public double[][] d2func(double x[])
{int n=x.length;
 double c[][]=new double[n][n];

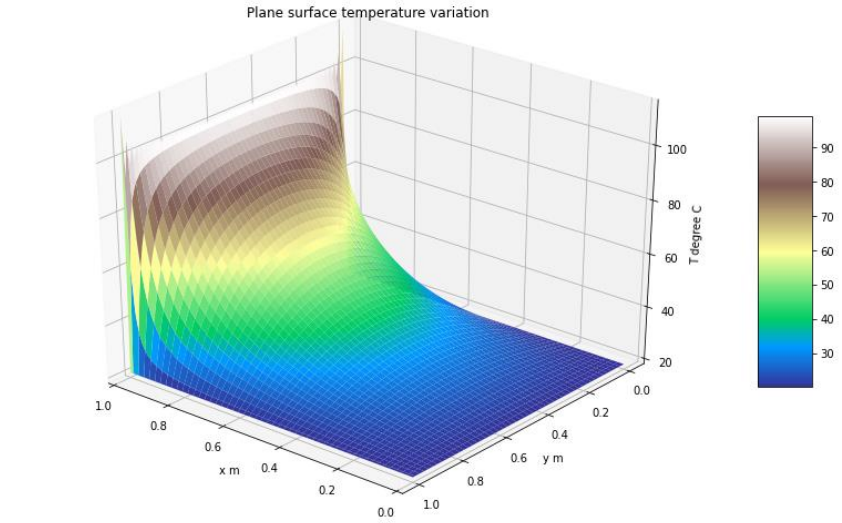
 for(int i=0;i<n;i++)
 {for(int j=0;j<n;j++)
   {   int xref[]={i,j};
                c[i][j]=d2func(x,xref);
   }
 }
 return c;
}

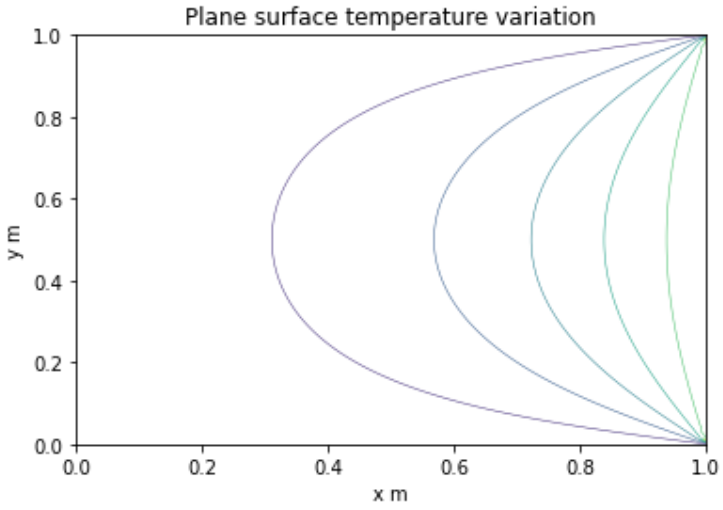
public double d2func(double x[],int x_ref[])
{// derivative of the function with respect to x_ref
double h0=0.0256;
int i,m;
int n=7;
double f1,f2;
double x1[];
x1=new double[x.length];
double x2[];
x2=new double[x.length];
for(i=0;i<x.length;i++)
{
x1[i]=x[i];
x2[i]=x[i];
}
//derivative of a simple function
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
//vector<double> h(n,0);
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
h[i]=h0*Math.pow(r,i);
}

for(i=0;i<n;i++)
{
x1[x_ref[1]]+=h[i];
x2[x_ref[1]]-=h[i];
f1=dfunc(x1,x_ref[0]);
f2=dfunc(x2,x_ref[0]);
```

```
T[i][0]=( f1 - f2)/(2.0*h[i]);
x1[x_ref[1]]=x[x_ref[1]];
x2[x_ref[1]]=x[x_ref[1]];
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i]
  - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}
public double dfunc(double x[],int x_ref)
{ // derivative of the function with respect to x_ref
double h0=0.015675863;
int i,m;
int n=7;
double f1,f2;
double x1[];
x1=new double[x.length];
double x2[];
x2=new double[x.length];
for(i=0;i<x.length;i++)
{
x1[i]=x[i];
x2[i]=x[i];
}
//derivative of a simple function
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
//vector<double> h(n,0);
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
h[i]=h0*Math.pow(r,i);
}

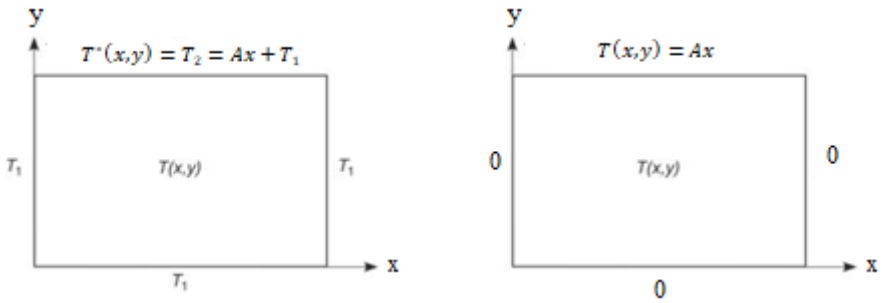
for(i=0;i<n;i++)
{
x1[x_ref]+=h[i];
x2[x_ref]-=h[i];
f1=func(x1);
f2=func(x2);
T[i][0]=( f1 - f2)/(2.0*h[i]);
x1[x_ref]=x[x_ref];
x2[x_ref]=x[x_ref];
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i]
  - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}
}
```





PROBLEM

A two dimensional rectangular plate is subjected to the boundary conditions shown in the figure. Derive an expression for the steady-state temperature distribution T(x,y)Denklemi buraya yazın.



$T(x,y) = Ax$

$$\frac{\partial^2 T^*(x,y)}{\partial x^2} + \frac{\partial^2 T^*(x,y)}{\partial y^2} = 0$$

$T^*(x,y) = T_1$ x=0 $0 \le y \le W$

$T^*(x,y) = T_1$ x=L $0 \le y \le W$

$T^*(x,y) = T_1$ y=0 $0 \le x \le L$

$T^*(x,y) = Ax + T_1$ y=W $0 \le x \le L$

$T(x,y) = T^*(x,y) - T_1$

$\frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial y^2} = 0$	
$T(x,y) = 0$	x=0 $0 \le y \le W$
$T(x,y) = 0$	x=L $0 \le y \le W$
$T(x,y) = 0$	y=0 $0 \le x \le L$
$T(x,y) = Ax$	y=W $0 \le x \le L$

Separation of variables:

$T(x,y) = X(x) * Y(y)$

$$\frac{1}{X(x)}\frac{\partial^2 X(x)}{\partial x^2} = -\frac{1}{Y(y)}\frac{\partial^2 Y(y)}{\partial y^2} = -\lambda^2$$

$\frac{\partial^2 X(x)}{\partial x^2} + \lambda^2 X(x) = 0$

$\frac{\partial^2 Y(y)}{\partial y^2} - \lambda^2 Y(y) = 0$

$X(x) = 0 \quad x = 0$

$X(x) = 0 \quad x = L$

$Y(y) = 0 \quad y = 0$

$T(x,y) = T_2 - T_1 = T_3 \quad y = W$

$X(x) = C_1 \sin(\lambda x) + C_2 \cos(\lambda x)$

$Y(y) = C_3 \sin\mathrm{h}(\lambda y) + C_4 \cosh(\lambda y)$

Boundary values:

$X(0) = C_1 \sin(\lambda 0) + C_2 \cos(\lambda 0) = 0 \xrightarrow{yields} C_2 = 0$

$X(L) = C_1 \sin(\lambda L) = 0 \xrightarrow{yields} \lambda_n = \frac{n\pi}{L}$

$Y(0) = C_3 \sin\mathrm{h}(\lambda 0) + C_4 \cosh(\lambda 0) = 0 \xrightarrow{yields} C_4 = 0$

$$T(x,y) = \sum_{n=1}^{\infty} C_n \sin\left(\frac{n\pi x}{L}\right) \sinh\left(\frac{n\pi y}{L}\right)$$

$$T(x,W) = \sum_{n=1}^{\infty} C_n \sin\left(\frac{n\pi x}{L}\right) \sin\mathrm{h}\left(\frac{n\pi W}{L}\right) = Ax$$

This equation has n coefficients. In order to solve these coefficients, we will use properties of orthogonal functions.

$$\int_0^L X(\lambda_n x)\, X(\lambda_m x) dx = \begin{cases} 0 & m \neq n \\ N(\lambda_m) & m = n \end{cases}$$

$\sin(\lambda_n x)$ is an orthogonal function. Therefore:

$$\int_0^L \frac{Ax}{\sin\mathrm{h}\left(\frac{n\pi W}{L}\right)} \sin\left(\frac{n\pi x}{L}\right) dx = C_n \int_0^L \sin^2\left(\frac{n\pi x}{L}\right) dx$$

$$\cos^2\left(\frac{n\pi x}{L}\right) + \sin^2\left(\frac{n\pi x}{L}\right) = 1$$

$$\cos^2\left(\frac{n\pi x}{L}\right) - \sin^2\left(\frac{n\pi x}{L}\right) = cos\left(\frac{2n\pi x}{L}\right)$$

$$\sin^2\left(\frac{n\pi x}{L}\right) = \frac{1}{2} - \frac{1}{2} cos\left(\frac{2n\pi x}{L}\right)$$

$$\int_0^L \sin^2\left(\frac{n\pi x}{L}\right)dx = \int_0^L \left[\frac{1}{2} - \frac{1}{2}cos\left(\frac{2n\pi x}{L}\right)\right]dx = \left[\frac{x}{2} - \frac{L}{4n\pi}sin\left(\frac{2n\pi x}{L}\right)\right]_0^L = \frac{L}{2}$$

$$C_n = \frac{AL}{2\sin h\left(\frac{n\pi W}{L}\right)}\int_0^L x\sin\left(\frac{n\pi x}{L}\right)dx$$

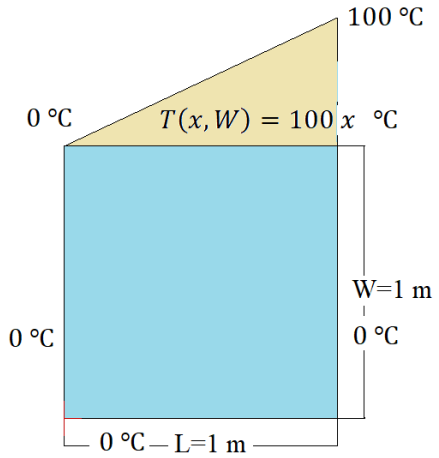
$$\int udv = uv - \int vdu$$

$$u = x \quad du = dx \quad dv = \sin\left(\frac{n\pi x}{L}\right)dx \quad v = -\frac{L}{n\pi}\cos\left(\frac{n\pi x}{L}\right)$$

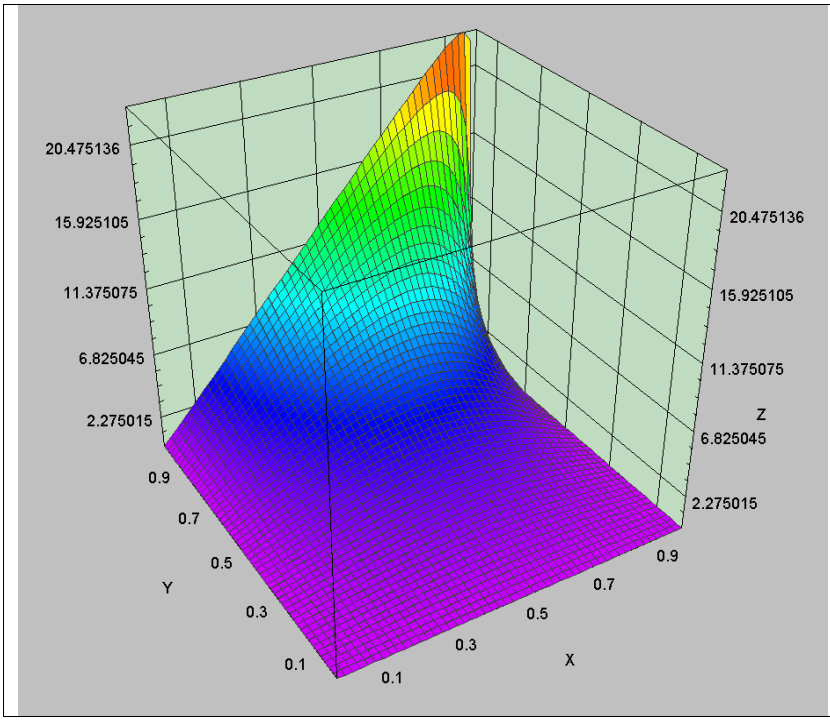
$$\int_0^L x\sin\left(\frac{n\pi x}{L}\right)dx = \left[-\frac{Lx}{n\pi}\cos\left(\frac{n\pi x}{L}\right)\right]_0^L - \int_0^L \sin\left(\frac{n\pi x}{L}\right)dx = -\frac{L^2}{n\pi}\cos(n\pi) = \frac{L^2}{n\pi}(-1)^{n+1}$$

$$C_n = \frac{AL^3}{2\sin h\left(\frac{n\pi W}{L}\right)}\frac{1}{n\pi}(-1)^{n+1}$$

Numerical example: A rectangle with L=1 m and height W=1 m , has 0 ℃ temperature and at the top side (y=W, $0 \le x \le L$) temperature profile is given as $T(x, W) = 100x$ ℃. Calculate the temperature variation in plate.




```
public class HT_plateX
{
public static void main(String arg[])
{double A=100.0;
double L=1.0;
double W=1.0;
if_xj T=(double z[])->{double x=z[0];double y=z[1];
double TT=0;
double Cn=0;
double pi=Math.PI;
int plus=1;
for(int n=1;n<100;n++)
{Cn=A*L*L*L/(2.0*Math.sinh(n*pi*W/L))/(n*pi)*plus;
  TT+=Cn*Math.sin(n*pi*x/L)*Math.sinh(n*pi*y/L);
  plus*=-1;
}
return TT;
}; //end of Temperature formula T
surfacePlot pp=new surfacePlot(T,0,L,0,W,50);
pp.plot();
}}
```



Python version:

```
from math import *
from f_xj import *;
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt


class f_rectangle(f_xj):
  def __init__(self, Li,Wi,T1i,Ai,ki):
    self.L=Li
    self.W=Wi
    self.T1=T1i
    self.A=Ai
    self.k=ki

  def func(self,x):
      C=0
      plusminus=1
      tetha=0.0
      for n in range(1,100):
          C=A*L*L*L/(2.0*sinh(n*pi*W/L))/(n*pi)*plusminus
          plusminus*=-1
          tetha=tetha+C*sin(n*pi*x[0]/L)*sinh(n*pi*x[1]/L)
      return tetha+T1

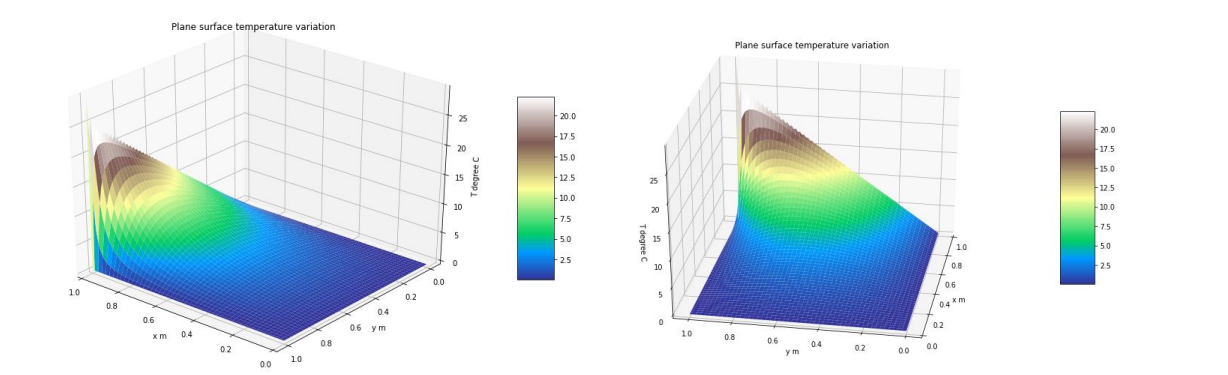
def plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLAbel,YLable,ZLabel):
    #f=f2()
    x=[0.0 for j in range(Nx)]
    y=[0.0 for j in range(Ny)]
    z=[[0.0 for j in range(Ny)] for i in range(Nx)]
    dx=(xmax-xmin)/(Nx-1)
    dy=(ymax-ymin)/(Ny-1)
    for i in range(Nx):
        for j in range(Ny):
            x[i]=xmin+dx*i
            y[j] = ymin+dy*j
            z[i][j]=f.funcxy(x[i],y[j])
    xi=np.array(x)
    yi=np.array(y)
    xi, yi = np.meshgrid(xi, yi)
    zi=np.array(z)
    #print(x[i][j]," ",y[i][j]," ",z[i][j])
    fig = plt.figure(figsize =(14, 9))
    ax = plt.axes(projection ='3d')
```

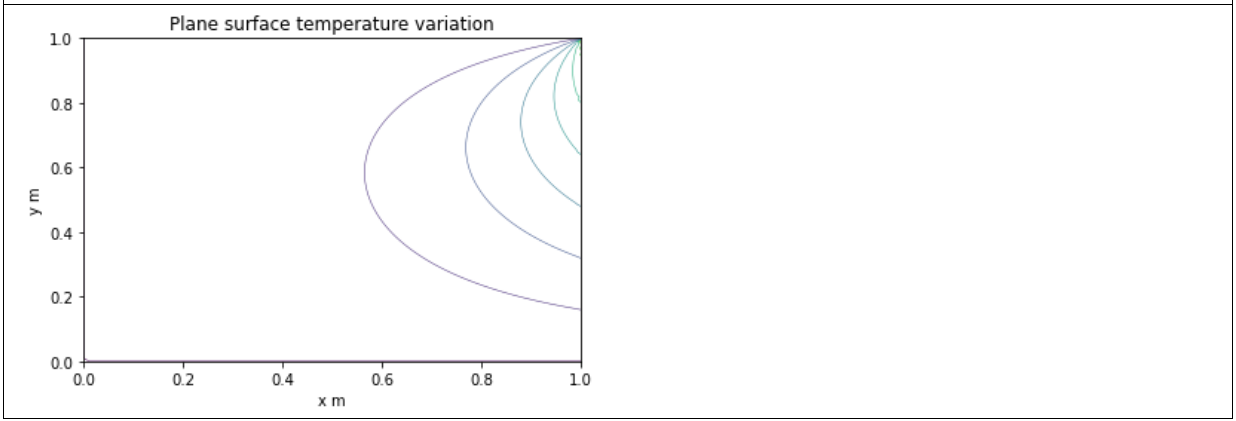
```
    # Creating color map
    my_cmap = plt.get_cmap('terrain')
    # Creating plot
    surf = ax.plot_surface(xi, yi, zi,
                    cmap = my_cmap,
                    edgecolor ='none')
    fig.colorbar(surf, ax = ax,
            shrink = 0.5,
            aspect = 5)

    ax.set_title(PLabel)
    ax.set_xlabel(XLabel)
    ax.set_xlim(xmin, xmax)
    ax.set_ylabel(YLabel)
    ax.set_xlim(ymin, ymax)
    ax.set_zlabel(ZLabel)
    ax.set_zlim(np.min(zi), np.max(zi))
    ax.view_init(30, 130)
    plt.show()

# Creating dataset
def contour_plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLable,ZLabel):
    #f=f2()
    x=[0.0 for j in range(Nx)]
    y=[0.0 for j in range(Ny)]
    z=[[0.0 for j in range(Ny)] for i in range(Nx)]
    dx=(xmax-xmin)/(Nx-1)
    dy=(ymax-ymin)/(Ny-1)
    for i in range(Nx):
        for j in range(Ny):
            x[i]=xmin+dx*i
            y[j] = ymin+dy*j
            z[i][j]=f.funcxy(x[i],y[j])
    xi=np.array(x)
    yi=np.array(y)
    xi, yi = np.meshgrid(xi, yi)
    zi=np.array(z)
    fig, ax = plt.subplots(1, 1)
    # plots contour lines
    ax.contour(xi, yi, zi,linewidths=0.5)
    ax.set_title(PLabel)
    ax.set_xlabel(XLabel)
    ax.set_xlim(xmin, xmax)
    ax.set_ylabel(YLabel)
    ax.set_xlim(ymin, ymax)
    plt.show()

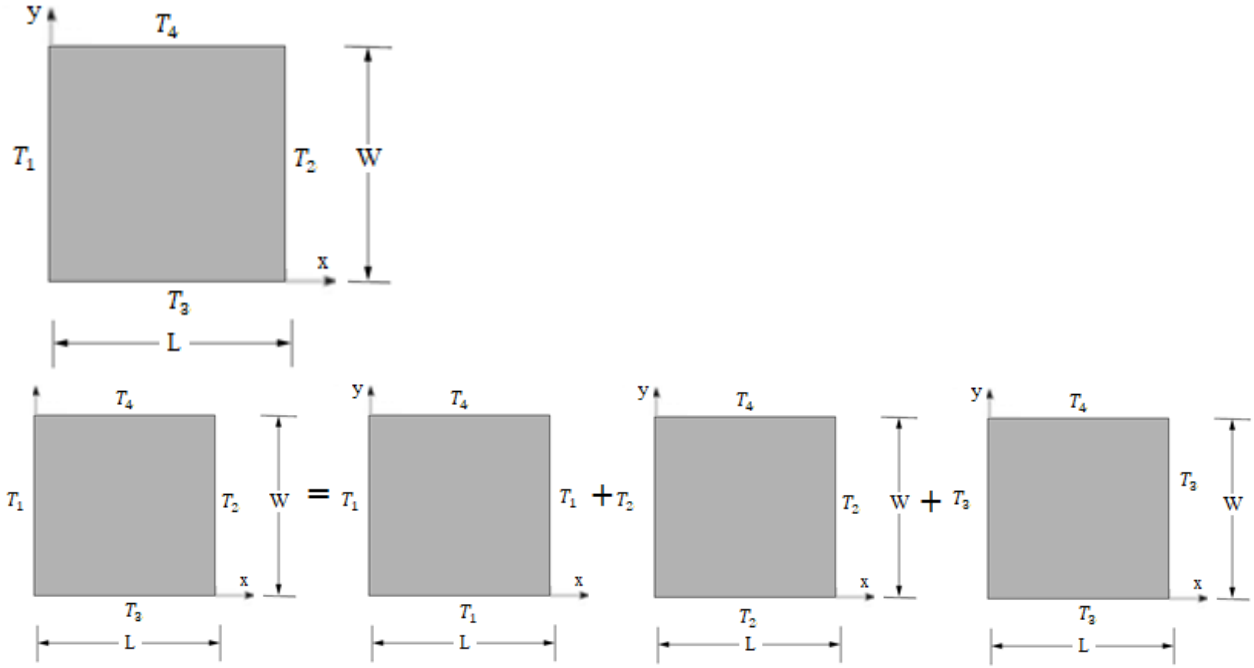

W=1.0
L=1.0
k=50.0
T1=0.0
A=100.0
f=f_rectangle(W,L,T1,A,k)
xmin=0.0
xmax=L
Nx=100
ymin=0.0
ymax=W
Ny=100
PLabel="Plane surface temperature variation"
XLabel="x m"
YLabel="y m"
ZLabel="T degree C"
plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLabel,ZLabel)
contour_plot(f,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLabel,ZLabel)
```





PROBLEM 3.7 SEPARATION OF VARIABLES, NON-HOMOGENEOUS, STEADY STATE SOLUTION; CARTESIAN COORDINATES , SUPERPOSITION PRINCIPLE

Simple homegeneous problems can be added to obtain a non-homogeneous case solution. This is called superposition principle. As an example case, consider a plate with each side has a different constant temperature.



$$\frac{\partial^2 T^*(x,y)}{\partial x^2}+\frac{\partial^2 T^*(x,y)}{\partial y^2}=0$$

$T^*(x,y)=T_1$ x=0 $0\le y\le W$

$T^*(x,y)=T_2$ x=L $0\le y\le W$

$T^*(x,y)=T_3$ y=0 $0\le x\le L$

$T^*(x,y) = T_4$ y=W $0 \le x \le L$

$T(x,y) = T^*(x,y) - T_1$

$$\frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial y^2} = 0$$

$T(x,y) = 0$ x=0 $0 \le y \le W$

$T(x,y) = (T_2 - T_1)$ x=L $0 \le y \le W$

$T(x,y) = (T_3 - T_1)$ y=0 $0 \le x \le L$

$T(x,y) = (T_4 - T_1)$ y=W $0 \le x \le L$

New problem can be constitute as superposition of three homegenous problem

$T(x,y) = T^*(x,y) - T_1 = \varphi_1(x,y) + \varphi_2(x,y) + \varphi_3(x,y)$

$$\frac{\partial^2 \varphi_1(x,y)}{\partial x^2} + \frac{\partial^2 \varphi_1(x,y)}{\partial y^2} = 0$$

$\varphi_1(x,y) = 0$ x=0 $0 \le y \le W$

$\varphi_1(x,y) = (T_2 - T_1)$ x=L $0 \le y \le W$

$\varphi_1(x,y) = 0$ y=0 $0 \le x \le L$

$\varphi_1(x,y) = 0$ y=W $0 \le x \le L$

$$\frac{\partial^2 \varphi_2(x,y)}{\partial x^2} + \frac{\partial^2 \varphi_2(x,y)}{\partial y^2} = 0$$

$\varphi_2(x,y) = 0$ x=0 $0 \le y \le W$

$\varphi_2(x,y) = 0$ x=L $0 \le y \le W$

$\varphi_2(x,y) = (T_3 - T_1)$ y=0 $0 \le x \le L$

$\varphi_2(x,y) = 0$ y=W $0 \le x \le L$

$$\frac{\partial^2 \varphi_3(x,y)}{\partial x^2} + \frac{\partial^2 \varphi_3(x,y)}{\partial y^2} = 0$$

$\varphi_3(x,y) = 0$ x=0 $0 \le y \le W$

$\varphi_3(x,y) = 0$ x=L $0 \le y \le W$

$\varphi_3(x,y) = 0$ y=0 $0 \le x \le L$

$\varphi_3(x,y) = (T_4 - T_1)$ y=W $0 \le x \le L$

Solution of each homegenous problem can be carried out similar to the previous problem

$$\varphi_1(x,y) = \sum_{n=1}^{\infty} C_{n1} sinh\left(\frac{n\pi}{W}x\right) sin\left(\frac{n\pi}{W}y\right)$$

$$C_{n1} = \frac{(T_2 - T_1)\frac{2W}{\pi}}{sinh\left(\frac{n\pi L}{W}\right)}\left[\frac{(-1)^{n+1}+1}{n}\right]$$

$$\varphi_2(x,y) = \sum_{n=1}^{\infty} C_{n2} sin\left(\frac{n\pi}{L}x\right) sinh\left(\frac{n\pi}{L}(W-y)\right)$$

$$C_{n2} = \frac{(T_3 - T_1)\frac{2L}{\pi}}{sinh\left(\frac{n\pi W}{L}\right)}\left[\frac{(-1)^{n+1}+1}{n}\right]$$

$$\varphi_3(x,y) = \sum_{n=1}^{\infty} C_{n3} sin\left(\frac{n\pi}{L}x\right) sinh\left(\frac{n\pi}{L}y\right)$$

$$C_{n3} = \frac{(T_4 - T_1)\frac{2L}{\pi}}{sinh\left(\frac{n\pi W}{L}\right)}\left[\frac{(-1)^{n+1}+1}{n}\right]$$

$T^*(x,y) = \varphi_1(x,y) + \varphi_2(x,y) + \varphi_3(x,y) + T_1$

Numerical problem: A plate has a length of L=1 m and width of W=1 m. side of the plate has the following temperature profiles: at x=0 temperature of T_1=20 ℃ , at x=L temperature of T_2=40 ℃, at y=0 temperature of T_3=60 at y=W temperature of T_3=80 ℃. Calculate the temperature distribution in the plate

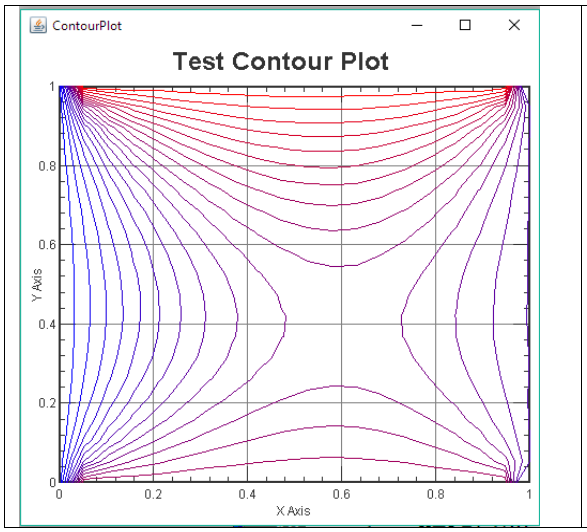
```
 import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

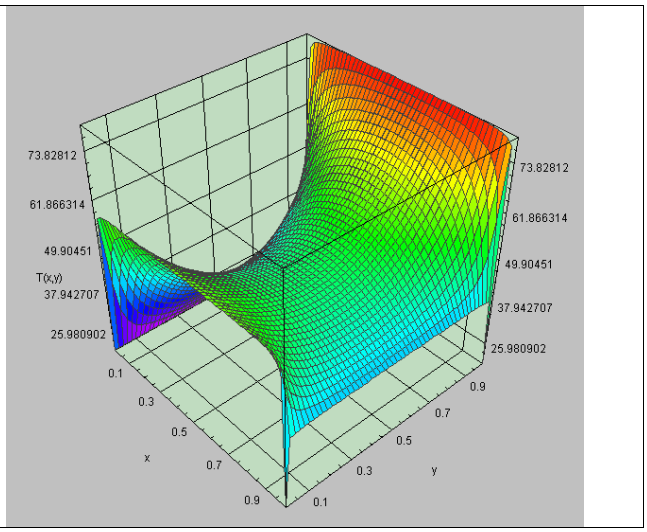
class f2 implements if_xj
{
public double T1,T2,T3,T4,W,L;
public f2(double T1i,double T2i,double T3i,double T4i,double Wi,double Li)
{T1=T1i;T2=T2i;T3=T3i;T4=T4i;W=Wi;L=Li;}
public double func(double x[])
{
double ff;
double PI=Math.PI;
double sum1=0;double sum2=0;double sum3=0;
int plusminus=1;
double Cn1=0;double Cn2=0;double Cn3=0;
double T5=(T2-T1);
double T6=(T3-T1);
double T7=(T4-T1);
//System.out.println("T1="+T1+"T2="+T2+"T3="+T3+"T4="+T4+"T5="+T5+"T6="+T6+"T7="+T7);
for(double n=1;n<200;n++)
{   Cn1=T5*2.0*W/PI/Math.sinh(n*PI*L/W)*(plusminus+1)/n;
              sum1+=Cn1*Math.sinh(n*PI*x[0]/W)*Math.sin(n*PI*x[1]/W);
              Cn2=T6*2.0*L/PI/Math.sinh(n*PI*W/L)*(plusminus+1)/n;
              sum2+=Cn2*Math.sin(n*PI*x[0]/L)*Math.sinh(n*PI*(W-x[1])/L);
              Cn3=T7*2.0*L/PI/Math.sinh(n*PI*W/L)*(plusminus+1)/n;
              sum3+=Cn3*Math.sin(n*PI*x[0]/L)*Math.sinh(n*PI*x[1]/L);
   plusminus*=-1;
}
double T=sum1+sum2+sum3+T1;
return T;
}
}

 public class HT_duzlem1A
 {
 public static void main(String[] args)
 { double T1=20.0;
   double T2=40.0;
   double T3=60.0;
   double T4=80.0;
   double L=1.0;
   double W=1.0;
             double TT[][]=new double[21][21];
   double x=0,y=0;
   f2 f=new f2(T1,T2,T3,T4,L,W);
   for(int i=0;i<=20;i++)
   {   x=i*L/20.0;
                for(int j=0;j<=20;j++)
                {y=j*W/20.0;
                double xx[]={x,y};
                 TT[j][i]=f.func(xx);
                }
              }
   Text.print(TT,"Temperature profile");
   ContourPlot p = new ContourPlot(f,0.0,L,20,0.0,W,20,20, false,"Test Contour Plot", "X Axis", "Y Axis");
   p.plot();
   surfacePlot sp=new surfacePlot(f,0.0,L,0.0,W,"x","y","T(x,y)");
   sp.plot();
 }
 }
```

Temperature profile

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
20.0	59.18771...	59.58817...	59.71960...	59.78340...	59.81995...	59.84262...	59.85710...	59.86612...	59.87109...	59.87267...	59.87109...	59.86612...	59.85710...	59.84262...	59.81995...	59.78340...	59.71960...	59.58817...	59.18771...	20.00000...
20.0	40.10942...	48.40764...	52.11702...	54.15484...	55.43218...	56.29796...	56.90996...	57.34811...	57.65590...	57.85750...	57.96536...	57.98323...	57.90636...	57.71871...	57.38491...	56.83002...	55.88303...	54.09157...	49.99999...	39.59385...
20.0	32.03005...	40.43761...	45.66050...	48.99688...	51.24917...	52.83830...	53.98825...	54.82361...	55.41615...	55.80738...	56.01937...	56.05944...	55.92059...	55.57774...	54.97748...	54.01544...	52.48794...	50.0	45.90842...	39.79408...
20.0	28.53976...	35.65233...	40.98401...	44.86202...	47.69913...	49.80255...	51.37295...	52.53715...	53.37463...	53.93414...	54.24303...	54.31171...	54.13441...	53.68633...	52.91659...	51.73575...	50.00000...	47.51205...	44.11696...	39.85980...
20.0	26.72185...	32.75462...	37.76132...	41.74635...	44.86460...	47.28780...	49.15624...	50.57249...	51.60766...	52.30880...	52.70428...	52.80691...	52.61480...	52.11032...	51.25756...	50.00000...	48.26424...	45.98455...	43.16997...	39.89170...
20.0	25.66643...	30.94422...	35.58242...	39.49770...	42.71886...	45.32129...	47.38647...	48.98495...	50.17169...	50.98658...	51.45599...	51.59427...	51.40453...	50.87905...	50.00000...	48.74243...	47.08340...	45.02251...	42.61508...	39.90997...
20.0	25.02657...	29.80395...	34.14509...	37.95002...	41.19222...	43.88985...	46.08072...	47.80659...	49.10534...	50.00784...	50.53698...	50.70785...	50.52827...	50.0	49.12094...	47.88967...	46.31366...	44.42225...	42.28128...	39.92131...
20.0	24.64758...	29.11560...	33.25660...	36.97235...	40.21310...	42.96563...	45.23921...	47.05410...	48.43390...	49.40113...	49.97504...	50.17083...	50.0	49.47172...	48.59546...	47.38519...	45.86558...	44.07940...	42.09363...	39.92855...
20.0	24.45396...	28.76307...	32.80171...	36.47583...	39.72584...	42.52207...	44.85687...	46.73646...	48.17443...	49.18705...	49.79058...	50.00000...	49.82916...	49.29214...	48.40572...	47.19308...	45.68828...	43.94055...	42.01676...	39.93306...
20.0	24.40862...	28.68657...	32.71730...	36.40850...	39.69602...	42.54222...	44.93088...	46.86086...	48.34026...	49.38183...	50.00000...	50.20941...	50.02495...	49.46301...	48.54400...	47.29571...	45.75696...	43.98062...	42.03463...	39.93554...
20.0	24.49646...	28.86150...	32.97727...	36.74937...	40.11111...	43.02242...	45.46508...	47.43658...	48.94414...	50.0	50.61816...	50.81294...	50.59886...	49.99215...	49.01341...	47.69119...	46.06585...	44.19261...	42.14249...	39.93633...
20.0	24.71808...	29.28979...	33.58569...	37.50512...	40.98032...	43.97387...	46.47203...	48.47701...	50.0	51.05585...	51.65973...	51.82556...	51.56609...	50.89465...	49.82830...	48.39233...	46.62536...	44.58384...	42.34409...	39.93554...
20.0	25.08908...	29.99890...	34.57627...	38.71026...	42.33517...	45.42334...	47.97359...	50.0	51.52298...	52.56341...	53.13913...	53.26353...	52.94589...	52.19340...	51.01504...	49.42750...	47.46284...	45.17638...	42.65188...	39.93306...
20.0	25.64398...	31.04794...	36.01806...	40.43092...	44.23115...	47.41317...	49.99999...	52.02640...	53.52796...	54.53491...	55.06911...	55.14312...	54.76078...	53.91927...	52.61352...	50.84375...	48.62704...	46.01174...	43.09003...	39.92855...
20.0	26.44732...	32.54338...	38.02888...	42.77254...	46.74998...	50.00000...	52.58682...	54.57665...	56.02612...	56.97757...	57.45777...	57.47792...	57.03436...	56.11014...	54.67870...	52.71219...	50.19744...	47.16169...	43.70203...	39.92131...
20.0	27.61916...	34.67253...	40.79989...	45.89065...	50.00000...	53.25001...	55.76884...	57.66482...	59.01967...	59.88888...	60.30397...	60.27415...	59.78689...	58.80777...	57.28113...	55.13539...	52.30086...	48.75082...	44.56781...	39.90997...
20.0	29.39702...	37.77318...	44.63505...	50.0	54.10934...	57.22745...	59.56907...	61.28973...	62.49487...	63.25062...	63.59149...	63.52416...	63.02764...	62.04997...	60.50229...	58.25364...	55.13797...	51.00311...	45.84515...	39.89170...
20.0	32.30576...	42.47977...	49.99999...	55.36494...	59.20010...	61.97111...	63.98193...	65.42372...	66.41430...	67.02272...	67.28269...	67.19828...	66.74339...	65.85490...	64.41757...	62.23867...	59.01598...	54.33949...	47.88297...	39.85980...
20.0	37.71398...	50.00000...	57.52022...	62.22681...	65.32746...	67.45661...	68.95205...	70.00109...	70.71020...	71.13849...	71.31342...	71.23692...	70.88439...	70.19604...	69.05577...	67.24537...	64.34766...	59.56238...	51.59235...	39.79408...
20.0	49.99999...	62.28601...	67.69423...	70.60297...	72.38083...	73.55267...	74.35601...	74.91091...	75.28191...	75.50353...	75.59137...	75.54603...	75.35241...	74.97342...	74.33356...	73.27814...	71.46023...	67.96994...	59.89057...	39.59385...
20.0	78.78156...	79.38226...	79.57940...	79.67511...	79.72992...	79.76394...	79.78565...	79.79919...	79.80663...	79.80901...	79.80663...	79.79919...	79.78565...	79.76394...	79.72992...	79.67511...	79.57940...	79.38226...	78.78156...	20.00000...

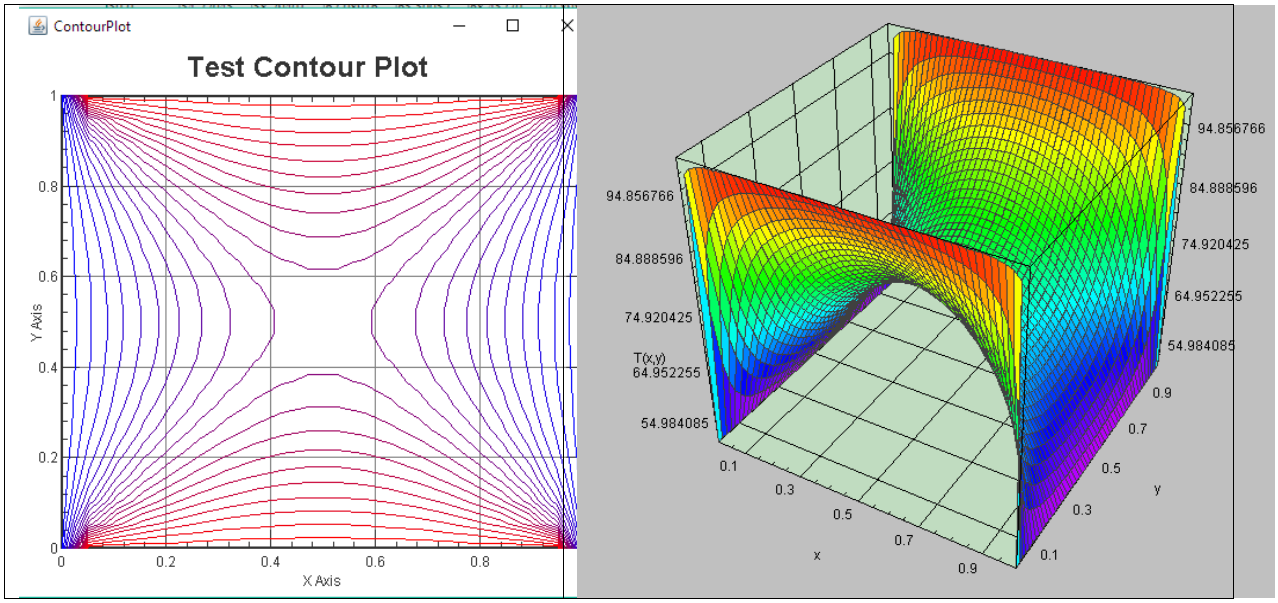




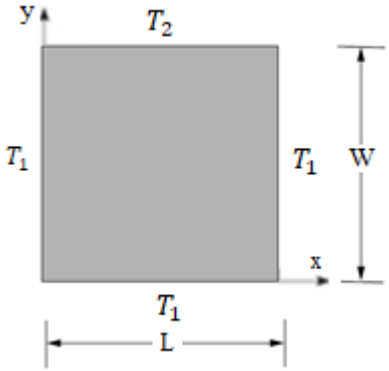
Numerical problem: A plate has a length of L=1 m and width of W=1 m. side of the plate has the following temperature profiles: at x=0 temperature of T_1=50 ℃ , at x=L temperature of T_2=50 ℃, at y=0 temperature of T_3=100 ℃ at y=W temperature of T_3=100 ℃. Calculate the temperature distribution in the plate. double T1=50.0; double T2=50.0;double T3=100.0;double T4=100.0;

Temperature profile

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
50.0	98.98463...	99.48521...	99.64950...	99.72926...	99.77493...	99.80328...	99.82138...	99.83265...	99.83886...	99.84084...	99.83886...	99.83265...	99.82138...	99.80328...	99.77493...	99.72926...	99.64950...	99.48521...	98.98463...	50.00000...
50.0	74.99999...	85.23599...	89.73578...	92.14561...	93.60359...	94.54461...	95.16399...	95.55884...	95.77954...	95.85065...	95.77954...	95.55884...	95.16399...	94.54461...	93.60359...	92.14561...	89.73578...	85.23599...	75.0	50.00000...
50.0	64.76400...	75.0	81.25510...	85.15141...	87.69058...	89.39647...	90.54540...	91.28783...	91.70598...	91.84117...	91.70598...	91.28783...	90.54540...	89.39647...	87.69058...	85.15141...	81.25510...	75.0	64.76400...	50.00000...
50.0	60.26421...	68.74489...	74.99999...	79.43793...	82.57293...	84.78590...	86.32271...	87.33464...	87.91083...	88.09804...	87.91083...	87.33464...	86.32271...	84.78590...	82.57293...	79.43793...	75.0	68.74489...	60.26421...	50.00000...
50.0	57.85438...	64.84858...	70.56206...	75.0	78.35431...	80.83611...	82.61492...	83.81040...	84.49947...	84.72464...	84.49947...	83.81040...	82.61492...	80.83611...	78.35431...	75.0	70.56206...	64.84858...	57.85438...	50.00000...
50.0	56.39640...	62.30941...	67.42706...	71.64568...	74.99999...	77.58066...	79.48336...	80.78694...	81.54729...	81.79716...	81.54729...	80.78694...	79.48336...	77.58066...	75.0	71.64568...	67.42706...	62.30941...	56.39640...	50.00000...
50.0	55.45538...	60.60352...	65.21409...	69.16388...	72.41933...	75.0	76.94693...	78.30282...	79.10194...	79.36588...	79.10194...	78.30282...	76.94693...	75.0	72.41933...	69.16388...	65.21409...	60.60352...	55.45538...	50.00000...
50.0	54.83600...	59.45459...	63.67728...	67.38507...	70.51663...	73.05306...	75.0	76.37327...	77.18938...	77.46003...	77.18938...	76.37327...	75.0	73.05306...	70.51663...	67.38507...	63.67728...	59.45459...	54.83600...	50.00000...
50.0	54.44115...	58.71216...	62.66535...	66.18959...	69.21305...	71.69717...	73.62672...	75.0	75.82098...	76.09404...	75.82098...	75.0	73.62672...	71.69717...	69.21305...	66.18959...	62.66535...	58.71216...	54.44115...	50.0
50.0	54.22045...	58.29401...	62.08916...	65.50052...	68.45270...	70.89805...	72.81061...	74.17901...	75.0	75.27355...	75.0	74.17901...	72.81061...	70.89805...	68.45270...	65.50052...	62.08916...	58.29401...	54.22045...	50.0
50.0	54.14934...	58.15882...	61.90195...	65.27535...	68.20283...	70.63411...	72.53996...	73.90595...	74.72644...	75.0	74.72644...	73.90595...	72.53996...	70.63411...	68.20283...	65.27535...	61.90195...	58.15882...	54.14934...	50.0
50.0	54.22045...	58.29401...	62.08916...	65.50052...	68.45270...	70.89805...	72.81061...	74.17901...	75.0	75.27355...	75.0	74.17901...	72.81061...	70.89805...	68.45270...	65.50052...	62.08916...	58.29401...	54.22045...	50.0
50.0	54.44115...	58.71216...	62.66535...	66.18959...	69.21305...	71.69717...	73.62672...	75.0	75.82098...	76.09404...	75.82098...	75.0	73.62672...	71.69717...	69.21305...	66.18959...	62.66535...	58.71216...	54.44115...	50.0
50.0	54.83600...	59.45459...	63.67728...	67.38507...	70.51663...	73.05306...	75.0	76.37327...	77.18938...	77.46003...	77.18938...	76.37327...	75.0	73.05306...	70.51663...	67.38507...	63.67728...	59.45459...	54.83600...	50.00000...
50.0	55.45538...	60.60352...	65.21409...	69.16388...	72.41933...	75.0	76.94693...	78.30282...	79.10194...	79.36588...	79.10194...	78.30282...	76.94693...	75.0	72.41933...	69.16388...	65.21409...	60.60352...	55.45538...	50.00000...
50.0	56.39640...	62.30941...	67.42706...	71.64568...	74.99999...	77.58066...	79.48336...	80.78694...	81.54729...	81.79716...	81.54729...	80.78694...	79.48336...	77.58066...	75.0	71.64568...	67.42706...	62.30941...	56.39640...	50.00000...
50.0	57.85438...	64.84858...	70.56206...	75.0	78.35431...	80.83611...	82.61492...	83.81040...	84.49947...	84.72464...	84.49947...	83.81040...	82.61492...	80.83611...	78.35431...	75.0	70.56206...	64.84858...	57.85438...	50.00000...
50.0	60.26421...	68.74489...	74.99999...	79.43793...	82.57293...	84.78590...	86.32271...	87.33464...	87.91083...	88.09804...	87.91083...	87.33464...	86.32271...	84.78590...	82.57293...	79.43793...	75.0	68.74489...	60.26421...	50.00000...
50.0	64.76400...	75.0	81.25510...	85.15141...	87.69058...	89.39647...	90.54540...	91.28783...	91.70598...	91.84117...	91.70598...	91.28783...	90.54540...	89.39647...	87.69058...	85.15141...	81.25510...	75.0	64.76400...	50.00000...
50.0	74.99999...	85.23599...	89.73578...	92.14561...	93.60359...	94.54461...	95.16399...	95.55884...	95.77954...	95.85065...	95.77954...	95.55884...	95.16399...	94.54461...	93.60359...	92.14561...	89.73578...	85.23599...	75.0	50.00000...
50.0	98.98463...	99.48521...	99.64950...	99.72926...	99.77493...	99.80328...	99.82138...	99.83265...	99.83886...	99.84084...	99.83886...	99.83265...	99.82138...	99.80328...	99.77493...	99.72926...	99.64950...	99.48521...	98.98463...	50.00000...



PROBLEM 3.8 SEPARATION OF VARIABLES, NON-HOMOGENEOUS, STEADY STATE SOLUTION; CARTESIAN COORDINATES, HEAT GENERATION



$$\frac{\partial^2 T^*(x,y)}{\partial x^2}+\frac{\partial^2 T^*(x,y)}{\partial y^2}+\frac{g}{k}=0$$

$T^*(x,y)=T_1$ x=0 $0\le y\le W$
$T^*(x,y)=T_1$ x=L $0\le y\le W$
$T^*(x,y)=T_1$ y=0 $0\le x\le L$
$T^*(x,y)=T_2$ y=W $0\le x\le L$

$T(x,y)=T^*(x,y)-T_1$

$$\frac{\partial^2 T(x,y)}{\partial x^2}+\frac{\partial^2 T(x,y)}{\partial y^2}+\frac{g}{k}=0$$

$T(x,y)=0$ x=0 $0\le y\le W$
$T(x,y)=0$ x=L $0\le y\le W$
$T(x,y)=0$ y=0 $0\le x\le L$
$T(x,y)=T_2-T_1=T_3$ y=W $0\le x\le L$

This equation can be divided into two equation to define a homogeneous problem
$T(x,y)=\psi(x,y)+\phi(x)$
$\psi(x,y)=T(x,y)-\phi(x)$
where

$$\frac{\partial^2 \psi(x,y)}{\partial x^2}+\frac{\partial^2 \psi(x,y)}{\partial y^2}=0$$

$\psi(x,y) = 0$ x=0 $0 \le y \le W$
$\psi(x,y) = 0$ x=L $0 \le y \le W$
$\psi(x,y) = 0$ y=0 $0 \le x \le L$
$\psi(x,y) = (T_2 - T_1) - \phi(x) = T_3 - \phi(x)$ y=W $0 \le x \le L$

and

$$\frac{\partial^2 \phi(x)}{\partial x^2} + \frac{g}{k} = 0$$
$\phi(x) = 0$ x=0
$\phi(x) = 0$ x=L

Solution:

$$\frac{\partial \phi(x)}{\partial x} = -\frac{g}{k}x + C_1$$
$$\phi(x) = -\frac{g}{k}\frac{x^2}{2} + C_1 x + C_2$$
$$\phi(x) = 0 \quad x=0 \xrightarrow{yields} C_2 = 0$$
$$\phi(x) = 0 \quad x=L \xrightarrow{yields} C_1 = \frac{g}{k}\frac{L}{2}$$
$$\phi(x) = -\frac{g}{k}\frac{x^2}{2} + \frac{g}{k}\frac{L}{2}x$$
$$\psi(x,y) = \sum_{n=1}^{\infty} C_n sin\left(\frac{n\pi}{L}x\right) sinh\left(\frac{n\pi}{L}y\right)$$
$$\psi(x,W) = \sum_{n=1}^{\infty} C_n sin\left(\frac{n\pi}{L}x\right) sinh\left(\frac{n\pi}{L}W\right) = T_3 - \phi(x) = (T_2 - T_1) + \frac{g}{k}\frac{x^2}{2} - \frac{g}{k}\frac{L}{2}x$$

Orthogonality conditions:

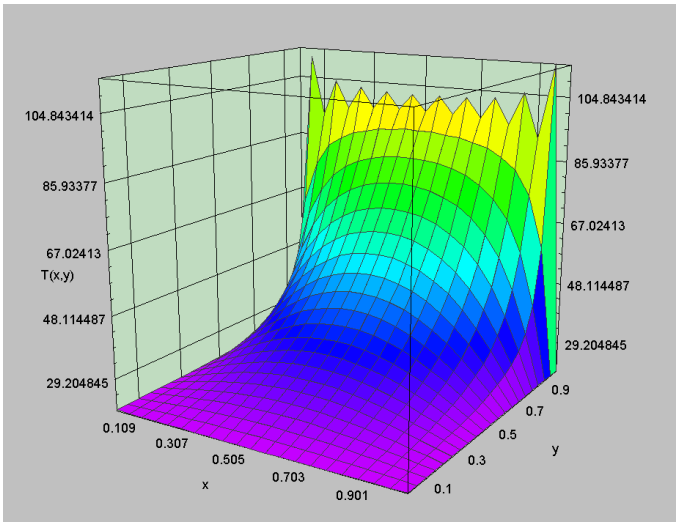
$$C_n\left[\int_{x=0}^{L} sin^2\left(\frac{n\pi}{L}x\right)dx\right] = \int_{x=0}^{L}\left((T_2 - T_1) + \frac{g}{k}\frac{x^2}{2} - \frac{g}{k}\frac{L}{2}x\right) sin\left(\frac{n\pi}{L}x\right)dx$$
$$N = \left[\int_{x=0}^{L} sin^2\left(\frac{n\pi}{L}x\right)dx\right] = \frac{L}{2}$$
$$C_n = \frac{2}{Lsinh\left(\frac{n\pi}{L}W\right)}\int_{x=0}^{L}\left((T_2 - T_1) + \frac{g}{k}\frac{x^2}{2} - \frac{g}{k}\frac{L}{2}x\right) sin\left(\frac{n\pi}{L}x\right)dx$$
$$T(x,y) = T^*(x,y) - T_1 = \sum_{n=1}^{\infty} C_n sin\left(\frac{n\pi}{L}x\right) sinh\left(\frac{n\pi}{L}y\right) + \frac{g}{k}\frac{x^2}{2} - \frac{g}{k}\frac{L}{2}x$$
$$T^*(x,y) = \sum_{n=1}^{\infty} C_n sin\left(\frac{n\pi}{L}x\right) sinh\left(\frac{n\pi}{L}y\right) + \frac{g}{k}\frac{x^2}{2} - \frac{g}{k}\frac{L}{2}x + T_1$$

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

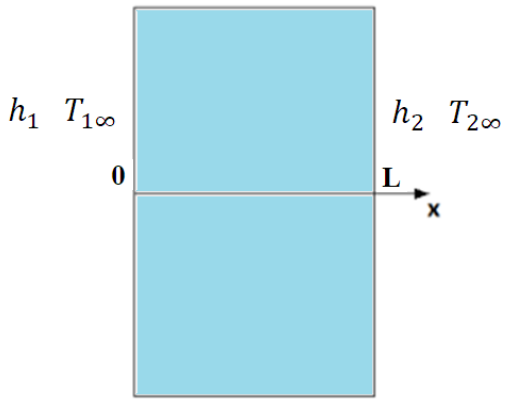
class f2 implements if_xj
{
double T1,T2,T3,W,L,g,k;
public f2(double T1i,double T2i,double Wi,double Li,double gi,double ki)
{T1=T1i;T2=T2i;W=Wi;L=Li;T3=T2-T1;g=gi;k=ki;}
public double func(double x[])
{
//çözümü istenen fonksiyon
double ff;
double PI=Math.PI;
double sum=g/k*x[0]*x[0]/2.0-g/k*L/2.0*x[0];
double plusminus=1;
```

```
double Cn=0;
for(int n=1;n<20;n++)
{   final int n1=n;
                if_x fa=(xx)->(T3+g/k*xx*xx/2.0-g/k*L/2.0*xx)*Math.sin(n1*Math.PI*xx/L);
                Cn=2.0/(L*Math.sinh(n*Math.PI*W/L))*fa.integral(0,L,20);
                sum+=Cn*Math.sin(n*PI*x[0]/L)*Math.sinh(n*PI*x[1]/L);
}
sum+=T1;
return sum;
}
}

 public class HT_duzlem_Heat_generationA
 {
 public static void main(String[] args)
 { double T1=20.0;
  double T2=100.0;
  double L=1.0;
  double W=1.0;
  double g=100.0;
  double k=50.0;
                f2 f=new f2(T1,T2,L,W,g,k);
  ContourPlot p = new ContourPlot(f,0.0,L,20,0.0,W,20,50, false,"Test Contour Plot", "X Axis", "Y Axis");
  p.plot();
  surfacePlot sp=new surfacePlot(f,0.01,L,0.0,W,"x","y","T(x,y)");
  sp.plot();
 }
 }
```



PROBLEM 3.9 SEPARATION OF VARIABLES, NON-HOMOGENEOUS, TIME DEPENDENT SOLUTION; CARTESIAN COORDINATES



$$\frac{\partial^2 T^*(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T^*(x,t)}{\partial t}$$

Boundary conditions

$-\frac{\partial T^*(x,t)}{\partial x} + H_1(T_{1\infty} - T^*(x,t)) = 0$ x=0 t >0 $H_1 = \frac{h_1}{k}$

$\frac{\partial T^*(x,t)}{\partial x} + H_2(T^*(x,t) - T_{2\infty}) = 0$ x=L t>0 $H_2 = \frac{h_2}{k}$

$T^*(x,t) = F^*(x)$ t=0 $0 \le x \le L$

$$\boldsymbol{T(x,t) = T^*(x,t) - T_{2\infty}}$$

$$\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T(x,t)}{\partial t}$$

Boundary conditions

$-\frac{\partial T(x,t)}{\partial x} + H_1[T_{1\infty} - (T(x,t) + T_{1\infty})] = 0$ x=0 t >0 $H_1 = \frac{h_1}{k}$

$-\frac{\partial T(x,t)}{\partial x} + H_1[(T_{1\infty} - T_{1\infty}) - T(x,t)] = 0$ x=0 t >0 $H_1 = \frac{h_1}{k}$

$\frac{\partial T(x,t)}{\partial x} + H_2 T(x,t) = 0$ x=L t>0 $H_2 = \frac{h_2}{k}$

$T(x,t) = F^*(x) - T_{2\infty}$ t=0 $0 \le x \le L$

$$\boldsymbol{T(x,t) = T_H(x,t) + T_{SS}(x)}$$

$$\frac{\partial^2 T_H(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T_H(x,t)}{\partial t}$$

$-\frac{\partial T_H(x,t)}{\partial x} + H_1 T_H(x,t) = 0$ x=0 t >0 $H_1 = \frac{h_1}{k}$

$-\frac{\partial T_H(x,t)}{\partial x} - H_2 T_H(x,t) = 0$ x=L t>0 $H_2 = \frac{h_2}{k}$

$T_H(x,t) = F^*(x) - T_{2\infty} - T_{SS}(x)$ t=0 $0 \le x \le L$

$$\frac{\partial^2 T_{SS}(x)}{\partial x^2} = \mathbf{0}$$

$-\frac{\partial T_{SS}(x)}{\partial x} + H_1 T_{SS}(x) = H_1(T_{1\infty} - T_{1\infty})$ x=0 t >0 $H_1 = \frac{h_1}{k}$

$-\frac{\partial T_{SS}(x)}{\partial x} - H_2 T_{SS}(x) = 0$ x=L t>0 $H_2 = \frac{h_2}{k}$

Solution of T_H is given before, so it will not be repeated here.

Solution of T_{SS}:

$\frac{\partial T_{SS}(x,t)}{\partial x} = C_1$

$T_{SS}(x) = C_1 x + C_2$

$-C_1 + H_1(C_1 0 + C_2) = H_1(T_{1\infty} - T_{2\infty})$

$C_1 + H_2(C_1 L + C_2) = 0$

$C_1 = -\frac{H_2}{1+H_2 L} C_2$

$C_2 = \frac{H_1(T_{1\infty} - T_{2\infty})}{H_1 + \frac{H_2}{1+H_2 L}}$

$$T_H(x,t) = \sum_{m=1}^{\infty}\left[\frac{1}{N(\beta_m)}\int_0^L F(x)[\beta_m \cos(\beta_m x) + H_1 \sin(\beta x)]dx\right][\beta_m \cos(\beta_m x) + H_1 \sin(\beta x)]\exp(-\beta_m^2 \alpha t)$$

Program code:

```
import java.io.*;
import javax.swing.*;
import java.awt.*;
import static java.lang.Math.*;
public class HT_plane_wall3
{
public if_x F;
public double h1,h2,L,k,alpha,Bi1,Bi2,Ti,T1inf,T2inf,H1,H2,C1,C2;
public  HT_plane_wall3(double h1i,double h2i,double Li,double ki,double alphai,double Tii,double Tinf1i,double Tinf2i)
{h1=h1i;h2=h2i;k=ki;L=Li;alpha=alphai;Bi1=h1*L/k;Bi2=h2*L/k;Ti=Tii;T1inf=Tinf1i;T2inf=Tinf2i;H1=h1/k;H2=h2/k;
 //coefficients of Tss Tss=C1*x+C2
 C2=H1*(T1inf-T2inf)/(H1+H2/(1+H2*L));
 C1=-H2/(1.0+H2*L)*C2;
 //initial temperature profile is defined as constant change
 //if you desired to do so
 F=xx->Ti-T2inf-Tss(xx);
```

```
}
  public double Tss(double x)  {return C1*x+C2;}

  public static double ksi(int n,double Bi1,double Bi2)
  {double x1=n*PI+0.0000001;
   double x2=x1+(PI/2-0.000001);
   if_x f1=x->(x*x-Bi1*Bi2)*Math.tan(x)-x*(Bi1+Bi2);
   double ksi=f1.illinois(x1,x2);
   return ksi;
  }


  public double teta(double x,double t)
  {double ksi=0;
   double beta=0;
   double C=0;
   double N=0,C1=0,C2=0;
   double teta=0;
   //coefficient of equation
   for(int n=0;n<10;n++)
   {ksi=ksi(n,Bi1,Bi2);
    beta=ksi/L;
    final double beta1=beta;
    N=0.5*((beta*beta+H1*H1)*(L+H2/(beta*beta+H2*H2))+H1);
    if_x Cn=xx->F.func(xx)*(beta1*cos(beta1*xx)+H1*sin(beta1*xx));
    C=1.0/N*Cn.integral(0,L,20);
    teta+=C*(beta*cos(beta*x)+H1*sin(beta*x))*exp(-beta*beta*alpha*t);
    }
    return teta;
  }
  public double dteta(double x,double t)
  {double ksi=0;
   double beta=0;
   double C=0;
   double N=0,C1=0,C2=0;
   double dteta=0;
   //coefficient of equation
   for(int n=0;n<10;n++)
   {ksi=ksi(n,Bi1,Bi2);
    beta=ksi/L;
    final double beta1=beta;
    N=0.5*((beta*beta+H1*H1)*(L+H2/(beta*beta+H2*H2))+H1);
    if_x Cn=xx->F.func(xx)*(beta1*cos(beta1*xx)+H1*sin(beta1*xx));
    C=1.0/N*Cn.integral(0,L,20);
    dteta+=C*(-beta*beta*sin(beta*x)+H1*beta*cos(beta*x))*exp(-beta*beta*alpha*t);
    }
    return dteta;
  }
  public double T(double x,double t)
  {double T=teta(x,t)+T2inf+Tss(x);
   return T;
  }
  public double q(double x,double t)
  {double q=-k*(dteta(x,t)+C1);
   return q;
  }
}
```

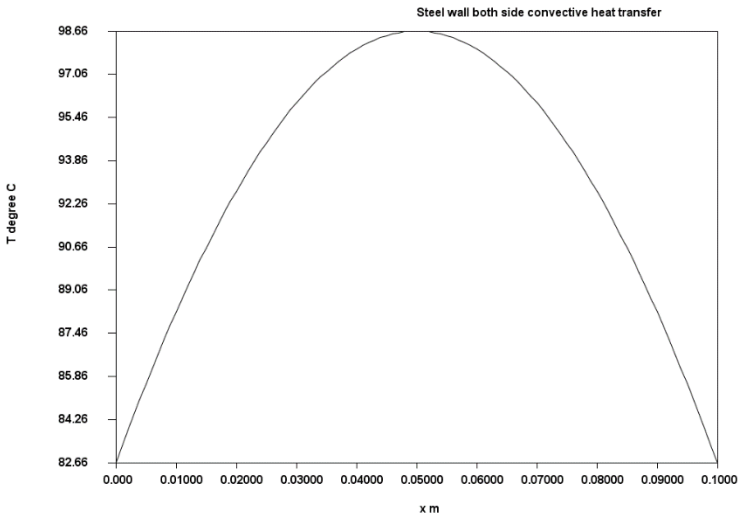
Numerical example:

A steel plate [α=1.2e-5 and k=43 W/(mK)] of thickness L=0.1 m, initially at a uniform temperature of T_i=240 ℃, is suddenly immersed in an oil bath at $T_{1\infty} = 40$ ℃ and $T_{2\infty} = 40$ ℃ . The convective heat transfer coefficients are h_1=600 W/(m^2K), h_2=600 W/(m^2K) calculate temperature profile for t=478.8 seconds.

F(x)= T_i -$T_{2\infty} - Tss(x)$

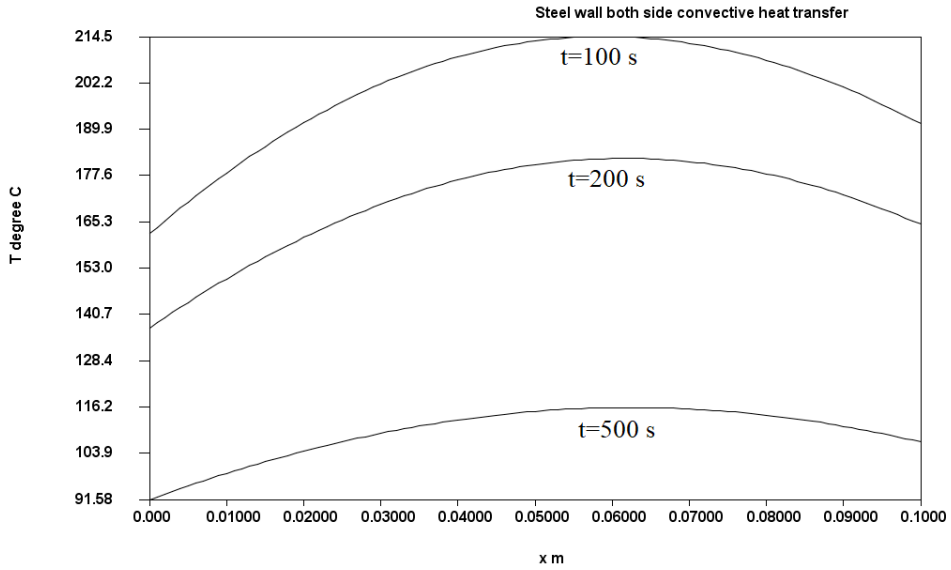
```
 public class HT_plane_wall3_test1
{
public static void main (String args[])
{ double h1=600; //W/m^2K
  double h2=600; //W/m^2K
  double k=42;
  double alpha=1.2e-5;
```

```
  double L=0.1;
  double t[]={478.8};  //s
  double T1inf=40;
  double T2inf=40;
  double Ti=240;
  int n1=t.length;
  double x[]=new double[101];
  double T[][]=new double[n1][101];
  double q[][]=new double[n1][101];
  HT_plane_wall3 htpw=new HT_plane_wall3(h1,h2,L,k,alpha,Ti,T1inf,T2inf);
  //initial temperature variation t=0
  htpw.F=xx->(Ti-T2inf)-htpw.Tss(xx);
  for(int j=0;j<n1;j++)
  {
  for(int i=0;i<=100;i++)
      {   x[i]=i*L/100;
                  T[j][i]=htpw.T(x[i],t[j]);
                  q[j][i]=htpw.q(x[i],t[j]);
      }
   }
      Plot pp=new Plot(x,T[0]);
      for(int j=1;j<n1;j++)
      {pp.addData(x,T[j]);}
      pp.setPlabel("Steel wall both side convective heat transfer");
      pp.setXlabel(" x m");
      pp.setYlabel(" T degree C");
      pp.plot();
      Plot pp1=new Plot(x,q[0]);
      for(int j=1;j<n1;j++)
      {pp1.addData(x,q[j]);}
      pp1.setPlabel("Steel wall both side convective heat transfer");
      pp1.setXlabel(" x m");
      pp1.setYlabel(" q W/m^2");
      pp1.plot();
}
}
```



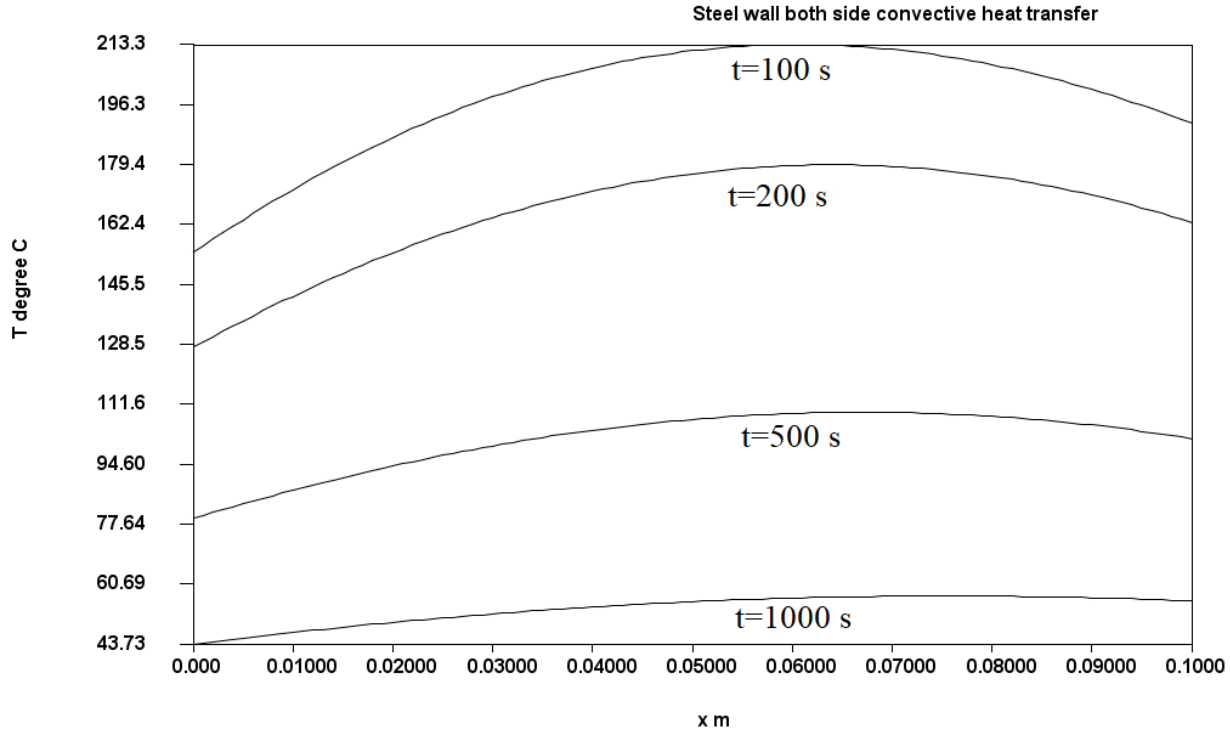
Numerical example:

A steel plate [α=1.2e-5 and k=43 W/(mK)] of thickness L=0.1 m, initially at a uniform temperature of 240 ℃, is suddenly immersed in oil baths at $T_{1\infty} = 40$ ℃ and $T_{2\infty} = 40$ ℃ . The convective heat transfer coefficients are h_1=600 W/(m^2K), h_2=300 W/(m^2K) calculate temperature profile for t=100,200 and 500 seconds.



Numerical example:

A steel plate [α=1.2e-5 and k=43 W/(mK)] of thickness L=0.1 m, initially at a uniform temperature of T_i= 240 ℃, is suddenly immersed in oil baths at one side $T_{1\infty} = 20$ ℃ and the other side $T_{2\infty} = 40$ ℃. The convective heat transfer coefficients are h_1=600 W/(m^{2}K), h_2=300 W/(m^{2}K) calculate temperature profile for t=100,200, 500 and 1000 seconds.



Numerical example:

A steel plate [α=1.2e-5 and k=43 W/(mK)] of thickness L=0.1 m, initially has a temperature profile as;

0	60
0.01	45.6
0.02	34.4
0.03	26.4
0.04	21.6
0.05	20
0.06	21.6
0.07	26.4
0.08	34.4
0.09	45.6
0.1	60

is suddenly immersed in an oil bath at$T_{1\infty} = 20$ ℃ and the other side $T_{2\infty} = 40$ ℃The convective heat transfer coefficients are h_1=100 W/(m^2K), h_2=100 W/(m^2K) calculate temperature profile for t=20,200,500 and 1000 seconds.

F(x)=?

Least square curve fitting $f(x)$

Data:

n		x	y	x^2	x^3	x^4	xy	x^2y
1		0	60	0	0	0	0	0
2		0.01	45.6	0.0001	0.000001	1E-08	0.456	0.00456
3		0.02	34.4	0.0004	0.000008	1.6E-07	0.688	0.01376
4		0.03	26.4	0.0009	0.000027	8.1E-07	0.792	0.02376
5		0.04	21.6	0.0016	0.000064	2.56E-06	0.864	0.03456
6		0.05	20	0.0025	0.000125	6.25E-06	1	0.05
7		0.06	21.6	0.0036	0.000216	1.3E-05	1.296	0.07776
8		0.07	26.4	0.0049	0.000343	2.4E-05	1.848	0.12936
9		0.08	34.4	0.0064	0.000512	4.1E-05	2.752	0.22016
10		0.09	45.6	0.0081	0.000729	6.56E-05	4.104	0.36936
11		0.1	60	0.01	0.001	0.0001	6	0.6
11	$\sum$	0.55	396	0.0385	0.003025	0.000253	19.8	1.52328

Gauss elimination

11	0.55	0.0385	a0	396
0.55	0.0385	0.003025	a1	19.8
0.0385	0.003025	0.000253	a2	1.52328

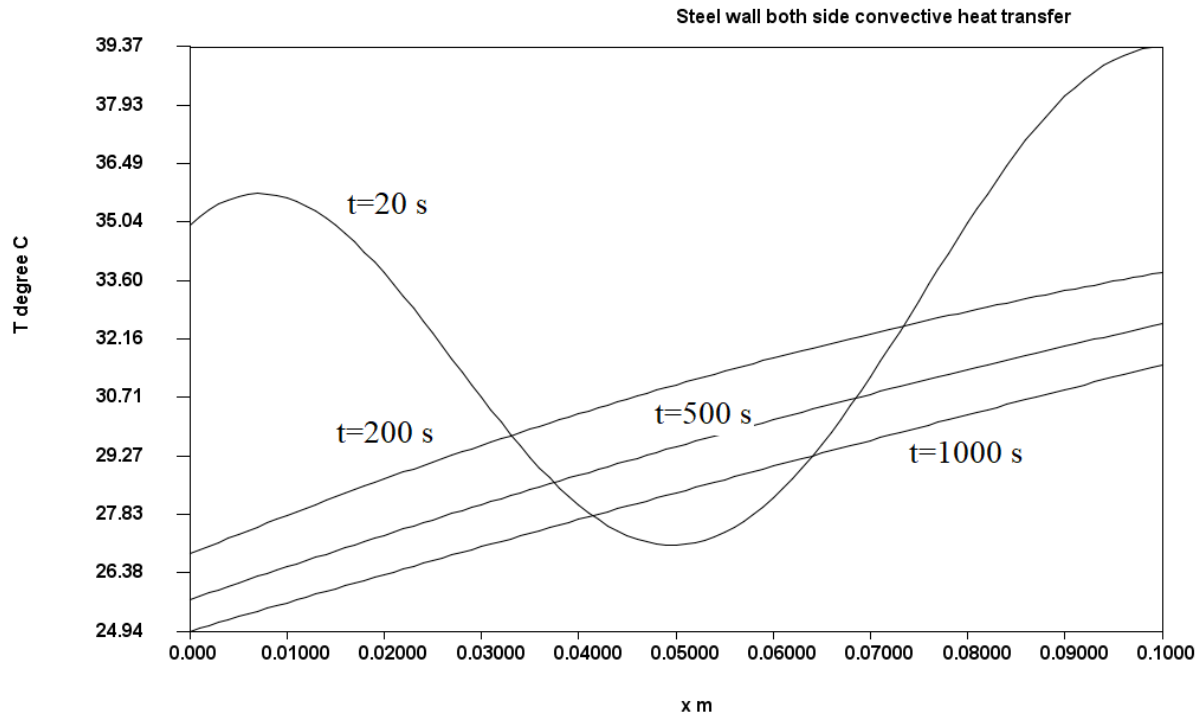
C					
0.05	11	0.55	0.0385	a0	396
0.0035	0	0.011	0.0011	a1	0
	0	0.0011	0.000119	a2	0.13728

C					
	11	0.55	0.0385	a0	396
	0	0.011	0.0011	a1	0
0.1	0	0	8.58E-06	a2	0.13728

a0	**60**
a1	**-1600**
a2	**16000**

```
 public class HT_plane_wall3_test2
{
public static void main (String args[])
{ double h1=600; //W/m^2K
  double h2=300; //W/m^2K
  double k=42;
  double alpha=1.2e-5;
  double L=0.1;
  double t[]={20,200,500,1000};  //s
  double T1inf=20;
  double T2inf=40;
  double Ti=240;
  int n1=t.length;
  double x[]=new double[101];
  double T[][]=new double[n1][101];
  double q[][]=new double[n1][101];
  HT_plane_wall3 htpw=new HT_plane_wall3(h1,h2,L,k,alpha,Ti,T1inf,T2inf);
  //initial temperature variation t=0
  htpw.F=xx->{double T1i=16000*xx*xx-1600*xx+60;
          double Fx=(T1i-T2inf)-htpw.Tss(xx);
          return Fx;
          };
  for(int j=0;j<n1;j++)
  {
  for(int i=0;i<=100;i++)
     {  x[i]=i*L/100;
                T[j][i]=htpw.T(x[i],t[j]);
                q[j][i]=htpw.q(x[i],t[j]);
     }
```

```
    }
      Plot pp=new Plot(x,T[0]);
      for(int j=1;j<n1;j++)
      {pp.addData(x,T[j]);}
      pp.setPlabel("Steel wall both side convective heat transfer");
      pp.setXlabel(" x m");
      pp.setYlabel(" T degree C");
      pp.plot();
      Plot pp1=new Plot(x,q[0]);
      for(int j=1;j<n1;j++)
      {pp1.addData(x,q[j]);}
      pp1.setPlabel("Steel wall both side convective heat transfer");
      pp1.setXlabel(" x m");
      pp1.setYlabel(" q W/m^2");
      pp1.plot();
}
}
```



Python version:

```
from math import *
from f_x import *;
import matplotlib.pyplot as plt

def bisection(f,a,b):
    b1=1.1*b
    r=(a+b)/2.0
    fr=f.func(r)
    fa=f.func(a)
    eps=1.0e-8;
    nmax=100;
    i=1;
    while abs(fr)>eps and i<nmax:
        if fa*fr<0:
            b=r
        else:
            a=r
            fa=fr
        r=(a+b)/2.0;
        fr=f.func(r)
        i=i+1
    if i>=nmax:
       r=bisection(f,a,b1)
    return r

def gauss_legendre_coefficients(x1,x2,n):
    #calculates legendre gauss-coefficients as coefficients of the integral
```

```
    #for n terms
        eps=3e-15
        m=int((n+1.0)/2.0)
        xm=0.5*(x2+x1)
        xl=0.5*(x2-x1)
        nmax=100
        a=[[0.0 for i in range(n)] for j in range(2)]
        for i in range(1,m+1):
            z=cos(pi*((i-0.25)/(n+0.5)))
            for ii in range(nmax):
                #while abs(z-z1) > eps:
                p1=1.0;
                p2=0.0;
                for j in range(1,n+1):
                    p3=p2
                    p2=p1
                    p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
                pp=(float)(n*(z*p1-p2)/(z*z-1.0))
                z1=z;
                z=z1-p1/pp
                if abs(z-z1) < eps: break
            a[0][i-1]=xm-xl*z
            a[0][n-i]=xm+xl*z
            a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
            a[1][n-i]=a[1][i-1]
        return a


def integral(f,x1,x2,n):
    #integral func(x)dx
    #integral of a function by using gauss-legendre quadrature
    #between x1 and x2
    a=gauss_legendre_coefficients(x1,x2,n)
    z=0
    for i in range(n):
        z=z+a[1][i]*f.func(a[0][i])
    return z

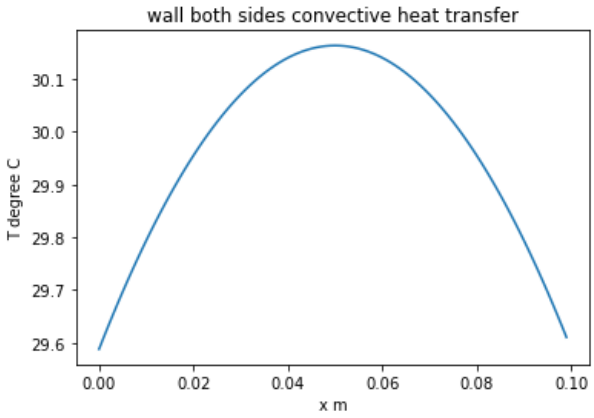

def ksi(n,Bi1,Bi2):
    # eigenvalue function to be solved
    x1=n*pi+0.00000001
    x2=x1+(pi/2.0-0.00000001)
    class f1(f_x):func=lambda self,x: (x*x-Bi1*Bi2)*tan(x)-x*(Bi1+Bi2)
    f = f1()
    y= bisection(f,x1,x2)
    return y

def tetha(F,h1,h2,k,alpha,L,x,t):
    beta=0.0
    C=0.0
    N=0.0
    teta=0.0
    H1=h1/k
    H2=h2/k
    Bi1=H1*L
    Bi2=H2*L
    for n in range(20):
        k1=ksi(n,Bi1,Bi2)
        beta=k1/L
        N=L/2.0*(beta*beta+H1*H1)+sin(2.0*beta*L)/(4*beta)*(beta*beta-H1*H1)+H1/2.0*(1.0-cos(2.0*beta*L))
        class Cn(f_x):func=lambda self,x:F.func(x)*(beta*cos(beta*x)+H1*sin(beta*x))
        fCn=Cn()
        C=1.0/N*integral(fCn,0.0,L,20)
        teta=teta+C*(beta*cos(beta*x)+H1*sin(beta*x))*exp(-beta*beta*alpha*t)
    return teta

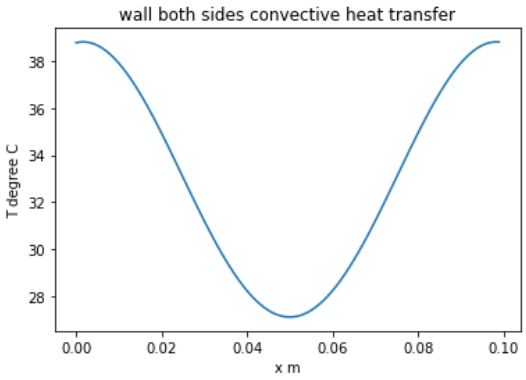
h1=100.0
h2=100.0
k=42
alpha=1.2e-5
L=0.1
t=500
Ti=60
Tinf=20
class F(f_x):func=lambda self,x: 16000.0*x*x-1600.0*x+60.0-Tinf
F1 = F()
```

```
n=100
T=[0.0 for i in range(n)]
x=[0.0 for i in range(n)]
for i in range(n):
    x[i]=i*L/n
    T[i]=tetha(F1,h1,h2,k,alpha,L,x[i],t)+Tinf
plt.plot(x,T)
plt.xlabel("x m")
plt.ylabel("T degree C")
plt.title("wall both sides convective heat transfer")
plt.show()
```

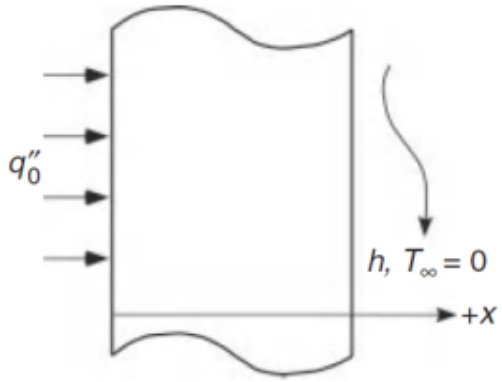
T=500 s



t=20 s



PROBLEM 3.10 SEPARATION OF VARIABLES, NON-HOMOGENEOUS, TIME DEPENDENT SOLUTION; CARTESIAN COORDINATES



$$\frac{\partial^2 T^*(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T^*(x,t)}{\partial t}$$

Boundary conditions

$-k\frac{\partial T^*(x,t)}{\partial x} = q''_0$ x=0 t>0

$$-k\frac{\partial T^*(x,t)}{\partial x} = h(T^*(x,t) - T_\infty) \quad x = L \quad t > 0$$
$$T^*(x,t) = F^*(x) \quad t=0 \quad 0 \le x \le L$$

$$T(x,t) = T^*(x,t) - T_\infty$$

$$\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T(x,t)}{\partial t}$$

Boundary conditions

$$-k\frac{\partial T(x,t)}{\partial x} = q''_0 \qquad x=0 \quad t>0$$
$$\frac{\partial T(x,t)}{\partial x} + HT(x,t) = 0 \quad x = L \quad t > 0$$
$$T(x,t) = F^*(x) - T_\infty = F(x) \quad t=0 \quad 0 \le x \le L$$

$$T(x,t) = T_H(x,t) + T_{SS}(x)$$

$$\frac{\partial^2 T_H(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T_H(x,t)}{\partial t}$$

Boundary conditions

$$\frac{\partial T_H(x,t)}{\partial x} = 0 \quad x=0 \quad t>0$$
$$\frac{\partial T_H(x,t)}{\partial x} + HT_H(x,t) = 0 \quad x = L \quad t > 0$$
$$T(x,t) = F^*(x) - T_\infty - T_{SS}(x) = F(x) - T_{SS}(x) \quad t=0 \quad 0 \le x \le L$$

$$\frac{\partial^2 T_{ss}(x)}{\partial x^2} = 0$$

Boundary conditions

$$-\frac{\partial T(x)}{\partial x} = \frac{q''_0}{k} \quad x=0$$
$$\frac{\partial T_{ss}(x)}{\partial x} + H * T_{ss}(x) = 0 \quad x=L \quad t>0$$

Solution:

$$\frac{\partial T_{ss}(x)}{\partial x} = C_1 \qquad T_{ss}(x) = C_1 x + C_2$$

$$T_{ss}(x) = \frac{q''_0}{k}(L-x) + \frac{q''_0}{h}$$

The first part of the problem is solved as problem 1.

$$H = \frac{h}{k}$$

$\beta_n \tan(\beta_n L) = H$ equation has more than one solution.

$$N(\beta_m) = \int_0^L cos^2(\beta_m x)dx = \frac{L(\beta_m^2 + H^2) + H}{2(\beta_m^2 + H^2)}$$

$$C_m = \frac{1}{N(\beta_m)}\int_0^L F(x)\cos(\beta_m x)dx$$

Depends on value of F(x), this integration can be solved by using numerical or analytical methods.

$$T_H(x,t) = \sum_{m=1}^{\infty} C_m \cos(\beta_m x)\exp(-\beta_m^2 \alpha t) = \sum_{m=1}^{\infty} \frac{1}{N(\beta_m)}\left[\int_0^L F(x)\cos(\beta_m x)dx\right]\cos(\beta_m x)\exp(-\beta_m^2 \alpha t)$$

Program code:

```
import java.io.*;
import javax.swing.*;
import java.awt.*;
import static java.lang.Math.*;
public class HT_plane_wall4
{
public if_x F;
public double h,L,k,alpha,Bi,Ti,Tinf,H,q0,C1,C2;
public  HT_plane_wall4(double hi,double q0i,double Li,double ki,double alphai,double Tii,double Tinfi)
{h=hi;q0=q0i;k=ki;L=Li;alpha=alphai;Bi=h*L/k;Ti=Tii;Tinf=Tinfi;H=h/k;
 C1=q0/k;C2=q0/h;
 F=xx->Ti-Tinf-Tss(xx);
}
public double Tss(double x)  {return C1*(L-x)+C2;}

  public static double ksi(int n,double Bi)
  {double x1=n*PI+0.0000001;
   double x2=x1+(PI/2-0.000001);
   if_x f1=x->x*sin(x)-Bi*cos(x);
   double ksi=f1.illinois(x1,x2);
   return ksi;
  }
  public double teta(double x,double t)
  {double ksi=0;
   double beta=0;
   double C=0;
   double N=0,N1=0,C1=0,C2=0;
   double teta=0;
   //initial temperature variation t=0
   if_x F=xx->Ti-Tinf;
   //coefficient of equation
   for(int n=0;n<20;n++)
   {ksi=ksi(n,Bi);
    beta=ksi/L;
    final double beta1=beta;
    N=L/2.0+sin(2*ksi)/(4.0*beta);
    N1=(L*(beta*beta+H*H)+H)/(2*(beta*beta+H*H));
    if_x Cn=xx->F.func(xx)*cos(beta1*xx);
    C=1.0/N1*Cn.integral(0,L,20);
    teta+=C*Math.cos(beta*x)*Math.exp(-beta*beta*alpha*t);
    }
    return teta;
  }
  public double dteta(double x,double t)
  {double ksi=0;
   double beta=0;
   double C=0;
   double N=0,N1=0,C1=0,C2=0;
   double dteta=0;
   //initial temperature variation t=0
   if_x F=xx->Ti-Tinf;
   //coefficient of equation
   for(int n=0;n<20;n++)
   {ksi=ksi(n,Bi);
    beta=ksi/L;
    final double beta1=beta;
    N=L/2.0+sin(2*ksi)/(4.0*beta);
    if_x Cn=xx->F.func(xx)*cos(beta1*xx);
    double x1=
    C=1.0/N*Cn.integral(0,L,20);
    dteta+=-C*beta*sin(beta*x)*exp(-beta*beta*alpha*t);
    }
    return dteta;
  }
  public double T(double x,double t)
  {double teta=teta(x,t);
   double T=teta+Tinf+Tss(x);
   return T;
```

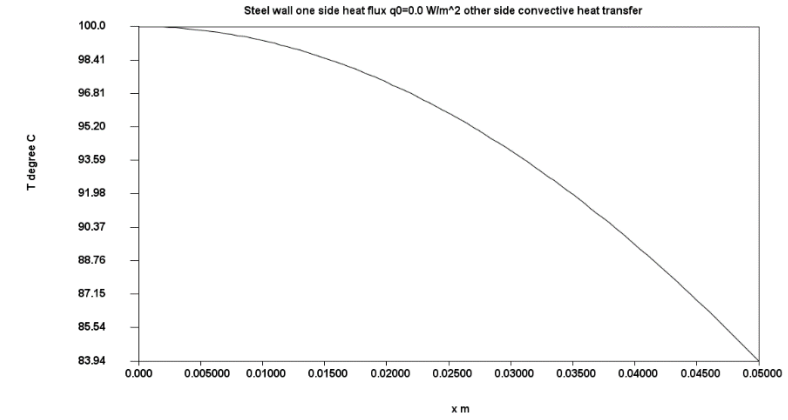
```
  }
  public double q(double x,double t)
  {double q=-k*(dteta(x,t)-C1);
   return q;
  }

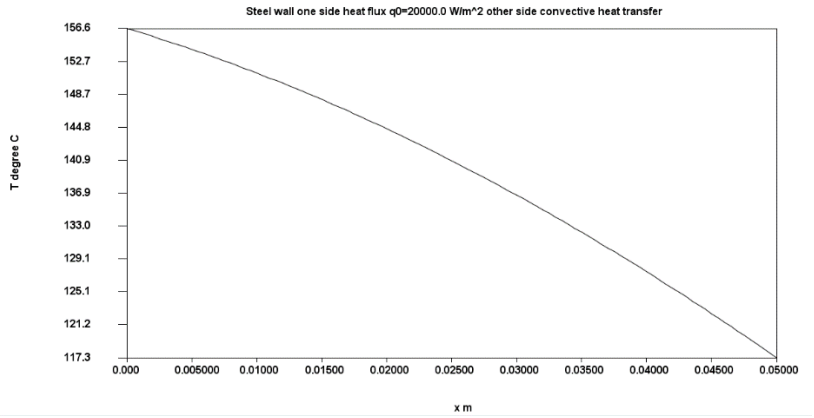
}
```

Numerical example:

A steel plate [α=1.2e-5 and k=43 W/(mK)] of thickness L=0.05 m, initially at a uniform temperature of T_i= 240 ℃, is suddenly immersed in oil baths at one side $T_{1\infty} = 20$ ℃ and the convective heat transfer coefficients h_1=600 W/(m^2K), the other side of wall exposed to heat flux of q0=0,10 and 20 kW/m^2 calculate temperature profile for t=478.8 seconds.

```
 public class HT_plane_wall4_test1
{
public static void main (String args[])
{ double h=600; //W/m^2K
  double k=43;
  double alpha=1.2e-5;
  double L=0.05;
  double t[]={478.8};  //s
  double q0[]={0,10e3,20e3};  //W/m^2
  double Tinf=40.0;
  double Ti=240.0;
  int n1=t.length;
  double x[]=new double[101];
  double T[][]=new double[n1][101];
  double q[][]=new double[n1][101];
  HT_plane_wall4 htpw;
  for(int kk=0;kk<q0.length;kk++)
  {
  htpw=new HT_plane_wall4(h,q0[kk],L,k,alpha,Ti,Tinf);
  //initial temperature variation t=0
  htpw.F=xx->(Ti-Tinf);
  for(int j=0;j<n1;j++)
  {
  for(int i=0;i<=100;i++)
      {   x[i]=i*L/100;
                  T[j][i]=htpw.T(x[i],t[j]);
                  q[j][i]=htpw.q(x[i],t[j]);
      }
   }
      Plot pp=new Plot(x,T[0]);
      for(int j=1;j<n1;j++)
      {pp.addData(x,T[j]);}
      pp.setPlabel("Steel wall one side heat flux q0="+q0[kk]+" W/m+2 other side convective heat transfer");
      pp.setXlabel(" x m");
      pp.setYlabel(" T degree C");
      pp.plot();
      Plot pp1=new Plot(x,q[0]);
      for(int j=1;j<n1;j++)
      {pp1.addData(x,q[j]);}
      pp1.setPlabel("Steel wall one side heat flux q0="+q0[kk]+"W/m+2 other side convective heat transfer");
      pp1.setXlabel(" x m");
      pp1.setYlabel(" q W/m^2");
      pp1.plot();
   }
   }
}
```





Numerical example:

A steel plate [α=1.2e-5 and k=43 W/(mK)] of thickness L=0.1 m, initially has a temperature profile as;

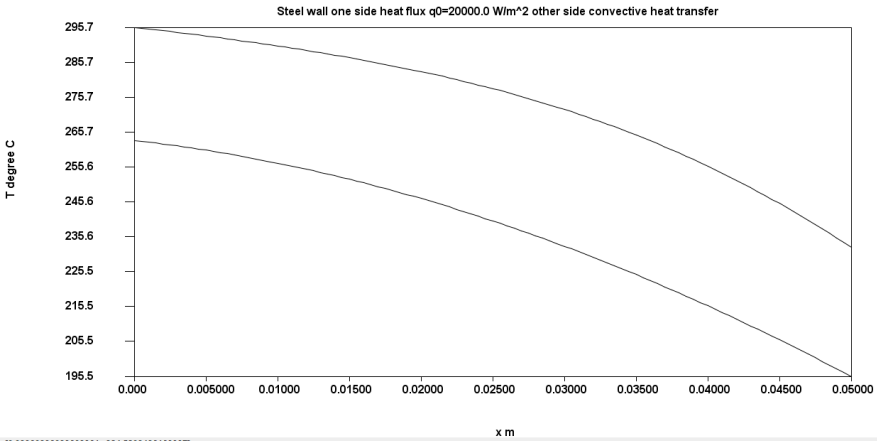
0	60
0.01	45.6
0.02	34.4
0.03	26.4
0.04	21.6
0.05	20
0.06	21.6
0.07	26.4
0.08	34.4
0.09	45.6
0.1	60

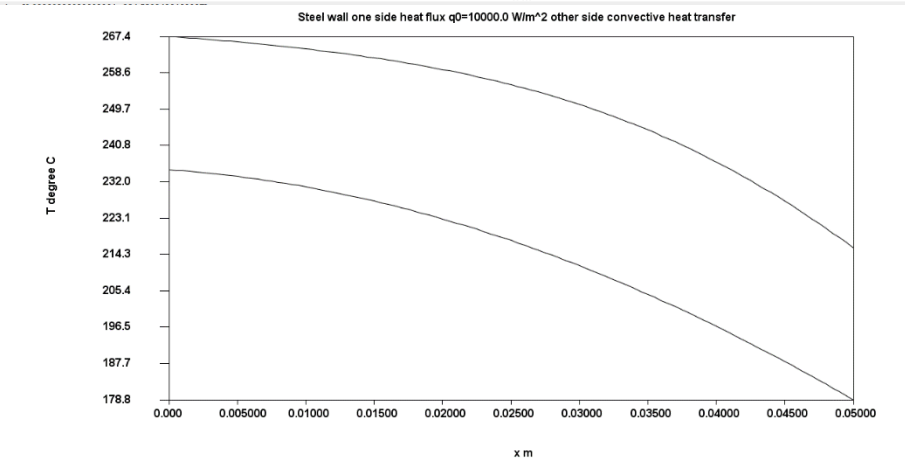
is suddenly immersed in an oil bath at $T_{2\infty} = 20$ ℃ and the convective heat transfer coefficients are h_2=100 W/(m^{2}K), the other side of wall exposed to heat flux of q0=0,10 and 20 kW/m^2 calculate temperature profile for t=20,100 seconds.

F(x)=?

```
 public class HT_plane_wall4_test2
{
public static void main (String args[])
{ double h=600; //W/m^2K
  double k=43;
  double alpha=1.2e-5;
  double L=0.05;
  double t[]={20,100};  //s
  double q0[]={0,10e3,20e3};  //W/m^2
  double Tinf=40.0;
  double Ti=240.0;
  int n1=t.length;
  double x[]=new double[101];
  double T[][]=new double[n1][101];
  double q[][]=new double[n1][101];
  //HT_plane_wall4 htpw;
  for(int kk=0;kk<q0.length;kk++)
```

```
  {
  HT_plane_wall4 htpw=new HT_plane_wall4(h,q0[kk],L,k,alpha,Ti,Tinf);
  //initial temperature variation t=0
  htpw.F=xx->{double T1i=16000*xx*xx-1600*xx+60;
          double Fx=(T1i-Tinf)-htpw.Tss(xx);
          return Fx;
          };
  for(int j=0;j<n1;j++)
  {
  for(int i=0;i<=100;i++)
     {   x[i]=i*L/100;
                 T[j][i]=htpw.T(x[i],t[j]);
                 q[j][i]=htpw.q(x[i],t[j]);
     }
   }
     Plot pp=new Plot(x,T[0]);
     for(int j=1;j<n1;j++)
     {pp.addData(x,T[j]);}
     pp.setPlabel("Steel wall one side heat flux q0="+q0[kk]+" W/m^2 other side convective heat transfer");
     pp.setXlabel(" x m");
     pp.setYlabel(" T degree C");
     pp.plot();
     Plot pp1=new Plot(x,q[0]);
     for(int j=1;j<n1;j++)
     {pp1.addData(x,q[j]);}
     pp1.setPlabel("Steel wall one side heat flux q0="+q0[kk]+"W/m^2 other side convective heat transfer");
     pp1.setXlabel(" x m");
     pp1.setYlabel(" q W/m^2");
     pp1.plot();
   }
   }
}
```





Separation of variables of spherical coordinate system

Solid sphere

$$\frac{\partial^2 T(r,t)}{\partial r^2}+\frac{2}{r}\frac{\partial T(r,t)}{\partial r}=\frac{1}{\alpha}\frac{\partial T(r,t)}{\partial t}$$

$$U(r,t) = rT(r,t)$$

$$\frac{\partial U(r,t)}{\partial r} = T + r\frac{\partial T(r,t)}{\partial r}$$

$$\frac{\partial^2 U(r,t)}{\partial r^2} = \frac{\partial T(r,t)}{\partial r} + \frac{\partial T(r,t)}{\partial r} + r\frac{\partial^2 T(r,t)}{\partial r^2} = 2\frac{\partial T(r,t)}{\partial r} + r\frac{\partial^2 T(r,t)}{\partial r^2}$$

$$\frac{1}{r}\frac{\partial^2 U(r,t)}{\partial r^2} = \frac{2}{r}\frac{\partial T(r,t)}{\partial r} + \frac{\partial^2 T(r,t)}{\partial r^2}$$

$$\frac{\partial U(r,t)}{\partial t} = r\frac{\partial T(r,t)}{\partial t}$$

$$\frac{\partial T(r,t)}{\partial t} = \frac{1}{r}\frac{\partial U(r,t)}{\partial t}$$

$$\frac{\partial^2 U(r,t)}{\partial r^2} = \frac{1}{\alpha}\frac{\partial U(r,t)}{\partial t}$$

Boundary conditions: $U(r,t) = 0 \quad r = 0 \quad t > 0$

$$\frac{\partial U(r,t)}{\partial r} + \left(H - \frac{1}{b}\right)U(r,t) = 0 \quad r = b \quad t > 0$$

$$K = \left(H - \frac{1}{b}\right)$$

$$\frac{\partial U(r,t)}{\partial r} + KU(r,t) = 0 \quad r = b \quad t > 0$$

$$U(r,t) = rF(r) \quad t = 0 \quad 0 \le r \le b$$

Separation of variables: $U(r,t) = R(r)\Gamma(t)$

$$\frac{1}{R(r)}\frac{\partial^2 R(r)}{\partial r^2} = \frac{1}{\alpha\Gamma(t)}\frac{\partial \Gamma(t)}{\partial t} = -\lambda^2$$

$$\frac{\partial^2 R(r)}{\partial r^2} + \lambda^2 R(r) = 0$$

$$\frac{\partial \Gamma(t)}{\partial t} + \lambda^2 \alpha\Gamma(t) = 0$$

Boundary conditions:

$R(r) = 0 \quad r = 0 \quad t > 0$

$\frac{\partial R(r)}{\partial r} + KR(r) = 0 \quad r = b \quad t > 0$

$U(r,t) = rF(r) \quad t = 0 \quad 0 \le r \le b$

$R(r) = C_1 \sin(\lambda r) + C_2 \cos(\lambda r)$

$\Gamma(t) = C_3 \exp(-\lambda^2 \alpha t)$

B:C. $R(r) = 0 \quad r = 0 \quad t > 0 \rightarrow C_1 \sin(0) + C_2 \cos(0) = 0 \qquad C_2 = 0$

$R(r) = C_1 \sin(\lambda r)$

$\frac{\partial R(r)}{\partial r} = C_1 \lambda \cos(\lambda r)$

$B.C.: \quad \frac{\partial R(r)}{\partial r} + KR(r) = 0 \quad at\ r = b$

$C_1 \lambda \cos(\lambda b) + KC_1 \sin(\lambda b) = 0 \qquad C_1 \neq 0$

$\lambda \cos(\lambda b) + K\sin(\lambda b) = 0$

Multiple roots are existed to solve this equation

$\lambda_n \cos(\lambda_n b) + K\sin(\lambda_n b) = 0$ or multiplying with b gives

$\lambda_n b\cos(\lambda_n b) + Kb\sin(\lambda_n b) = 0$

$$\lambda_n b = \xi_n$$

$\xi_n \cos(\xi_n) + Kb\sin(\xi_n) = 0$

$\frac{\xi_n}{\tan(\xi_n)} + Kb = 0$ solution of this equation will give eigenvalues $\lambda_n b = \xi_n$

$$U(r,t) = \sum_{n=0}^{\infty} C_n \sin(\lambda_n r) \exp(-\lambda_n^2 \alpha t)$$

Last boundary condition:

$U(r,t) = rF(r) \quad t = 0 \quad 0 \le r \le b$

$$rF(r) = \sum_{n=0}^{\infty} C_n \sin(\lambda_n r)$$

This equation has n coefficients. In order to solve these coefficients, we will use properties of orthogonal functions.

$$\int_0^L X(\beta_n x) X(\beta_m x) dx = \begin{cases} 0 & m \neq n \\ N(\beta_m) & m = n \end{cases}$$

$$\int_0^b rF(r) \sin(\lambda_n r)\, dr = C_n \int_0^b \sin(\lambda_n r)^2\, dr$$

$$\int_0^b \sin(\lambda_n r)^2\, dr = \int_0^b \frac{1-\cos(2\lambda_n r)}{2} dr = \left[\frac{r}{2} - \frac{1}{4\lambda_n}\sin(2\lambda_n r)\right]_0^b = \frac{b}{2} - \frac{1}{4\lambda_n}\sin(2\lambda_n b)$$

$$= \frac{b}{2}\left[1 - \frac{1}{2\lambda_n b}\sin(2\lambda_n b)\right] = \frac{b}{2}\left[1 - \frac{1}{2\xi_n}\sin(2\xi_n)\right] = \frac{b}{2}\left[\frac{2\xi_n - \sin(2\xi_n)}{2\xi_n}\right]$$

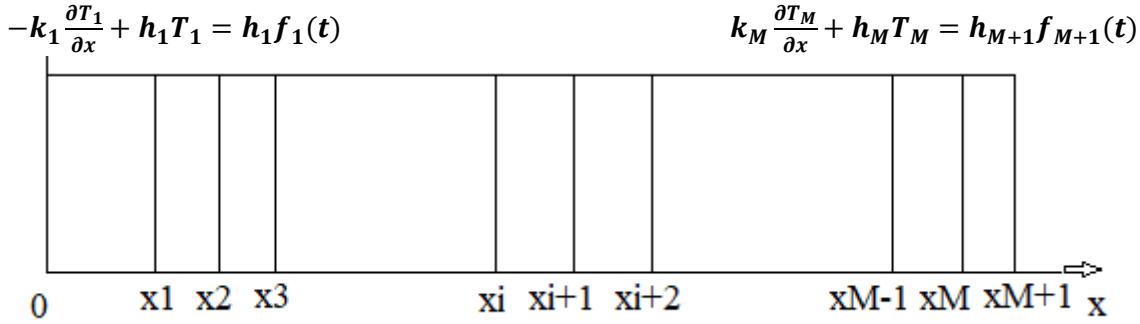
$$C_n = \frac{2\lambda_n}{(2\lambda_n b - \sin(2\lambda_n b))} \int_0^b rF(r) \sin(\lambda_n r)\, dr$$

$$U(r,t) = \sum_{n=0}^{\infty} \frac{2\lambda_n}{(2\lambda_n b - \sin(2\lambda_n b))} \int_0^b rF(r) \sin(\lambda_n r)\, dr \sin(\lambda_n r) \exp(-\lambda_n^2 \alpha t)$$

$$T(r,t) = \sum_{n=0}^{\infty} \left[\frac{2\lambda_n}{r(2\lambda_n b - \sin(2\lambda_n b))} \int_0^b rF(r) \sin(\lambda_n r)\, dr\right] \sin(\lambda_n r) \exp(-\lambda_n^2 \alpha t)$$

3.2 SEPARATION OF VARIABLES FOR COMPOSITE MEDIUM

Transient temperature distribution in a composite medium with several layers will be investigated in this section. Such problem has important engineering applications. When you have composite layers, there usually is an additional thermal resistance between the layers due to gaps left between the surfaces.



The mathematical formulation of the problem is given as follows (for cartesian coordinate system):

$\alpha_i \frac{\partial^2 T_i(x,t)}{\partial x^2} + \frac{\alpha_i}{k_i} g_i(x,t) = \frac{\partial T_i(x,t)}{\partial t} \quad x_i \le x \le x_{i+1}$ t>0 i=1,2,…,M

Boundary conditions:

$-k_1 \frac{\partial T_1(x,t)}{\partial x} + h_1 T_1(x,t) = h_1 f_1(t)$ at the outer boundary x=x₁ t>0

$-k_i \frac{\partial T_i(x,t)}{\partial x} = h_{i+1}(T_i(x,t) - T_{i+1}(x,t))$

$k_i \frac{\partial T_i(x,t)}{\partial x} = k_{i+1} \frac{\partial T_{i+1}(x,t)}{\partial x}$ at the interface $x=x_{i+1}$ i=1,2,…,M

$k_M \frac{\partial T_M(x,t)}{\partial x} + h_M T_M(x,t) = h_{M+1} f_{M+1}(t)$ at the outer boundary x=x₁ t>0

$T_i(x,t) = F_i(x)$ t=0 $x_i \le x \le x_{i+1}$

It should be note that for $h_{i+1} \to \infty$ boundary value equation

$k_i \frac{\partial T_i(x,t)}{\partial x} = h_{i+1}(T_i(x,t) - T_{i+1}(x,t))$ will be converted to $T_i(x,t) = T_{i+1}(x,t)$

For cylindrical coordinates:

$\alpha_i \frac{1}{r} \frac{\partial}{\partial r}\left(\frac{\partial T_i(x,t)}{\partial r}\right) + \frac{\alpha_i}{k_i} g_i(r,t) = \frac{\partial T_i(r,t)}{\partial t}$ $r_i \le r \le r_{i+1}$ t>0 i=1,2,…,M

Boundary conditions:

$-k_1 \frac{\partial T_1(r,t)}{\partial r} + h_1 T_1(r,t) = h_1 f_1(t)$ at the outer boundary r=r₁ t>0

$-k_i \frac{\partial T_i(r,t)}{\partial r} = h_{i+1}(T_i(r,t) - T_{i+1}(r,t))$

$k_i \frac{\partial T_i(r,t)}{\partial r} = k_{i+1} \frac{\partial T_{i+1}(r,t)}{\partial r}$ at the interface $r=r_{i+1}$ i=1,2,…,M

$k_M \frac{\partial T_M(r,t)}{\partial x} + h_M T_M(r,t) = h_{M+1} f_{M+1}(t)$ at the outer boundary r=r₁ t>0

$T_i(r,t) = F_i(r)$ t=0 $r_i \le r \le r_{i+1}$

PROBLEM 3.11 SEPARATION OF VARIABLES, COMPOSITE MEDIUM, TIME DEPENDENT SOLUTION; CARTESIAN COORDINATES

Let us consider two layer composite medium. The first layer in the region $0 \le x \le a$ and the second layer in the region $a \le x \le b$. We will be assuming that a perfect thermal contact between the layers. Diferential equation and boundary layer equations are as follows:

$\alpha_1 \frac{\partial^2 T_1^*(x,t)}{\partial x^2} = \frac{\partial T_1^*(x,t)}{\partial t}$ $0 \le x \le a$ t>0

$\alpha_2 \frac{\partial^2 T_2^*(x,t)}{\partial x^2} = \frac{\partial T_2^*(x,t)}{\partial t}$ $a \le x \le b$ t>0

$T_1^*(x,t) = T_{1\infty}$ x=0 t>0

$T_1^*(x,t) = T_2^*(x,t)$ x=a t>0

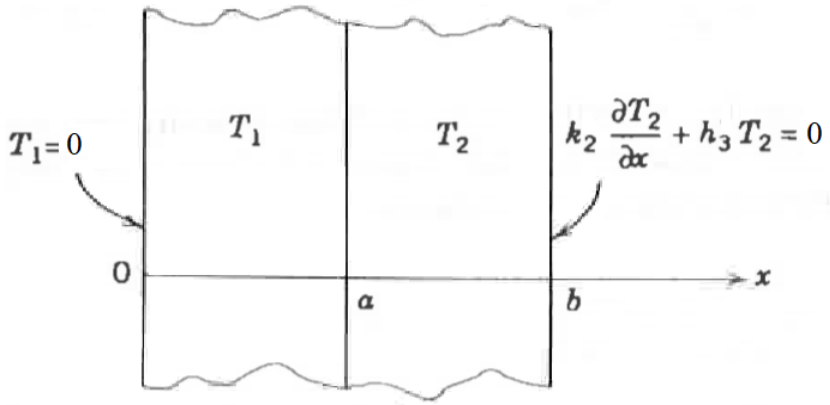
$k_1 \frac{\partial T_1^*(x,t)}{\partial x} = k_2 \frac{\partial T_2^*(x,t)}{\partial x}$ x=a t>0

$k_2 \frac{\partial T_2^*(x,t)}{\partial x} + h_3(T_2^*(x,t) - T_{1\infty}) = 0$

Initial conditions:

$T_1^*(x,t) = F_1(x) + T_{1\infty}$ t=0 $0 \le x \le a$

$T_2^*(x,t) = F_2(x) + T_{2\infty}$ t=0 $a \le x \le b$



By applying change $T_1(x,t) = T_1^*(x,t) - T_{1\infty}$ and $T_2(x,t) = T_2^*(x,t) - T_{1\infty}$ equation becomes

$\alpha_1 \frac{\partial^2 T_1(x,t)}{\partial x^2} = \frac{\partial T_1(x,t)}{\partial t}$ $0 \le x \le a$ t>0

$\alpha_2 \frac{\partial^2 T_2(x,t)}{\partial x^2} = \frac{\partial T_2(x,t)}{\partial t}$ $a \le x \le b$ t>0

$T_1(x,t) = 0$ x=0 t>0

$T_1(x,t) = T_2(x,t)$ x=a t>0

$k_1 \frac{\partial T_1(x,t)}{\partial x} = k_2 \frac{\partial T_2(x,t)}{\partial x}$ x=a t>0

$k_2 \frac{\partial T_2(x,t)}{\partial x} + h_3 T_2(x,t) = 0$ x=b t>0

Initial conditions:

$T_1(x,t) = F_1(x)$ t=0 $0 \le x \le a$

$T_2(x,t) = F_2(x)$ t=0 $a \le x \le b$

Separation of variable method wil be apply to solve this differential equation.

$T_1(x,t) = \varphi_1(x)\Gamma(t)$

$T_2(x,t) = \varphi_2(x)\Gamma(t)$

Equations becomes

$\alpha_1 \Gamma(t) \frac{\partial^2 \varphi_1(x)}{\partial x^2} = \varphi_1(x) \frac{\partial \Gamma(t)}{\partial t} = \beta^2$

$\alpha_2 \Gamma(t) \frac{\partial^2 \varphi_2(x)}{\partial x^2} = \varphi_2(x) \frac{\partial \Gamma(t)}{\partial t} = \beta^2$

$\frac{\partial^2 \varphi_1(x)}{\partial x^2} + \frac{\beta^2}{\alpha_1} \varphi_1(x) = 0$

$\frac{\partial^2 \varphi_2(x)}{\partial x^2} + \frac{\beta^2}{\alpha_1} \varphi_2(x) = 0$

$\frac{\partial \Gamma(t)}{\partial t} + \beta^2 \Gamma(t) = 0$

$\Gamma(t) = C_5 \exp(-\beta^2 t)$

$\varphi_1(x) = C_1 sin\left(\frac{\beta x}{\sqrt{\alpha_1}}\right) + C_2 cos\left(\frac{\beta x}{\sqrt{\alpha_1}}\right)$

$\varphi_2(x) = C_3 sin\left(\frac{\beta x}{\sqrt{\alpha_2}}\right) + C_4 cos\left(\frac{\beta x}{\sqrt{\alpha_2}}\right)$

Boundary value: $\varphi_1(x) = 0$ at x=0 $\rightarrow C_2 = 0$

Boundary value: $\varphi_1(x) = \varphi_2(x)$ at x=a

$C_1 sin\left(\frac{\beta a}{\sqrt{\alpha_1}}\right) = C_3 sin\left(\frac{\beta a}{\sqrt{\alpha_2}}\right) + C_4 cos\left(\frac{\beta a}{\sqrt{\alpha_2}}\right)$

$A_2 = C_3/C_1 \quad B_2 = C_4/C_1$

$sin\left(\frac{\beta a}{\sqrt{\alpha_1}}\right) = A_2 sin\left(\frac{\beta a}{\sqrt{\alpha_2}}\right) + B_2 cos\left(\frac{\beta a}{\sqrt{\alpha_2}}\right)$ equation has more than one solution for β , A_2, B_2 so that it becomes

$sin\left(\frac{\beta_n a}{\sqrt{\alpha_1}}\right) = A_{2n} sin\left(\frac{\beta_n a}{\sqrt{\alpha_2}}\right) + B_{2n} cos\left(\frac{\beta_n a}{\sqrt{\alpha_2}}\right)$

$\varphi_{1n}(x) = sin\left(\frac{\beta_n x}{\sqrt{\alpha_1}}\right)$

$\varphi_{2n}(x) = A_{2n} sin\left(\frac{\beta_n x}{\sqrt{\alpha_2}}\right) + B_{2n} cos\left(\frac{\beta_n x}{\sqrt{\alpha_2}}\right)$

Boundary condition: $k_1 \frac{\partial \varphi_{1n}(x)}{\partial x} = k_2 \frac{\varphi_{2n}(x)}{\partial x}$ x=a t>0

$k_1 \frac{\partial \varphi_{1n}(x)}{\partial x} = k_2 \frac{\varphi_{2n}(x)}{\partial x}$

$\frac{\partial \varphi_{1n}(x)}{\partial x} = \frac{\beta_n}{\sqrt{\alpha_1}} cos\left(\frac{\beta_n x}{\sqrt{\alpha_1}}\right)$

$\frac{\partial \varphi_{2n}(x)}{\partial x} = \frac{\beta_n}{\sqrt{\alpha_2}} \left[A_{2n} cos\left(\frac{\beta_n x}{\sqrt{\alpha_2}}\right) - B_{2n} sin\left(\frac{\beta_n x}{\sqrt{\alpha_2}}\right) \right]$

$k_1 \left[\frac{\beta_n}{\sqrt{\alpha_1}} cos\left(\frac{\beta_n a}{\sqrt{\alpha_1}}\right)\right] = k_2 \left[\frac{\beta_n}{\sqrt{\alpha_2}} \left[A_{2n} cos\left(\frac{\beta_n a}{\sqrt{\alpha_2}}\right) - B_{2n} sin\left(\frac{\beta_n a}{\sqrt{\alpha_2}}\right) \right]\right]$

$\frac{k_1}{k_2} \frac{\sqrt{\alpha_2}}{\sqrt{\alpha_1}} cos\left(\frac{\beta_n a}{\sqrt{\alpha_1}}\right) = \left[A_{2n} cos\left(\frac{\beta_n a}{\sqrt{\alpha_2}}\right) - B_{2n} sin\left(\frac{\beta_n a}{\sqrt{\alpha_2}}\right) \right]$

Boundary condition: $k_2 \frac{\partial \varphi_{2n}(x)}{\partial x} = h_3 \varphi_{2n}(x)$ at x=b

$$k_2\left[\frac{\beta_n}{\sqrt{\alpha_2}}\left[A_{2n}cos\left(\frac{\beta_n b}{\sqrt{\alpha_2}}\right)-B_{2n}sin\left(\frac{\beta_n b}{\sqrt{\alpha_2}}\right)\right]\right]=h_3\left[A_{2n}sin\left(\frac{\beta_n b}{\sqrt{\alpha_2}}\right)+B_{2n}cos\left(\frac{\beta_n b}{\sqrt{\alpha_2}}\right)\right]$$

We have a set of equation of three unknown for each n. This set is non-linear due to nature of variable β_n. Bu notice that A_{2n} and B_{2n} forms linear equations, therefore for a given β_n A_{2n} and B_{2n} can be calculated by solving the equations as a linear system of equation. Out of 3 equations 2 can be used to form a linear set of equations and then β_n can be solved by a single non-linear set of equation by some non-linear root finding method such as bisection. So root finding basic function can be defined as:

$\boldsymbol{function\ f(\beta_n)}$

$$\left\{\begin{array}{c}\begin{bmatrix}sin\left(\frac{a}{b}\frac{\beta_n b}{\sqrt{\alpha_2}}\right) & cos\left(\frac{a}{b}\frac{\beta_n b}{\sqrt{\alpha_2}}\right)\\ cos\left(\frac{a}{b}\frac{\beta_n b}{\sqrt{\alpha_2}}\right) & -sin\left(\frac{a}{b}\frac{\beta_n b}{\sqrt{\alpha_2}}\right)\end{bmatrix}\begin{bmatrix}A_{2n}\\ B_{2n}\end{bmatrix}=\begin{bmatrix}sin\left(\frac{\beta_n a}{\sqrt{\alpha_1}}\right)\\ \frac{k_1}{k_2}\frac{\sqrt{\alpha_2}}{\sqrt{\alpha_1}}sin\left(\frac{\beta_n a}{\sqrt{\alpha_1}}\right)\end{bmatrix}\\ f=k_2\left[\frac{\beta_n}{\sqrt{\alpha_2}}\left[A_{2n}cos\left(\frac{\beta_n b}{\sqrt{\alpha_2}}\right)-B_{2n}sin\left(\frac{\beta_n b}{\sqrt{\alpha_2}}\right)\right]\right]-h_3\left[A_{2n}sin\left(\frac{\beta_n b}{\sqrt{\alpha_2}}\right)+B_{2n}cos\left(\frac{\beta_n b}{\sqrt{\alpha_2}}\right)\right]\end{array}\right\}$$

So the equation becomes:

$$T_1(x,t)=\sum_{n=0}^{\infty}C_n e^{-\beta_n t}\varphi_{1n}(x)=\sum_{n=0}^{\infty}C_n e^{-\beta_n t}sin\left(\frac{\beta_n a}{\sqrt{\alpha_1}}\right)$$

$$T_2(x,t)=\sum_{n=0}^{\infty}C_n e^{-\beta_n t}\varphi_{2n}(x)=\sum_{n=0}^{\infty}C_n e^{-\beta_n t}\left[A_{2n}sin\left(\frac{\beta_n x}{\sqrt{\alpha_2}}\right)+B_{2n}cos\left(\frac{\beta_n x}{\sqrt{\alpha_2}}\right)\right]$$

Now initial conditions can be applied to these equations

Initial conditions:

$T_1(x,t)=F_1(x)$ t=0 $0\le x\le a$

$T_2(x,t)=F_2(x)$ t=0 $a\le x\le b$

$$T_1(x,0)=F_1(x)=\sum_{n=0}^{\infty}C_n\varphi_{1n}(x)=\sum_{n=0}^{\infty}C_n sin\left(\frac{\beta_n a}{\sqrt{\alpha_1}}\right)\quad 0\le x\le a$$

$$T_2(x,0)=F_2(x)=\sum_{n=0}^{\infty}C_n\varphi_{2n}(x)=\sum_{n=0}^{\infty}C_n\left[A_{2n}sin\left(\frac{\beta_n x}{\sqrt{\alpha_2}}\right)+B_{2n}cos\left(\frac{\beta_n x}{\sqrt{\alpha_2}}\right)\right]\quad a\le x\le b$$

Sin and cos functions are orthogonal functions, therefore orthogonal relations can be applied to solve C_n coefficients. $\frac{k_1}{\alpha_1}\int_0^a F_1(x)\,sin\left(\frac{\beta_n x}{\sqrt{\alpha_1}}\right)dx+\frac{k_2}{\alpha_2}\int_a^b F_2(x)\left[A_{2n}sin\left(\frac{\beta_n x}{\sqrt{\alpha_2}}\right)+B_{2n}cos\left(\frac{\beta_n x}{\sqrt{\alpha_2}}\right)\right]dx=$

$$C_n\left[\frac{k_1}{\alpha_1}\int_0^a sin^2\left(\frac{\beta_n a}{\sqrt{\alpha_1}}\right)dx+\frac{k_2}{\alpha_2}\int_a^b\left[A_{2n}sin\left(\frac{\beta_n x}{\sqrt{\alpha_2}}\right)+B_{2n}cos\left(\frac{\beta_n x}{\sqrt{\alpha_2}}\right)\right]^2dx\right]$$

$$N_n=\left[\frac{k_1}{\alpha_1}\int_0^a sin^2\left(\frac{\beta_n a}{\sqrt{\alpha_1}}\right)dx+\frac{k_2}{\alpha_2}\int_a^b\left[A_{2n}sin\left(\frac{\beta_n x}{\sqrt{\alpha_2}}\right)+B_{2n}cos\left(\frac{\beta_n x}{\sqrt{\alpha_2}}\right)\right]^2dx\right]$$

$$C_n=\frac{1}{N_n}\left[\frac{k_1}{\alpha_1}\int_0^a F_1(x)\,sin\left(\frac{\beta_n x}{\sqrt{\alpha_1}}\right)dx+\frac{k_2}{\alpha_2}\int_a^b F_2(x)\left[A_{2n}sin\left(\frac{\beta_n x}{\sqrt{\alpha_2}}\right)+B_{2n}cos\left(\frac{\beta_n x}{\sqrt{\alpha_2}}\right)\right]dx\right]$$

PROBLEM 3.12 Solution of the equation in java HT_composite_wall.java

```
import static java.lang.Math.*;
class f_composite implements if_x
{public double a,b,alpha1,alpha2,h3,k1,k2,sqrt_alpha2,sqrt_alpha1,A2,B2;
public f_composite(double ai,double bi,double alpha1i,double alpha2i,double h3i,double k1i,double k2i)
{a=ai;b=bi;alpha1=alpha1i;alpha2=alpha2i;h3=h3i;k1=k1i;k2=k2i;sqrt_alpha2=sqrt(alpha2);sqrt_alpha1=sqrt(alpha1);}

public double func(double x)
{double beta=x;
A2=0;
B2=0;
double gamma=a*beta/sqrt_alpha1;
double eta=b*beta/sqrt_alpha2;
double H=h3*b/k2;
double K=k1/k2*sqrt_alpha2/sqrt_alpha1;
double y=a/b*eta;
double A[][]={{sin(y),cos(y)},{cos(y),-sin(y)}};
double B[]={sin(gamma),K*cos(gamma)};
double z[]=pivotlugauss(A,B);
A2=z[0];B2=z[1];
double z1=A2*(H/eta*sin(eta)+cos(eta))+B2*(H/eta*cos(eta)-sin(eta));
return z1;
}
public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
```

```
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss eleme
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivoting
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                  if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
                  b[k]=dummy;
                }
                //Solving gauss elimination
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
                }
  }
  //back substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }

}
class f_xt_composite implements if_xt
{public double a,b,alpha1,alpha2,h3,k1,k2,sqrt_alpha2,sqrt_alpha1,A2,B2;
 f_composite f;
public  f_xt_composite(double ai,double bi,double alpha1i,double alpha2i,double h3i,double k1i,double k2i)
{a=ai;b=bi;alpha1=alpha1i;alpha2=alpha2i;h3=h3i;k1=k1i;k2=k2i;sqrt_alpha2=sqrt(alpha2);sqrt_alpha1=sqrt(alpha1);
 f =new f_composite(a,b,alpha1,alpha2,h3,k1,k2);}
 public double func(double x,double t)
 {double x0=0;
    double dx=0.0001;
    double x1=0;
    int nn=20;
    double beta[]=new double[nn];
    double A2[]=new double[nn];
    double B2[]=new double[nn];
    double TT=0;
    for(int n=0;n<nn;n++)
    {
    x1=f.enlarge(x0,dx);
    beta[n]= f.bisection(x0,x1);
    A2[n]=f.A2;
    B2[n]=f.B2;
    x0=beta[n]+dx;
    }
    //for(int n=0;n<nn;n++)
    //{System.out.println("beta["+n+"] = "+beta[n]+" A2["+n+"] = "+A2[n]+" B2["+n+"] = "+B2[n]);}
    for(int n=0;n<nn;n++)
    {
    final double fbeta=beta[n];
     final double fA2=A2[n];
```

```
      final double fB2=B2[n];
    if_x fi1=xx->sin(fbeta/sqrt(alpha1)*xx);
    if_x fi2=xx->fA2*sin(fbeta/sqrt(alpha2)*xx)+fB2*cos(fbeta/sqrt(alpha2)*xx);
    if_x fi1_2=xx->fi1.func(xx)*fi1.func(xx);
    if_x fi2_2=xx->fi2.func(xx)*fi2.func(xx);
    if_x F1=xx->20.0;
    if_x F2=xx->20.0;
    if_x fi1_F1=xx->fi1.func(xx)*F1.func(xx);
    if_x fi2_F2=xx->fi2.func(xx)*F2.func(xx);
    double N=k1/alpha1*fi1_2.integral(0.000001,a,10)+k2/alpha2*fi2_2.integral(a,b,10);
    double w=exp(-fbeta*fbeta*t);
    if(x<=a)
    {  TT+=w*fi1.func(x)/N*(k1/alpha1*fi1_F1.integral(0.000001,a,10)+k2/alpha2*fi2_F2.integral(a,b));}
    else //x>a &6 x<=b
    {  TT+=w*fi2.func(x)/N*(k1/alpha2*fi1_F1.integral(0.000001,a,10)+k2/alpha2*fi2_F2.integral(a,b,10));}
    }
    return TT;
   }
}

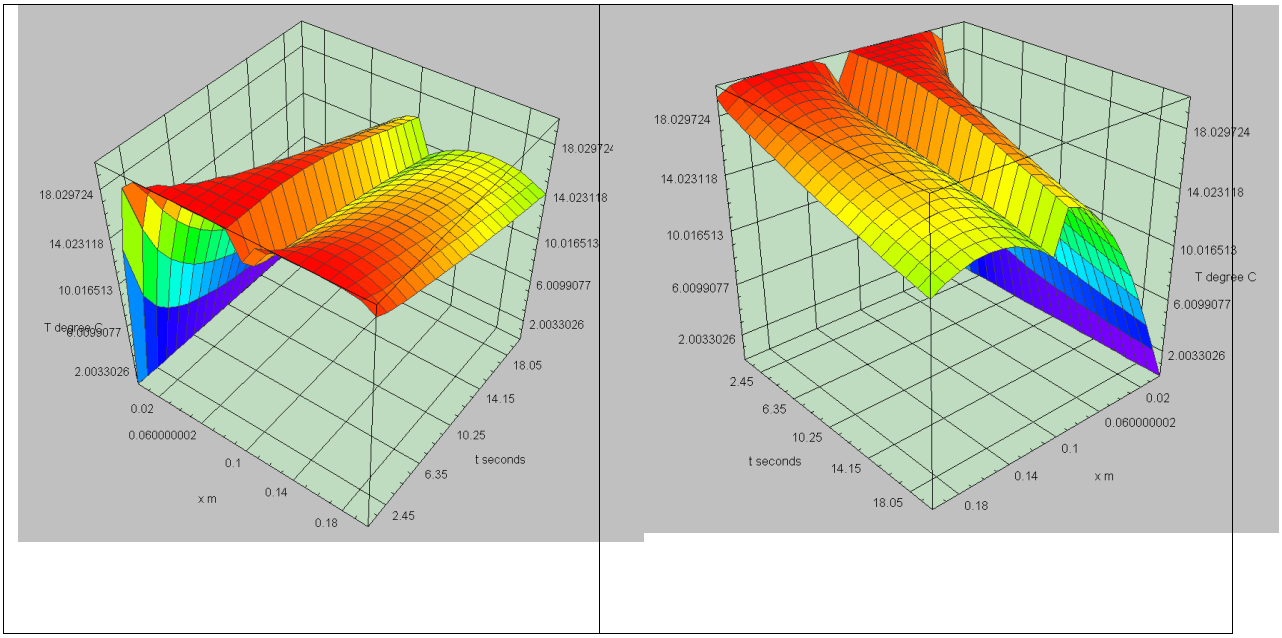
public class HT_composite_wall
{

 public static void main(String arg[])
 {   double a=0.1;  //m
             double b=0.2;  //m
             double alpha1=1e-4;
             double alpha2=1.4e-4;
             double h3=53.0;    //W/(m^2K)
             double k1=55.0;   //W/(mK)
             double k2=12.0; //W/mK
             f_xt_composite fc =new f_xt_composite(a,b,alpha1,alpha2,h3,k1,k2);
    System.out.println("T="+fc.func(0.05,20));
 }
}
```

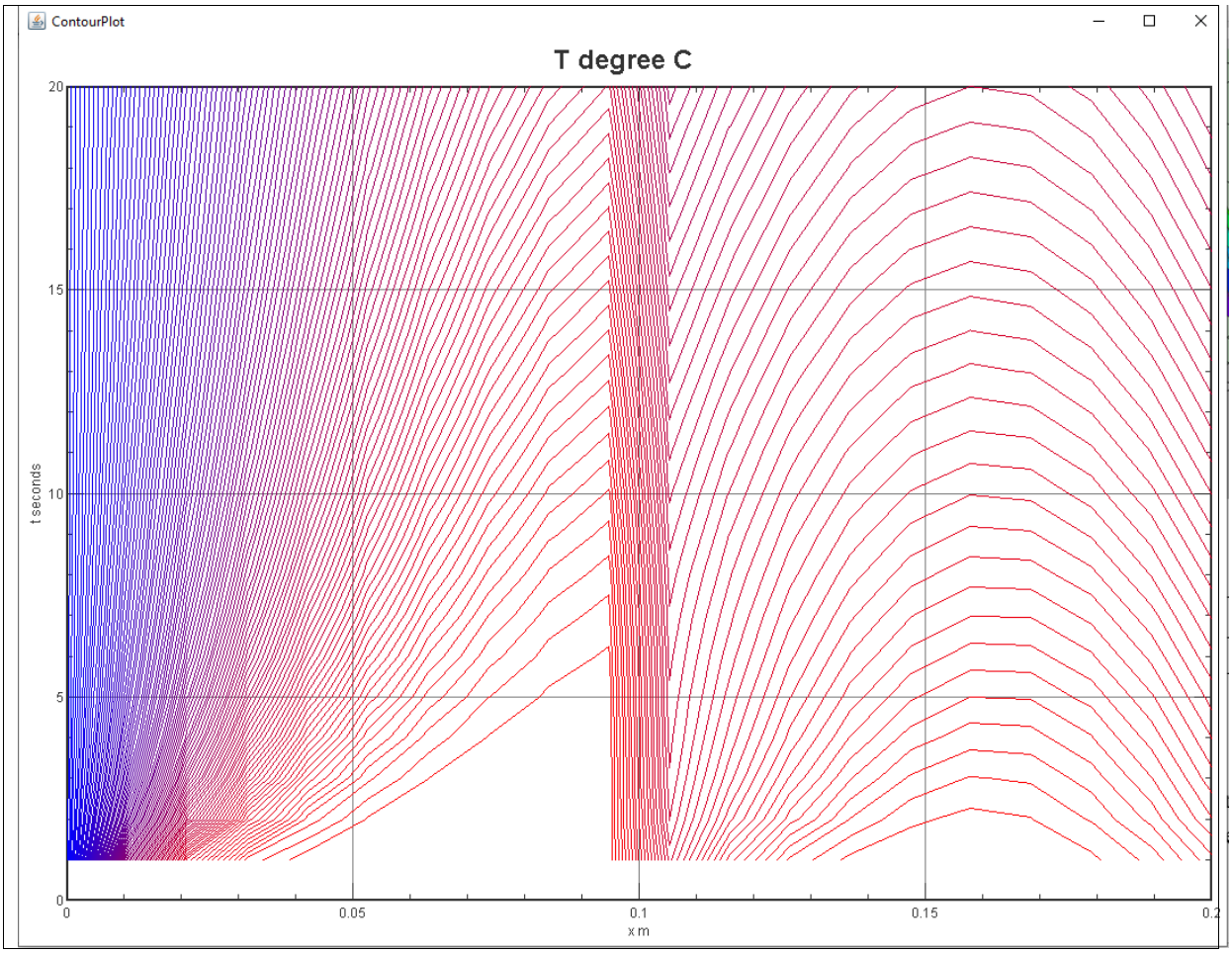
```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_composite_wall
beta[0] = 0.13811260496372413 A2[0] = 1.3046670126210391 B2[0] = -0.554690240236761
beta[1] = 0.23408221644278576 A2[1] = 2.1555031964932487 B2[1] = 3.1815110205440558
beta[2] = 0.4582181625436879 A2[2] = 1.1862087166018016 B2[2] = 0.2680488547300807
beta[3] = 0.5797074548857309 A2[3] = 1.3506191504257223 B2[3] = 4.624352947660564
beta[4] = 0.7807261489068871 A2[4] = 0.550425981927039 B2[4] = 0.8712049567857558
beta[5] = 0.9370686691255451 A2[5] = 0.40943525535619624 B2[5] = 5.399911388796983
beta[6] = 1.102435365343533 A2[6] = -0.26239748584904976 B2[6] = 0.9770729093454886
beta[7] = 1.296587082247933 A2[7] = -0.5755608297964429 B2[7] = 4.977990924063209
beta[8] = 1.4251775840634222 A2[8] = -1.0326046443761248 B2[8] = 0.5525070225731262
beta[9] = 1.6532019716823405 A2[9] = -1.3291941458271517 B2[9] = 3.5129655238712267
beta[10] = 1.752105011959081 A2[10] = -1.5692600307762328 B2[10] = -0.4157233723330962
beta[11] = 2.0005845420206096 A2[11] = -1.644422157631557 B2[11] = 1.7040792351236744
beta[12] = 2.0888173158243237 A2[12] = -1.7261838032482328 B2[12] = -1.9486866223758104
beta[13] = 2.3349956029970755 A2[13] = -1.4792626319695532 B2[13] = 0.2635981006957179
beta[14] = 2.4386749466581237 A2[14] = -1.4194766046610574 B2[14] = -3.782633153387838
beta[15] = 2.6608425786742433 A2[15] = -0.9273952651604207 B2[15] = -0.62941671314879
beta[16] = 2.7969537257679744 A2[16] = -0.6952386511893701 B2[16] = -5.134958573789521
beta[17] = 2.9833394086833103 A2[17] = -0.1444196478956086 B2[17] = -0.9914917262519245
beta[18] = 3.157960665630442 A2[18] = 0.23729795707451354 B2[18] = -5.347808413284253
beta[19] = 3.3054723692301806 A2[19] = 0.6784693333405873 B2[19] = -0.8190523548230823
T=11.151114935837422

> Terminated with exit code 0.
```

3D plot of temperature T(x,t)



Contour plot of the temperature T(x,t)



Temperature profile

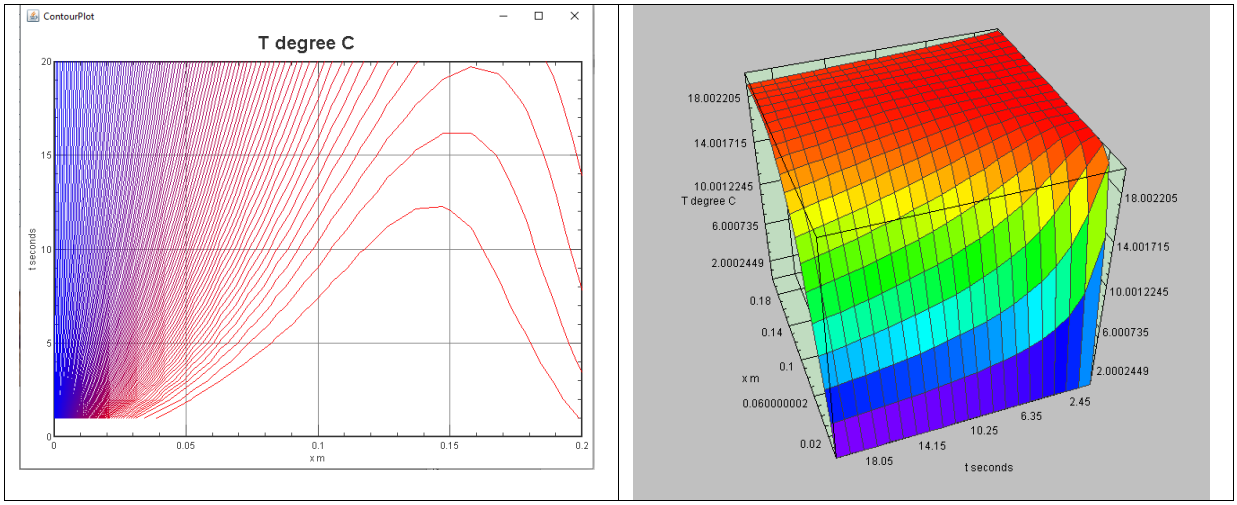
0.00	10.87	17.27	19.49	19.94	20.00	20.00	20.00	20.00	20.00	16.36	18.33	19.45	19.87	19.97
0.00	8.03	14.15	17.71	19.29	19.83	19.97	20.00	20.00	20.00	16.02	17.56	18.72	19.42	19.76
0.00	6.65	12.20	16.05	18.29	19.37	19.80	19.95	19.99	20.00	15.87	17.17	18.24	18.99	19.44
0.00	5.80	10.87	14.72	17.27	18.74	19.49	19.82	19.94	19.98	15.77	16.91	17.89	18.63	19.10
0.00	5.22	9.89	13.64	16.34	18.08	19.08	19.60	19.84	19.93	15.68	16.72	17.62	18.31	18.78

0.00	4.78	9.13	12.76	15.52	17.43	18.63	19.33	19.69	19.84	15.59	16.55	17.38	18.04	18.49	18.71	18.69
0.00	4.43	8.53	12.03	14.79	16.81	18.17	19.01	19.48	19.70	15.48	16.39	17.17	17.78	18.21	18.42	18.40
0.00	4.15	8.03	11.40	14.15	16.23	17.70	18.67	19.24	19.52	15.34	16.22	16.96	17.54	17.94	18.14	18.12
0.00	3.92	7.60	10.87	13.58	15.70	17.25	18.31	18.96	19.29	15.19	16.05	16.76	17.32	17.69	17.87	17.84
0.00	3.72	7.24	10.40	13.06	15.20	16.81	17.94	18.66	19.04	15.01	15.86	16.56	17.09	17.45	17.61	17.58
0.00	3.55	6.93	9.98	12.60	14.74	16.38	17.57	18.34	18.76	14.82	15.66	16.35	16.87	17.21	17.36	17.32
0.00	3.40	6.65	9.61	12.18	14.31	15.98	17.20	18.02	18.46	14.61	15.46	16.14	16.64	16.97	17.12	17.07
0.00	3.27	6.40	9.27	11.79	13.90	15.58	16.83	17.68	18.15	14.40	15.24	15.92	16.42	16.74	16.88	16.82
0.00	3.15	6.17	8.97	11.43	13.52	15.20	16.47	17.34	17.83	14.17	15.02	15.70	16.19	16.51	16.64	16.58
0.00	3.04	5.97	8.68	11.10	13.16	14.84	16.11	17.00	17.51	13.94	14.80	15.47	15.96	16.27	16.40	16.34
0.00	2.94	5.78	8.42	10.79	12.82	14.48	15.76	16.66	17.18	13.71	14.57	15.24	15.73	16.04	16.17	16.11
0.00	2.85	5.61	8.18	10.49	12.49	14.14	15.42	16.32	16.85	13.48	14.34	15.01	15.50	15.81	15.94	15.88
0.00	2.77	5.45	7.95	10.22	12.18	13.81	15.08	15.98	16.52	13.24	14.10	14.78	15.27	15.58	15.71	15.65
0.00	2.69	5.30	7.74	9.95	11.88	13.49	14.75	15.65	16.19	13.00	13.87	14.55	15.04	15.35	15.48	15.42
0.00	2.62	5.15	7.54	9.70	11.60	13.18	14.43	15.33	15.87	12.77	13.63	14.31	14.81	15.12	15.25	15.20

To check the solution if the same material is taken for both layers

$$\alpha_1 = \alpha_2 = 10^{-4}\ m^2/s$$

$$k_1 = k_2 = 55\ W/(mK)$$



4. FINITE DIFFERENCE SOLUTIONS OF HEAT CONDUCTION

4.1 FINITE DIFFERENCES

Introduction to finite difference method

Finite difference concept based on the difference concept. We actually used finite differences to approximate to derivative of the function. In order to approximate derivation difference equations can be used. Diference equations can basically be derived from Taylor formulations

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f"(x_i)}{2}h^2 + \frac{f^{(3)}(x_i)}{3!}h^3 + \frac{f^{(4)}(x_i)}{4!}h^4 + \cdots..$$

If the first derivative is solved from this equation it is found as

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \left[\frac{f"(x_i)}{2}h + \frac{f^{(3)}(x_i)}{3!}h^2 + \frac{f^{(4)}(x_i)}{4!}h^3 + O(h^4)..\right]$$

Or

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + O(h)$$

Term O(h) is the order of error. Error is proportional to the difference parameter h. When h is getting smaller, error will also shrinked. The second derivative can be written the same way:

$$f"(x_i) = \frac{f(x_{i+2}) - 2(x_{i+1}) + f(x_i)}{h^2} + O(h)$$

In order to decrease the error, if the second derivative difference equation is substituted into the first derivative Taylor equation a first derivative equation with a second order error can be obtained.

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \left[\frac{f(x_{i+2}) - 2(x_{i+1}) + f(x_i)}{2h^2}h + O(h^2)\right] = \frac{-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)}{2h} + O(h^2)$$

Difference equation can be written as forward or bakward way. The backward difference equation: $f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h}$. By taken avarage of forward and backward formulas can create a central formula

$$f'(x_i) = \frac{f'(x_i)_{forward} + f'(x_i)_{backward}}{2} = \frac{\frac{f(x_{i+1}) - f(x_i)}{h} - \frac{f(x_i) - f(x_{i-1})}{h}}{2} = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} + O(h^2)$$

Difference formulas for calculating derivatives are given (as coefficients) in the following table

Central differences

Derivative	Accuracy	−4	−3	−2	−1	0	1	2	3	4
1	2				−1/2	0	1/2			
	4			1/12	−2/3	0	2/3	−1/12		
	6		−1/60	3/20	−3/4	0	3/4	−3/20	21916	
	8	1/280	−4/105	1/5	−4/5	0	4/5	−1/5	4/105	−1/280
2	2				1	−2	1			
	4			−1/12	4/3	−5/2	4/3	−1/12		
	6		32874	−3/20	3/2	−49/18	3/2	−3/20	32874	
	8	−1/560	8/315	−1/5	8/5	−205/72	8/5	−1/5	8/315	−1/560
3	2			−1/2	1	0	−1	1/2		
	4		41122	−1	13/8	0	−13/8	1	−1/8	
	6	−7/240	41185	−169/120	61/30	0	−61/30	169/120	−3/10	7/240
4	2			1	−4	6	−4	1		
	4		−1/6	2	−13/2	28/3	−13/2	2	−1/6	
	6	7/240	−2/5	169/60	−122/15	91/8	−122/15	169/60	−2/5	7/240

Derivative	Accuracy	0	1	2	3	4	5	6	7	8
1	1	−1	1							
	2	−3/2	2	−1/2						
	3	−11/6	3	−3/2	1/3					
	4	−25/12	4	−3	4/3	−1/4				
	5	−137/60	5	−5	10/3	−5/4	41030			
	6	−49/20	6	−15/2	20/3	−15/4	41035	−1/6		
2	1	1	−2	1						
	2	2	−5	4	−1					
	3	35/12	−26/3	40958	−14/3	41254				
	4	41014	−77/6	107/6	−13	61/12	−5/6			
	5	203/45	−87/5	117/4	−254/9	33/2	−27/5	137/180		
	6	469/90	−223/10	879/20	−949/18	41	−201/10	1019/180	−7/10	
3	1	−1	3	−3	1					
	2	−5/2	9	−12	7	−3/2				
	3	−17/4	71/4	−59/2	49/2	−41/4	7/4			
	4	−49/8	29	−461/8	62	−307/8	13	−15/8		
	5	−967/120	638/15	−3929/40	389/3	−2545/24	268/5	−1849/120	29/15	
	6	−801/80	349/6	−18353/120	2391/10	−1457/6	4891/30	−561/8	527/30	−469/240
4	1	1	−4	6	−4	1				
	2	3	−14	26	−24	11	−2			
	3	35/6	−31	137/2	−242/3	107/2	−19	41077		
	4	28/3	−111/2	142	−1219/6	176	−185/2	82/3	−7/2	
	5	1069/80	−1316/15	15289/60	−2144/5	10993/24	−4772/15	2803/20	−536/15	967/240

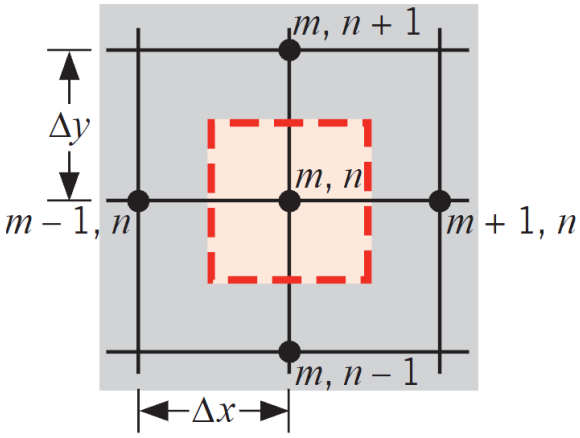
Conversion of differential equations and boundaries to finite element form

Assuming energy flowing from all the neigbour nodes to node m,n, it is possible to calculate any boundary or inner cell heat balance from basic energy conservation rule

Total energies coming to node m,n=0

$t = p\Delta t \quad x = m\Delta x \quad y = n\Delta y$

Steady state inner cell (x,y) directions



$$k\Delta x\frac{(T_{m,n+1}-T_{m,n})}{\Delta y}+k\Delta x\frac{(T_{m,n-1}-T_{m,n})}{\Delta y}+k\Delta y\frac{(T_{m,n-1}-T_{m,n})}{\Delta x}+k\Delta y\frac{(T_{m,n+1}-T_{m,n})}{\Delta x}=0$$

For If $\Delta x=\Delta y$

$$T_{m,n+1}+T_{m,n-1}+T_{m,n-1}+T_{m,n+1}-4T_{m,n}=0$$

$$0\le\phi\le 1$$

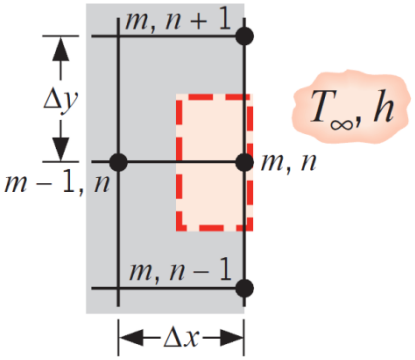
Same equation can be obtained from conservation of energy form of the differential equation as well

$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}=0$$

$$\frac{T_{m+1,n}-2T_{m,n}+T_{m-1,n}}{\Delta x^2}+\frac{T_{m,n+1}-2T_{m,n}+T_{m,n-1}}{\Delta y^2}=0$$

For $\Delta x=\Delta y$ $T_{m+1,n}+T_{m-1,n}+T_{m,n+1}+T_{m,n-1}-4T_{m,n}=0$

Steady state Convective boundary at surface:



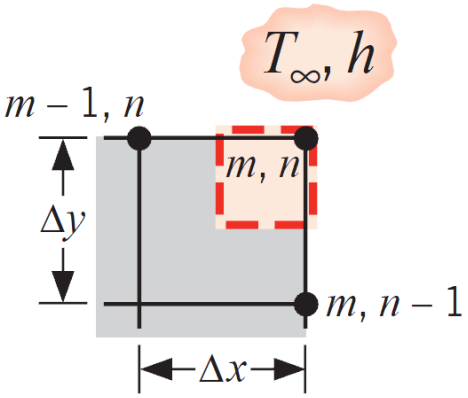
$$h\Delta y(T_\infty-T_{m,n})+k\frac{\Delta x}{2}\frac{(T_{m,n+1}-T_{m,n})}{\Delta y}+k\frac{\Delta x}{2}\frac{(T_{m,n-1}-T_{m,n})}{\Delta y}+k\Delta y\frac{(T_{m-1,n}-T_{m,n})}{\Delta x}$$

If $\Delta x=\Delta y$

$$\frac{2\Delta xh}{k}(T_\infty-T_{m,n})+T_{m,n+1}+T_{m,n-1}+2T_{m-1,n}-4T_{m,n}=0$$

$$\frac{2\Delta xhT_\infty}{k}+T_{m,n+1}+T_{m,n-1}+2T_{m-1,n}-\left(\frac{2\Delta xh}{k}+4\right)T_{m,n}=0$$

Steady state Convective boundary at corner:



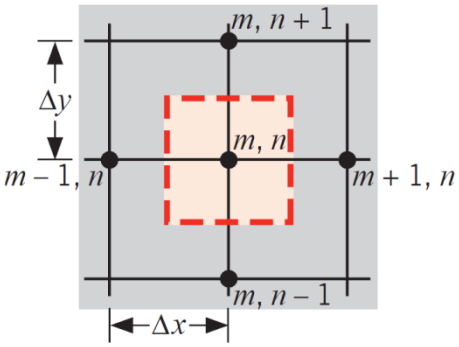
$$h\frac{\Delta y}{2}(T_\infty - T_{m,n}) + h\frac{\Delta x}{2}(T_\infty - T_{m,n}) + k\frac{\Delta x}{2}\frac{(T_{m,n-1} - T_{m,n})}{\Delta y} + k\frac{\Delta y}{2}\frac{(T_{m-1,n} - T_{m,n})}{\Delta x} = 0$$

If $\Delta x = \Delta y$

$$\frac{2\Delta x h}{k}(T_\infty - T_{m,n}) + T_{m,n-1} + T_{m-1,n} - 2T_{m,n} = 0$$

$$\frac{2\Delta x h}{k}T_\infty + T_{m,n-1} + T_{m-1,n} - \left(\frac{2\Delta x h}{k} + 2\right)T_{m,n} = 0$$

Time dependent inner cell (x,y,t) directions



$$\phi\left[k\Delta y\frac{(T^p_{m-1,n} - T^p_{m,n})}{\Delta x} + k\Delta y\frac{(T^p_{m+1,n} - T^p_{m,n})}{\Delta x} + k\Delta x\frac{(T^p_{m,n-1} - T^p_{m,n})}{\Delta y} + k\Delta x\frac{(T^p_{m,n+1} - T^p_{m,n})}{\Delta y}\right]$$
$$+ (1-\phi)\left[k\Delta y\frac{(T^{p+1}_{m-1,n} - T^{p+1}_{m,n})}{\Delta x} + k\Delta y\frac{(T^{p+1}_{m+1,n} - T^p_{m,n})}{\Delta x} + k\Delta x\frac{(T^{p+1}_{m,n-1} - T^{p+1}_{m,n})}{\Delta y}\right.$$
$$\left. + k\Delta x\frac{(T^{p+1}_{m,n+1} - T^{p+1}_{m,n})}{\Delta y} + k\frac{\Delta x}{2}\frac{(T^{p+1}_{m,n-1} - T^{p+1}_{m,n})}{\Delta y}\right] = \rho(\Delta x\Delta y)C_p\frac{T^{p+1}_{m,n} - T^p_{m,n}}{\Delta t}$$

$$0 \le \phi \le 1$$

ϕ =1 explicit solution: error level: $O[\Delta t, (\Delta x)^2]$
ϕ =0 implicit solution: error level: $O[\Delta t, (\Delta x)^2]$
ϕ =0.5 Crank-Nicolson solution: error level: $O[\Delta t, (\Delta x)^2]$
ϕ =0.5$\frac{(\Delta x)^2}{12\alpha\Delta t}$: error level: $O[(\Delta t)^2, (\Delta x)^4]$
Any other ϕmixed solution
The stability criterion depends on the value of the ϕ value
$0 \le \phi \le 0.5$ unconditionally stable
$0.5 \le \phi \le 1$ stable only if $0 \le Fo \le \frac{1}{4\phi\text{-}2}$

Same equation can be obtained from conservation of energy form of the differential equation as well

$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}=\frac{\rho C_p}{k}\frac{\partial T}{\partial t}$$

$$\phi\left[\frac{T^p_{m+1,n}-2T^p_{m,n}+T^p_{m-1,n}}{\Delta x^2}+\frac{T^p_{m,n+1}-2T^p_{m,n}+T^p_{m,n-1}}{\Delta y^2}\right]$$
$$+(1-\phi)\left[\frac{T^{p+1}_{m+1,n}-2T^{p+1}_{m,n}+T^{p+1}_{m-1,n}}{\Delta x^2}+\frac{T^{p+1}_{m,n+1}-2T^{p+1}_{m,n}+T^{p+1}_{m,n-1}}{\Delta y^2}\right]=\frac{\rho C_p}{k}\frac{T^{p+1}_m-T^p_m}{\Delta t}$$

For $\Delta x=\Delta y$

$$\phi\left[T^p_{m+1,n}+T^p_{m-1,n}+T^p_{m,n+1}+T^p_{m,n-1}-4T^p_{m,n}\right]+(1-\phi)\left[T^{p+1}_{m+1,n}+T^{p+1}_{m-1,n}+T^{p+1}_{m,n+1}+T^{p+1}_{m,n-1}-4T^{p+1}_{m,n}\right]$$
$$=\frac{\Delta x^2}{\alpha\Delta t}\left(T^{p+1}_m-T^p_m\right)$$

$Fo=\frac{\alpha\Delta t}{\Delta x^2}$ Fourier number $Bi=\frac{h\Delta x}{k}=H\Delta x$ Biot number $\alpha=\frac{k}{\rho C_p}$ thermal diffusivity

$$\mathrm{Fo}\phi\left[T^p_{m+1,n}+T^p_{m-1,n}+T^p_{m,n+1}+T^p_{m,n-1}-4T^p_{m,n}\right]+Fo(1-\phi)\left[T^{p+1}_{m+1,n}+T^{p+1}_{m-1,n}+T^{p+1}_{m,n+1}+T^{p+1}_{m,n-1}-4T^{p+1}_{m,n}\right]$$
$$=\left(T^{p+1}_m-T^p_m\right)$$

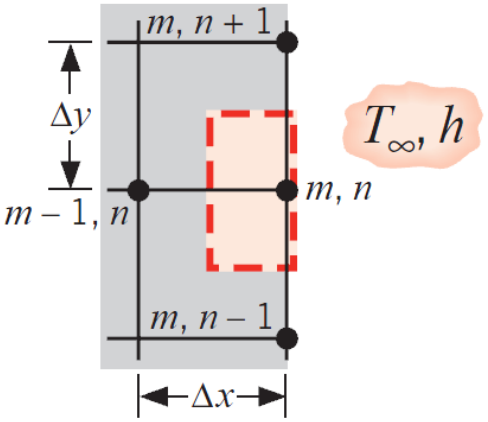
For ϕ=1

$$\mathrm{Fo}\left[T^p_{m+1,n}+T^p_{m-1,n}+T^p_{m,n+1}+T^p_{m,n-1}\right]+(1-4Fo)T^p_m=T^{p+1}_m$$

Two dimensional case:

$$\mathrm{Fo}\left[T^p_{m+1}+T^p_{m-1}\right]+(1-2Fo)T^p_m=T^{p+1}_m$$

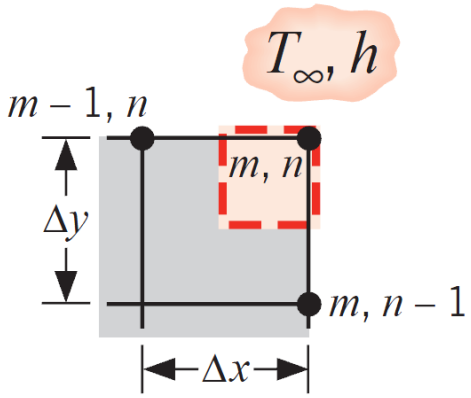
Time dependent Convective boundary at surface:



For $0\le\phi\le1$

$$\phi\left[h\Delta y\left(T_\infty-T^p_{m,n}\right)+k\Delta y\frac{\left(T^p_{m-1,n}-T^p_{m,n}\right)}{\Delta x}+k\frac{\Delta x}{2}\frac{\left(T^p_{m,n+1}-T^p_{m,n}\right)}{\Delta y}+k\frac{\Delta x}{2}\frac{\left(T^p_{m,n-1}-T^p_{m,n}\right)}{\Delta y}\right]$$
$$+(1-\phi)\left[h\Delta y\left(T_\infty-T^{p+1}_{m,n}\right)+k\Delta y\frac{\left(T^{p+1}_{m-1,n}-T^{p+1}_{m,n}\right)}{\Delta x}+k\frac{\Delta x}{2}\frac{\left(T^{p+1}_{m,n+1}-T^{p+1}_{m,n}\right)}{\Delta y}\right.$$
$$\left.+k\frac{\Delta x}{2}\frac{\left(T^{p+1}_{m,n-1}-T^{p+1}_{m,n}\right)}{\Delta y}\right]=\rho\left(\frac{\Delta x}{2}\Delta y\right)C_p\frac{T^{p+1}_{m,n}-T^p_{m,n}}{\Delta t}$$

Time dependent Convective boundary at corner:



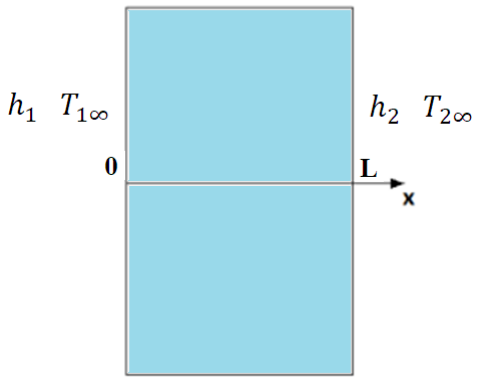
For $0 \le \phi \le 1$

$$\phi\left[h\frac{\Delta y}{2}\left(T_\infty - T_{m,n}^{p}\right) + h\frac{\Delta x}{2}\left(T_\infty - T_{m,n}^{p}\right) + k\frac{\Delta y}{2}\frac{\left(T_{m-1,n}^{p} - T_{m,n}^{p}\right)}{\Delta x} + k\frac{\Delta x}{2}\frac{\left(T_{m,n-1}^{p} - T_{m,n}^{p}\right)}{\Delta y}\right]$$
$$+ (1-\phi)\left[h\frac{\Delta y}{2}\left(T_\infty - T_{m,n}^{p+1}\right) + h\frac{\Delta x}{2}\left(T_\infty - T_{m,n}^{p+1}\right) + k\frac{\Delta y}{2}\frac{\left(T_{m-1,n}^{p+1} - T_{m,n}^{p+!}\right)}{\Delta x}\right.$$
$$\left. + k\frac{\Delta x}{2}\frac{\left(T_{m,n-1}^{p+1} - T_{m,n}^{p+1}\right)}{\Delta y}\right] = \rho\left(\frac{\Delta x}{2}\Delta y\right)C_p\frac{T_{m,n}^{p+1} - T_{m,n}^{p}}{\Delta t}$$

If $\Delta x = \Delta y$ and assuming $Fo = \frac{\alpha \Delta t}{\Delta x^2}$ Fourier number $Bi = \frac{h\Delta x}{k} = H\Delta x$ Biot number $\alpha = \frac{k}{\rho C_p}$ thermal diffusivity, equations can be simplified further

Now let us consider a differential equation such as heat conduction equation in cartesian coordinates. It should be note that when a differential equation is given, in order to solve the differential equations boundary conditions should also be specified.

PROBLEM 4.1.1 FINITE DIFFERENCE SOLUTION, NON-HOMOGENEOUS, TIME DEPENDENT , CARTESIAN COORDINATES



$$\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T(x,t)}{\partial t}$$

Boundary conditions

$$-\frac{\partial T(x,t)}{\partial x} + H_1(T_{1\infty} - T(x,t)) = 0 \quad \text{x=0} \quad \text{t>0} \quad H_1 = \frac{h_1}{k}$$

$$\frac{\partial T^*(x,t)}{\partial x} + H_2(T(x,t) - T_{2\infty}) = 0 \quad \text{x=L} \quad \text{t>0} \quad H_2 = \frac{h_2}{k}$$

$$T^*(x,t) = F^*(x) \quad \text{t=0} \quad 0 \le x \le L$$

$$\phi\frac{T_{m+1}^{p} - 2T_m^{p} + T_{m-1}^{p}}{\Delta x^2} + (1-\phi)\frac{T_{m+1}^{p+1} - 2T_m^{p+1} + T_{m-1}^{p+1}}{\Delta x^2} = \frac{1}{\alpha}\frac{T_m^{p+1} - T_m^{p}}{\Delta t}$$

$Fo = \frac{\alpha \Delta t}{\Delta x^2}$ Fourier number $Bi = \frac{h\Delta x}{k} = H\Delta x$ Biot number $\alpha = \frac{k}{\rho C_p}$ thermal diffusivity

$t = p\Delta t \quad x = m\Delta x$

By rearranging the equation

$\phi \mathrm{Fo} T_{m+1}^{p} + (1 - 2\phi \mathrm{Fo}) T_{m}^{p} + \phi \mathrm{Fo} T_{m-1}^{p} = -(1-\phi)\mathrm{Fo} T_{m+1}^{p+1} + (1 - 2(1-\phi)\mathrm{Fo}) T_{m}^{p+1} - (1-\phi) T_{m-1}^{p+1}$

ϕ =1 explicit solution: error level: $O[\Delta t, (\Delta x)^2]$

ϕ =0 implicit solution: error level: $O[\Delta t, (\Delta x)^2]$

ϕ =0.5 Crank-Nicolson solution: error level: $O[\Delta t, (\Delta x)^2]$

ϕ =0.5$\frac{(\Delta x)^2}{12\alpha\Delta t}$: error level: $O[(\Delta t)^2, (\Delta x)^4]$

Any other ϕmixed solution

The stability criterion depends on the value of the ϕ value

$0 \leq \phi \leq 0.5$ unconditionally stable

$0.5 \leq \phi \leq 1$ stable only if $0 \leq Fo \leq \frac{1}{4\phi\text{-}2}$

Boundary conditions:

$$\phi h_1 A(T_{1\infty} - T_0^p) + \phi \frac{kA}{\Delta x}(T_1^p - T_0^p) + (1-\phi) h_1 A(T_{1\infty} - T_0^{p+1}) + (1-\phi)\frac{kA}{\Delta x}(T_1^{p+1} - T_0^{p+1}) = \rho C_p \left(A\frac{\Delta x}{2}\right)\frac{T_0^{p+1} - T_0^p}{\Delta t}$$

$$\phi h_2 A(T_{2\infty} - T_{k-1}^p) + \phi \frac{kA}{\Delta x}(T_{k-2}^p - T_{k-1}^p) + (1-\phi) h_2 A(T_{2\infty} - T_{k-1}^{p+1}) + (1-\phi)\frac{kA}{\Delta x}(T_{k-2}^{p+1} - T_{k-1}^{p+1}) = \rho C_p \left(A\frac{\Delta x}{2}\right)\frac{T_{k-1}^{p+1} - T_{k-1}^p}{\Delta t}$$

By rearranging the equations

$$2\phi Bi_1 Fo(T_{1\infty} - T_0^p) + 2\phi(T_1^p - T_0^p) + 2(1-\phi)Bi_1 Fo\left(T_{1\infty} - T_0^{p+1}\right) + 2(1-\phi)\left(T_1^{p+1} - T_0^{p+1}\right) = T_0^{p+1} - T_0^p$$

$$2\phi Bi_2 Fo(T_{2\infty} - T_{k-1}^p) + 2\phi(T_{k-2}^p - T_{k-1}^p) + 2(1-\phi)Bi_2 Fo\left(T_{2\infty} - T_{k-1}^{p+1}\right) + 2(1-\phi)\left(T_{k-2}^{p+1} - T_{k-1}^{p+1}\right) = T_{k-1}^{p+1} - T_{k-1}^p$$

Problem 4.1 Java solution:

PROBLEM 4.1.1 :

L=0.8 m thick Wall has an initial temperature profile $T_0 = 20$ ℃. Sudenly Wall exposed to convective flow a fluid from both side . At x=0 side convective heat transfer coefficient h_1=100 W/(m^{2}K) and temperature $T_{1\infty} = 200$ ℃ and x=L=0.8 m side convective heat transfer coefficient h_2=100 W/(m^{2}K) and temperature $T_{2\infty} = 200$ ℃ . Thermal conductivity of Wall k=50 W/(mK). Thermal diffusivity of wall is α =8.4x10^{-5} m^2/s. Calculate Wall temperature profile at t=600 s

```
import java.io.*;

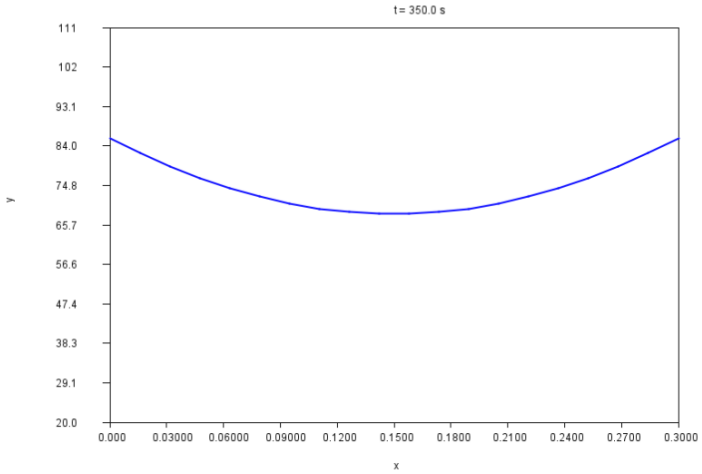
class HT_xt_FDB
{ public static double [] thomas(double a[][],double b[])
{
//3 band matrix
int n=a.length;
double f[]=new double[n];
double e[]=new double[n];
double g[]=new double[n];
double x[]=new double[n];
for(int i=0;i<n;i++)     {f[i]=a[i][i];}
for(int i=0;i<(n-1);i++) {g[i]=a[i][i+1];}
for(int i=0;i<(n-1);i++) {e[i+1]=a[i+1][i];}
for(int k=1;k<n;k++)
{e[k]=e[k]/f[k-1];
f[k]=f[k]-e[k]*g[k-1];
}
for(int k=1;k<n;k++)
{b[k]=b[k]-e[k]*b[k-1];
}
x[n-1]=b[n-1]/f[n-1];
for(int k=(n-2);k>=0;k--)
{x[k]=(b[k]-g[k]*x[k+1])/f[k];}
return x;
}
public static double [][] q(double k, double dx,double T[][])
{  //heat transfer
int nx=T[0].length;
int nt=T.length;
double qi[][]=new double[nt][nx-1];
for(int p=0;p<nt;p++)
{  for(int m=1;m<nx;m++)
```

```
{ qi[p][m-1]=-k*(T[p][m]-T[p][m-1])/dx;}
}
return qi;
}

public static double [][] HT_xtcoz(double fi, double L,double t, double dt,double T0[],double alpha,double Bi1,double Bi2,double
T1inf,double T2inf)
{
//  L duvar kalınlığı
//  t maksimum zaman
//  dt zaman stebi
//  T1 x=0 t>=0 boundary temperature
//  T2 x=L t>=0 boundary temperature
int nx=T0.length; //x step sayısı
//System.out.println(nx="+nx)
double A[][]=new double[nx][nx];
double B[]=new double[nx];
double dx=L/(nx-1); //x steb uzunluğu
int nt=(int)(t/dt+0.001);
double Fo=alpha*dt/(dx*dx);
double T[][]=new double[nt+1][nx];
for(int m=0;m<nx;m++) {T[0][m]=T0[m];}
for(int p=0;p<nt;p++)
{ //t>0 wall temperatures
for(int m=0;m<nx;m++)
{
if(m==0) B[m]=(1.0-2.0*Fo*fi-2*Fo*Bi1*fi)*T[p][m]+2.0*fi*Fo*T[p][m+1]+2.0*Bi1*Fo*T1inf;
else if(m==(nx-1)) B[m]=(1.0-2.0*Fo*fi-2*Fo*Bi2*fi)*T[p][m]+2.0*fi*Fo*T[p][m-1]+2.0*Bi2*Fo*T2inf;
else  B[m]=fi*Fo*T[p][m-1]+(1.0-2.0*Fo*fi)*T[p][m]+fi*Fo*T[p][m+1];
if(m==0)
{A[m][m]=(1+2.0*Fo*Bi1*(1.0-fi)+2.0*Fo*(1.0-fi));
A[m][m+1]=-2.0*(1.0-fi)*Fo;
}
else if(m==(nx-1))
{A[m][m]=(1+2.0*Fo*Bi2*(1.0-fi)+2.0*Fo*(1.0-fi));
A[m][m-1]=-2.0*(1.0-fi)*Fo;
}
else
{A[m][m]=(2.0*Fo*(1.0-fi)+1);
A[m][m-1]=-(1.0-fi)*Fo;
A[m][m+1]=-(1.0-fi)*Fo;
}
}
T[p+1]=thomas(A,B);
}
return T;
}

public static void main(String args[])
{
double T0[]=new double[81];
double L=0.8;
// t=0 daki T0(x) sıcaklık profili
int N=T0.length;
for(int m=0;m<N;m++) {T0[m]=300-300.0*Math.sin(m*Math.PI/N);}
double T1inf=100.0;  //x=0 da t>0 zamanı için duvar sıcaklığı sınır şartı
double T2inf=25.0;  //x=L de t>0 zamanı için duvar sıcaklığı sınır şartı
double h1=250.0;  //x=0 da t>0 zamanı için duvar sıcaklığı sınır şartı
double h2=250.0;  //x=L de t>0 zamanı için duvar sıcaklığı sınır şartı
double t=600.0; //sn toplam zaman
double dt=1.0; //sn gerçek zaman adımı
double dtplot=0.3; //sn her bir plotta bekleme zamanı
double alpha=8.4e-5;
double x0=0.0;
double xn=L;
double fi=0.5;
double k=200.0;
double dx=L/T0.length;
double Bi1=h1*dx/k;
double Bi2=h2*dx/k;
double T[][]=HT_xtcoz(fi,L,t,dt,T0,alpha,Bi1,Bi2,T1inf,T2inf);
double b[][]=Matrix.T(T);
double q[][]=q(k,dx,T);
timeplot p=new timeplot(T,x0,xn,dtplot,dt);
//timeplot p=new timeplot(q,x0,xn,dtplot,dt);
p.start();
}
```

```
}
```



Note: time plot is time dependent real time plot program, and scene above captured from the real time plot .

t sec	x m																			
0	0	0.015789	0.031578	0.047367	0.063156	0.078945	0.094734	0.110523	0.126312	0.142101	0.15789	0.173679	0.189468	0.205257	0.221046	0.236835	0.252624	0.268413	0.284202	0.299991
10	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00
20	32.62	25.95	22.80	21.32	20.62	20.29	20.14	20.07	20.03	20.02	20.02	20.03	20.07	20.14	20.29	20.62	21.32	22.80	25.95	32.62
30	33.75	30.76	27.08	24.29	22.47	21.38	20.75	20.41	20.24	20.17	20.17	20.24	20.41	20.75	21.38	22.47	24.29	27.08	30.76	33.75
40	38.47	33.04	29.64	26.87	24.66	23.03	21.91	21.20	20.79	20.60	20.60	20.79	21.20	21.91	23.03	24.66	26.87	29.64	33.04	38.47
50	39.91	36.13	32.14	29.00	26.57	24.70	23.30	22.32	21.71	21.41	21.41	21.71	22.32	23.30	24.70	26.57	29.00	32.14	36.13	39.91
60	42.79	37.91	34.29	31.10	28.45	26.35	24.76	23.62	22.88	22.52	22.52	22.88	23.62	24.76	26.35	28.45	31.10	34.29	37.91	42.79
70	44.26	40.19	36.21	32.94	30.23	28.02	26.30	25.04	24.22	23.81	23.81	24.22	25.04	26.30	28.02	30.23	32.94	36.21	40.19	44.26
80	46.38	41.80	38.08	34.75	31.95	29.66	27.87	26.54	25.66	25.23	25.23	25.66	26.54	27.87	29.66	31.95	34.75	38.08	41.80	46.38
90	47.83	43.67	39.76	36.45	33.64	31.30	29.46	28.08	27.17	26.72	26.72	27.17	28.08	29.46	31.30	33.64	36.45	39.76	43.67	47.83
100	49.59	45.20	41.46	38.12	35.28	32.93	31.05	29.66	28.73	28.26	28.26	28.73	29.66	31.05	32.93	35.28	38.12	41.46	45.20	49.59
110	51.02	46.86	43.03	39.73	36.90	34.54	32.65	31.24	30.30	29.83	29.83	30.30	31.24	32.65	34.54	36.90	39.73	43.03	46.86	51.02
120	52.60	48.34	44.62	41.32	38.49	36.13	34.25	32.83	31.89	31.42	31.42	31.89	32.83	34.25	36.13	38.49	41.32	44.62	48.34	52.60
130	54.01	49.89	46.14	42.88	40.06	37.71	35.83	34.42	33.48	33.01	33.01	33.48	34.42	35.83	37.71	40.06	42.88	46.14	49.89	54.01
140	55.48	51.33	47.66	44.41	41.61	39.28	37.41	36.00	35.06	34.59	34.59	35.06	36.00	37.41	39.28	41.61	44.41	47.66	51.33	55.48
150	56.87	52.81	49.14	45.92	43.15	40.83	38.97	37.57	36.64	36.17	36.17	36.64	37.57	38.97	40.83	43.15	45.92	49.14	52.81	56.87
160	58.29	54.22	50.61	47.42	44.67	42.37	40.52	39.13	38.20	37.74	37.74	38.20	39.13	40.52	42.37	44.67	47.42	50.61	54.22	58.29
170	59.65	55.65	52.06	48.90	46.17	43.89	42.06	40.68	39.76	39.30	39.30	39.76	40.68	42.06	43.89	46.17	48.90	52.06	55.65	59.65
180	61.02	57.04	53.49	50.36	47.66	45.40	43.58	42.21	41.30	40.84	40.84	41.30	42.21	43.58	45.40	47.66	50.36	53.49	57.04	61.02
190	62.36	58.43	54.91	51.81	49.13	46.89	45.09	43.74	42.83	42.38	42.38	42.83	43.74	45.09	46.89	49.13	51.81	54.91	58.43	62.36
200	63.70	59.80	56.31	53.24	50.59	48.37	46.58	45.24	44.35	43.90	43.90	44.35	45.24	46.58	48.37	50.59	53.24	56.31	59.80	63.70
210	65.01	61.16	57.70	54.66	52.03	49.83	48.07	46.74	45.85	45.40	45.40	45.85	46.74	48.07	49.83	52.03	54.66	57.70	61.16	65.01
220	66.32	62.50	59.08	56.06	53.46	51.28	49.53	48.21	47.33	46.89	46.89	47.33	48.21	49.53	51.28	53.46	56.06	59.08	62.50	66.32
230	67.61	63.83	60.44	57.45	54.88	52.72	50.98	49.68	48.81	48.37	48.37	48.81	49.68	50.98	52.72	54.88	57.45	60.44	63.83	67.61
240	68.89	65.14	61.79	58.83	56.28	54.14	52.42	51.13	50.27	49.84	49.84	50.27	51.13	52.42	54.14	56.28	58.83	61.79	65.14	68.89
250	70.15	66.45	63.12	60.19	57.67	55.55	53.85	52.57	51.71	51.29	51.29	51.71	52.57	53.85	55.55	57.67	60.19	63.12	66.45	70.15
260	71.41	67.74	64.44	61.54	59.04	56.94	55.26	53.99	53.15	52.72	52.72	53.15	53.99	55.26	56.94	59.04	61.54	64.44	67.74	71.41
270	72.65	69.01	65.75	62.88	60.40	58.33	56.66	55.40	54.56	54.14	54.14	54.56	55.40	56.66	58.33	60.40	62.88	65.75	69.01	72.65
280	73.88	70.28	67.05	64.20	61.75	59.69	58.04	56.80	55.97	55.55	55.55	55.97	56.80	58.04	59.69	61.75	64.20	67.05	70.28	73.88
290	75.10	71.53	68.33	65.52	63.09	61.05	59.41	58.18	57.36	56.95	56.95	57.36	58.18	59.41	61.05	63.09	65.52	68.33	71.53	75.10
300	76.30	72.77	69.61	66.81	64.41	62.39	60.77	59.55	58.74	58.33	58.33	58.74	59.55	60.77	62.39	64.41	66.81	69.61	72.77	76.30
310	77.50	74.00	70.86	68.10	65.72	63.72	62.11	60.91	60.10	59.70	59.70	60.10	60.91	62.11	63.72	65.72	68.10	70.86	74.00	77.50
320	78.68	75.22	72.11	69.37	67.01	65.04	63.45	62.25	61.45	61.05	61.05	61.45	62.25	63.45	65.04	67.01	69.37	72.11	75.22	78.68
330	79.85	76.42	73.35	70.64	68.30	66.34	64.77	63.58	62.79	62.39	62.39	62.79	63.58	64.77	66.34	68.30	70.64	73.35	76.42	79.85
340	81.01	77.62	74.57	71.89	69.57	67.63	66.07	64.90	64.11	63.72	63.72	64.11	64.90	66.07	67.63	69.57	71.89	74.57	77.62	81.01
350	82.16	78.80	75.78	73.12	70.83	68.91	67.36	66.20	65.43	65.04	65.04	65.43	66.20	67.36	68.91	70.83	73.12	75.78	78.80	82.16
360	83.30	79.97	76.98	74.35	72.08	70.17	68.65	67.49	66.73	66.34	66.34	66.73	67.49	68.65	70.17	72.08	74.35	76.98	79.97	83.30
370	84.43	81.13	78.17	75.56	73.31	71.43	69.91	68.77	68.01	67.63	67.63	68.01	68.77	69.91	71.43	73.31	75.56	78.17	81.13	84.43
380	85.54	82.27	79.35	76.76	74.54	72.67	71.17	70.04	69.29	68.91	68.91	69.29	70.04	71.17	72.67	74.54	76.76	79.35	82.27	85.54
390	86.65	83.41	80.51	77.95	75.75	73.90	72.41	71.30	70.55	70.18	70.18	70.55	71.30	72.41	73.90	75.75	77.95	80.51	83.41	86.65

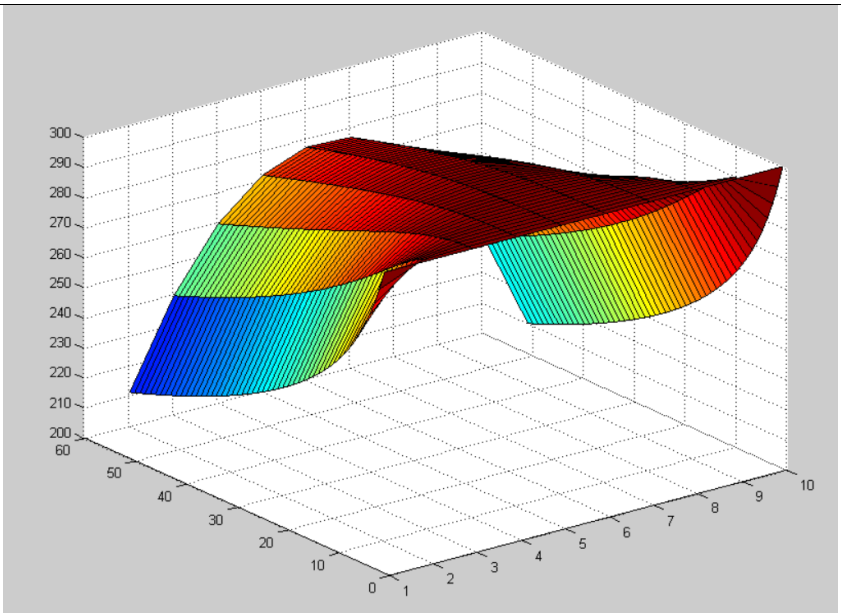
400	87.74	84.54	81.66	79.13	76.95	75.12	73.65	72.54	71.80	71.43	71.43	71.80	72.54	73.65	75.12	76.95	79.13	81.66	84.54	87.74
410	88.83	85.65	82.81	80.30	78.14	76.32	74.87	73.77	73.04	72.67	72.67	73.04	73.77	74.87	76.32	78.14	80.30	82.81	85.65	88.83
420	89.90	86.76	83.94	81.45	79.31	77.52	76.07	74.99	74.26	73.90	73.90	74.26	74.99	76.07	77.52	79.31	81.45	83.94	86.76	89.90
430	90.96	87.85	85.06	82.60	80.48	78.70	77.27	76.20	75.48	75.12	75.12	75.48	76.20	77.27	78.70	80.48	82.60	85.06	87.85	90.96
440	92.02	88.93	86.17	83.73	81.63	79.87	78.46	77.39	76.68	76.32	76.32	76.68	77.39	78.46	79.87	81.63	83.73	86.17	88.93	92.02
450	93.06	90.01	87.27	84.86	82.77	81.03	79.63	78.58	77.87	77.52	77.52	77.87	78.58	79.63	81.03	82.77	84.86	87.27	90.01	93.06
460	94.09	91.07	88.36	85.97	83.91	82.18	80.79	79.75	79.05	78.70	78.70	79.05	79.75	80.79	82.18	83.91	85.97	88.36	91.07	94.09
470	95.12	92.12	89.44	87.07	85.03	83.32	81.94	80.91	80.22	79.87	79.87	80.22	80.91	81.94	83.32	85.03	87.07	89.44	92.12	95.12
480	96.13	93.16	90.50	88.16	86.14	84.44	83.08	82.06	81.38	81.03	81.03	81.38	82.06	83.08	84.44	86.14	88.16	90.50	93.16	96.13
490	97.13	94.19	91.56	89.24	87.24	85.56	84.21	83.20	82.52	82.18	82.18	82.52	83.20	84.21	85.56	87.24	89.24	91.56	94.19	97.13
500	98.13	95.22	92.61	90.31	88.33	86.67	85.33	84.33	83.66	83.32	83.32	83.66	84.33	85.33	86.67	88.33	90.31	92.61	95.22	98.13
510	99.11	96.23	93.64	91.37	89.41	87.76	86.44	85.44	84.78	84.45	84.45	84.78	85.44	86.44	87.76	89.41	91.37	93.64	96.23	99.11
520	100.08	97.23	94.67	92.42	90.47	88.84	87.54	86.55	85.89	85.56	85.56	85.89	86.55	87.54	88.84	90.47	92.42	94.67	97.23	100.08
530	101.05	98.22	95.69	93.46	91.53	89.92	88.62	87.65	86.99	86.67	86.67	86.99	87.65	88.62	89.92	91.53	93.46	95.69	98.22	101.05
540	102.00	99.20	96.70	94.49	92.58	90.98	89.70	88.73	88.09	87.76	87.76	88.09	88.73	89.70	90.98	92.58	94.49	96.70	99.20	102.00
550	102.95	100.18	97.69	95.50	93.62	92.03	90.76	89.81	89.17	88.85	88.85	89.17	89.81	90.76	92.03	93.62	95.50	97.69	100.18	102.95
560	103.89	101.14	98.68	96.51	94.64	93.08	91.82	90.87	90.24	89.92	89.92	90.24	90.87	91.82	93.08	94.64	96.51	98.68	101.14	103.89
570	104.82	102.10	99.66	97.51	95.66	94.11	92.86	91.92	91.30	90.98	90.98	91.30	91.92	92.86	94.11	95.66	97.51	99.66	102.10	104.82
580	105.73	103.04	100.63	98.50	96.67	95.13	93.90	92.97	92.35	92.04	92.04	92.35	92.97	93.90	95.13	96.67	98.50	100.63	103.04	105.73
590	106.64	103.98	101.59	99.48	97.67	96.14	94.92	94.00	93.39	93.08	93.08	93.39	94.00	94.92	96.14	97.67	99.48	101.59	103.98	106.64
600	**107.55**	**104.91**	**102.54**	**100.45**	**98.65**	**97.15**	**95.94**	**95.02**	**94.42**	**94.11**	**94.11**	**94.42**	**95.02**	**95.94**	**97.15**	**98.65**	**100.45**	**102.54**	**104.91**	**107.55**

Problem 4. 1 Matlab solution

```
function x=gauss1(a,b)
% Gauss elimination
n=length(b);
carpan=0;
toplam=0;
for k=1:n
%pivoting
p=k;
buyuk=abs(a(k,k));
for ii=k+1:n
dummy=abs(a(ii,k));
if dummy > buyuk buyuk=dummy;p=ii;end
end
if p~=k
for jj=k:n
dummy=a(p,jj);
a(p,jj)=a(k,jj);
a(k,jj)=dummy;
end
dummy=b(p);
b(p)=b(k);
b(k)=dummy;
end
%gauss eleme
for i=k+1:n
carpan=a(i,k)/a(k,k);
a(i,k)=0;
for j=k+1:n
a(i,j)=a(i,j)-carpan*a(k,j);
end
b(i)   =b(i)   -carpan*b(k);
end
end
%back substituting
x(n)=b(n)/a(n,n);
for i=n-1:-1:1
toplam=0;
for j=i+1:n
toplam=toplam+a(i,j)*x(j);end
x(i)=(b(i)-toplam)/a(i,i);
end
end
```

```
function [T] = HT_xtcoz(fi,L,t,dt,TT,alpha,Bi1,Bi2,T1inf,T2inf)
%  L wall thickness
%  t maksimum time
```

```
% dt time step
% T1 x=0 t>=0 boundary temperature
% T2 x=L t>=0 boundary temperature
nx=length(TT); %x number of steps
dx=L/(nx-1);
nt=t/dt;
Fo=alpha*dt/(dx*dx);
for p=1:nt
for m=1:nx
if p==1 T(p,m)=TT(m);
else T(p,m)=0.0;
end
end
end
A=zeros(nx,nx);
B=zeros(nx);
for p=1:nt
for m=1:nx
if(m==1)
A(m,m)=(1+2.0*Fo*Bi1*(1.0-fi)+2.0*Fo*(1.0-fi));
A(m,m+1)=-2.0*(1.0-fi)*Fo;
elseif(m==nx)
A(m,m)=(1+2.0*Fo*Bi2*(1.0-fi)+2.0*Fo*(1.0-fi));
A(m,m-1)=-2.0*(1.0-fi)*Fo;
else
A(m,m)=(2.0*Fo*(1.0-fi)+1);
A(m,m-1)=-(1.0-fi)*Fo;
A(m,m+1)=-(1.0-fi)*Fo;
end
if m==1
B(m)=(1.0-2.0*Fo*fi)*T(p,m)+fi*Fo*T(p,m+1)+Fo*T1inf;
elseif m==nx
B(m)=(1.0-2.0*Fo*fi-2*Fo*Bi2*fi)*T(p,m)+2.0*fi*Fo*T(p,m-1)+2.0*Bi2*Fo*T2inf;
else
B(m)=fi*Fo*T(p,m-1)+(1.0-2.0*Fo*fi)*T(p,m)+fi*Fo*T(p,m+1);
end
end
W=gauss1(A,B);
for m=1:nx
T(p+1,m)=W(m);
end
end
end
```



PROBLEM 4.1.2:

A Wall initially at $T_i = 100$ ℃ and L=0.3 m. Thermal conductivity of Wall k=50 W/(mK). Thermal diffusivity of wall is α =8.4x10^{-5} m^2/s. Temperature of both side of the Wall is dropped to $T_0 = 20$ ℃ Calculate Wall temperature profile as a function of time

For ϕ=1

Two dimensional case:

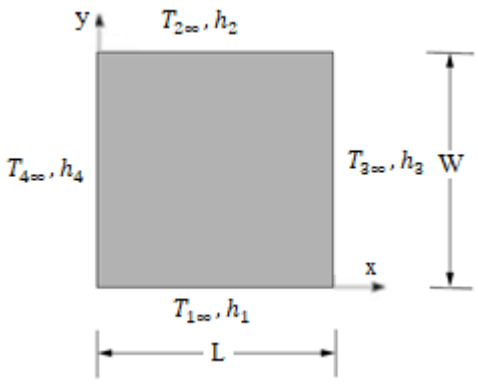
$$\mathrm{Fo}\left[T_{m+1}^{p}+T_{m-1}^{p}\right]+(1-2Fo)T_{m}^{p}=T_{m}^{p+1}$$

$Fo=\frac{\alpha\Delta t}{\Delta x^2}$ Fourier number stability condition for ϕ=1 : $Fo \leq 0.5$

dt	1	second
dx	0.05	m
α	6.40E-05	m^2/s
Fo	0.0256	

t s/x m	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5
0	100	100	100	100	100	100	100	100	100	100	100
1	20	100	100	100	100	100	100	100	100	100	20
2	20	97.952	100	100	100	100	100	100	100	97.952	20
3	20	96.00886	99.94757	100	100	100	100	100	99.94757	96.00886	20
4	20	94.16386	99.84808	99.99866	100	100	100	99.99866	99.84808	94.16386	20
5	20	92.41078	99.70642	99.99484	99.99997	100	99.99997	99.99484	99.70642	92.41078	20
6	20	90.74384	99.52704	99.98759	99.99984	100	99.99984	99.98759	99.52704	90.74384	20
7	20	89.15764	99.31398	99.97611	99.99953	99.99999	99.99953	99.97611	99.31398	89.15764	20
8	20	87.64721	99.07092	99.95976	99.99894	99.99997	99.99894	99.95976	99.07092	87.64721	20
9	20	86.20789	98.80123	99.93801	99.99796	99.99991	99.99796	99.93801	98.80123	86.20789	20
10	20	84.83536	98.50794	99.91044	99.99648	99.99981	99.99648	99.91044	98.50794	84.83536	20
11	20	83.52559	98.19383	99.87674	99.99436	99.99964	99.99436	99.87674	98.19383	83.52559	20
12	20	82.27484	97.8614	99.83667	99.99148	99.99937	99.99148	99.83667	97.8614	82.27484	20
13	20	81.07962	97.51296	99.79006	99.98772	99.99897	99.98772	99.79006	97.51296	81.07962	20
14	20	79.93668	97.15056	99.73683	99.98295	99.99839	99.98295	99.73683	97.15056	79.93668	20
15	20	78.84297	96.77609	99.67692	99.97704	99.9976	99.97704	99.67692	96.77609	78.84297	20
16	20	77.79568	96.39126	99.61034	99.96989	99.99655	99.96989	99.61034	96.39126	77.79568	20
17	20	76.79216	95.99762	99.53714	99.96137	99.99518	99.96137	99.53714	95.99762	76.79216	20
18	20	75.82994	95.59658	99.45739	99.95137	99.99345	99.95137	99.45739	95.59658	75.82994	20
19	20	74.90672	95.18939	99.3712	99.9398	99.9913	99.9398	99.3712	95.18939	74.90672	20
20	20	74.02034	94.7772	99.2787	99.92656	99.98866	99.92656	99.2787	94.7772	74.02034	20
21	20	73.1688	94.36107	99.18005	99.91157	99.98548	99.91157	99.18005	94.36107	73.1688	20
22	20	72.3502	93.94191	99.07541	99.89473	99.9817	99.89473	99.07541	93.94191	72.3502	20
23	20	71.56278	93.52058	98.96496	99.87599	99.97725	99.87599	98.96496	93.52058	71.56278	20
24	20	70.80489	93.09784	98.84891	99.85526	99.97206	99.85526	98.84891	93.09784	70.80489	20
25	20	70.07499	92.67437	98.72745	99.83248	99.96608	99.83248	98.72745	92.67437	70.07499	20
26	20	69.37161	92.25078	98.60078	99.80761	99.95924	99.80761	98.60078	92.25078	69.37161	20
27	20	68.69341	91.82763	98.46911	99.7806	99.95148	99.7806	98.46911	91.82763	68.69341	20
28	20	68.03909	91.40542	98.33266	99.7514	99.94273	99.7514	98.33266	91.40542	68.03909	20
29	20	67.40747	90.98458	98.19165	99.71998	99.93293	99.71998	98.19165	90.98458	67.40747	20
30	20	66.79741	90.56551	98.04627	99.68631	99.92203	99.68631	98.04627	90.56551	66.79741	20

PROBLEM 4.2 FINITE DIFFERENCE SOLUTION, NON-HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CARTESIAN COORDINATES, TWO DIMENSION



$$\frac{\partial^2 T(x,y)}{\partial x^2}+\frac{T(x,y)}{\partial y^2}=0$$

$$-k\frac{\partial T(x,y)}{\partial x}=h_1\big(T_{1\infty}-T(x,y)\big) \quad x=0, \quad 0\le y\le W$$

$$-k\frac{\partial T(x,y)}{\partial x}=h_2\big(T_{2\infty}-T(x,y)\big) \quad x=L, \quad 0\le y\le W$$

$$-k\frac{\partial T(x,y)}{\partial y}=h_3\big(T_{3\infty}-T(x,y)\big) \quad y=0, \quad 0\le x\le L$$

$$-k\frac{\partial T(x,y)}{\partial y}=h_4\big(T_{4\infty}-T(x,y)\big) \quad y=W, \quad 0\le x\le L$$

$$\frac{T_{m+1,n}-2T_{m,n}+T_{m-1,n}}{\Delta x^2}+\frac{T_{m,n+1}-2T_{m,n}+T_{m,n-1}}{\Delta y^2}=0$$

If $\Delta x=\Delta y$

$$T_{m+1,n}+T_{m-1,n}+T_{m,n+1}+T_{m,n-1}-4T_{m,n}=0$$

Boundary conditions:

m =0 $n\neq 0$

$$-k(\Delta y.1)\frac{T_{m+1,n}-T_{m,n}}{\Delta x}-k\left(\frac{\Delta x}{2}.1\right)\frac{T_{m,n+1}-T_{m,n}}{\Delta y}-k\left(\frac{\Delta x}{2}.1\right)\frac{T_{m,n+1}-T_{m,n}}{\Delta y}=h_1(\Delta y.1)\big(T_{m,n}-T_{1\infty}\big)$$

m=0 n=0

$$-k\left(\frac{\Delta y}{2}.1\right)\frac{T_{m+1,n}-T_{m+1,n}}{\Delta x}-k\left(\frac{\Delta x}{2}.1\right)\frac{T_{m,n+1}-T_{m,n}}{\Delta y}=h_1\left(\frac{\Delta y}{2}.1\right)\left(T_{m,n}-T_{1\infty}\right)+h_3\left(\frac{\Delta x}{2}.1\right)\left(T_{m,n}-T_{3\infty}\right)$$

m=M (x=L) $n\neq 0$

$$-k(\Delta y.1)\frac{T_{m-1,n}-T_{m,n}}{\Delta x}-k\left(\frac{\Delta x}{2}.1\right)\frac{T_{m,n+1}-T_{m,n}}{\Delta y}-k\left(\frac{\Delta x}{2}.1\right)\frac{T_{m,n-1}-T_{m,n}}{\Delta y}=h_2(\Delta y.1)\big(T_{m,n}-T_{2\infty}\big)$$

m=M (x=L) n=0

$$-k\left(\frac{\Delta y}{2}1\right)\frac{T_{m-1,n}-T_{m,n}}{\Delta x}-k\left(\frac{\Delta x}{2}1\right)\frac{T_{m,n+1}-T_{m,n}}{\Delta y}=h\left(\frac{\Delta y}{2}1\right)\left(T_{m,n}-T_{1\infty}\right)+h_3\left(\frac{\Delta x}{2}.1\right)\left(T_{m,n}-T_{3\infty}\right)$$

$m\neq 0$ n=0

$$-k(\Delta x.1)\frac{T_{m,n+1}-T_{m,n}}{\Delta y}-k(\frac{\Delta y}{2}.1)\frac{T_{m+1,n}-T_{m,n}}{\Delta x}-k(\frac{\Delta y}{2}.1)\frac{T_{m-1,n}-T_{m,n}}{\Delta x}=h_3(\Delta y.1)(T_{m,n}-T_{3\infty})$$

$m\neq 0$ n=N (y=W)

$$-k(\Delta x.1)\frac{T_{m,n-1}-T_{m,n}}{\Delta y}-k(\frac{\Delta y}{2}.1)\frac{T_{m+1,n}-T_{m,n}}{\Delta x}-k(\frac{\Delta y}{2}.1)\frac{T_{m-1,n}-T_{m,n}}{\Delta x}=h_4(\Delta y.1)(T_{m,n}-T_{4\infty})$$

m=0 n=N (y=W)

$$-k(\frac{\Delta x}{2}.1)\frac{T_{m,n-1}-T_{m,n}}{\Delta y}-k(\frac{\Delta y}{2}.1)\frac{T_{m+1,n}-T_{m,n}}{\Delta x}=h_4(\frac{\Delta y}{2}.1)(T_{m,n}-T_{4\infty})+h_1(\frac{\Delta x}{2}.1)(T_{m,n}-T_{1\infty})$$

m=M(x=L) n=N (y=W)

$$-k(\frac{\Delta x}{2}.1)\frac{T_{m,n-1}-T_{m,n}}{\Delta y}-k(\frac{\Delta y}{2}.1)\frac{T_{m-1,n}-T_{m,n}}{\Delta x}=h_4(\frac{\Delta y}{2}.1)(T_{m,n}-T_{4\infty})+h_2(\frac{\Delta x}{2}.1)(T_{m,n}-T_{1\infty})$$

Java solution:

```
import java.io.*;


class HT_plane_FD2
{
  public static double [] gauss_seidel(double a[][],double b[],double x[],double lambda)
  {
  //denklem sistemi çözümü b iteratif gauss_seidel metodu metodu
  //yakınsama faktörü lambda ile birlikte verilmiş formül
  int n=a.length;
```

```
int imax=1000;//maksimum iterasyon sayısı
double es=1.0e-10;
double dummy=0;
double ea=1e-1;
double sum=0;
double old=0;
int iter=0;
int sentinel=0;
//System.out.print("iter = "+(iter+1)+" ");
for(int i=0;i<n;i++)
{  dummy=a[i][i];
   for(int j=0;j<n;j++)
     {a[i][j]/=dummy;}
   b[i]/=dummy;
}
for(int i=0;i<n;i++)
{  sum=b[i];
   for(int j=0;j<n;j++)
     {if(i!=j)sum-=a[i][j]*x[j];}
   x[i]=sum;
   //System.out.print(x[i]+ "  ");
}
 //System.out.println();
iter++;
do
{
sentinel=1;
//System.out.print("iter = "+(iter+1)+" ");
for(int i=0;i<n;i++)
   {
              old=x[i];
              sum=b[i];
              for(int j=0;j<n;j++)
                 {if(i!=j) sum-=a[i][j]*x[j];}
                 x[i]=lambda*sum+(1.0-lambda)*old;
                 if(sentinel==1 && x[i]!=0.0 ) {ea=Math.abs((x[i]-old)/x[i])*100.0;}
                 if(ea<es) sentinel=0;
               //System.out.print(x[i]+ "  ");
   }
//System.out.println();
iter++;
}while(sentinel==1 && iter<imax);
if(iter>=imax) System.out.println("Maksimum iterasyon sayısı aşıldı sonuç geçerli olmayabilir iter = "+iter);
return x;
}

public static double[] pivotlugauss(double a[][],double b[])
{ //kısmi pivotlu gauss eleme yöntemi
int n=b.length;
double x[]=new double[n];
double carpan=0;
double toplam=0;
double buyuk;
double dummy=0;
//gauss eleme
int i,j,k,p,ii,jj;
//System.out.println("Orijinal Matris :\n"+Matrix.toString(a));
for(k=0;k<(n-1);k++)
{ //pivotlama
            p=k;
            buyuk=Math.abs(a[k][k]);
            for(ii=k+1;ii<n;ii++)
            {  dummy=Math.abs(a[ii][k]);
               if(dummy > buyuk) {buyuk=dummy;p=ii;}
            }
            if(p!=k)
            { for(jj=k;jj<n;jj++)
              { dummy=a[p][jj];
                a[p][jj]=a[k][jj];
                a[k][jj]=dummy;
              }
              dummy=b[p];
              b[p]=b[k];
              b[k]=dummy;
            }
              //System.out.println("pivotlu Matris : k="+k+"\n"+Matrix.toString(a));
            //gauss elemeyi çözme
```

```
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
          b[i]   =b[i]   -carpan*b[k];
               }
  }
  //geriye doğru yerine koyma
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }

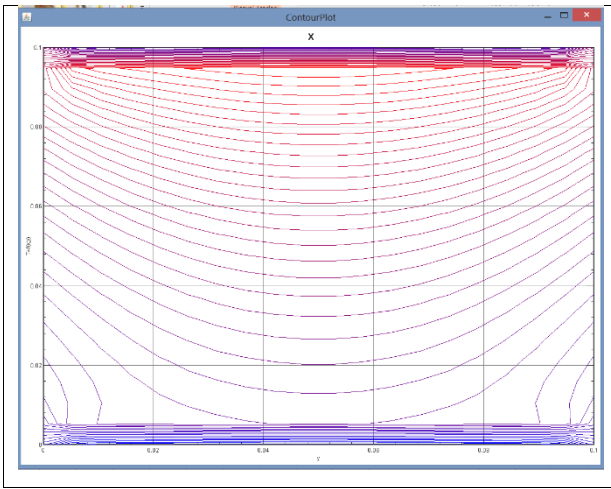
  public static double [][] HT_planecoz( double L,double W,double dx, double Tx0,double Txn,double Ty0,double Tyn,double hx0,double
hxn,double hy0,double hyn,double kc)
  {
  //  Assume dx=dy=const;
  //  Temperature of 4 sides are known
  int nx=(int)(L/dx+0.01);
  int ny=(int)(W/dx+0.01);
  System.out.println("nx="+nx+"ny="+ny);

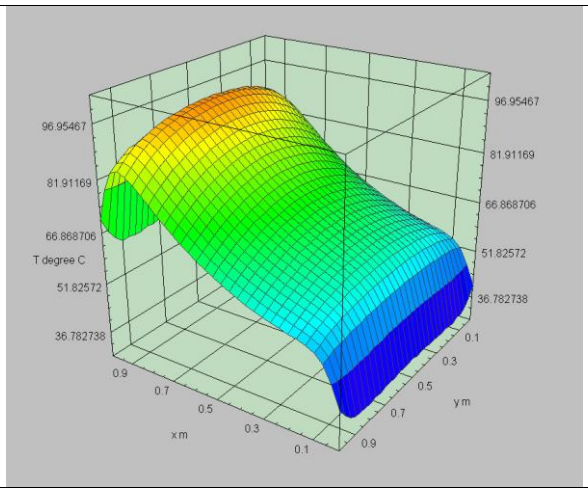
  int mx=nx;
  int my=ny;
  int mt=mx*my;
  double A[][]=new double[mt][mt];
  double b[]=new double[mt];
  double x[]=new double[mt];
  double x1,x2,y1,y2;
  // boundary conditions
  int i=0,j=0;
  for(i=0;i<my;i++)
  {
    for(j=0;j<mx;j++)
    {
                x1=dx*j;
                x2=(L-x1);
                y1=dx*i;
                y2=(W-y1);
                int k=i*my+j;
                x[k]=((x1*Tx0+x2*Txn)/L+(y1*Ty0+y2*Tyn)/W)/2.0;
               int kp1=(i+1)*my+j;
               int km1=(i-1)*my+j;
                // 1
    if((i==0)&&(j==0)) {A[k][k]+=-(hy0*dx/kc+hx0*dx/kc+2.0);A[kp1][k]+=1.0;A[k][k+1]+=1.0;b[k]+=-hy0*dx/kc*Tx0-
hx0*dx/kc*Ty0;}
    // 4
    else if((i==0)&&(j==(mx-1))) {A[k][k]+=-(hxn*dx/kc+hy0*dx/kc+2.0);A[kp1][k]+=1.0;A[k][k-1]+=1.0;b[k]+=-hxn*dx/kc*Txn-
hy0*dx/kc*Ty0;}
    // 2
    else if((i==(my-1))&&(j==0)) {A[k][k]+=-(hx0*dx/kc+hyn*dx/kc+2.0);A[km1][k]+=1.0;A[k][k+1]+=1.0;b[k]+=-hyn*dx/kc*Tyn-
hx0*dx/kc*Tx0;}
    // 3
    else if((i==(my-1))&&(j==(mx-1))) {A[k][k]+=-(hxn*dx/kc+hyn*dx/kc+2.0);A[km1][k]+=1.0;A[k][k-1]+=1.0;b[k]+=-hyn*dx/kc*Tyn-
hxn*dx/kc*Txn;}
    // 7
    else if(j==0) {A[k][k]+=-(2.0*hx0*dx/kc+4.0);A[kp1][k]+=1.0;A[km1][k]+=1.0;A[k][k+1]+=2.0;b[k]+=-2.0*hx0*dx/kc*Tx0;}
    // 5
    else if(i==0) {A[k][k]+=-(2.0*hy0*dx/kc+4.0);A[kp1][k]+=2.0;A[k][k-1]+=1.0;A[k][k+1]+=1.0;b[k]+=-2.0*hy0*dx/kc*Ty0;}
    // 8
    else if(j==(mx-1)) {A[k][k]+=-(2.0*hxn*dx/kc+4.0);A[kp1][k]+=1.0;A[km1][k]+=1.0;A[k][k-1]+=2.0;b[k]+=-2.0*hxn*dx/kc*Txn;}
    // 6
    else if(i==(my-1)) {A[k][k]+=-(2.0*hyn*dx/kc+4.0);A[k][k-1]+=1.0;A[k][k+1]+=1.0;A[km1][k]+=2.0;b[k]+=-2.0*hyn*dx/kc*Tyn;}
    // iç bölge
    else {A[k][k]+=-4.0;A[km1][k]+=1.0;A[kp1][k]+=1.0;A[k][k-1]+=1.0;A[k][k+1]+=1.0;}
    }
  }
  //Text.print(A);
  double T[][]=new double[ny][nx];
  //double a[]=conjugate_gradient1.conjugate_gradient(x,A,b);
```

```
//double a[]=gauss_seidel(A,b,x,1.2);
double a[]=pivotlugauss(A,b);
for(i=0;i<ny;i++)
{ for(j=0;j<nx;j++)
 {
  {int k=i*my+j;
   T[i][j]=a[k];
  }
 }
}

return T;
}

public static void main(String args[])
{  double L=1;
   double W=1;
   double dx=0.05;
   double Tx0=20.0;
   double Txn=20.0;
   double Ty0=20.0;
   double Tyn=100.0;
   double hx0=50;  //W/m2K
   double hxn=50;  //W/m2K
   double hy0=50;  //W/m2K
   double hyn=50;  //W/m2K
   double kc=50;    //W/mK
            double a[][]=HT_planecoz(L,W,dx, Tx0,Txn,Ty0,Tyn,hx0,hxn,hy0,hyn,kc);
            Text.print(a);
     double bb[][]=surfacePlot.dataformat(0.0,1.0,0.0,1.0,a);
     surfacePlot sp=new surfacePlot(bb,"x m","y m","T degree C");
     sp.plot();
      ContourPlot cp1=new ContourPlot(a,0.0,1.0,0.0,1.0,50, true,"T derece", "x","y");
      cp1.plot();
 }
}
```





0.00	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95	1.0
0.05	40.9	31.8	29.7	29.3	29.3	29.4	29.5	29.6	29.7	29.8	29.8	29.7	29.6	29.5	29.4	29.3	29.3	29.7	31.8	40.9
0.10	52.0	57.9	58.7	59.0	59.3	59.7	60.0	60.3	60.5	60.6	60.6	60.5	60.3	60.0	59.7	59.3	59.0	58.7	57.9	52.0
0.15	54.6	57.2	58.6	59.5	60.1	60.6	61.0	61.4	61.6	61.7	61.7	61.6	61.4	61.0	60.6	60.1	59.5	58.6	57.2	54.6
0.20	55.7	57.5	59.0	60.1	61.0	61.6	62.2	62.6	62.8	63.0	63.0	62.8	62.6	62.2	61.6	61.0	60.1	59.0	57.5	55.7
0.25	56.5	58.3	59.8	61.0	62.0	62.8	63.4	63.9	64.2	64.3	64.3	64.2	63.9	63.4	62.8	62.0	61.0	59.8	58.3	56.5
0.30	57.5	59.3	60.9	62.2	63.3	64.2	64.8	65.4	65.7	65.9	65.9	65.7	65.4	64.8	64.2	63.3	62.2	60.9	59.3	57.5
0.35	58.7	60.6	62.2	63.6	64.7	65.7	66.4	67.0	67.4	67.5	67.5	67.4	67.0	66.4	65.7	64.7	63.6	62.2	60.6	58.7
0.40	60.1	62.0	63.7	65.2	66.4	67.4	68.2	68.8	69.2	69.4	69.4	69.2	68.8	68.2	67.4	66.4	65.2	63.7	62.0	60.1
0.45	61.8	63.7	65.5	67.0	68.3	69.4	70.2	70.9	71.3	71.5	71.5	71.3	70.9	70.2	69.4	68.3	67.0	65.5	63.7	61.8
0.50	63.6	65.6	67.5	69.1	70.4	71.6	72.5	73.1	73.5	73.8	73.8	73.5	73.1	72.5	71.6	70.4	69.1	67.5	65.6	63.6
0.55	65.6	67.8	69.7	71.4	72.8	74.0	74.9	75.6	76.0	76.3	76.3	76.0	75.6	74.9	74.0	72.8	71.4	69.7	67.8	65.6
0.60	67.9	70.2	72.2	73.9	75.4	76.6	77.6	78.3	78.8	79.0	79.0	78.8	78.3	77.6	76.6	75.4	73.9	72.2	70.2	67.9
0.65	70.5	72.8	74.9	76.8	78.3	79.5	80.5	81.2	81.7	81.9	81.9	81.7	81.2	80.5	79.5	78.3	76.8	74.9	72.8	70.5
0.70	73.3	75.8	78.0	79.9	81.4	82.7	83.7	84.4	84.9	85.1	85.1	84.9	84.4	83.7	82.7	81.4	79.9	78.0	75.8	73.3
0.75	76.3	79.0	81.3	83.3	84.9	86.2	87.1	87.9	88.3	88.6	88.6	88.3	87.9	87.1	86.2	84.9	83.3	81.3	79.0	76.3
0.80	79.7	82.6	85.1	87.1	88.7	89.9	90.8	91.5	92.0	92.2	92.2	92.0	91.5	90.8	89.9	88.7	87.1	85.1	82.6	79.7
0.85	83.3	86.6	89.3	91.3	92.8	93.9	94.8	95.5	95.9	96.1	96.1	95.9	95.5	94.8	93.9	92.8	91.3	89.3	86.6	83.3
0.90	86.5	91.3	94.2	96.0	97.2	98.2	99.0	99.6	100.	100.	100.	100.	99.6	99.0	98.2	97.2	96.0	94.2	91.3	86.5
0.95	86.7	98.0	100.	101.	102.	102.	103.	103.	104.	104.	104.	104.	103.	103.	102.	102.	101.	100.	98.0	86.7

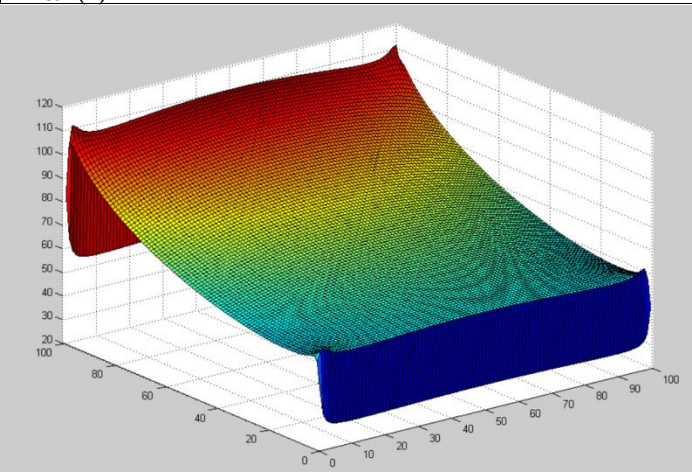
1.00	71.2	56.8	53.7	53.2	53.4	53.7	54.0	54.2	54.4	54.5	54.5	54.4	54.2	54.0	53.7	53.4	53.2	53.7	56.8	71.2

Matlab solution:

```
function [T] = HT_plane_xy(L,W,dx,Tx0,Txn,Ty0,Tyn,hx0,hxn,hy0,hyn,kc)
 nx=L/dx;
 ny=W/dx;
 mx=nx;
 my=ny;
 mt=mx*my;
 A=zeros(mt,mt);
 B=zeros(mt);
 T=zeros(mt,mt);
 % boundary conditions
 for i=1:my
   for j=1:mx
   k=(i-1)*my+j;
   kp1=i*my+j;
   km1=(i-2)*my+j;
   % 1
   if( i==1 && j==1 )
      A(k,k)=A(k,k)-(hy0*dx/kc+hx0*dx/kc+2.0);
      A(kp1,k)=A(kp1,k)+1.0;
      A(k,k+1)=A(k,k+1)+1.0;
      B(k)=B(k)-hy0*dx/kc*Tx0-hx0*dx/kc*Ty0;
   % 4
   elseif( i==1&& j==mx )
      A(k,k)=A(k,k)-(hxn*dx/kc+hy0*dx/kc+2.0);
      A(kp1,k)=A(kp1,k)+1.0;
      A(k,k-1)=A(k,k-1)+1.0;
      B(k)=B(k)-hxn*dx/kc*Txn-hy0*dx/kc*Ty0;
   % 2
   elseif( i==my && j==1)
      A(k,k)=A(k,k)-(hx0*dx/kc+hyn*dx/kc+2.0);
      A(km1,k)=A(km1,k)+1.0;
      A(k,k+1)=A(k,k+1)+1.0;
      B(k)=B(k)-hyn*dx/kc*Tyn-hx0*dx/kc*Tx0;
   % 3
   elseif( i==my && j==mx)
      A(k,k)=A(k,k)-(hxn*dx/kc+hyn*dx/kc+2.0);
      A(km1,k)=A(km1,k)+1.0;
      A(k,k-1)=A(k,k-1)+1.0;
      B(k)=B(k)-hyn*dx/kc*Tyn-hxn*dx/kc*Txn;
   % 7
   elseif(j==1)
      A(k,k)=A(k,k)-(2.0*hx0*dx/kc+4.0);
      A(kp1,k)=A(kp1,k)+1.0;
      A(km1,k)=A(km1,k)+1.0;
      A(k,k+1)=A(k,k+1)+2.0;
      B(k)=B(k)-2.0*hx0*dx/kc*Tx0;
   % 5
   elseif(i==1)
      A(k,k)=A(k,k)-(2.0*hy0*dx/kc+4.0);
      A(kp1,k)=A(kp1,k)+2.0;
      A(k,k-1)=A(k,k-1)+1.0;
      A(k,k+1)=A(k,k+1)+1.0;
      B(k)=B(k)-2.0*hy0*dx/kc*Ty0;
   % 8
   elseif(j==mx)
      A(k,k)=A(k,k)-(2.0*hxn*dx/kc+4.0);
      A(kp1,k)=A(kp1,k)+1.0;
      A(km1,k)=A(km1,k)+1.0;
      A(k,k-1)=A(k,k-1)+2.0;
      B(k)=B(k)-2.0*hxn*dx/kc*Txn;
   % 6
   elseif(i==my) A(k,k)=A(k,k)-(2.0*hyn*dx/kc+4.0);
      A(k,k-1)=A(k,k-1)+1.0;
      A(k,k+1)=A(k,k+1)+1.0;
      A(km1,k)=A(km1,k)+2.0;
      B(k)=B(k)-2.0*hyn*dx/kc*Tyn;
   % internal region
   else
     A(k,k)=A(k,k)-4.0;
     A(km1,k)=A(km1,k)+1.0;
     A(kp1,k)=A(kp1,k)+1.0;
     A(k,k-1)=A(k,k-1)+1.0;
     A(k,k+1)=A(k,k+1)+1.0;
```

```
    end
    end
  end
   T=zeros(ny,nx);
  B=B(:,1);
  aa=linsolve(A,B);
 for i=1:ny
 for j=1:nx
   k=(i-1)*my+j;
   T(i,j)=aa(k);
 end
end
end
```

```
>> [T]=HT_plane_xy(1.0,1.0,0.01,20.0,20.0,20.0,100.0,200.0,200.0,200.0,200.0,200.0);
>> surf(T)
```



Python version:

```
from math import *
from f_xj import *;
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt

def gauss(a,b):
    #Gauss elimination with pivoting
  b1=b
  n=len(a)
  x=[0 for i in range(n)]
  for k in range(n):
    p=k
    buyuk=abs(a[k][k])
    for ii in range(k+1,n):
      dummy=abs(a[ii][k])
      if dummy>buyuk:
        buyuk=dummy
        p=ii
    if p!=k:
      for jj in range(k,n):
        dummy=a[p][jj]
        a[p][jj]=a[k][jj]
        a[k][jj]=dummy
      dummy=b1[p]
      b1[p]=b1[k]
      b1[k]=dummy
    for i in range(k+1,n):
      carpan=a[i][k]/a[k][k]
      a[i][k]=0
      for j in range(k+1,n):
        a[i][j]=a[i][j]-carpan*a[k][j]
      b1[i]   =b1[i]-carpan*b[k]
  x[n-1]=b1[n-1]/a[n-1][n-1]
  for i in range(n-1,-1,-1):
    toplam=0
    for j in range(i+1,n):
```

```
      toplam=toplam+a[i][j]*x[j]
    x[i]=(b[i]-toplam)/a[i][i]
  return x

def HT_planecoz(L,W,dx,Tx0,Txn,Ty0,Tyn,hx0,hxn,hy0,hyn,kc):
    #Assume dx=dy=const
    # 4 sides convective heat transfer hi,Ti_inf
    nx=int(L/dx+0.01)
    ny=int(W/dx+0.01)
    mx=nx
    my=ny
    mt=mx*my
    A=[[0.0 for j in range(mt)] for i in range(mt)]
    T=[[0.0 for j in range(ny+1)] for i in range(nx+1)]
    b=[0.0 for j in range(mt)]
    x=[0.0 for j in range(mt)]
    T=[[0.0 for j in range(ny+1)] for i in range(nx+1)]
    x1=0.0
    x2=0.0
    y1=0.0
    y2=0.0
    #boundary conditions
    i=0
    j=0
    for i in range(my):
        for j in range(mx):
            x1=dx*j
            x2=L-x1
            y1=dx*i
            y2=(W-y1)
            k=i*my+j
            x[k]=((x1*Tx0+x2*Txn)/L+(y1*Ty0+y2*Tyn)/W)/2.0
            kp1=(i+1)*my+j
            km1=(i-1)*my+j
            if ((i==0) and (j==0)):
                A[k][k]+=-(hy0*dx/kc+hx0*dx/kc+2.0)
                A[kp1][k]+=1.0
                A[k][k+1]+=1.0
                b[k]+=-hy0*dx/kc*Tx0-hx0*dx/kc*Ty0
            #4
            elif ((i==0) and (j==(mx-1))):
                A[k][k]+=-(hxn*dx/kc+hy0*dx/kc+2.0)
                A[kp1][k]+=1.0
                A[k][k-1]+=1.0
                b[k]+=-hxn*dx/kc*Txn-hy0*dx/kc*Ty0;
            #2
            elif ((i==(my-1)) and (j==0)):
                A[k][k]+=-(hx0*dx/kc+hyn*dx/kc+2.0)
                A[km1][k]+=1.0;A[k][k+1]+=1.0
                b[k]+=-hyn*dx/kc*Tyn-hx0*dx/kc*Tx0
            # 3
            elif ((i==(my-1)) and (j==(mx-1))):
                A[k][k]+=-(hxn*dx/kc+hyn*dx/kc+2.0)
                A[km1][k]+=1.0;A[k][k-1]+=1.0
                b[k]+=-hyn*dx/kc*Tyn-hxn*dx/kc*Txn
            # 7
            elif (j==0):
                A[k][k]+=-(2.0*hx0*dx/kc+4.0)
                A[kp1][k]+=1.0;A[km1][k]+=1.0
                A[k][k+1]+=2.0
                b[k]+=-2.0*hx0*dx/kc*Tx0
            # 5
            elif (i==0):
                A[k][k]+=-(2.0*hy0*dx/kc+4.0)
                A[kp1][k]+=2.0
                A[k][k-1]+=1.0
                A[k][k+1]+=1.0
                b[k]+=-2.0*hy0*dx/kc*Ty0
            # 8
            elif (j==(mx-1)):
                A[k][k]+=-(2.0*hxn*dx/kc+4.0)
                A[kp1][k]+=1.0
                A[km1][k]+=1.0
                A[k][k-1]+=2.0
                b[k]+=-2.0*hxn*dx/kc*Txn
            # 6
            elif (i==(my-1)):
```

```
            A[k][k]+=-(2.0*hyn*dx/kc+4.0)
            A[k][k-1]+=1.0;A[k][k+1]+=1.0
            A[km1][k]+=2.0
            b[k]+=-2.0*hyn*dx/kc*Tyn
        # internal region
        else:
            A[k][k]+=-4.0
            A[km1][k]+=1.0
            A[kp1][k]+=1.0
            A[k][k-1]+=1.0
            A[k][k+1]+=1.0
    a=gauss(A,b)
    for i in range(ny):
        for j in range(nx):
            k=i*my+j
            T[i][j]=a[k];
    return T

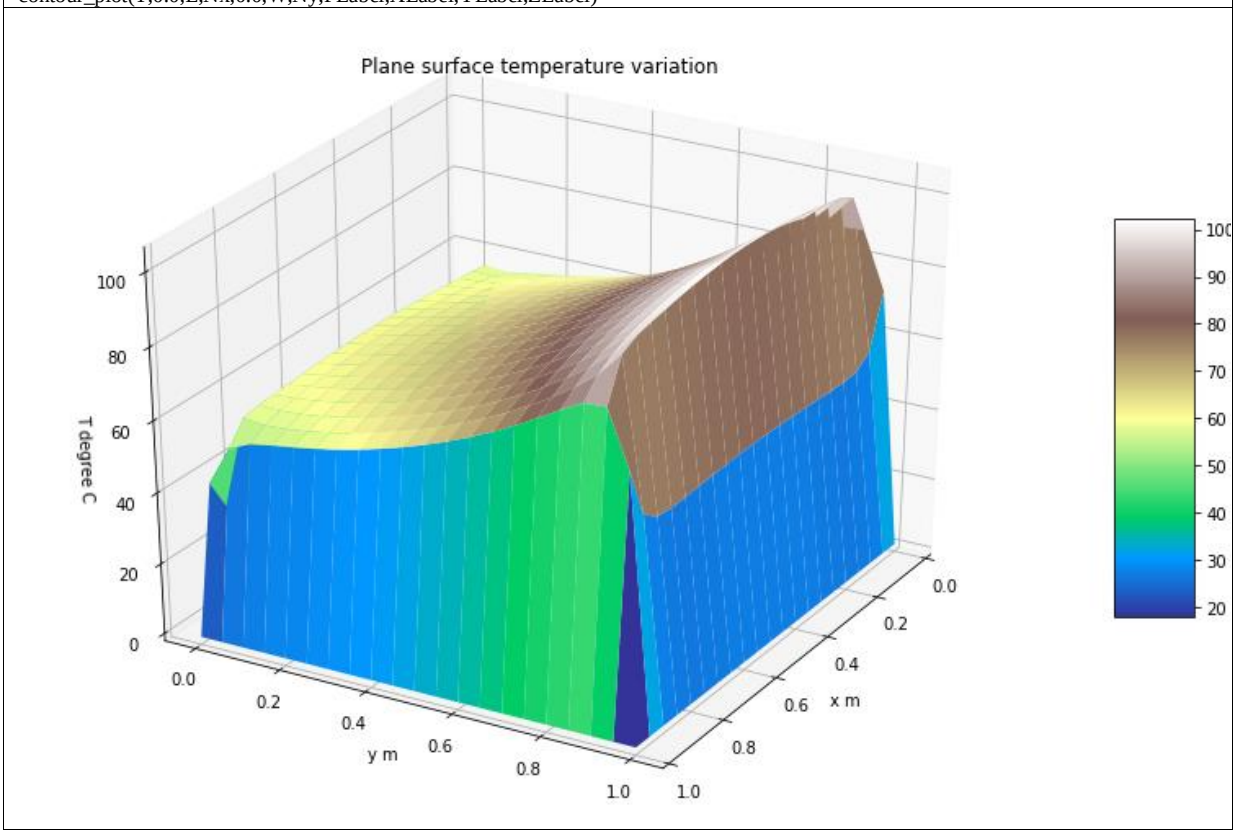

def plot(A,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLAbel,YLable,ZLabel):
    #f=f2()
    x=[0.0 for j in range(Nx)]
    y=[0.0 for j in range(Ny)]
    z=[[0.0 for j in range(Ny)] for i in range(Nx)]
    dx=(xmax-xmin)/(Nx-1)
    dy=(ymax-ymin)/(Ny-1)
    for i in range(Nx):
        for j in range(Ny):
            x[i]=xmin+dx*i
            y[j] = ymin+dy*j
            z[i][j]=A[i][j]
    xi=np.array(x)
    yi=np.array(y)
    xi, yi = np.meshgrid(xi, yi)
    zi=np.array(z)
    print(zi)
    fig = plt.figure(figsize =(14, 9))
    ax = plt.axes(projection ='3d')
    # Creating color map
    my_cmap = plt.get_cmap('terrain')
    # Creating plot
    surf = ax.plot_surface(xi, yi, zi,
                           cmap = my_cmap,
                           edgecolor ='none')
    fig.colorbar(surf, ax = ax,
                 shrink = 0.5,
                 aspect = 5)

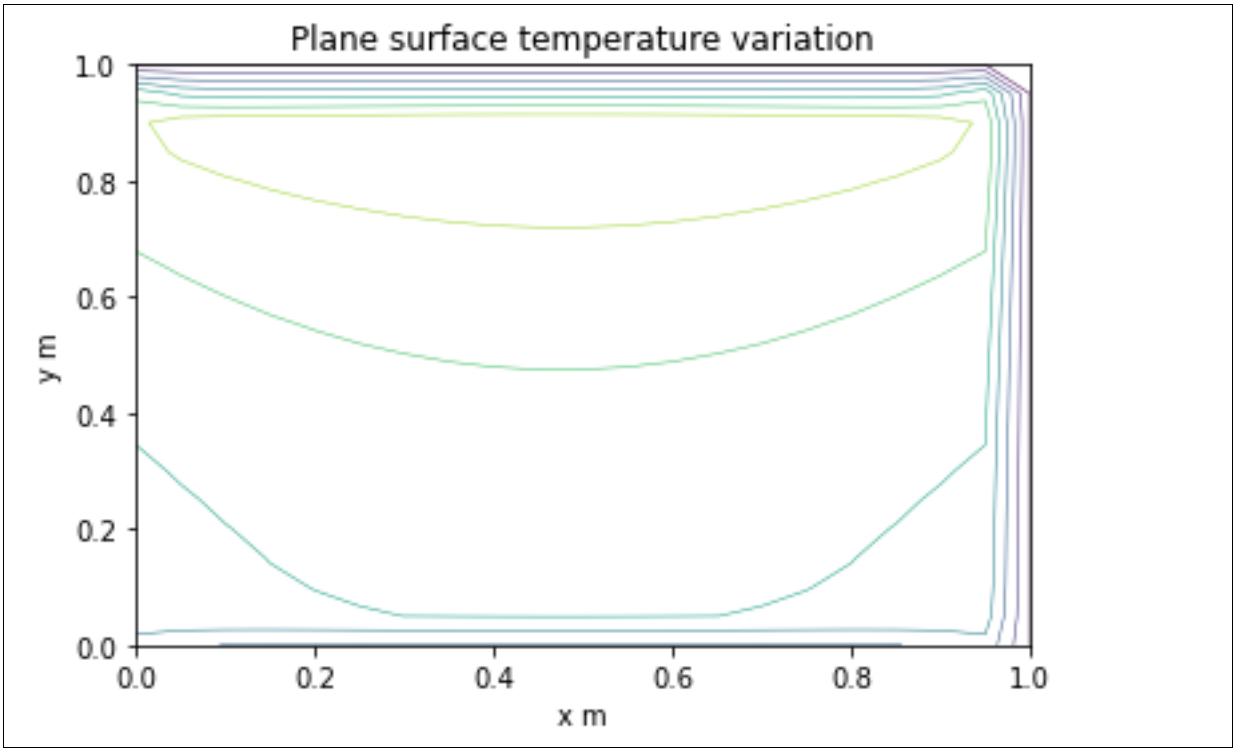
    ax.set_title(PLabel)
    ax.set_xlabel(XLabel)
    ax.set_xlim(xmin, xmax)
    ax.set_ylabel(YLabel)
    ax.set_xlim(ymin, ymax)
    ax.set_zlabel(ZLabel)
    ax.set_zlim(np.min(zi), np.max(zi))
    ax.view_init(30, 30)
    plt.show()

# Creating dataset
def contour_plot(A,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLable,ZLabel):
    #f=f2()
    x=[0.0 for j in range(Nx)]
    y=[0.0 for j in range(Ny)]
    z=[[0.0 for j in range(Ny)] for i in range(Nx)]
    dx=(xmax-xmin)/(Nx-1)
    dy=(ymax-ymin)/(Ny-1)
    for i in range(Nx):
        for j in range(Ny):
            x[i]=xmin+dx*i
            y[j] = ymin+dy*j
            z[i][j]=A[i][j]
    xi=np.array(x)
    yi=np.array(y)
    xi, yi = np.meshgrid(xi, yi)
    zi=np.array(z)
    fig, ax = plt.subplots(1, 1)
    # plots contour lines
```

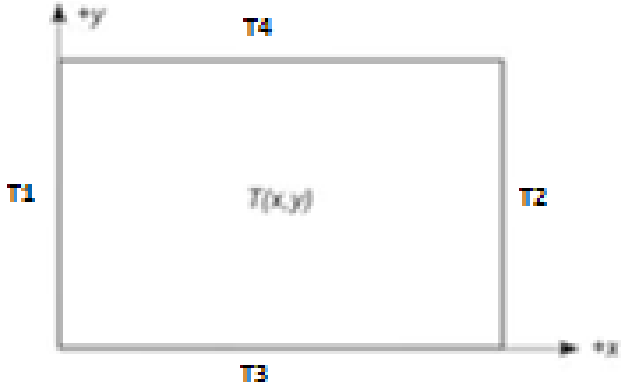
```
    ax.contour(xi, yi, zi,linewidths=0.5)
    ax.set_title(PLabel)
    ax.set_xlabel(XLabel)
    ax.set_xlim(xmin, xmax)
    ax.set_ylabel(YLabel)
    ax.set_xlim(ymin, ymax)
    plt.show()


W=1.0
L=1.0
dx=0.05
Tx0=20.0
Txn=20.0
Ty0=20.0
Tyn=100.0
hx0=50;  #W/m2K
hxn=50;  #W/m2K
hy0=50;  #W/m2K
hyn=50;  #W/m2K
kc=50;   #W/mK
T=HT_planecoz(L,W,dx,Tx0,Txn,Ty0,Tyn,hx0,hxn,hy0,hyn,kc)
print(T)
PLabel="Plane surface temperature variation"
XLabel="x m"
YLabel="y m"
ZLabel="T degree C"
Nx=len(T)
Ny=len(T[0])
plot(T,0.0,L,Nx,0.0,W,Ny,PLabel,XLabel,YLabel,ZLabel)
contour_plot(T,0.0,L,Nx,0.0,W,Ny,PLabel,XLabel,YLabel,ZLabel)
```





PROBLEM 4.3 FINITE DIFFERENCE SOLUTION, NON-HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CARTESIAN COORDINATES, TWO DIMENSION



$$\frac{\partial^2 T(x,y)}{\partial x^2}+\frac{\partial^2 T(x,y)}{\partial y^2}=0$$

$$T(x,y)=T_1 \quad x=0,\ \ 0\le y\le W$$
$$T(x,y)=T_2 \quad x=L,\ \ 0\le y\le W$$
$$T(x,y)=T_3 \quad y=0,\ \ 0\le x\le L$$
$$T(x,y)=T_4 \quad y=W,\ \ 0\le x\le L$$

$$\frac{T_{m+1,n}-2T_{m,n}+T_{m-1,n}}{\Delta x^2}+\frac{T_{m,n+1}-2T_{m,n}+T_{m,n-1}}{\Delta y^2}=0$$

$$\frac{T_{m+1,n}-2T_{m,n}+T_{m-1,n}}{\Delta x^2}+\frac{T_{m,n+1}-2T_{m,n}+T_{m,n-1}}{\Delta y^2}=0$$

If $\Delta x=\Delta y$

$$T_{m+1,n}+T_{m-1,n}+T_{m,n+1}+T_{m,n-1}-4T_{m,n}=0$$

Java program:

```
class HT_plane_FD_T
{
  public static double [] gauss_seidel(double a[][],double b[],double x[],double lambda)
  {
  //denklem sistemi çözümü b iteratif gauss_seidel metodu metodu
  //yakınsama faktörü lambda ile birlikte verilmiş formül
  int n=a.length;
  int imax=1000;//maksimum iterasyon sayısı
  double es=1.0e-10;
  double dummy=0;
  double ea=1e-1;
  double sum=0;
  double old=0;
  int iter=0;
  int sentinel=0;
  //System.out.print("iter = "+(iter+1)+" ");
  for(int i=0;i<n;i++)
  {  dummy=a[i][i];
     for(int j=0;j<n;j++)
       {a[i][j]/=dummy;}
     b[i]/=dummy;
  }
  for(int i=0;i<n;i++)
  {  sum=b[i];
     for(int j=0;j<n;j++)
       {if(i!=j)sum-=a[i][j]*x[j];}
     x[i]=sum;
     //System.out.print(x[i]+ "   ");
  }
   //System.out.println();
  iter++;
  do
  {
  sentinel=1;
  //System.out.print("iter = "+(iter+1)+" ");
  for(int i=0;i<n;i++)
     {
                old=x[i];
                sum=b[i];
                for(int j=0;j<n;j++)
                   {if(i!=j) sum-=a[i][j]*x[j];}
                   x[i]=lambda*sum+(1.0-lambda)*old;
                   if(sentinel==1 && x[i]!=0.0 ) {ea=Math.abs((x[i]-old)/x[i])*100.0;}
                   if(ea<es) sentinel=0;
                 //System.out.print(x[i]+ "   ");
     }
  //System.out.println();
  iter++;
  }while(sentinel==1 && iter<imax);
  if(iter>=imax) System.out.println("Maksimum iterasyon sayısı aşıldı sonuç geçerli olmayabilir iter = "+iter);
  return x;
  }

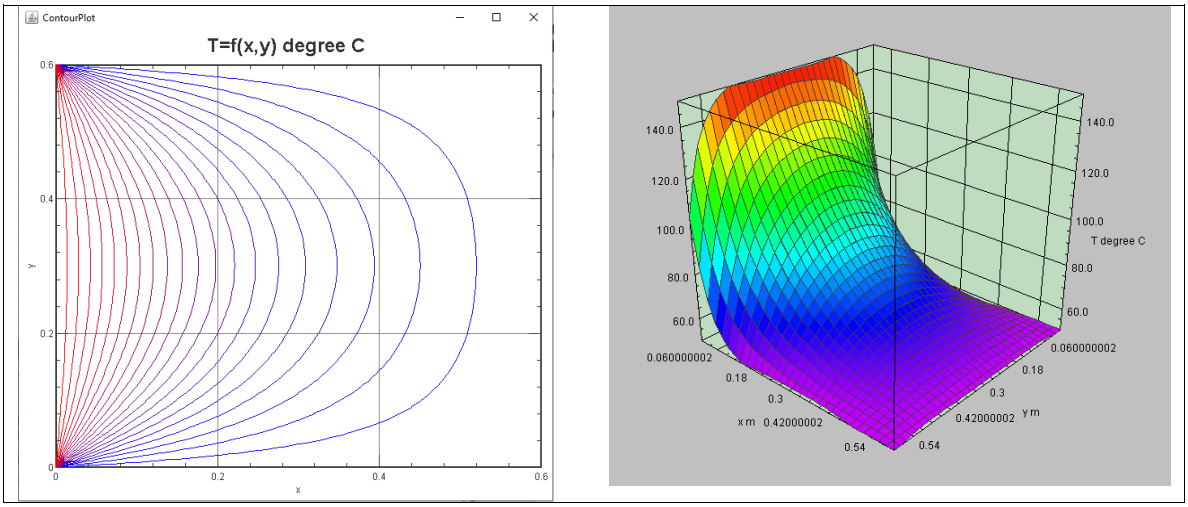
  public static double[] pivotlugauss(double a[][],double b[])
  { //kısmi pivotlu gauss eleme yöntemi
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss eleme
  int i,j,k,p,ii,jj;
  //System.out.println("Orijinal Matris :\n"+Matrix.toString(a));
  for(k=0;k<(n-1);k++)
  { //pivotlama
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                   if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
```

```
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
               }
                //System.out.println("pivotlu Matris : k="+k+"\n"+Matrix.toString(a));
               //gauss elemeyi çözme
  for(i=k+1;i<n;i++)
  {  carpan=a[i][k]/a[k][k];
     a[i][k]=0;
     for(j=k+1;j<n;j++)
     {   a[i][j]-=carpan*a[k][j]; }
         b[i]   =b[i]   -carpan*b[k];
               }
}
//back substitution
x[n-1]=b[n-1]/a[n-1][n-1];
for(i=n-2;i>=0;i--)
{
  toplam=0;
  for(j=i+1;j<n;j++)
  {  toplam+=a[i][j]*x[j];}
  x[i]=(b[i]-toplam)/a[i][i];
}
return x;
}

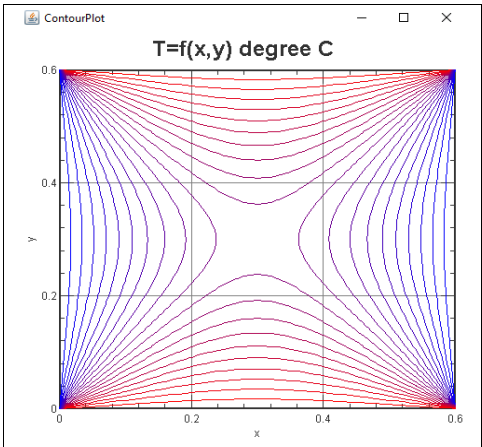

public static double [][] HT_planecoz( double L,double W,double dx, double Tx0,double Txn,double Ty0,double Tyn)
{
int nx=(int)(L/dx+0.01);
int ny=(int)(W/dx+0.01);
int mx=(nx-1);
int my=(ny-1);
int mt=mx*my;
double A[][]=new double[mt][mt];
double b[]=new double[mt];
double x[]=new double[mt];
double x1,x2,y1,y2;
// boundary conditions
int i=0,j=0;
for(i=0;i<my;i++)
{
   for(j=0;j<mx;j++)
   {
               x1=dx*j;
               x2=(L-x1);
               y1=dx*i;
               y2=(W-y1);
               int k=i*my+j;
               x[k]=((x1*Tx0+x2*Txn)/L+(y1*Ty0+y2*Tyn)/W)/2.0;
               int kp1=(i+1)*my+j;
               int km1=(i-1)*my+j;
               if(j==0)          {b[k]+=-Tx0;}
   if(j== (mx-1))     {b[k]+=-Txn;}
   if(i==0)          {b[k]+=-Ty0;}
   if(i==(my-1))  {b[k]+=-Tyn;}
   A[k][k]+=-4.0;
   if((i==0)&&(j==0)) {A[kp1][k]+=1.0;A[k][k+1]+=1.0;}
   else if((i==0)&&(j==(mx-1))) {A[kp1][k]+=1.0;A[k][k-1]+=1.0;}
   else if((i==(my-1))&&(j==0)) {A[km1][k]+=1.0;A[k][k+1]+=1.0;}
   else if((i==(my-1))&&(j==(mx-1))) {A[km1][k]+=1.0;A[k][k-1]+=1.0;}
   else if(j==0) {A[kp1][k]+=1.0;A[km1][k]+=1.0;A[k][k+1]+=1.0;}
   else if(i==0) {A[k][k+1]+=1.0;A[k][k-1]+=1.0;A[kp1][k]+=1.0;}
   else if(j==(mx-1)) {A[kp1][k]+=1.0;A[km1][k]+=1.0;A[k][k-1]+=1.0;}
   else if(i==(my-1)) {A[km1][k]+=1.0;A[k][k-1]+=1.0;A[k][k+1]+=1.0;}
   else {A[km1][k]+=1.0;A[kp1][k]+=1.0;A[k][k-1]+=1.0;A[k][k+1]+=1.0;}
   }
}
Text.print(A,"A");
Text.print(b,"B");
double T[][]=new double[ny+1][nx+1];
double a[]=conjugate_gradient1.conjugate_gradient(A,b);
for(i=0;i<=ny;i++)
{ for(j=0;j<=nx;j++)
{if(j==0) {T[i][j]=Tx0;}
  else if(i==0) {T[i][j]=Ty0;}
```

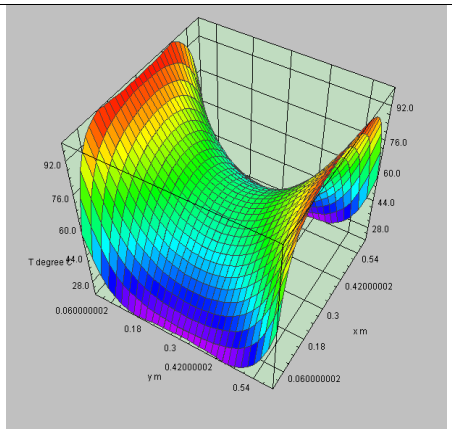
```
    else if(j==nx) {T[i][j]=Txn;}
    else if(i==ny) {T[i][j]=Tyn;}
    else
    {int k=(i-1)*my+(j-1);
     T[i][j]=a[k];
    }
    }

  }
   return T;
}
  public static void main(String args[]) //throws IOException,RemoteException, VisADException
  {  double L=0.6;
     double W=0.6;
     double dx=0.01;
     double Tx0=150.0;
     double Txn=50.0;
     double Ty0=50.0;
     double Tyn=50.0;
               double a[][]=HT_planecoz(L,W,dx,Tx0,Txn,Ty0,Tyn);
    Text.print(a);
               ContourPlot pp=new ContourPlot(a,0.0,L,0.0,W,20, false,"T=f(x,y) degree C","x","y");
               pp.plot();
      double bb[][]=surfacePlot.dataformat(0.0,L,0.0,W,a);
      surfacePlot sp=new surfacePlot(bb,"y m","y m","T degree C");
      sp.plot();
  }
}
```

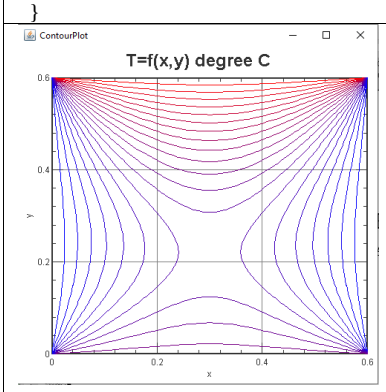


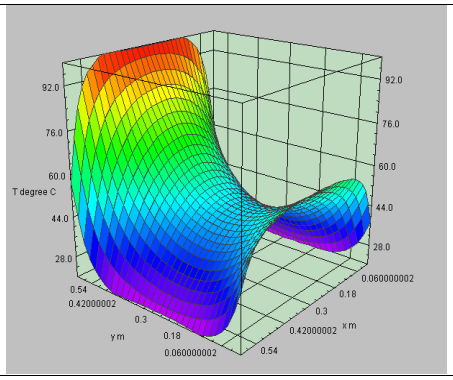

```
  public static void main(String args[]) //throws IOException,RemoteException, VisADException
  {  double L=0.6;
     double W=0.6;
     double dx=0.01;
     double Tx0=20.0;
     double Txn=20.0;
     double Ty0=100.0;
     double Tyn=100.0;
               double a[][]=HT_planecoz(L,W,dx,Tx0,Txn,Ty0,Tyn);
    Text.print(a);
               ContourPlot pp=new ContourPlot(a,0.0,L,0.0,W,20, false,"T=f(x,y) degree C","x","y");
               pp.plot();
     double bb[][]=surfacePlot.dataformat(0.0,L,0.0,W,a);
     surfacePlot sp=new surfacePlot(bb,"y m","x m","T degree C");
     sp.plot();
  }
```






```
public static void main(String args[]) //throws IOException,RemoteException, VisADException
  {  double L=0.6;
     double W=0.6;
     double dx=0.01;
     double Tx0=20.0;
     double Txn=20.0;
     double Ty0=60.0;
     double Tyn=100.0;
              double a[][]=HT_planecoz(L,W,dx,Tx0,Txn,Ty0,Tyn);
    Text.print(a);
              ContourPlot pp=new ContourPlot(a,0.0,L,0.0,W,20, false,"T=f(x,y) degree C","x","y");
              pp.plot();
      double bb[][]=surfacePlot.dataformat(0.0,L,0.0,W,a);
      surfacePlot sp=new surfacePlot(bb,"y m","x m","T degree C");
      sp.plot();
  }
```





Python version of the program:

```
from math import *
from f_xj import *;
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt

def gauss(a,b):
   #Gauss elimination with pivoting
  b1=b
  n=len(a)
  x=[0 for i in range(n)]
  for k in range(n):
```

```
    p=k
    buyuk=abs(a[k][k])
    for ii in range(k+1,n):
      dummy=abs(a[ii][k])
      if dummy>buyuk:
        buyuk=dummy
        p=ii
    if p!=k:
      for jj in range(k,n):
        dummy=a[p][jj]
        a[p][jj]=a[k][jj]
        a[k][jj]=dummy
      dummy=b1[p]
      b1[p]=b1[k]
      b1[k]=dummy
    for i in range(k+1,n):
       carpan=a[i][k]/a[k][k]
       a[i][k]=0
       for j in range(k+1,n):
          a[i][j]=a[i][j]-carpan*a[k][j]
       b1[i]   =b1[i]-carpan*b[k]
  x[n-1]=b1[n-1]/a[n-1][n-1]
  for i in range(n-1,-1,-1):
    toplam=0
    for j in range(i+1,n):
      toplam=toplam+a[i][j]*x[j]
    x[i]=(b[i]-toplam)/a[i][i]
  return x

def HT_planecoz(L,W,dx,Tx0,Txn,Ty0,Tyn):
    nx=int(L/dx+0.01)
    ny=int(W/dx+0.01)
    mx=(nx-1)
    my=(ny-1)
    mt=mx*my
    A=[[0.0 for j in range(mt)] for i in range(mt)]
    b=[0.0 for j in range(mt)]
    x=[0.0 for j in range(mt)]
    T=[[0.0 for j in range(ny+1)] for i in range(nx+1)]
    x1=0.0
    x2=0.0
    y1=0.0
    y2=0.0
    #boundary conditions
    i=0
    j=0
    for i in range(my):
        for j in range(mx):
            x1=dx*j
            x2=L-x1
            y1=dx*i
            y2=(W-y1)
            k=i*my+j
            x[k]=((x1*Tx0+x2*Txn)/L+(y1*Ty0+y2*Tyn)/W)/2.0
            kp1=(i+1)*my+j
            km1=(i-1)*my+j
            if j==0:
                b[k]=b[k]-Tx0
            if j== (mx-1):
                b[k]=b[k]-Txn;
            if i==0:
                b[k]=b[k]-Ty0;
            if i==(my-1):
                b[k]=b[k]-Tyn;
            A[k][k]+=-4.0
            if i==0 and j==0:
                A[kp1][k]=A[kp1][k]+1.0
                A[k][k+1]+=1.0
            elif (i==0) and (j==(mx-1)):
                A[kp1][k]=A[kp1][k]+1.0
                A[k][k-1]+=1.0
            elif (i==(my-1)) and (j==0):
                A[km1][k]+=1.0
                A[k][k+1]+=1.0
            elif (i==(my-1)) and (j==(mx-1)):
                A[km1][k]=A[km1][k]+1.0
                A[k][k-1]+=1.0
```

```
            elif j==0:
                A[kp1][k]+=1.0
                A[km1][k]+=1.0
                A[k][k+1]+=1.0
            elif i==0:
                A[k][k+1]+=1.0
                A[k][k-1]+=1.0
                A[kp1][k]+=1.0
            elif j==(mx-1):
                A[kp1][k]+=1.0
                A[km1][k]+=1.0
                A[k][k-1]+=1.0
            elif i==(my-1):
                A[km1][k]+=1.0
                A[k][k-1]+=1.0
                A[k][k+1]+=1.0
            else:
                A[km1][k]+=1.0
                A[kp1][k]+=1.0
                A[k][k-1]+=1.0
                A[k][k+1]+=1.0
    a=gauss(A,b)

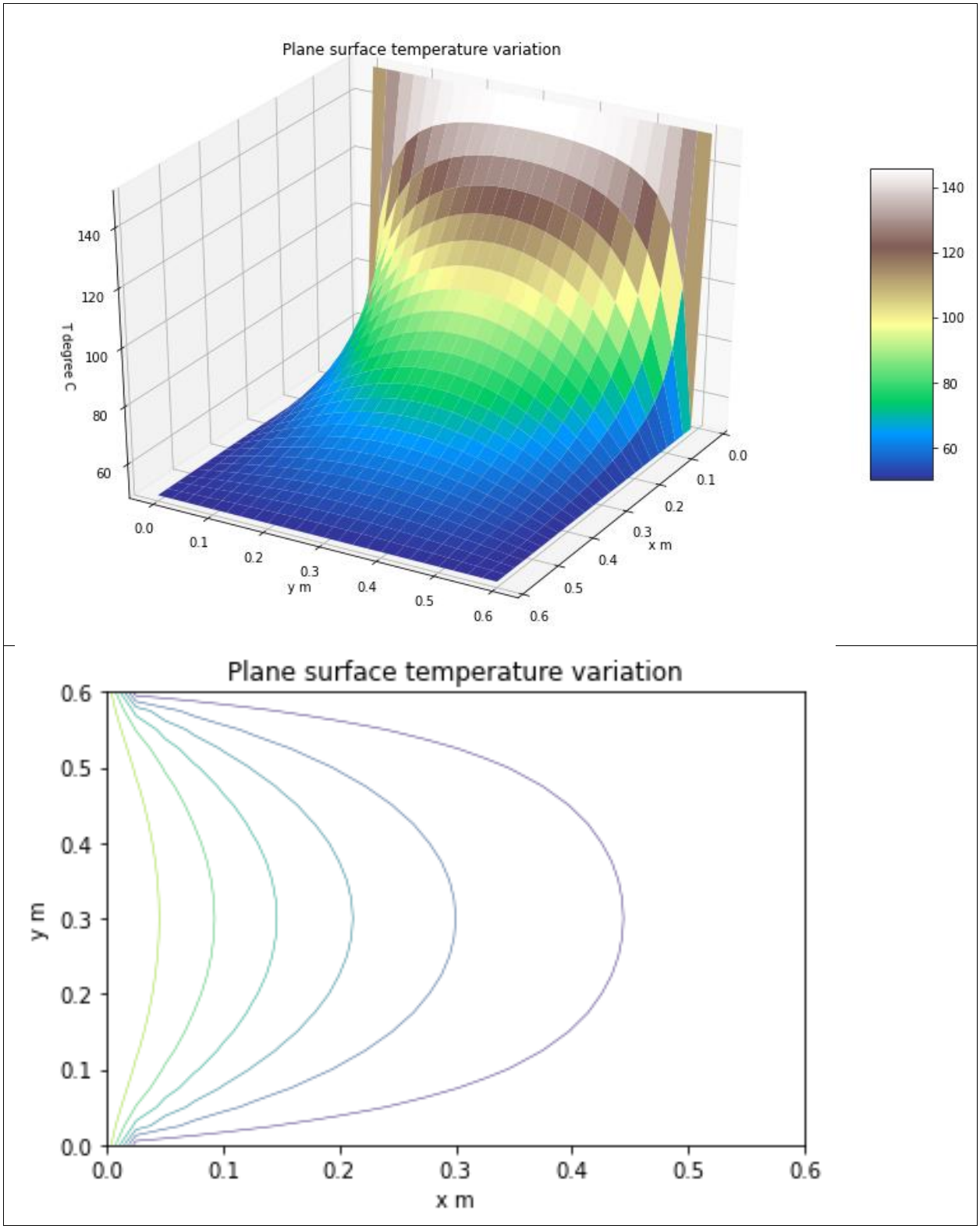
    for i in range(ny+1):
        for j in range(nx+1):
            if j==0:
                T[i][j]=Tx0
            elif i==0:
                T[i][j]=Ty0
            elif j==nx:
                T[i][j]=Txn
            elif i==ny:
                T[i][j]=Tyn
            else:
                k=(i-1)*my+(j-1)
                T[i][j]=a[k];
    return T


def plot(A,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLAbel,YLable,ZLabel):
    #f=f2()
    x=[0.0 for j in range(Nx)]
    y=[0.0 for j in range(Ny)]
    z=[[0.0 for j in range(Ny)] for i in range(Nx)]
    dx=(xmax-xmin)/(Nx-1)
    dy=(ymax-ymin)/(Ny-1)
    for i in range(Nx):
        for j in range(Ny):
            x[i]=xmin+dx*i
            y[j] = ymin+dy*j
            z[i][j]=A[i][j]
    xi=np.array(x)
    yi=np.array(y)
    xi, yi = np.meshgrid(xi, yi)
    zi=np.array(z)
    print(zi)
    fig = plt.figure(figsize =(14, 9))
    ax = plt.axes(projection ='3d')
    # Creating color map
    my_cmap = plt.get_cmap('terrain')
    # Creating plot
    surf = ax.plot_surface(xi, yi, zi,
                           cmap = my_cmap,
                           edgecolor ='none')
    fig.colorbar(surf, ax = ax,
                 shrink = 0.5,
                 aspect = 5)

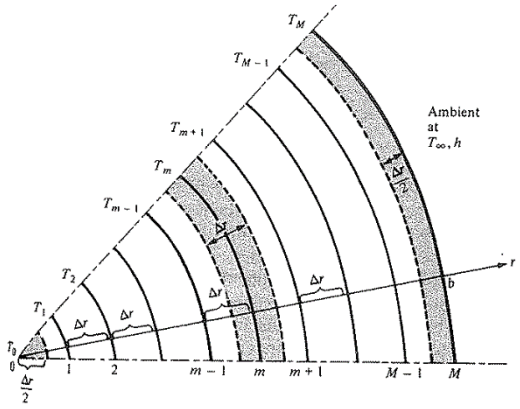
    ax.set_title(PLabel)
    ax.set_xlabel(XLabel)
    ax.set_xlim(xmin, xmax)
    ax.set_ylabel(YLabel)
    ax.set_xlim(ymin, ymax)
    ax.set_zlabel(ZLabel)
    ax.set_zlim(np.min(zi), np.max(zi))
    ax.view_init(30, 30)
    plt.show()
```

```
# Creating dataset
def contour_plot(A,xmin,xmax,Nx,ymin,ymax,Ny,PLabel,XLabel,YLable,ZLabel):
    #f=f2()
    x=[0.0 for j in range(Nx)]
    y=[0.0 for j in range(Ny)]
    z=[[0.0 for j in range(Ny)] for i in range(Nx)]
    dx=(xmax-xmin)/(Nx-1)
    dy=(ymax-ymin)/(Ny-1)
    for i in range(Nx):
        for j in range(Ny):
            x[i]=xmin+dx*i
            y[j] = ymin+dy*j
            z[i][j]=A[i][j]
    xi=np.array(x)
    yi=np.array(y)
    xi, yi = np.meshgrid(xi, yi)
    zi=np.array(z)
    fig, ax = plt.subplots(1, 1)
    # plots contour lines
    ax.contour(xi, yi, zi,linewidths=0.5)
    ax.set_title(PLabel)
    ax.set_xlabel(XLabel)
    ax.set_xlim(xmin, xmax)
    ax.set_ylabel(YLabel)
    ax.set_xlim(ymin, ymax)
    plt.show()


W=0.6
L=0.6
dx=0.025
Tx0=150.0
Txn=50.0
Ty0=50.0
Tyn=50.0
T=HT_planecoz(L,W,dx,Tx0,Txn,Ty0,Tyn)
print(T)
PLabel="Plane surface temperature variation"
XLabel="x m"
YLabel="y m"
ZLabel="T degree C"
Nx=len(T)
Ny=len(T[0])
plot(T,0.0,L,Nx,0.0,W,Ny,PLabel,XLabel,YLabel,ZLabel)
contour_plot(T,0.0,L,Nx,0.0,W,Ny,PLabel,XLabel,YLabel,ZLabel)
```



PROBLEM 4.4 FINITE DIFFERENCE SOLUTION, NON-HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CYLINDRICAL COORDINATES, ONE DIMENSION WITH HEAT GENERATION



$$\Delta r = \frac{b}{M}$$

$$\frac{T_{m-1} - T_m}{R_{m-1,m}} + \frac{T_{m+1} - T_m}{R_{m+1,m}} + \Delta V_m g_m = 0$$

where g_m is heat generation, ΔV_m is volume of element m.

$$R_{m-1,m} = \frac{\Delta r}{2\pi\left(m\Delta r - \frac{\Delta r}{2}\right)Hk} = \frac{1}{2\pi m\left(1 - \frac{1}{2m}\right)Hk}$$

$$R_{m+1,m} = \frac{\Delta r}{2\pi\left(m\Delta r + \frac{\Delta r}{2}\right)Hk} = \frac{1}{2\pi m\left(1 + \frac{1}{2m}\right)Hk}$$

$$\Delta V_m = 2\pi m\Delta r\Delta r H$$

H=the length of the cylinder. Equation can be arranged as:

$$\left(1 - \frac{1}{2m}\right)T_{m-1} - 2T_m + \left(1 + \frac{1}{2m}\right)T_{m+1} + \frac{(\Delta r)^2 g_m}{k} = 0$$

Boundary conditions:

At the center of the cylinder:

$$\frac{T_1 - T_0}{R_{1,0}} + \Delta V_0 g_0 = 0$$

$$R_{1,0} = \frac{\Delta r}{2\pi\left(\frac{\Delta r}{2}\right)Hk} = \frac{1}{\pi Hk}$$

$$\Delta V_0 = \pi\left(\frac{\Delta r}{2}\right)^2 H$$

After the arrangement

$$4(T_1 - T_0) + \frac{(\Delta r)^2 g_0}{k} = 0$$

Second boundary condition (convective boundary condition)

$$-kA\frac{T_{M+1} - T_{M-1}}{2\Delta r} + Ah(T_M - T_\infty) = 0$$

Combining this equation with previous finite difference formula :

$$\left(1 - \frac{1}{2M}\right)T_{m-1} - 2T_m + \left(1 + \frac{1}{2M}\right)T_{m+1} + \frac{(\Delta r)^2 g_m}{k} = 0$$

Gives

$$2T_{M-1} - 2\beta_M T_M + \gamma_M + \frac{(\Delta r)^2 g_m}{k} = 0$$

$$\beta_M = 1 + \left(1 + \frac{1}{2M}\right)\frac{(\Delta r)h}{k}$$

$$\gamma_M = \left(1 + \frac{1}{2M}\right)\frac{(\Delta r)h}{k}T_\infty$$

Problem 4.4 Java Solution:

```
import java.io.*;

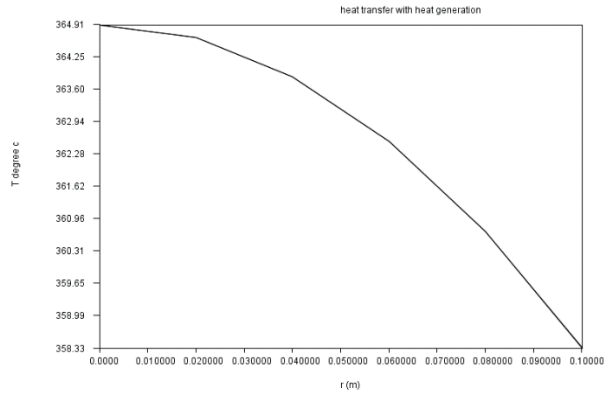
class HT_cylinder_FD_g
{
  public static double [] thomas(double a[][],double b[])
  {
  //3 band matrix
  int n=a.length;
  double f[]=new double[n];
  double e[]=new double[n];
  double g[]=new double[n];
  double x[]=new double[n];
  for(int i=0;i<n;i++)     {f[i]=a[i][i];}
  for(int i=0;i<(n-1);i++) {g[i]=a[i][i+1];}
  for(int i=0;i<(n-1);i++) {e[i+1]=a[i+1][i];}
  for(int k=1;k<n;k++)
    {e[k]=e[k]/f[k-1];
     f[k]=f[k]-e[k]*g[k-1];
    }
  for(int k=1;k<n;k++)
    {b[k]=b[k]-e[k]*b[k-1];
    }
  x[n-1]=b[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
     {x[k]=(b[k]-g[k]*x[k+1])/f[k];}
  return x;
  }

  public static double [] HT_cylinder( double b,double gm[],double dr, double Tr,double hr,double kr)
  {
  //b : cylinder radius
  //gm : heat generatşon per unit volume
  int M=(int)(b/dr);
  int mt=M+1;
  double A[][]=new double[mt][mt];
  double B[]=new double[mt];
  double x[]=new double[mt];
  double x1,x2,y1,y2;

  // boundary conditions
  int m=0;
  double GM=0,betaM=0,gammaM=0;
  for(m=0;m<=M;m++)
  {  GM=dr*dr*gm[m]/kr;
    betaM=1+(1+1.0/(2.0*m))*dr*hr/kr;
    gammaM=(1+1.0/(2.0*m))*dr*hr/kr*Tr;
              if(m==0)          {B[m]+=-dr*dr*gm[0]/kr;}
    else if(m==M)      {B[m]+=-2.0*gammaM-GM;}
    else               {B[m]+=-dr*dr*gm[m]/kr;}
    if(m==0)     {A[m][m]+=-4.0;A[m][m+1]+=4.0;}
    else if(m==M) {A[m][m-1]=2.0;A[m][m]+=-2*betaM;}
    else         {A[m][m-1]+=1.0-1/(2.0*m);A[m][m]+=-2.0;A[m][m+1]+=1.0+1/(2.0*m);}
  }
  Text.print(A,"A");
  Text.printT(B,"B");
  //double T[]=conjugate_gradient1.conjugate_gradient(A,B);
  double T[]=thomas(A,B);
  return T;
}
  public static void main(String args[])
  {  int M=5;
    double b=10.0e-2;
    double dr=b/M;
    double gm[]=new double[M+1];
    double r[]=new double[M+1];
    for(int m=0;m<M+1;m++)
    {gm[m]=1.0e6;//W/m3
    r[m]=dr*m;
    }
    double Tr=25.0;//degree C
    double hr=150.0;//W/(m2K)
    double k=380.0;//W/(mK)
              double T[]=HT_cylinder( b,gm,dr,Tr,hr,k);
```

```
            //compare result with analytical case with qm=const.
            double T1[]=new double[T.length];
            for(int m=0;m<M+1;m++)
    {T1[m]=gm[0]*b*b/(4.0*k)*(1.0-(m*dr/b)*(m*dr/b))+gm[0]*b/(2.0*hr)+Tr;
    }
    double TT[][]={T,T1};
    Text.print(TT);
            Plot pp=new Plot(r,T);
            pp.addData(r,T1);
            pp.setPlabel("heat transfer with heat generation");
            pp.setXYlabel("r (m)","T degree c");
            pp.plot();
  }
}
```

Graphic output:



Solution and analytical solution to compare the results

0	1	2	3	4	5
364.9122807017613	364.64912280702447	363.8596491228139	362.5438596491297	360.70175438597175	358.3333333333401
364.91228070175436	364.64912280701753	363.859649122807	362.5438596491228	360.7017543859649	358.3333333333333

The matrix equations: (M=5)

A

0	1	2	3	4	5
-4.0	4.0	0.0	0.0	0.0	0.0
0.5	-2.0	1.5	0.0	0.0	0.0
0.0	0.75	-2.0	1.25	0.0	0.0
0.0	0.0	0.8333333...	-2.0	1.1666666...	0.0
0.0	0.0	0.0	0.875	-2.0	1.125
0.0	0.0	0.0	0.0	2.0	-2.0173684...

B

0
-1.0526315789473684
-1.0526315789473684
-1.0526315789473684
-1.0526315789473684
-1.0526315789473684
-1.486842105263158

If the same problem solved by M=10, matrices will become :

A

0	1	2	3	4	5	6	7	8	9	10
-4.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.5	-2.0	1.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.75	-2.0	1.25	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.83...	-2.0	1.16...	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.875	-2.0	1.125	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.9	-2.0	1.1	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.91...	-2.0	1.08...	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.92...	-2.0	1.07...	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.9375	-2.0	1.0625	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.94...	-2.0	1.05...
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	-2.00...

B

0
-0.2631578947368421
-0.2631578947368421
-0.2631578947368421
-0.2631578947368421
-0.2631578947368421
-0.2631578947368421
-0.2631578947368421
-0.2631578947368421
-0.2631578947368421
-0.2631578947368421
-0.47039473684210525

data output(veri çıktısı) :

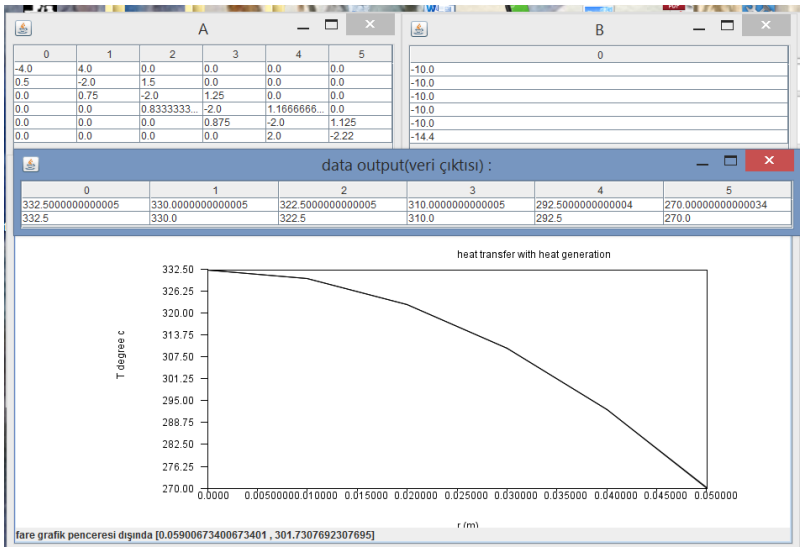
0	1	2	3	4	5	6	7	8	9	10
364.912280...	364.846491...	364.649122...	364.320175...	363.859649...	363.267543...	362.543859...	361.688596...	360.701754...	359.583333...	358.3333
364.912280...	364.846491...	364.649122...	364.320175...	363.859649...	363.267543...	362.543859...	361.688596...	360.701754...	359.583333...	358.3333

A $10x10^{-2}$ m diameter solid steel bar of thermal conductivity k=40 W/(mK) is heated electrically by the passage of electrical current which henerates energy within the rod at a rate of $g=4x10^{6}$ W/m^3. Heat is dissipated from the

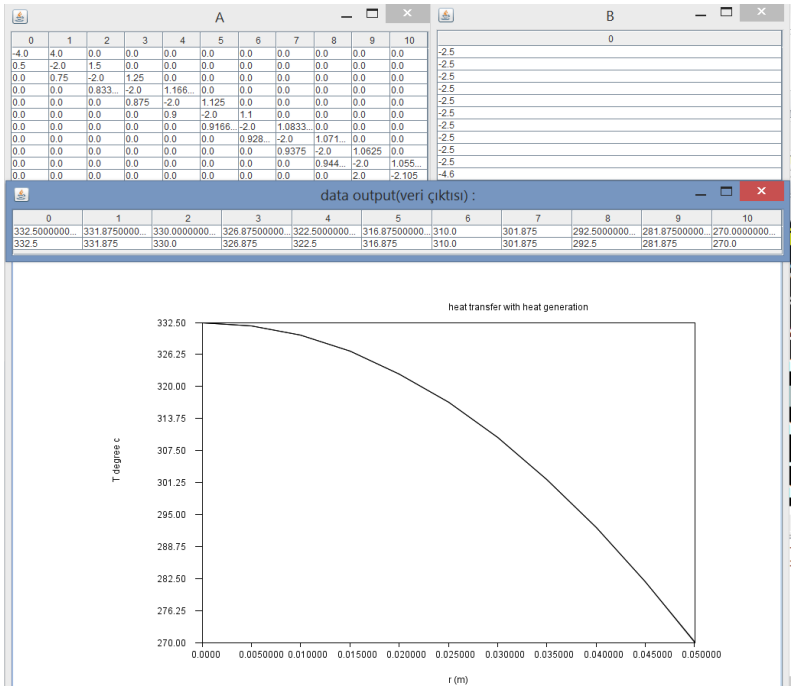
surface of the rod by convection with a heat transfer coefficient h=400 W/(m^2K) into an ambient temperature T_∞ =20 C. By dividing the Radius into 5 and 10 equal parts develop and solve the finite difference equation.

M=5 division

```
public static void main(String args[])
  {  int M=5;
     double b=5.0e-2;
     double dr=b/M;
     double gm[]=new double[M+1];
     double r[]=new double[M+1];
     for(int m=0;m<M+1;m++)
     {gm[m]=4.0e6;//W/m3
     r[m]=dr*m;
     }
     double Tr=20.0;//degree C
     double hr=400.0;//W/(m2K)
     double k=40.0;//W/(mK)
                double T[]=HT_cylinder( b,gm,dr,Tr,hr,k);
                //compare result with analytical case with qm=const.
                double T1[]=new double[T.length];
                for(int m=0;m<M+1;m++)
     {T1[m]=gm[0]*b*b/(4.0*k)*(1.0-(m*dr/b)*(m*dr/b))+gm[0]*b/(2.0*hr)+Tr;
     }
     double TT[][]={T,T1};
     Text.print(TT);
                Plot pp=new Plot(r,T);
                pp.addData(r,T1);
                pp.setPlabel("heat transfer with heat generation");
                pp.setXYlabel("r (m)","T degree c");
                pp.plot();
  }
```

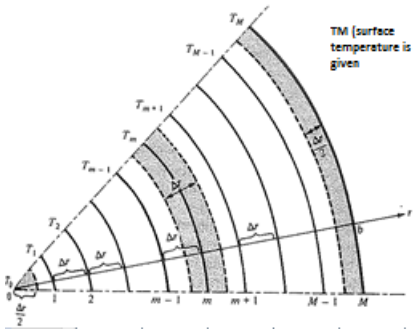


M=10 division



PROBLEM 4.5 FINITE DIFFERENCE SOLUTION NON-HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CYLINDRICAL COORDINATES, ONE DIMENSION WITH HEAT GENERATION

A long cylindrical fuel element of Radius b=1x10^{-2}m and thermal conductivity k=25 W/(mK) generates energy at a constant rate of g=5x10^{8}W/m^{3}. The boundary surface at r=b can be assumed to be maintained at 100 C. Assuming one-dimensional, radial heat flow, calculate the radial temperature distributio in the fuel element by finite difference method.



Problem is similar to the previous problem except that temperature boundary is given in the outside. Therefore the Boundary condition are:

$T_M = T(b)$

So the boundary condition will become: for m=M-1

$$\left(1-\frac{1}{2m}\right)T_{m-1} - 2T_m = -\frac{(\Delta r)^2 g_m}{k} - \left(1-\frac{1}{2m}\right)T_M$$

Problem 4.5 Java Solution:

```
import java.io.*;

class HT_cylinder_FD_g1
{

 public static double[] pivotlugauss(double a[][],double b[])
 { //kısmi pivotlu gauss eleme yöntemi
 int n=b.length;
 double x[]=new double[n];
 double carpan=0;
 double toplam=0;
 double buyuk;
 double dummy=0;
 //gauss eleme
 int i,j,k,p,ii,jj;
```

```
  //System.out.println("Orijinal Matris :\n"+Matrix.toString(a));
  for(k=0;k<(n-1);k++)
  { //pivotlama
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
                //System.out.println("pivotlu Matris : k="+k+"\n"+Matrix.toString(a));
              //gauss elemeyi çözme
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
              }
  }
  //back substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }

  public static double [] HT_cylinder( double b,double gm[],double dr, double Tr,double k)
  {
  //b : cylinder radius
  //gm : heat generatşon per unit volume
  int M=(int)(b/dr+0.01);
  double A[][]=new double[M][M];
  double B[]=new double[M];
  double x[]=new double[M+1];
  double x1,x2,y1,y2;

  // boundary conditions
  int m=0;
  for(m=0;m<M;m++)
  {
              if(m==0)          {B[m]+=-dr*dr*gm[m]/k;}
    else if(m==(M-1))   {B[m]+=-dr*dr*gm[m]/k-(1+1/(2.0*m))*Tr;}
    else              {B[m]+=-dr*dr*gm[m]/k;}
    if(m==0)     {A[m][m]+=-4.0;A[m][m+1]+=4.0;}
    else if(m==(M-1)) {A[m][m-1]+=1.0-1/(2.0*m);A[m][m]+=-2.0;}
    else         {A[m][m-1]+=1.0-1/(2.0*m);A[m][m]+=-2.0;A[m][m+1]+=1.0+1/(2.0*m);}
  }
  Text.print(A,"A");
  Text.printT(B,"B");
  //double T[]=conjugate_gradient1.conjugate_gradient(A,B);
  double T[]=pivotlugauss(A,B);
  return T;
}
  public static void main(String args[])
  {  int M=10;
     double b=1.0e-2;
     double dr=b/M;
     double gm[]=new double[M+1];
     double r[]=new double[M+1];
     for(int m=0;m<=M;m++)
     {gm[m]=5.0e8;//W/m3
      r[m]=dr*m;
```

```
    }
    double Tr=100.0;//degree C
    double k=25.0;//W/(mK)
    System.out.println("dr="+dr+"gg="+gm[0]*dr*dr/k);
                double a[]=HT_cylinder( b,gm,dr,Tr,k);
                double T[]=new double[a.length+1];
                //exact solution
                double T1[]=new double[a.length+1];
                T[M]=Tr;
                T1[M]=Tr;
                for(int m=0;m<M;m++)
    {T[m]=a[m];
     T1[m]=gm[m]*b*b/(4.0*k)*(1.0-(r[m]/b)*(r[m]/b))+Tr;
    }
    double TT[][]={T,T1};
    Text.print(TT);
                Plot pp=new Plot(r,T);
                pp.addData(r,T1);
                pp.setPlabel("heat transfer with heat generation");
                pp.setXYlabel("r (m)","T degree c");
                pp.plot();
  }
}
```

A

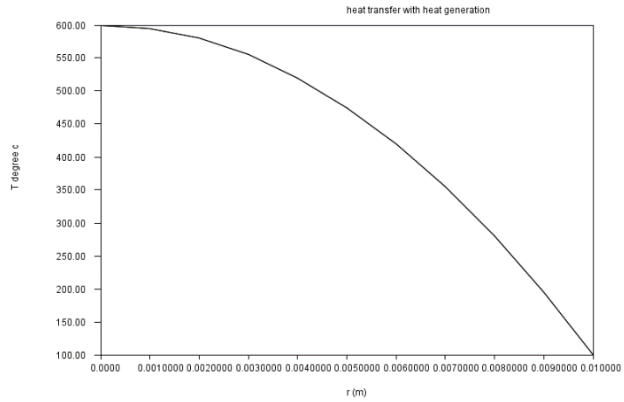
0	1	2	3	4	5	6	7	8	9
-4.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.5	-2.0	1.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.75	-2.0	1.25	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.833...	-2.0	1.166...	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.875	-2.0	1.125	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.9	-2.0	1.1	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.916...	-2.0	1.083...	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.928...	-2.0	1.071...	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.9375	-2.0	1.0625
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.944...	-2.0

B

0
-20.0
-20.0
-20.0
-20.0
-20.0
-20.0
-20.0
-20.0
-20.0
-125.55555555555556

data output(veri çıktısı) :

0	1	2	3	4	5	6	7	8	9	10
600.000000...	595.000000...	580.000000...	555.000000...	520.000000...	475.000000...	420.000000...	355.000000...	280.0	195.0	100.0
600.0	595.0	580.0	555.0	520.0	475.0	420.0	355.0	279.999999...	194.999999...	100.0



PROBLEM 4.6 FINITE DIFFERENCE SOLUTION, HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CYLINDRICAL COORDINATES, ONE DIMENSION , FIN PROBLEM

An iron rod , L=$5x10^{-2}$ m long of diameter = $2x10^{-2}$ m with thermal conductivity k= 50 W/(m K) protrudes from a wall and is exposed to an ambient at T_∞=27 C and h = 100 W/(mK). The base of the rod is at T_0= 327 ℃, and its tip is insulated. Assuming one-dimensional steady-state heat flow, calculate the temperature distribution along the rod and the rate of heat loss into the ambient by using finite differences. compaıe the finite difference solution with the exact analytical solution of this problem.

$$\frac{\partial}{\partial x}\left(-kA_c\frac{\partial T}{\partial x}\right)dx + hPdx(T - T_\infty) = 0$$

k=const A_c=const $\theta = T - T_\infty$

$\frac{\partial^2\theta}{\partial x} + \frac{hP}{kA_c}\theta = 0 \quad m^2 = \frac{hP}{kA_c} \quad \frac{\partial^2\theta(x)}{\partial x} + m^2\theta(x) = 0$ x'=L-x $\frac{\partial^2\theta(x\prime)}{\partial x} + m^2\theta(x') = 0$

Boundary conditions: x'=0 $\frac{\partial\theta}{\partial x} = 0$ x'=L $\theta(x') = T_b - T_\infty = \theta_b$

$$\theta(x) = \theta_b\frac{\tanh\,[\mathrm{m}(L - x)]}{\tanh\,[\mathrm{m}L]}$$

Finite difference equations:

$$\frac{\theta_{i-1} - 2\theta_i + \theta_{i+1}}{(\Delta x)^2} - m^2\theta_n = 0$$

$$\theta_{i-1} - [2 + (m\Delta x)^2]\theta_i + \theta_{i+1} = 0$$

Boundries:

$\theta_b - [2 + (m\Delta x)^2]\theta_i + \theta_{i+1} = 0$

$-[2 + (m\Delta x)^2]\theta_i + \theta_{i+1} = -\theta_b$ i=0

$\theta_{i-1} - [2 + (m\Delta x)^2]\theta_i = 0$ for i=n

```
public class HT_FD_fin
{ double L;double D;double Tb;double Tinf;double h;double k;double m;double tetab;
  public double x[];
  public HT_FD_fin(double Li, double Di,double Tbi,double Tinfi,double hi,double ki)
  {L=Li;D=Di;Tb=Tbi;Tinf=Tinfi;h=hi;k=ki;tetab=Tb-Tinf;double Ac=Math.PI*D*D/4.0; double P=Math.PI*D;
   m=Math.sqrt(h*P/(k*Ac));
  }
  public double[] T1(int n)
  {double T1[]=new double[n];
   x=new double[n];
   double dx=L/n;
   double xx=1;
   for(int i=0;i<n;i++)
   {xx=dx*(i+1);
    x[i]=xx;
    T1[i]=tetab*Math.cosh(m*(L-xx))/Math.cosh(m*L);
   }
   Text.print(T1,"T1");
   return T1;
  }
  public double[] T2(int n)
  {double T2[]=new double[n];
   double dx=L/n;
   double A[][]=new double[n][n];
   double B[]=new double[n];
   B[0]=-tetab;
   for(int i=0;i<n;i++)
   { if(i==0) {A[i][i]+=-(2.0+(m*dx)*(m*dx));A[i][i+1]+=1;}
     else if(i==(n-1)) {A[i][i-1]+=2;A[i][i]+=-(2.0+(m*dx)*(m*dx));}
     else {A[i][i-1]+=1;A[i][i]+=-(2.0+(m*dx)*(m*dx));A[i][i+1]+=1;}
   }
   Text.print(A,"A");
   Text.print(B,"B");
   T2=NA28.AXB(A,B);
   Text.print(T2,"T2");
   return T2;
  }
  public static void main(String arg[])
  {double L=5e-2; //m
   double D=2e-2; //m
   double k=50; //W/mK
   double Tinf=27; //degree C
   double h=100; //W/m^2K
   double Tb=327; //degree C
   HT_FD_fin f=new HT_FD_fin(L,D,Tb,Tinf,h,k);
   int n=5;
   double T1[]=f.T1(n);
   double x[]=f.x;
   double T2[]=f.T2(n);
   Plot pp=new Plot(x,T1);
   pp.addData(x,T2);
   pp.plot();
```

```
  }
}
```

N=5

A

0	1	2	3	4
-2.04	1.0	0.0	0.0	0.0
1.0	-2.04	1.0	0.0	0.0
0.0	1.0	-2.04	1.0	0.0
0.0	0.0	1.0	-2.04	1.0
0.0	0.0	0.0	2.0	-2.04

B

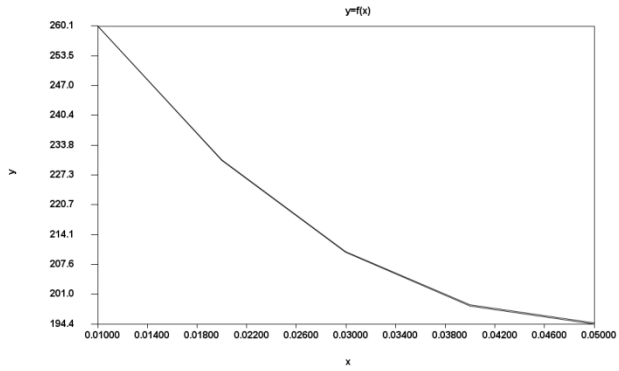
0	1	2	3	4
-300.0	0.0	0.0	0.0	0.0

T1

0	1	2	3	4
260.0191298...	230.4737402...	210.1780712...	198.3175861...	194.4162820...

T2

0	1	2	3	4
260.1185119...	230.6417644...	210.3906874...	198.5552379...	194.6619980...



N=10

A

0	1	2	3	4	5	6	7	8	9
-2.01	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	-2.01	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.0	-2.01	1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	1.0	-2.01	1.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	1.0	-2.01	1.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	-2.01	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	1.0	-2.01	1.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	1.0	-2.01	1.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-2.01	1.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	-2.01

B

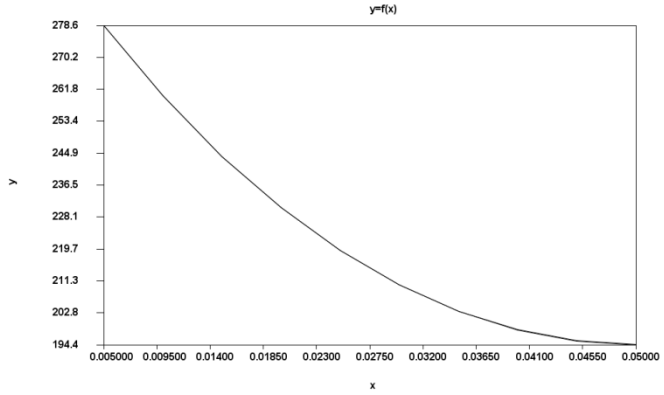
0	1	2	3	4	5	6	7	8	9
-300.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

T1

0	1	2	3	4	5	6	7	8	9
278.615326...	260.019129...	244.025291...	230.473740...	219.2288477...	210.178071...	203.2308274...	198.317586...	195.3891738...	194.416282...

T2

0	1	2	3	4	5	6	7	8	9
278.628885...	260.044059...	244.059674...	230.515885...	219.277256...	210.231399...	203.2878563...	198.377191...	195.4502993...	194.477909...



N=20

T1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
288.946...	278.615...	268.980...	260.019...	251.707...	244.025...	236.953...	230.473...	224.570...	219.228...	214.435...	210.178...	206.446...	203.230...	200.523...	198.317...	196.607...	195.389...	194.659...	194.416...

T2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
288.948...	278.618...	268.985...	260.025...	251.714...	244.033...	236.962...	230.484...	224.581...	219.240...	214.448...	210.191...	206.460...	203.245...	200.538...	198.332...	196.622...	195.404...	194.674...	194.431...

A

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
-2.0025	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	-2.0025	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.0	-2.0025	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	1.0	-2.0025	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	1.0	-2.0025	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	-2.0025	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	1.0	-2.0025	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	1.0	-2.0025	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-2.0025	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-2.0025	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-2.0025	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-2.0025	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-2.0025	1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-2.0025	1.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-2.0025	1.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-2.0025	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-2.0025	1.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-2.0025	1.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-2.0025	1.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	-2.0025

B

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
-300.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Python code for the problem:

```
from math import *
from f_x import *
import numpy as np;
import matplotlib.pyplot as plt;

class  HT_FD_fin:
  def __init__(self,Li,Di,Tbi,Tinfi, hi,ki):
      self.name = 'HT_FD_fin'
      self.L=Li
      self.D=Di
      self.Tb=Tbi
      self.Tinf=Tinfi
      self.h=hi
      self.k=ki
      self.tetab=Tb-Tinf
      self.Ac=pi*D*D/4.0
```

```
        self.P=pi*D
        xx=self.h*self.P/(self.k*self.Ac)
        self.m=sqrt(xx)

    # analytical solution
    def T1(self,n):
        T1=[0 for i in range(n)]
        self.x=[0 for i in range(n)]
        dx=L/n
        xx=1.0
        for i in range(n):
            xx=dx*(i+1);
            self.x[i]=xx;
            T1[i]=self.tetab*cosh(self.m*(self.L-xx))/cosh(self.m*self.L);
        return T1

    def T2(self,n):
        T2=[0 for i in range(n)]
        dx=L/n
        A=[[0.0 for j in range(n)] for i in range(n)]
        B=[0.0 for j in range(n)]
        B[0]=-self.tetab
        for i in range(n):
            if i==0:
                A[i][i]+=-(2.0+(self.m*dx)*(self.m*dx))
                A[i][i+1]+=1
            elif i==(n-1):
                A[i][i-1]+=2
                A[i][i]+=-(2.0+(self.m*dx)*(self.m*dx))
            else:
                A[i][i-1]+=1
                A[i][i]+=-(2.0+(self.m*dx)*(self.m*dx))
                A[i][i+1]+=1
        a=gauss(A,B)
        return a

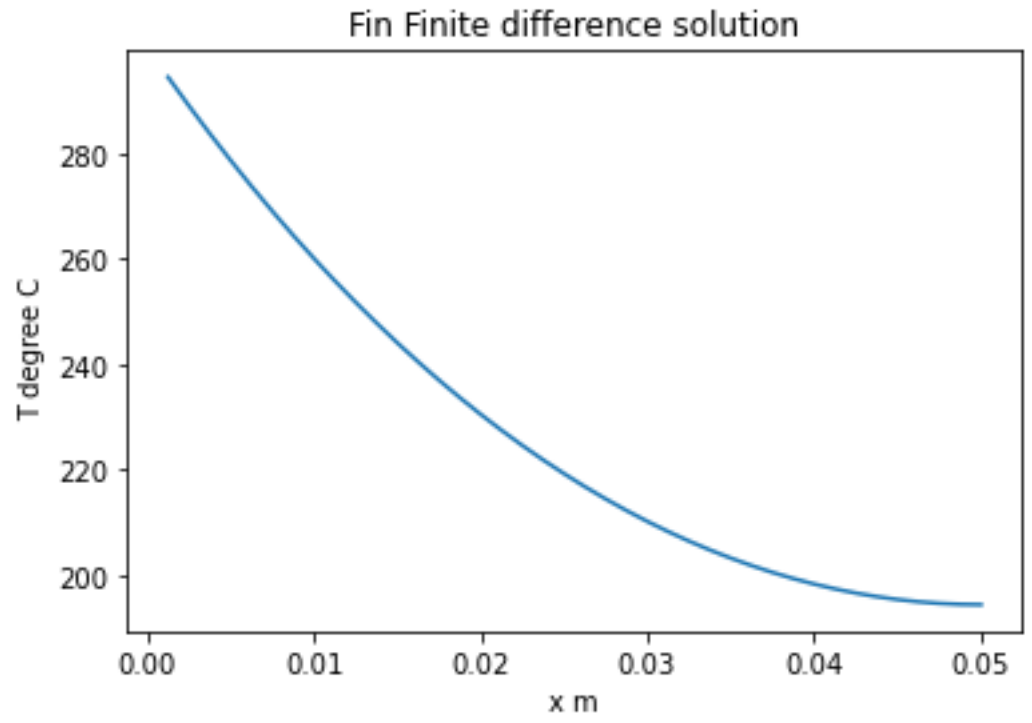
    def dT(self,n):
        d=[0.0 for j in range(n)]
        T1=self.T1(n)
        T2=self.T2(n)
        for i in range(n):
            d[i]=T1[i]-T2[i]
        return d

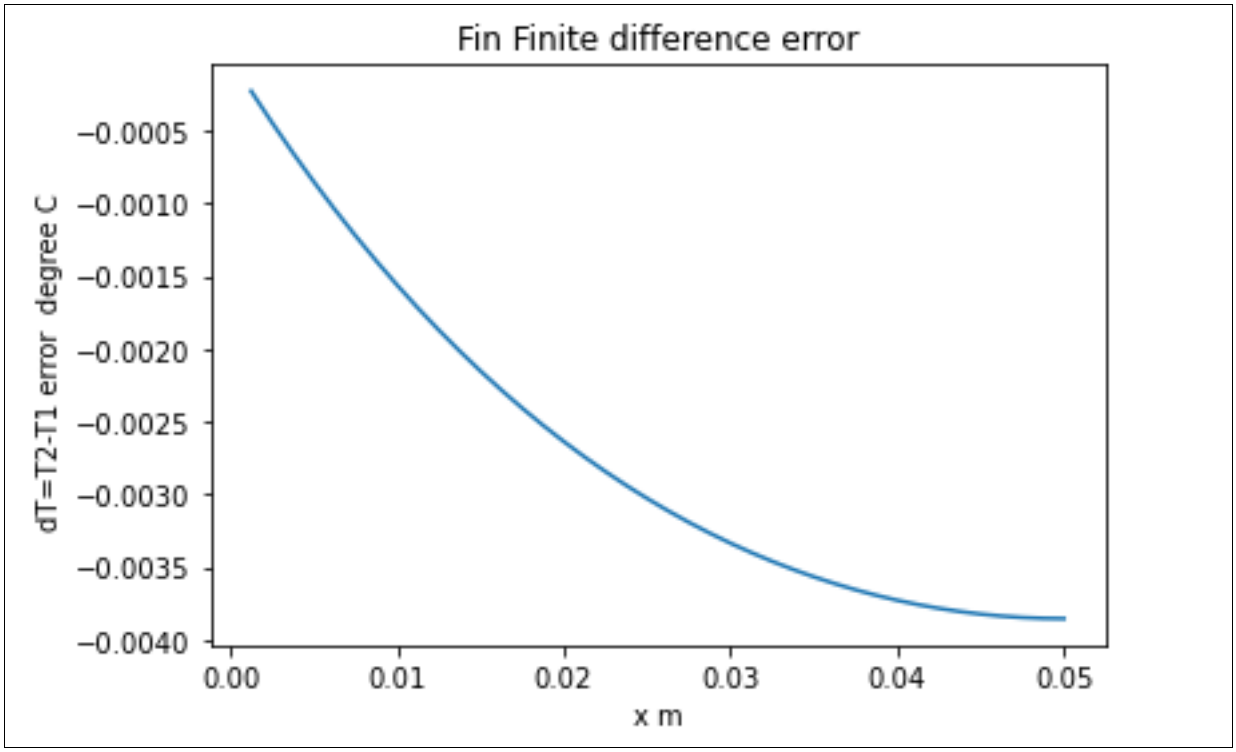
def gauss(a,b):
    #Gauss elimination with pivoting
    b1=b
    n=len(a)
    x=[0 for i in range(n)]
    for k in range(n):
      p=k
      buyuk=abs(a[k][k])
      for ii in range(k+1,n):
        dummy=abs(a[ii][k])
        if dummy>buyuk:
          buyuk=dummy
          p=ii
      if p!=k:
        for jj in range(k,n):
          dummy=a[p][jj]
          a[p][jj]=a[k][jj]
          a[k][jj]=dummy
        dummy=b1[p]
        b1[p]=b1[k]
        b1[k]=dummy
      for i in range(k+1,n):
        carpan=a[i][k]/a[k][k]
        a[i][k]=0
        for j in range(k+1,n):
          a[i][j]=a[i][j]-carpan*a[k][j]
        b1[i]   =b1[i]-carpan*b[k]
    x[n-1]=b1[n-1]/a[n-1][n-1]
    for i in range(n-1,-1,-1):
      toplam=0
      for j in range(i+1,n):
        toplam=toplam+a[i][j]*x[j]
      x[i]=(b[i]-toplam)/a[i][i]
```

```
  return x

#main program
L=5e-2; #m
D=2e-2;  # m
k=50;    # W/mK
Tinf=27; #degree C
h=100;   # W/m^2K
Tb=327;  # degree C
f=HT_FD_fin(L,D,Tb,Tinf,h,k)
n=40
T1=f.T1(n)
print("x=",f.x,"nx=",len(f.x))
print("T1=",T1,"nT1=",len(T1))
T2=f.T2(n)
print("T2=",T2,"nT2=",len(T2))
"""
plt.title('Fin Finite difference solution')
plt.xlabel('x m')
plt.ylabel('T degree C ')
plt.plot(f.x,T2)
"""
dT=f.dT(n)
plt.title('Fin Finite difference error')
plt.xlabel('x m')
plt.ylabel('dT=T2-T1 error  degree C ')
plt.plot(f.x,dT)
```

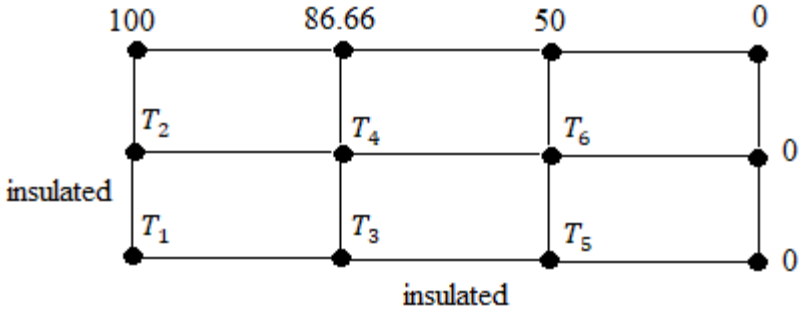
```
runfile('E:/okul/SCO1/HT_FD_fin.py', wdir='E:/okul/SCO1')
Reloaded modules: f_x
x= [0.00125, 0.0025, 0.00375, 0.005, 0.00625, 0.0075, 0.00875, 0.01, 0.01125, 0.0125, 0.01375, 0.015, 0.01625, 0.0175, 0.01875, 0.02,
0.02125, 0.0225, 0.02375, 0.025, 0.02625, 0.0275, 0.02875, 0.03, 0.03125, 0.0325, 0.03375, 0.035, 0.03625, 0.0375, 0.03875, 0.04,
0.04125, 0.0425, 0.043750000000000004, 0.045, 0.04625, 0.0475, 0.04875, 0.05] nx= 40
T1= [294.3812036992178, 288.94640523366866, 283.69220767739404, 278.6153269858826, 273.7125899434355, 268.9809321798038,
264.41739625485656, 260.01912981008513, 255.78338378578565, 251.7075107028078, 247.7889630077938, 244.0252914808746,
240.41414370482815, 236.95326259474146, 233.64048498725953, 230.47374028853767, 227.45104918005282, 224.57052238146528,
221.83035946975676, 219.22884775390764, 216.76436120440914, 214.43535943694195, 212.24038674958592, 210.17807121295914,
208.247123812717, 206.44633764387683, 204.77458715646247, 203.23082745199918, 201.81409363041806, 200.523500186962,
199.35824045871647, 198.3175861204189, 197.40088672923167, 196.6075693181942, 195.93713803809993, 195.38917384757437,
194.96333425116075, 194.65935308524902, 194.47704035171506, 194.41628209916564] nT1= 40
T2= [294.3814296296931, 288.94684765290475, 283.6928574558994, 278.616175294804, 273.7136282432678, 268.98215220938357,
264.4187900206302, 260.02068957563966, 255.78510206163386, 251.70938023641662, 247.79097677384715, 244.02744267176124,
240.41642572134518, 236.95566903700492, 233.6430096458128, 230.47637713564927, 227.45379236119547, 224.5733662069674,
221.83329840661867, 219.23187641777406, 216.76747435169048, 214.43855195707667, 212.243653657436, 210.18140764133122,
208.25052500500223, 206.44979894680134, 204.77810401294215, 203.23439539409102, 201.81770827236116, 200.52715721830148,
199.3619356375032, 198.3213152664783, 197.4046457174949, 196.6113540720849, 195.94094452296994, 195.39299806418182,
194.96717222918375, 194.66320087682888, 194.48089402502202, 194.4201377319808] nT2= 40
```



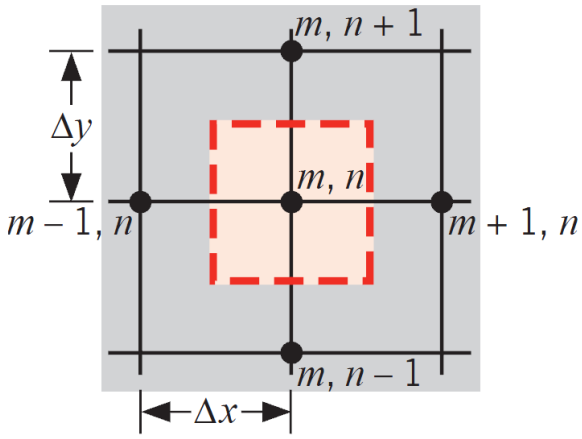


PROBLEM 4.7 FINITE DIFFERENCE SOLUTION, HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CARTESIAN COORDINATES, TWO DIMENSIONS

Consider the steady state problem given below. Temperature of top side has a changing temperature of 100,86.66,50 and 0 ℃, right hand side is 0 ℃. Bottom and left hand side is insulated. Calculate inner temperature profile. $\Delta x = \Delta y = 0.1\ m$



Steady state inner cell (x,y) directions



For $\Delta x = \Delta y$ $T_{m+1,n} + T_{m-1,n} + T_{m,n+1} + T_{m,n-1} - 4T_{m,n} = 0$

$$-4T_1 + 2T_3 + 2T_4 = 0$$
$$-4T_2 + 2T_2 + T_1 + 100 = 0$$
$$-4T_3 + 2T_4 + T_1 + T_5 = 0$$
$$-4T_4 + 86.66 + T_3 + T_2 + T_6 = 0$$
$$-4T_5 + 2T_6 + T_3 + 0 = 0$$
$$-4T_6 + 50 + T_4 + T_2 + 0 + T_5 = 0$$

-4	2	2	0	0	0	$\mathbf{T_1}$	0
1	-4	0	2	0	0	$\mathbf{T_2}$	-100
1	0	-4	2	1	0	$\mathbf{T_3}$	0
0	1	1	-4	0	1	$\mathbf{T_4}$	-86.66
0	0	1	0	-4	2	$\mathbf{T_5}$	0
0	0	0	1	1	-4	$\mathbf{T_6}$	-50

```
import java.io.*;
import java.util.Locale;
class NA15
{ public static String toString(double a[])
  {int n=a.length;
   String s="";
   for(int i=0;i<n;i++)
   {s+=String.format(Locale.US,"%10g",a[i])+" ";}
   return s;
  }
  public static String toString(double a[][])
  {int n=a.length;
   int m=a[0].length;
   String s="";
   for(int i=0;i<n;i++)
   { for(int j=0;j<m;j++)
                {s+=String.format(Locale.US,"%10g",a[i][j])+" ";}
                s+="\n";
   }
   return s;
  }

  public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss eleme
  int i,j,k,p,ii,jj;
  System.out.println("Original matrix :\n"+toString(a));
  for(k=0;k<(n-1);k++)
  { //pivoting
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
              System.out.println("pivoted matrix : k="+k+"\n"+toString(a));
              // Solving gauss elimination
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
```

```
        for(j=k+1;j<n;j++)
        {   a[i][j]-=carpan*a[k][j]; }
            b[i]   =b[i]   -carpan*b[k];
                  }
   }
   System.out.println("final upper triangular matrix : \n"+toString(a));
   // back substitution
   x[n-1]=b[n-1]/a[n-1][n-1];
   for(i=n-2;i>=0;i--)
   {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
   }
   return x;
   }

   public static void main(String args[]) throws IOException
   {
   double a[][]={{-4,2,2,0,0,0},{1,-4,0,2,0,0},{1,0,-4,2,1,0},{0,1,1,-4,0,1},{0,0,1,0,-4,2},{0,0,0,1,1,-4}};
   Text.print(a);
   double b[]={0,-100,0,-86.66,0,-50};
   Text.printT(b);
    double x[]=pivotlugauss(a,b);
    System.out.println(toString(x));
   }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" NA15
Original matrix :
  -4.00000   2.00000   2.00000   0.00000   0.00000   0.00000
   1.00000  -4.00000   0.00000   2.00000   0.00000   0.00000
   1.00000   0.00000  -4.00000   2.00000   1.00000   0.00000
   0.00000   1.00000   1.00000  -4.00000   0.00000   1.00000
   0.00000   0.00000   1.00000   0.00000  -4.00000   2.00000
   0.00000   0.00000   0.00000   1.00000   1.00000  -4.00000

pivoted matrix : k=0
  -4.00000   2.00000   2.00000   0.00000   0.00000   0.00000
   1.00000  -4.00000   0.00000   2.00000   0.00000   0.00000
   1.00000   0.00000  -4.00000   2.00000   1.00000   0.00000
   0.00000   1.00000   1.00000  -4.00000   0.00000   1.00000
   0.00000   0.00000   1.00000   0.00000  -4.00000   2.00000
   0.00000   0.00000   0.00000   1.00000   1.00000  -4.00000

pivoted matrix : k=1
  -4.00000   2.00000   2.00000   0.00000   0.00000   0.00000
   0.00000  -3.50000   0.500000   2.00000   0.00000   0.00000
   0.00000   0.500000  -3.50000   2.00000   1.00000   0.00000
   0.00000   1.00000   1.00000  -4.00000   0.00000   1.00000
   0.00000   0.00000   1.00000   0.00000  -4.00000   2.00000
   0.00000   0.00000   0.00000   1.00000   1.00000  -4.00000

pivoted matrix : k=2
  -4.00000   2.00000   2.00000   0.00000   0.00000   0.00000
   0.00000  -3.50000   0.500000   2.00000   0.00000   0.00000
   0.00000   0.00000  -3.42857   2.28571   1.00000   0.00000
   0.00000   0.00000   1.14286  -3.42857   0.00000   1.00000
   0.00000   0.00000   1.00000   0.00000  -4.00000   2.00000
   0.00000   0.00000   0.00000   1.00000   1.00000  -4.00000

pivoted matrix : k=3
  -4.00000   2.00000   2.00000   0.00000   0.00000   0.00000
   0.00000  -3.50000   0.500000   2.00000   0.00000   0.00000
   0.00000   0.00000  -3.42857   2.28571   1.00000   0.00000
   0.00000   0.00000   0.00000  -2.66667   0.333333   1.00000
   0.00000   0.00000   0.00000   0.666667  -3.70833   2.00000
   0.00000   0.00000   0.00000   1.00000   1.00000  -4.00000

pivoted matrix : k=4
  -4.00000   2.00000   2.00000   0.00000   0.00000   0.00000
   0.00000  -3.50000   0.500000   2.00000   0.00000   0.00000
   0.00000   0.00000  -3.42857   2.28571   1.00000   0.00000
   0.00000   0.00000   0.00000  -2.66667   0.333333   1.00000
   0.00000   0.00000   0.00000   0.00000  -3.62500   2.25000
   0.00000   0.00000   0.00000   0.00000   1.12500  -3.62500
```

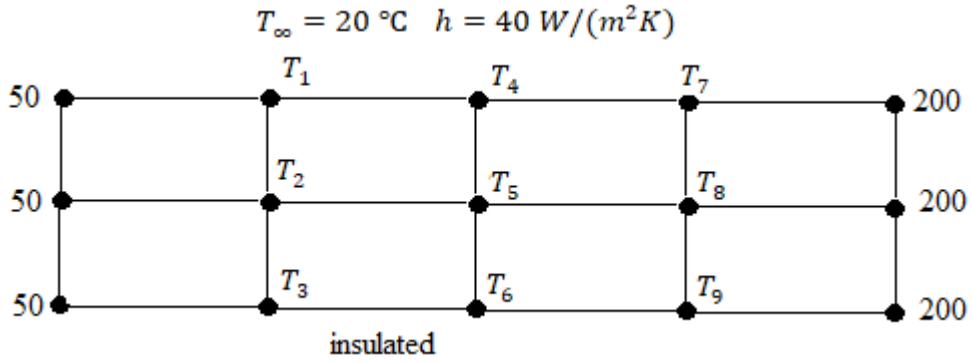
final upper triangular matrix :

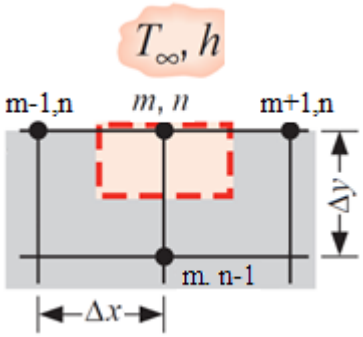
-4.00000	2.00000	2.00000	0.00000	0.00000	0.00000
0.00000	-3.50000	0.500000	2.00000	0.00000	0.00000
0.00000	0.00000	-3.42857	2.28571	1.00000	0.00000
0.00000	0.00000	0.00000	-2.66667	0.333333	1.00000
0.00000	0.00000	0.00000	0.00000	-3.62500	2.25000
0.00000	0.00000	0.00000	0.00000	0.00000	-2.92672

63.6404 72.1629 55.1179 62.5056 31.8202 36.0814

PROBLEM 4.8 FINITE DIFFERENCE SOLUTION, HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CARTESIAN COORDINATES, TWO DIMENSIONS

Consider the steady state problem given below. The Length is L=0.4 m and the width is 0.2 m. Temperature of left side 50 ℃, right side temperature is 200 ℃. Bottom side is insulated. Top side has convective heat transfer into an ambient of $T_\infty = 20$ ℃ with thermal convectivity h=40 W/(m^{2}K). Thermal conductivity k=3 W/(mK). Calculate inner temperature profile. $\Delta x = \Delta y = 0.1\ m$





$$h\Delta x\left(T_\infty - T_{m,n}\right) + k\frac{\Delta y}{2}\frac{\left(T_{m-1,n} - T_{m,n}\right)}{\Delta x} + k\frac{\Delta y}{2}\frac{\left(T_{m+1,n} - T_{m,n}\right)}{\Delta x} + k\Delta x\frac{\left(T_{m,n-1} - T_{m,n}\right)}{\Delta x}$$

If $\Delta x = \Delta y$

$$\frac{2\Delta xh}{k}\left(T_\infty - T_{m,n}\right) + T_{m+1,n} + T_{m-1,n} + 2T_{m,n-1} - 4T_{m,n} = 0$$

$$\frac{2\Delta xhT_\infty}{k} + T_{m+1,n} + T_{m-1,n} + 2T_{m,n-1} - \left(\frac{2\Delta xh}{k} + 4\right)T_{m,n} = 0$$

$\frac{2\Delta xh}{k}$ =2.666667

$\frac{2\Delta xh}{k} + 4$ =6.666667

$\frac{2\Delta xhT_\infty}{k}$ =53.33333

1: $53.33333 - 6.666667T_1 + 50 + T_4 + 2T_2 = 0$

2: $-4T_2 + 50 + T_1 + T_3 + T_5 = 0$

3: $-4T_3 + 50 + 2T_2 + T_6 = 0$

4: $53.33333 - 6.666667T_4 + T_1 + T_7 + 2T_5 = 0$

5: $-4T_5 + T_2 + T_4 + T_6 + T_8 = 0$

6: $:-4T_6 + T_3 + 2T_5 + T_9 = 0$

7: $53.33333 - 6.666667T_7 + 200 + T_4 + 2T_8 = 0$

8: $-4T_8 + T_5 + T_7 + T_9 + 200 = 0$

9: $-4T_9 + T_6 + 2T_8 + 200 = 0$

1	2	3	4	5	6	7	8	9		
-6.66667	2	0	1	0	0	0	0	0	T1	-103.333
1	-4	1	0	1	0	0	0	0	T2	-50
0	2	-4	0	0	1	0	0	0	T3	-50
1	0	0	-6.66667	2	0	1	0	0	T4	-53.3333
0	1	0	1	-4	1	0	1	0	T5	
0	0	1	0	2	-4	0	0	1	T6	
0	0	0	1	0	0	-6.66667	2	0	T7	-253.333
0	0	0	0	1	0	1	-4	1	T8	-200
0	0	0	0	0	1	0	2	-4	T9	-200

```
import java.io.*;
import java.util.Locale;
class NA15
{ public static String toString(double a[])
  {int n=a.length;
   String s="";
   for(int i=0;i<n;i++)
   {s+=String.format(Locale.US,"%10g",a[i])+" ";}
   return s;
  }
  public static String toString(double a[][])
  {int n=a.length;
   int m=a[0].length;
   String s="";
   for(int i=0;i<n;i++)
   { for(int j=0;j<m;j++)
                {s+=String.format(Locale.US,"%10g",a[i][j])+" ";}
                s+="\n";
   }
   return s;
  }

  public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss eleme
  int i,j,k,p,ii,jj;
  System.out.println("Original matrix :\n"+toString(a));
  for(k=0;k<(n-1);k++)
  { //pivoting
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
              System.out.println("pivoted matrix : k="+k+"\n"+toString(a));
              // Solving gauss elimination
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
```

```
          b[i]   =b[i]   -carpan*b[k];
                }
  }
  System.out.println("final upper triangular matrix : \n"+toString(a));
  // back substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
   toplam=0;
   for(j=i+1;j<n;j++)
   {  toplam+=a[i][j]*x[j];}
   x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }

  public static void main(String args[]) throws IOException
  {
  double a[][]={{-6.66667,2,0,1,0,0,0,0,0},
  {1,-4,1,0,1,0,0,0,0},
{0,2,-4,0,0,1,0, 0,0},
{1,0,0,-6.66667,2,0,1,0,0},
{0,1,0,1,-4,1,0,1,0},
{0,0,1,0,2, -4,0,0,1},
{0,0,0,1,0, 0,-6.66667,2,0},
{0,0,0,0,1, 0,1,-4,1},
{0,0,0,0,0,1,0,2,-4}};
   Text.print(a);
   double b[]={-103.333,
-50,
-50,
-53.3333,
0,
0,
-253.333,
-200,
-200};
   Text.printT(b);
    double x[]=pivotlugauss(a,b);
    System.out.println(toString(x));
  }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" NA15
Original matrix :
  -6.66667   2.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
   1.00000  -4.00000   1.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000
   0.00000   2.00000  -4.00000   0.00000   0.00000   1.00000   0.00000   0.00000   0.00000
   1.00000   0.00000   0.00000  -6.66667   2.00000   0.00000   1.00000   0.00000   0.00000
   0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000   0.00000   1.00000   0.00000
   0.00000   0.00000   1.00000   0.00000   2.00000  -4.00000   0.00000   0.00000   1.00000
   0.00000   0.00000   0.00000   1.00000   0.00000   0.00000  -6.66667   2.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000
   0.00000   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   2.00000  -4.00000

pivoted matrix : k=0
  -6.66667   2.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
   1.00000  -4.00000   1.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000
   0.00000   2.00000  -4.00000   0.00000   0.00000   1.00000   0.00000   0.00000   0.00000
   1.00000   0.00000   0.00000  -6.66667   2.00000   0.00000   1.00000   0.00000   0.00000
   0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000   0.00000   1.00000   0.00000
   0.00000   0.00000   1.00000   0.00000   2.00000  -4.00000   0.00000   0.00000   1.00000
   0.00000   0.00000   0.00000   1.00000   0.00000   0.00000  -6.66667   2.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000
   0.00000   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   2.00000  -4.00000

pivoted matrix : k=1
  -6.66667   2.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
   0.00000  -3.70000   1.00000   0.150000  1.00000   0.00000   0.00000   0.00000   0.00000
   0.00000   2.00000  -4.00000   0.00000   0.00000   1.00000   0.00000   0.00000   0.00000
   0.00000   0.300000  0.00000  -6.51667   2.00000   0.00000   1.00000   0.00000   0.00000
   0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000   0.00000   1.00000   0.00000
   0.00000   0.00000   1.00000   0.00000   2.00000  -4.00000   0.00000   0.00000   1.00000
   0.00000   0.00000   0.00000   1.00000   0.00000   0.00000  -6.66667   2.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000
   0.00000   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   2.00000  -4.00000
```

```
pivoted matrix : k=2
  -6.66667   2.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
   0.00000  -3.70000   1.00000   0.150000  1.00000   0.00000   0.00000   0.00000   0.00000
   0.00000   0.00000  -3.45946   0.0810810 0.540541  1.00000   0.00000   0.00000   0.00000
   0.00000   0.00000   0.0810810 -6.50451  2.08108   0.00000   1.00000   0.00000   0.00000
   0.00000   0.00000   0.270270  1.04054  -3.72973   1.00000   0.00000   1.00000   0.00000
   0.00000   0.00000   1.00000   0.00000   2.00000  -4.00000   0.00000   0.00000   1.00000
   0.00000   0.00000   0.00000   1.00000   0.00000   0.00000  -6.66667   2.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000
   0.00000   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   2.00000  -4.00000

pivoted matrix : k=3
  -6.66667   2.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
   0.00000  -3.70000   1.00000   0.150000  1.00000   0.00000   0.00000   0.00000   0.00000
   0.00000   0.00000  -3.45946   0.0810810 0.540541  1.00000   0.00000   0.00000   0.00000
   0.00000   0.00000   0.00000  -6.50261   2.09375   0.0234375 1.00000   0.00000   0.00000
   0.00000   0.00000   0.00000   1.04687  -3.68750   1.07812   0.00000   1.00000   0.00000
   0.00000   0.00000   0.00000   0.0234375 2.15625  -3.71094   0.00000   0.00000   1.00000
   0.00000   0.00000   0.00000   1.00000   0.00000   0.00000  -6.66667   2.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000
   0.00000   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   2.00000  -4.00000

pivoted matrix : k=4
  -6.66667   2.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
   0.00000  -3.70000   1.00000   0.150000  1.00000   0.00000   0.00000   0.00000   0.00000
   0.00000   0.00000  -3.45946   0.0810810 0.540541  1.00000   0.00000   0.00000   0.00000
   0.00000   0.00000   0.00000  -6.50261   2.09375   0.0234375 1.00000   0.00000   0.00000
   0.00000   0.00000   0.00000   0.00000  -3.35042   1.08190   0.160993  1.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   2.16380  -3.71085   0.00360432 0.00000  1.00000
   0.00000   0.00000   0.00000   0.00000   0.321986  0.00360432 -6.51289  2.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000
   0.00000   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   2.00000  -4.00000

pivoted matrix : k=5
  -6.66667   2.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
   0.00000  -3.70000   1.00000   0.150000  1.00000   0.00000   0.00000   0.00000   0.00000
   0.00000   0.00000  -3.45946   0.0810810 0.540541  1.00000   0.00000   0.00000   0.00000
   0.00000   0.00000   0.00000  -6.50261   2.09375   0.0234375 1.00000   0.00000   0.00000
   0.00000   0.00000   0.00000   0.00000  -3.35042   1.08190   0.160993  1.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   0.00000  -3.01213   0.107578  0.645828  1.00000
   0.00000   0.00000   0.00000   0.00000   0.00000   0.107578 -6.49741   2.09610   0.00000
   0.00000   0.00000   0.00000   0.00000   0.00000   0.322914  1.04805  -3.70153   1.00000
   0.00000   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   2.00000  -4.00000

pivoted matrix : k=6
  -6.66667   2.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
   0.00000  -3.70000   1.00000   0.150000  1.00000   0.00000   0.00000   0.00000   0.00000
   0.00000   0.00000  -3.45946   0.0810810 0.540541  1.00000   0.00000   0.00000   0.00000
   0.00000   0.00000   0.00000  -6.50261   2.09375   0.0234375 1.00000   0.00000   0.00000
   0.00000   0.00000   0.00000   0.00000  -3.35042   1.08190   0.160993  1.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   0.00000  -3.01213   0.107578  0.645828  1.00000
   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000  -6.49357   2.11917   0.0357150
   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   1.05958  -3.63229   1.10720
   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   0.0357150 2.21441  -3.66801

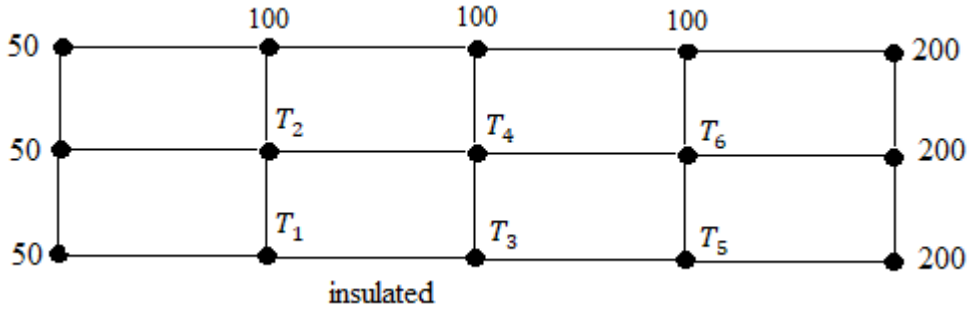
pivoted matrix : k=7
  -6.66667   2.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
   0.00000  -3.70000   1.00000   0.150000  1.00000   0.00000   0.00000   0.00000   0.00000
   0.00000   0.00000  -3.45946   0.0810810 0.540541  1.00000   0.00000   0.00000   0.00000
   0.00000   0.00000   0.00000  -6.50261   2.09375   0.0234375 1.00000   0.00000   0.00000
   0.00000   0.00000   0.00000   0.00000  -3.35042   1.08190   0.160993  1.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   0.00000  -3.01213   0.107578  0.645828  1.00000
   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000  -6.49357   2.11917   0.0357150
   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000  -3.28650   1.11303
   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   2.22606  -3.66781

final upper triangular matrix :
  -6.66667   2.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
   0.00000  -3.70000   1.00000   0.150000  1.00000   0.00000   0.00000   0.00000   0.00000
   0.00000   0.00000  -3.45946   0.0810810 0.540541  1.00000   0.00000   0.00000   0.00000
   0.00000   0.00000   0.00000  -6.50261   2.09375   0.0234375 1.00000   0.00000   0.00000
   0.00000   0.00000   0.00000   0.00000  -3.35042   1.08190   0.160993  1.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   0.00000  -3.01213   0.107578  0.645828  1.00000
   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000  -6.49357   2.11917   0.0357150
   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000  -3.28650   1.11303
   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000  -2.91392
```

40.9656	59.3449	64.8688	51.0814	81.5454	90.7852	83.1531	124.970	135.181

PROBLEM 4.9 FINITE DIFFERENCE SOLUTION, HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CARTESIAN COORDINATES, TWO DIMENSIONS

Consider the steady state problem given below. The Length is L=0.4 m and the width is 0.2 m. Temperature of left side 50 ℃, right side temperature is 200 ℃. Bottom side is insulated. Top side side temperature is 100 ℃. Thermal conductivity k=3 W/(mK). Calculate inner temperature profile. $\Delta x = \Delta y = 0.1\ m$



1: $-4T_1 + 2T_2 + 50 + T_3 = 0$
2: $T_1 - 4T_2 + 50 + 100 + T_4 = 0$
3: $T_1 - 4T_3 + 2T_4 + T_5 = 0$
4: $T_2 + T_3 - 4T_4 + 100 + T_6 = 0$
5: $T_3 - 4T_5 + 2T_6 + 200 = 0$
6: $T_4 + T_5 - 4T_6 + 100 = 0$

1	2	3	4	5	6		
-4	2	1	0	0	0	T1	-50
1	-4	0	1	0	0	T2	-150
1	0	-4	2	1	0	T3	0
0	1	1	-4	0	1	T4	0
0	0	1	0	-4	2	T5	-200
0	0	0	1	1	-4	T6	-100

```
import java.io.*;
import java.util.Locale;
class NA15E
{ public static String toString(double a[])
  {int n=a.length;
   String s="";
   for(int i=0;i<n;i++)
   {s+=String.format(Locale.US,"%10g",a[i])+" ";}
   return s;
  }
  public static String toString(double a[][])
  {int n=a.length;
   int m=a[0].length;
   String s="";
   for(int i=0;i<n;i++)
   { for(int j=0;j<m;j++)
                {s+=String.format(Locale.US,"%10g",a[i][j])+" ";}
                s+="\n";
   }
   return s;
  }

  public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
```

```
  double buyuk;
  double dummy=0;
  //gauss eleme
  int i,j,k,p,ii,jj;
  System.out.println("Original matrix :\n"+toString(a));
  for(k=0;k<(n-1);k++)
  { //pivoting
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
              System.out.println("pivoted matrix : k="+k+"\n"+toString(a));
              // Solving gauss elimination
   for(i=k+1;i<n;i++)
   {  carpan=a[i][k]/a[k][k];
      a[i][k]=0;
      for(j=k+1;j<n;j++)
      {   a[i][j]-=carpan*a[k][j]; }
         b[i]   =b[i]   -carpan*b[k];
              }
  }
  System.out.println("final upper triangular matrix : \n"+toString(a));
  // back substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
   toplam=0;
   for(j=i+1;j<n;j++)
   {  toplam+=a[i][j]*x[j];}
   x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }

  public static void main(String args[]) throws IOException
  {
   double a[][]={{-4,2,1,0,0,0},
{1,-4,0,1,0,0},
{1, 0,-4,2,1,0},
{0,1,1,-4,0,1},
{0,0,1, 0,-4,2},
{0,0,0,1,1,-4}};
  Text.print(a);
  double b[]={-50,-150,0,0,-200,-100};
  Text.printT(b);
   double x[]=pivotlugauss(a,b);
   System.out.println(toString(x));
  }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" NA15E
Original matrix :
  -4.00000    2.00000    1.00000    0.00000    0.00000    0.00000
   1.00000   -4.00000    0.00000    1.00000    0.00000    0.00000
   1.00000    0.00000   -4.00000    2.00000    1.00000    0.00000
   0.00000    1.00000    1.00000   -4.00000    0.00000    1.00000
   0.00000    0.00000    1.00000    0.00000   -4.00000    2.00000
   0.00000    0.00000    0.00000    1.00000    1.00000   -4.00000

pivoted matrix : k=0
  -4.00000    2.00000    1.00000    0.00000    0.00000    0.00000
   1.00000   -4.00000    0.00000    1.00000    0.00000    0.00000
   1.00000    0.00000   -4.00000    2.00000    1.00000    0.00000
   0.00000    1.00000    1.00000   -4.00000    0.00000    1.00000
   0.00000    0.00000    1.00000    0.00000   -4.00000    2.00000
```

```
  0.00000   0.00000   0.00000   1.00000   1.00000  -4.00000

pivoted matrix : k=1
 -4.00000   2.00000   1.00000   0.00000   0.00000   0.00000
  0.00000  -3.50000   0.250000  1.00000   0.00000   0.00000
  0.00000   0.500000 -3.75000   2.00000   1.00000   0.00000
  0.00000   1.00000   1.00000  -4.00000   0.00000   1.00000
  0.00000   0.00000   1.00000   0.00000  -4.00000   2.00000
  0.00000   0.00000   0.00000   1.00000   1.00000  -4.00000

pivoted matrix : k=2
 -4.00000   2.00000   1.00000   0.00000   0.00000   0.00000
  0.00000  -3.50000   0.250000  1.00000   0.00000   0.00000
  0.00000   0.00000  -3.71429   2.14286   1.00000   0.00000
  0.00000   0.00000   1.07143  -3.71429   0.00000   1.00000
  0.00000   0.00000   1.00000   0.00000  -4.00000   2.00000
  0.00000   0.00000   0.00000   1.00000   1.00000  -4.00000

pivoted matrix : k=3
 -4.00000   2.00000   1.00000   0.00000   0.00000   0.00000
  0.00000  -3.50000   0.250000  1.00000   0.00000   0.00000
  0.00000   0.00000  -3.71429   2.14286   1.00000   0.00000
  0.00000   0.00000   0.00000  -3.09615   0.288462  1.00000
  0.00000   0.00000   0.00000   0.576923 -3.73077   2.00000
  0.00000   0.00000   0.00000   1.00000   1.00000  -4.00000

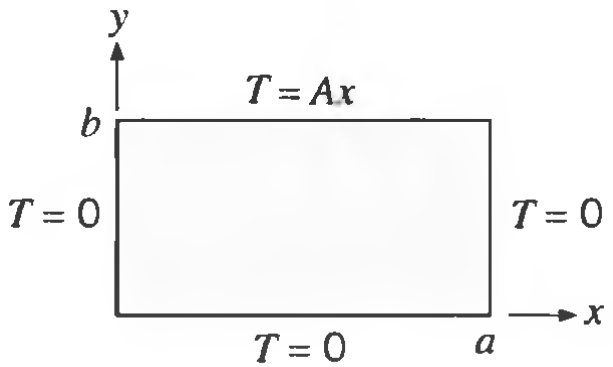
pivoted matrix : k=4
 -4.00000   2.00000   1.00000   0.00000   0.00000   0.00000
  0.00000  -3.50000   0.250000  1.00000   0.00000   0.00000
  0.00000   0.00000  -3.71429   2.14286   1.00000   0.00000
  0.00000   0.00000   0.00000  -3.09615   0.288462  1.00000
  0.00000   0.00000   0.00000   0.00000  -3.67702   2.18634
  0.00000   0.00000   0.00000   0.00000   1.09317  -3.67702

final upper triangular matrix :
 -4.00000   2.00000   1.00000   0.00000   0.00000   0.00000
  0.00000  -3.50000   0.250000  1.00000   0.00000   0.00000
  0.00000   0.00000  -3.71429   2.14286   1.00000   0.00000
  0.00000   0.00000   0.00000  -3.09615   0.288462  1.00000
  0.00000   0.00000   0.00000   0.00000  -3.67702   2.18634
  0.00000   0.00000   0.00000   0.00000   0.00000  -3.02703

 60.2679   64.2857   62.5000   46.8750   95.9821   60.7143
```

PROBLEM 4.10 FINITE DIFFERENCE SOLUTION, HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CARTESIAN COORDINATES, TWO DIMENSIONS

A two dimensional rectangular plate is subjected to the boundary conditions shown in the figure. Derive an expression for the steady-state temperature distribution T(x,y) by using finite difference method



Take: $\Delta x = \Delta y = 0.25$ **L=1 m W=1 m A=100 T=100x**

$$T_{m+1,n} + T_{m-1,n} + T_{m,n+1} + T_{m,n-1} - 4T_{m,n} = 0$$

0	25	50	75	100	
0	T7	T8	T9		0
0	T4	T5	T6		0
0	T1	T2	T3		0
0					0
	0	0	0	0	

The matrix form:

1	2	3	4	5	6	7	8	9		
-4	1	0	1	0	0	0	0	0	T1	0
1	-4	1	0	1	0	0	0	0	T2	0
0	1	-4	0	0	1	0	0	0	T3	0
1	0	0	-4	1	0	1	0	0	T4	0
0	1	0	1	-4	1	0	1	0	T5	0
0	0	1	0	1	-4	0	0	1	T6	0
0	0	0	1	0	0	-4	1	0	T7	-25
0	0	0	0	1	0	1	-4	1	T8	-50
0	0	0	0	0	1	0	1	-4	T9	-75

Gauss elimination method to solve the matrix

```
import java.io.*;
import java.util.Locale;
class NA15
{ public static String toString(double a[])
  {int n=a.length;
   String s="";
   for(int i=0;i<n;i++)
   {s+=String.format(Locale.US,"%10g",a[i])+" ";}
   return s;
  }
  public static String toString(double a[][])
  {int n=a.length;
   int m=a[0].length;
   String s="";
   for(int i=0;i<n;i++)
   { for(int j=0;j<m;j++)
                {s+=String.format(Locale.US,"%10g",a[i][j])+" ";}
                s+="\n";
   }
   return s;
  }

  public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss eleme
  int i,j,k,p,ii,jj;
  System.out.println("Original matrix :\n"+toString(a));
  for(k=0;k<(n-1);k++)
  { //pivoting
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
```

```
            if(p!=k)
            { for(jj=k;jj<n;jj++)
              { dummy=a[p][jj];
                a[p][jj]=a[k][jj];
                a[k][jj]=dummy;
              }
              dummy=b[p];
              b[p]=b[k];
              b[k]=dummy;
            }
            System.out.println("pivoted matrix : k="+k+"\n"+toString(a));
            // Solving gauss elimination
   for(i=k+1;i<n;i++)
   {  carpan=a[i][k]/a[k][k];
      a[i][k]=0;
      for(j=k+1;j<n;j++)
      {   a[i][j]-=carpan*a[k][j]; }
          b[i]   =b[i]   -carpan*b[k];
            }
  }
  System.out.println("final upper triangular matrix : \n"+toString(a));
  // back substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }

  public static void main(String args[]) throws IOException
  {
 double a[][]={{-4.0,1.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0},
{1.0,-4.0,1.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0},
{0.0,1.0,-4.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0},
{1.0,0.0,0.0,-4.0,1.0,0.0,1.0,0.0,0.0,0.0},
{0.0,1.0,0.0,1.0,-4.0,1.0,0.0,1.0,0.0,0.0},
{0.0,0.0,1.0,0.0,1.0,-4.0,0.0,0.0,1.0},
{0.0,0.0,0.0,1.0,0.0,0.0,0.0,-4.0,1.0,0.0},
{0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,1.0,-4.0,1.0},
{0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,1.0,-4.0}};
  Text.print(a);
  double b[]={0.0,0.0,0.0,0.0,0.0,0.0,0.0,-25.0,-50.0,-75.0};
  Text.printT(b);
   double x[]=pivotlugauss(a,b);
   System.out.println(toString(x));
  }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" NA15
Original matrix :
  -4.00000   1.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
   1.00000  -4.00000   1.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000
   0.00000   1.00000  -4.00000   0.00000   0.00000   1.00000   0.00000   0.00000   0.00000
   1.00000   0.00000   0.00000  -4.00000   1.00000   0.00000   1.00000   0.00000   0.00000
   0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000   0.00000   1.00000   0.00000
   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000   0.00000   0.00000   1.00000
   0.00000   0.00000   0.00000   1.00000   0.00000   0.00000  -4.00000   1.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000
   0.00000   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000

pivoted matrix : k=0
  -4.00000   1.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
   1.00000  -4.00000   1.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000
   0.00000   1.00000  -4.00000   0.00000   0.00000   1.00000   0.00000   0.00000   0.00000
   1.00000   0.00000   0.00000  -4.00000   1.00000   0.00000   1.00000   0.00000   0.00000
   0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000   0.00000   1.00000   0.00000
   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000   0.00000   0.00000   1.00000
   0.00000   0.00000   0.00000   1.00000   0.00000   0.00000  -4.00000   1.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000
   0.00000   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000

pivoted matrix : k=1
  -4.00000   1.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
```

```
  0.00000  -3.75000   1.00000   0.250000   1.00000   0.00000   0.00000   0.00000   0.00000
  0.00000   1.00000  -4.00000   0.00000    0.00000   1.00000   0.00000   0.00000   0.00000
  0.00000   0.250000   0.00000  -3.75000   1.00000   0.00000   1.00000   0.00000   0.00000
  0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000   0.00000   1.00000   0.00000
  0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000   0.00000   0.00000   1.00000
  0.00000   0.00000   0.00000   1.00000   0.00000   0.00000  -4.00000   1.00000   0.00000
  0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000
  0.00000   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000

pivoted matrix : k=2
 -4.00000   1.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
  0.00000  -3.75000   1.00000   0.250000  1.00000   0.00000   0.00000   0.00000   0.00000
  0.00000   0.00000  -3.73333   0.0666667 0.266667  1.00000   0.00000   0.00000   0.00000
  0.00000   0.00000   0.0666667 -3.73333  1.06667   0.00000   1.00000   0.00000   0.00000
  0.00000   0.00000   0.266667  1.06667  -3.73333   1.00000   0.00000   1.00000   0.00000
  0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000   0.00000   0.00000   1.00000
  0.00000   0.00000   0.00000   1.00000   0.00000   0.00000  -4.00000   1.00000   0.00000
  0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000
  0.00000   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000

pivoted matrix : k=3
 -4.00000   1.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
  0.00000  -3.75000   1.00000   0.250000  1.00000   0.00000   0.00000   0.00000   0.00000
  0.00000   0.00000  -3.73333   0.0666667 0.266667  1.00000   0.00000   0.00000   0.00000
  0.00000   0.00000   0.00000  -3.73214   1.07143   0.0178571 1.00000   0.00000   0.00000
  0.00000   0.00000   0.00000   1.07143  -3.71429   1.07143   0.00000   1.00000   0.00000
  0.00000   0.00000   0.00000   0.0178571 1.07143  -3.73214   0.00000   0.00000   1.00000
  0.00000   0.00000   0.00000   1.00000   0.00000   0.00000  -4.00000   1.00000   0.00000
  0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000
  0.00000   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000

pivoted matrix : k=4
 -4.00000   1.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
  0.00000  -3.75000   1.00000   0.250000  1.00000   0.00000   0.00000   0.00000   0.00000
  0.00000   0.00000  -3.73333   0.0666667 0.266667  1.00000   0.00000   0.00000   0.00000
  0.00000   0.00000   0.00000  -3.73214   1.07143   0.0178571 1.00000   0.00000   0.00000
  0.00000   0.00000   0.00000   0.00000  -3.40670   1.07656   0.287081  1.00000   0.00000
  0.00000   0.00000   0.00000   0.00000   1.07656  -3.73206   0.00478469 0.00000  1.00000
  0.00000   0.00000   0.00000   0.00000   0.287081  0.00478469 -3.73206  1.00000   0.00000
  0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000   1.00000
  0.00000   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000

pivoted matrix : k=5
 -4.00000   1.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
  0.00000  -3.75000   1.00000   0.250000  1.00000   0.00000   0.00000   0.00000   0.00000
  0.00000   0.00000  -3.73333   0.0666667 0.266667  1.00000   0.00000   0.00000   0.00000
  0.00000   0.00000   0.00000  -3.73214   1.07143   0.0178571 1.00000   0.00000   0.00000
  0.00000   0.00000   0.00000   0.00000  -3.40670   1.07656   0.287081  1.00000   0.00000
  0.00000   0.00000   0.00000   0.00000   0.00000  -3.39185   0.0955056 0.316011  1.00000
  0.00000   0.00000   0.00000   0.00000   0.00000   0.0955056 -3.70787  1.08427   0.00000
  0.00000   0.00000   0.00000   0.00000   0.00000   0.316011  1.08427  -3.70646   1.00000
  0.00000   0.00000   0.00000   0.00000   0.00000   1.00000   0.00000   1.00000  -4.00000

pivoted matrix : k=6
 -4.00000   1.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
  0.00000  -3.75000   1.00000   0.250000  1.00000   0.00000   0.00000   0.00000   0.00000
  0.00000   0.00000  -3.73333   0.0666667 0.266667  1.00000   0.00000   0.00000   0.00000
  0.00000   0.00000   0.00000  -3.73214   1.07143   0.0178571 1.00000   0.00000   0.00000
  0.00000   0.00000   0.00000   0.00000  -3.40670   1.07656   0.287081  1.00000   0.00000
  0.00000   0.00000   0.00000   0.00000   0.00000  -3.39185   0.0955056 0.316011  1.00000
  0.00000   0.00000   0.00000   0.00000   0.00000   0.00000  -3.70518  1.09317   0.0281573
  0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   1.09317  -3.67702   1.09317
  0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   0.0281573 1.09317  -3.70518

pivoted matrix : k=7
 -4.00000   1.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
  0.00000  -3.75000   1.00000   0.250000  1.00000   0.00000   0.00000   0.00000   0.00000
  0.00000   0.00000  -3.73333   0.0666667 0.266667  1.00000   0.00000   0.00000   0.00000
  0.00000   0.00000   0.00000  -3.73214   1.07143   0.0178571 1.00000   0.00000   0.00000
  0.00000   0.00000   0.00000   0.00000  -3.40670   1.07656   0.287081  1.00000   0.00000
  0.00000   0.00000   0.00000   0.00000   0.00000  -3.39185   0.0955056 0.316011  1.00000
  0.00000   0.00000   0.00000   0.00000   0.00000   0.00000  -3.70518  1.09317   0.0281573
  0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000  -3.35449   1.10148
  0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000   1.10148  -3.70496

final upper triangular matrix :
 -4.00000   1.00000   0.00000   1.00000   0.00000   0.00000   0.00000   0.00000   0.00000
```

```
  0.00000   -3.75000    1.00000   0.250000    1.00000    0.00000    0.00000    0.00000    0.00000
  0.00000    0.00000   -3.73333  0.0666667   0.266667    1.00000    0.00000    0.00000    0.00000
  0.00000    0.00000    0.00000   -3.73214    1.07143  0.0178571    1.00000    0.00000    0.00000
  0.00000    0.00000    0.00000    0.00000   -3.40670    1.07656   0.287081    1.00000    0.00000
  0.00000    0.00000    0.00000    0.00000    0.00000   -3.39185  0.0955056   0.316011    1.00000
  0.00000    0.00000    0.00000    0.00000    0.00000    0.00000   -3.70518    1.09317  0.0281573
  0.00000    0.00000    0.00000    0.00000    0.00000    0.00000    0.00000   -3.35449    1.10148
  0.00000    0.00000    0.00000    0.00000    0.00000    0.00000    0.00000    0.00000   -3.34328

  3.12500    4.91071    4.01786    7.58929    12.5000    11.1607    14.7321    26.3393    28.1250
> Terminated with exit code 0.
```

Another program (create matrix then solve it)

```
*
import java.io.*;
import visad.*;
import visad.java3d.DisplayImplJ3D;
import visad.java2d.DisplayImplJ2D;
import java.rmi.RemoteException;
*/
class HT_plane_FD_TX
{
  public static double [] gauss_seidel(double a[][],double b[],double x[],double lambda)
  {
  //denklem sistemi çözümü b iteratif gauss_seidel metodu metodu
  //yakınsama faktörü lambda ile birlikte verilmiş formül
  int n=a.length;
  int imax=1000;//maksimum iterasyon sayısı
  double es=1.0e-10;
  double dummy=0;
  double ea=1e-1;
  double sum=0;
  double old=0;
  int iter=0;
  int sentinel=0;
  //System.out.print("iter = "+(iter+1)+" ");
  for(int i=0;i<n;i++)
  {  dummy=a[i][i];
    for(int j=0;j<n;j++)
      {a[i][j]/=dummy;}
    b[i]/=dummy;
  }
  for(int i=0;i<n;i++)
  {  sum=b[i];
    for(int j=0;j<n;j++)
      {if(i!=j)sum-=a[i][j]*x[j];}
    x[i]=sum;
    //System.out.print(x[i]+ "   ");
  }
   //System.out.println();
  iter++;
  do
  {
  sentinel=1;
  //System.out.print("iter = "+(iter+1)+" ");
  for(int i=0;i<n;i++)
    {
                old=x[i];
                sum=b[i];
                for(int j=0;j<n;j++)
                   {if(i!=j) sum-=a[i][j]*x[j];}
                   x[i]=lambda*sum+(1.0-lambda)*old;
                   if(sentinel==1 && x[i]!=0.0 ) {ea=Math.abs((x[i]-old)/x[i])*100.0;}
                   if(ea<es) sentinel=0;
                 //System.out.print(x[i]+ "   ");
    }
  //System.out.println();
  iter++;
  }while(sentinel==1 && iter<imax);
  if(iter>=imax) System.out.println("Maksimum iterasyon sayısı aşıldı sonuç geçerli olmayabilir iter = "+iter);
  return x;
  }
```

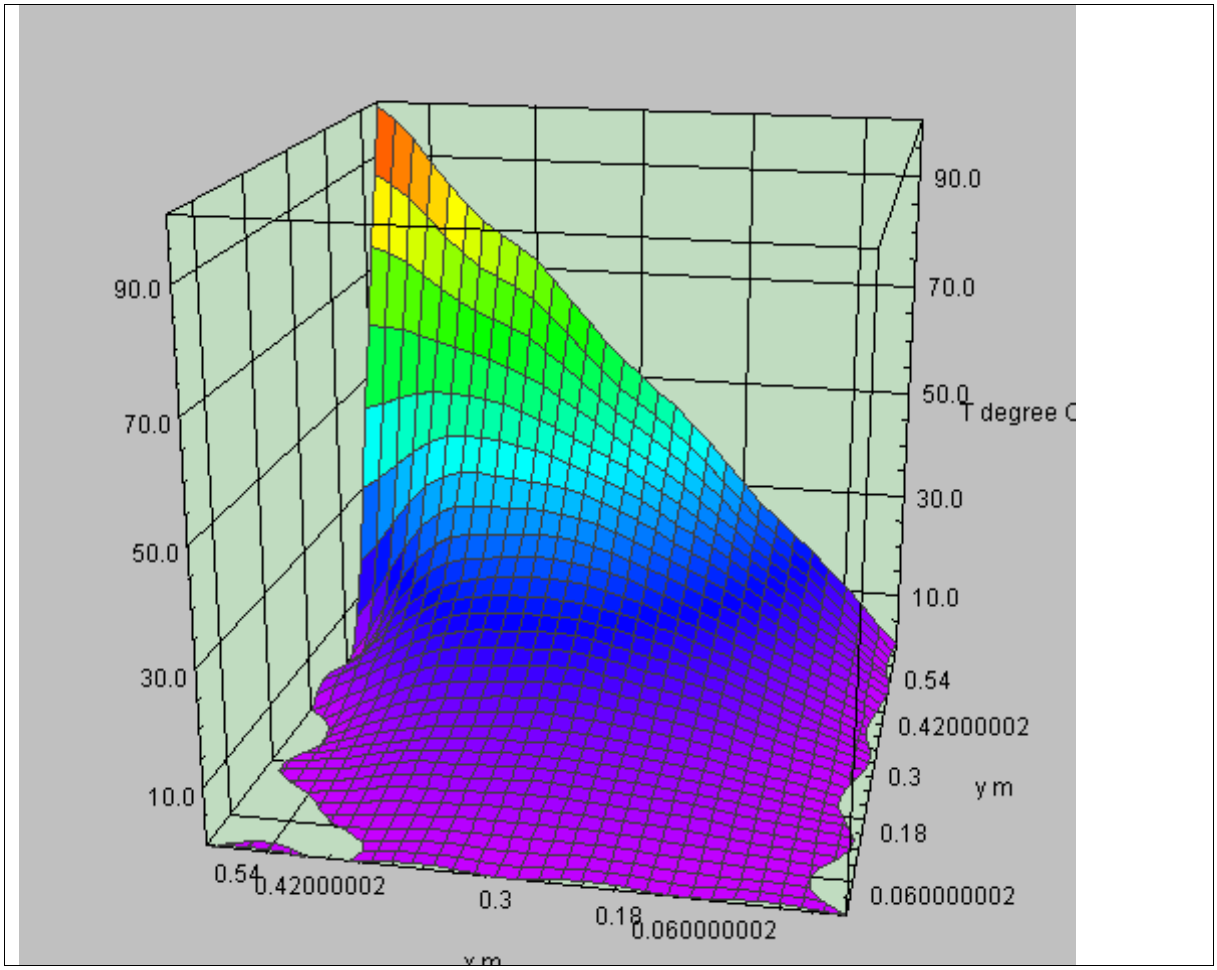
```
public static double[] pivotlugauss(double a[][],double b[])
{ //kısmi pivotlu gauss eleme yöntemi
int n=b.length;
double x[]=new double[n];
double carpan=0;
double toplam=0;
double buyuk;
double dummy=0;
//gauss eleme
int i,j,k,p,ii,jj;
//System.out.println("Orijinal Matris :\n"+Matrix.toString(a));
for(k=0;k<(n-1);k++)
{ //pivotlama
            p=k;
            buyuk=Math.abs(a[k][k]);
            for(ii=k+1;ii<n;ii++)
            {  dummy=Math.abs(a[ii][k]);
               if(dummy > buyuk) {buyuk=dummy;p=ii;}
            }
            if(p!=k)
            { for(jj=k;jj<n;jj++)
              { dummy=a[p][jj];
                a[p][jj]=a[k][jj];
                a[k][jj]=dummy;
              }
              dummy=b[p];
              b[p]=b[k];
              b[k]=dummy;
            }
              //System.out.println("pivotlu Matris : k="+k+"\n"+Matrix.toString(a));
            //gauss elemeyi çözme
  for(i=k+1;i<n;i++)
  {  carpan=a[i][k]/a[k][k];
     a[i][k]=0;
     for(j=k+1;j<n;j++)
     {   a[i][j]-=carpan*a[k][j]; }
         b[i]   =b[i]   -carpan*b[k];
            }
}
//back substitution
x[n-1]=b[n-1]/a[n-1][n-1];
for(i=n-2;i>=0;i--)
{
  toplam=0;
  for(j=i+1;j<n;j++)
  {  toplam+=a[i][j]*x[j];}
  x[i]=(b[i]-toplam)/a[i][i];
}
return x;
}


public static double [][] HT_planecoz( double L,double W,double dx, double Tx0,double Txn,double Ty0,double AA)
{
int nx=(int)(L/dx+0.01);
int ny=(int)(W/dx+0.01);
int mx=(nx-1);
int my=(ny-1);
int mt=mx*my;
double A[][]=new double[mt][mt];
double b[]=new double[mt];
double x[]=new double[mt];
double x1,x2,y1,y2;
double Tyn=0;
// boundary conditions
int i=0,j=0;
for(i=0;i<my;i++)
{
   for(j=0;j<mx;j++)
   {
            x1=dx*(j+1);
            x2=(L-x1);
            y1=dx*(i+1);
            Tyn=AA*x1;
            y2=(W-y1);
            int k=i*my+j;
            x[k]=((x1*Tx0+x2*Txn)/L+(y1*Ty0+y2*Tyn)/W)/2.0;
```

```
                int kp1=(i+1)*my+j;
                int km1=(i-1)*my+j;
                if(j==0)          {b[k]+=-Tx0;}
    if(j== (mx-1))     {b[k]+=-Txn;}
    if(i==0)          {b[k]+=-Ty0;}
    if(i==(my-1))  {b[k]+=-Tyn; System.out.println("i="+i+"j="+j+"x1="+x1+"b="+b[k]);}
    A[k][k]+=-4.0;
    if((i==0)&&(j==0)) {A[kp1][k]+=1.0;A[k][k+1]+=1.0;}
    else if((i==0)&&(j==(mx-1))) {A[kp1][k]+=1.0;A[k][k-1]+=1.0;}
    else if((i==(my-1))&&(j==0)) {A[km1][k]+=1.0;A[k][k+1]+=1.0;}
    else if((i==(my-1))&&(j==(mx-1))) {A[km1][k]+=1.0;A[k][k-1]+=1.0;}
    else if(j==0) {A[kp1][k]+=1.0;A[km1][k]+=1.0;A[k][k+1]+=1.0;}
    else if(i==0) {A[k][k+1]+=1.0;A[k][k-1]+=1.0;A[kp1][k]+=1.0;}
    else if(j==(mx-1)) {A[kp1][k]+=1.0;A[km1][k]+=1.0;A[k][k-1]+=1.0;}
    else if(i==(my-1)) {A[km1][k]+=1.0;A[k][k-1]+=1.0;A[k][k+1]+=1.0;}
    else {A[km1][k]+=1.0;A[kp1][k]+=1.0;A[k][k-1]+=1.0;A[k][k+1]+=1.0;}
    }
  }
  Text.print(A,"A");
  Text.print(b,"B");
  double T[][]=new double[ny+1][nx+1];
  double a[]=conjugate_gradient1.conjugate_gradient(A,b);
  for(i=0;i<=ny;i++)
  { for(j=0;j<=nx;j++)
  {  if(i==ny) {T[i][j]=AA*dx*j;}
                else if(j==0) {T[i][j]=Tx0;}
    else if(i==0) {T[i][j]=Ty0;}
    else if(j==nx) {T[i][j]=Txn;}
    else
    {int k=(i-1)*my+(j-1);
     T[i][j]=a[k];
    }
    }

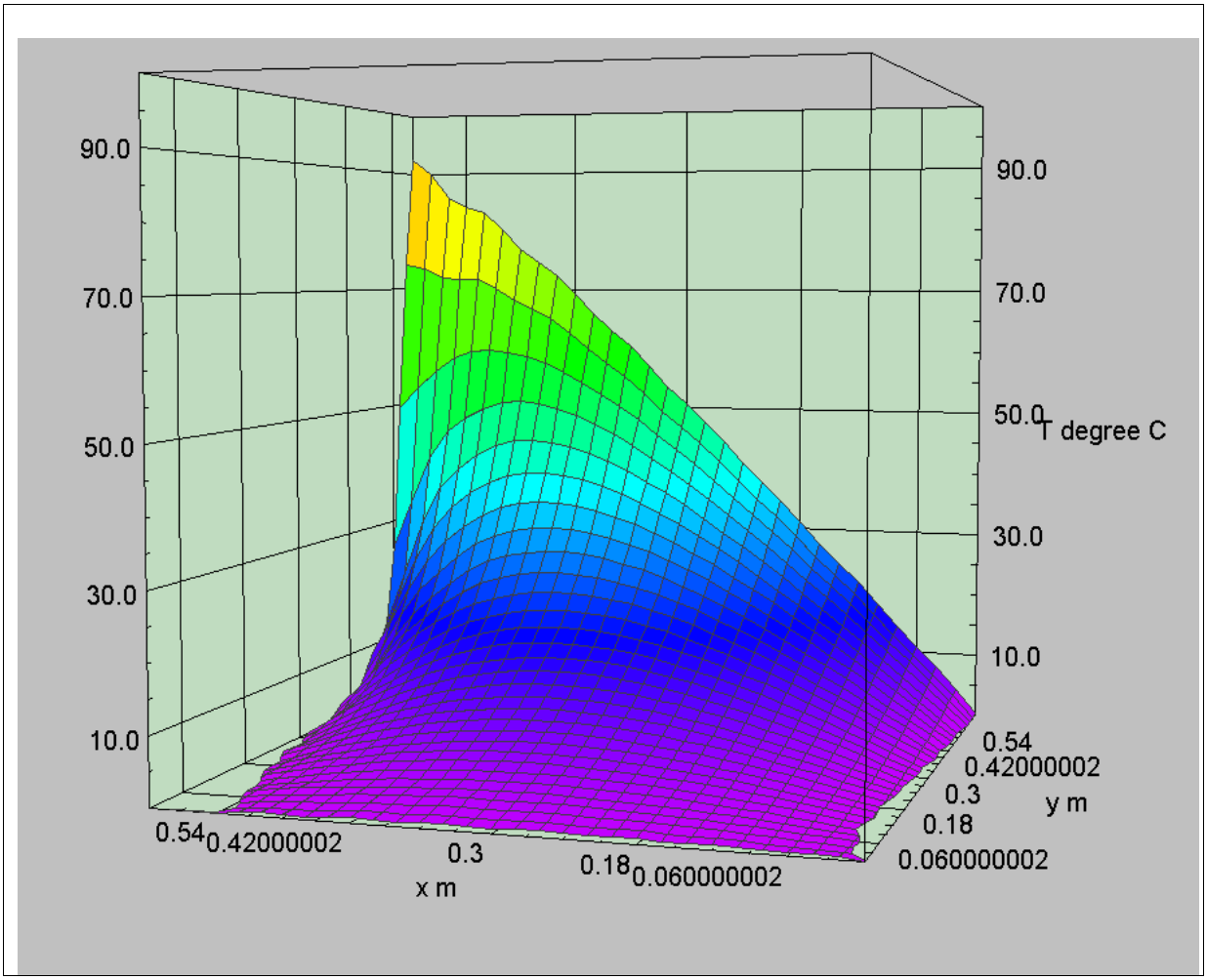
  }
   return T;
}
  public static void main(String args[]) //throws IOException,RemoteException, VisADException
  {  double a[][]=HT_planecoz(1.0,1.0,0.25,0.0,00.0,0.0,100.0);
     Text.print(a);
                //ContourPlot pp=new ContourPlot(a,0.0,0.6,0.0,0.6,40, false,"x","y","T=f(x,y)");
               //pp.plot();
               double b[][]=surfacePlot.dataformat(0.0,0.6,0.0,0.6,a);
                 surfacePlot p1=new surfacePlot(b,"y m","x m","T degree C");
                 p1.plot();
                //Plot3D.data(a,0.0,1.0,0.0,1.0,400,"x","y","T=f(x,y)");
  }
}
```

0	**0**	**0**	**0**	**0**
0	**3.125**	**4.9107143**	**4.0178571**	**0**
0	**7.5892857**	**12.5**	**11.160714**	**0**
0	**14.732143**	**26.339286**	**28.125**	**0**
0	**25**	**50**	**75**	**100**

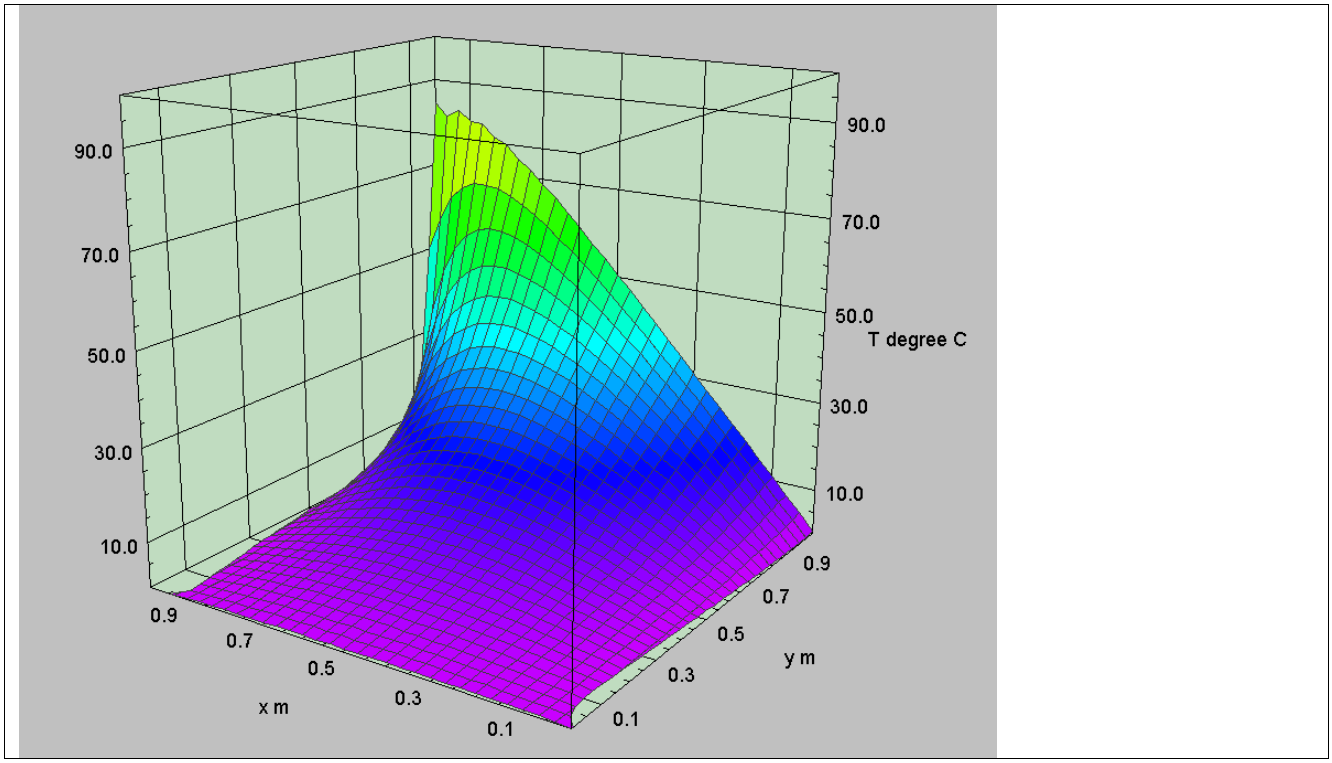


We can increase number of grids

0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.781	1.474	1.989	2.241	2.163	1.723	0.960	0.000
0.000	1.651	3.125	4.240	4.814	4.687	3.770	2.117	0.000
0.000	2.699	5.135	7.031	8.087	8.002	6.552	3.739	0.000
0.000	4.009	7.686	10.663	12.500	12.682	10.696	6.285	0.000
0.000	5.650	10.938	15.436	18.568	19.531	17.265	10.707	0.000
0.000	7.652	14.980	21.573	26.804	29.610	28.125	19.278	0.000
0.000	9.977	19.758	29.074	37.465	43.980	46.347	38.281	0.000
0.000	12.500	25.000	37.500	50.000	62.500	75.000	87.500	100.000



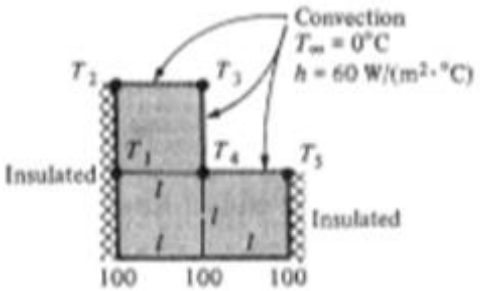
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.20	0.38	0.56	0.72	0.86	0.97	1.05	1.09	1.09	1.04	0.96	0.83	0.66	0.46	0.23	0.00
0.00	0.40	0.78	1.14	1.47	1.75	1.98	2.14	2.22	2.22	2.13	1.95	1.69	1.34	0.94	0.48	0.00
0.00	0.61	1.20	1.76	2.26	2.70	3.05	3.30	3.44	3.44	3.31	3.04	2.63	2.10	1.46	0.75	0.00
0.00	0.84	1.65	2.42	3.13	3.73	4.23	4.59	4.78	4.80	4.63	4.26	3.70	2.95	2.06	1.06	0.00
0.00	1.09	2.15	3.16	4.08	4.88	5.54	6.03	6.30	6.35	6.15	5.68	4.94	3.96	2.77	1.43	0.00
0.00	1.37	2.71	3.98	5.15	6.18	7.03	7.67	8.06	8.15	7.93	7.35	6.43	5.18	3.64	1.88	0.00
0.00	1.68	3.33	4.90	6.35	7.65	8.74	9.57	10.10	10.28	10.05	9.38	8.26	6.69	4.72	2.44	0.00
0.00	2.03	4.03	5.94	7.72	9.32	10.69	11.77	12.50	12.80	12.61	11.87	10.53	8.59	6.10	3.17	0.00
0.00	2.43	4.81	7.10	9.26	11.23	12.94	14.33	15.32	15.82	15.73	14.95	13.41	11.05	7.92	4.14	0.00
0.00	2.86	5.68	8.41	11.00	13.39	15.51	17.29	18.64	19.43	19.53	18.80	17.10	14.30	10.38	5.48	0.00
0.00	3.34	6.64	9.85	12.93	15.81	18.43	20.69	22.50	23.72	24.17	23.63	21.88	18.66	13.81	7.42	0.00
0.00	3.86	7.69	11.43	15.05	18.50	21.69	24.55	26.97	28.78	29.78	29.68	28.13	24.65	18.79	10.37	0.00
0.00	4.42	8.81	13.14	17.36	21.43	25.29	28.86	32.03	34.65	36.48	37.20	36.29	33.01	26.34	15.27	0.00
0.00	5.01	10.00	14.94	19.81	24.57	29.18	33.57	37.65	41.30	44.31	46.35	46.83	44.76	38.28	24.37	0.00
0.00	5.63	11.24	16.83	22.38	27.87	33.28	38.57	43.70	48.59	53.11	57.04	59.94	60.90	57.66	43.95	0.00
0.00	6.25	12.50	18.75	25.00	31.25	37.50	43.75	50.00	56.25	62.50	68.75	75.00	81.25	87.50	93.75	100.00

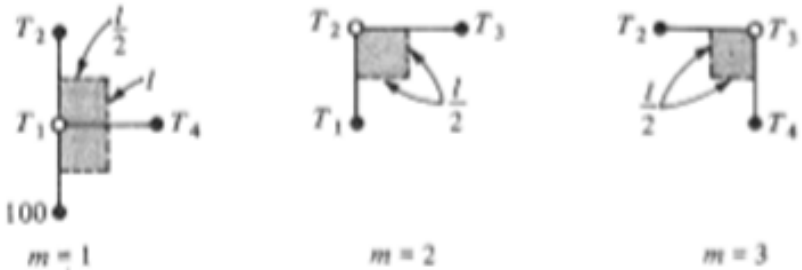


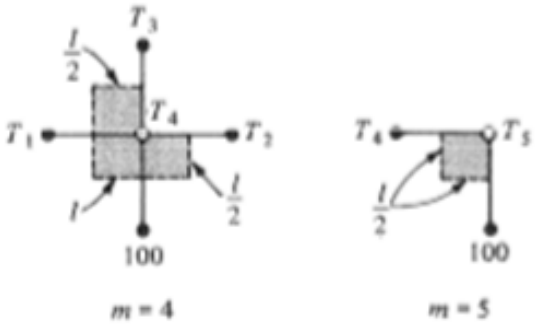
PROBLEM 4.11 FINITE DIFFERENCE SOLUTION, HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CARTESIAN COORDINATES, TWO DIMENSIONS

Consider two dimensional time dependent heat conduction for the region shown in the figure below. Boundary conditions are stated in the figure and a very course network and mesh size Δx= Δy=l=1 cm is used for finite difference formulation. Develop the finite difference equations for each of the five nodes m=1 to 5. Take h=60 W/(mK) k=30 W/(mK) and Tinf=0 C. $\alpha = 1.2x10^{-5}$

(You do not have to solve the equations just develop your equation matrix)







Node m=1

$$\emptyset\left[k\left(\frac{l}{2}\right)\frac{T_2^p - T_1^p}{l} + k(l)\frac{T_4^p - T_1^p}{l} + k\left(\frac{l}{2}\right)\frac{100 - T_1^p}{l}\right] + (1-\emptyset)\left[k\left(\frac{l}{2}\right)\frac{T_2^{p+1} - T_1^{p+1}}{l} + k(l)\frac{T_4^{p+1} - T_1^{p+!}}{l} + k\left(\frac{l}{2}\right)\frac{100 - T_1^{p+!}}{l}\right] = \rho C_p\left(\frac{l^2}{2}\right)\frac{T_1^{p+1} - T_1^p}{\Delta x}$$

Node m=2

$$\emptyset\left[k\left(\frac{l}{2}\right)\frac{T_1^p - T_2^p}{l} + k\left(\frac{l}{2}\right)\frac{T_3^p - T_2^p}{l} + h\left(\frac{l}{2}\right)(T_\infty - T_2^p)\right] + (1-\emptyset)\left[k\left(\frac{l}{2}\right)\frac{T_1^{p+1} - T_2^{p+1}}{l} + k\left(\frac{l}{2}\right)\frac{T_3^{p+!} - T_2^{p+!}}{l} + h\left(\frac{l}{2}\right)(T_\infty - T_2^{p+!})\right] = \rho C_p\left(\frac{l^2}{2}\right)\frac{T_2^{p+1} - T_2^p}{\Delta x}$$

Node m=3

$$\emptyset\left[k\left(\frac{l}{2}\right)\frac{T_2^p - T_3^p}{l} + k\left(\frac{l}{2}\right)\frac{T_4^p - T_3^p}{l} + h(l)(T_\infty - T_3^p)\right] + (1-\emptyset)\left[k\left(\frac{l}{2}\right)\frac{T_2^p - T_3^p}{l} + k\left(\frac{l}{2}\right)\frac{T_4^p - T_3^p}{l} + h(l)(T_\infty - T_3^p)\right] = \rho C_p\left(\frac{l^2}{2}\right)\frac{T_3^{p+1} - T_3^p}{\Delta x}$$

Node m=4

$$\emptyset\left[k\left(\frac{l}{2}\right)\frac{T_3^p - T_4^p}{l} + k(l)\frac{T_1^p - T_4^p}{l} + k\left(\frac{l}{2}\right)\frac{100 - T_4^p}{l} + k\left(\frac{l}{2}\right)\frac{T_5^p - T_4^p}{l} + h(l)(T_\infty - T_4^p)\right] + (1-\emptyset)\left[k\left(\frac{l}{2}\right)\frac{T_3^{p+1} - T_4^{p+1}}{l} + k(l)\frac{T_1^{p+1} - T_4^{p+1}}{l} + k\left(\frac{l}{2}\right)\frac{100 - T_4^{p+1}}{l} + k\left(\frac{l}{2}\right)\frac{T_5^{p+1} - T_4^{p+!}}{l} + h(l)(T_\infty - T_4^{p+1})\right] = \rho C_p\left(\frac{3l^2}{4}\right)\frac{T_4^{p+1} - T_4^p}{\Delta x}$$

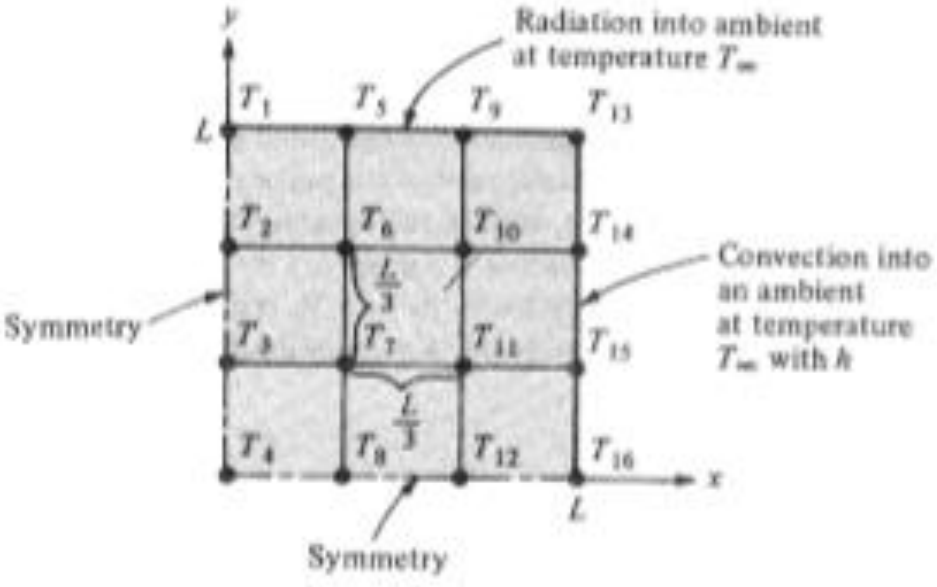
Node m=5

$$\emptyset\left[k\left(\frac{l}{2}\right)\frac{T_4^p - T_5^p}{l} + k(l)\frac{100 - T_4^p}{l} + h(l)\left(T_\infty - T_5^p\right)\right]$$
$$+ (1-\emptyset)\left[k\left(\frac{l}{2}\right)\frac{T_4^{p+1} - T_5^{p+1}}{l} + k(l)\frac{100 - T_4^{p+1}}{l} + h(l)\left(T_\infty - T_5^{p+1}\right)\right]$$
$$= \rho C_p\left(\frac{l^2}{4}\right)\frac{T_5^{p+1} - T_5^p}{\Delta x}$$

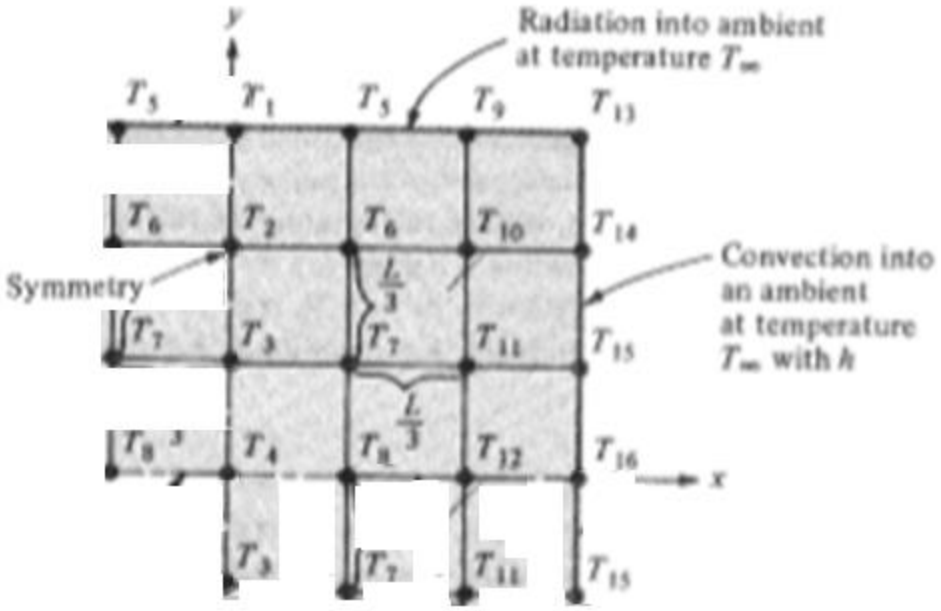
PROBLEM 4.12 FINITE DIFFERENCE SOLUTION, HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CARTESIAN COORDINATES, TWO DIMENSIONS

Consider two-dimensional time dependent heat conduction in a region 3 cm by 3 cm which is initially at a uniform temperature T_0=300 C and for t>0 is subjected to the boundary conditions shown in the figure. By using an explicit sheme and mesh size Δx= Δy=1 cm, develop the finite difference representation of this heat conduction problem.

(You do not have to solve the equations just develop your equation matrix)



Due to symmetry virtual points can be considered in symmetry sides



$$q_{radiation} = \varepsilon\sigma A\left(T^4 - T_\infty^4\right)$$

$$q_{rconvection} = hA(T - T_\infty)$$

$$l = \frac{L}{3} = 1\ cm$$

Node 1 :

$$\emptyset\left[\varepsilon\sigma\left(\frac{l}{2}\right)\left(T_\infty^4 - T_1^{p4}\right) + k\left(\frac{l}{2}\right)\frac{T_2^p - T_1^p}{l} + 2k\left(\frac{l}{2}\right)\frac{T_5^p - T_1^p}{l}\right]$$
$$+ (1 - \emptyset)\left[\varepsilon\sigma\left(\frac{l}{6}\right)\left(T_\infty^4 - T_1^{(p+1)4}\right) + k\left(\frac{l}{2}\right)\frac{T_2^{p+1} - T_1^{p+1}}{l} + 2k\left(\frac{l}{2}\right)\frac{T_5^{p+1} - T_1^{p+!}}{l}\right]$$
$$= \rho C_p\left(\frac{l^2}{4}\right)\frac{T_1^{p+1} - T_1^p}{\Delta x}$$

Node 2 :

$$\emptyset\left[2k(l)\frac{T_6^p - T_2^p}{l} + k\left(\frac{l}{2}\right)\frac{T_1^p - T_2^p}{l} + k\left(\frac{l}{2}\right)\frac{T_3^p - T_2^p}{l}\right]$$
$$+ (1 - \emptyset)\left[2k(l)\frac{T_6^{p+1} - T_2^{p+1}}{l} + k\left(\frac{l}{2}\right)\frac{T_1^{p+1} - T_2^{p+1}}{l} + k\left(\frac{l}{2}\right)\frac{T_3^{p+1} - T_2^{p+!}}{l}\right]$$
$$= \rho C_p\left(\frac{l^2}{2}\right)\frac{T_2^{p+1} - T_2^p}{\Delta x}$$

Node 3 :

$$\emptyset\left[2k\left(\frac{l}{2}\right)\frac{T_7^p - T_3^p}{l} + k\left(\frac{l}{2}\right)\frac{T_2^p - T_3^p}{l} + k\left(\frac{l}{2}\right)\frac{T_4^p - T_3^p}{l}\right]$$
$$+ (1-\emptyset)\left[2k\left(\frac{l}{2}\right)\frac{T_7^{p+1} - T_3^{p+1}}{l} + k\left(\frac{l}{2}\right)\frac{T_2^{p+1} - T_3^{p+1}}{l} + k\left(\frac{l}{2}\right)\frac{T_4^{p+1} - T_3^{p+1}}{l}\right]$$
$$= \rho C_p\left(\frac{l^2}{2}\right)\frac{T_3^{p+1} - T_3^p}{\Delta x}$$

Node 4 :

$$\emptyset\left[2k\left(\frac{l}{2}\right)\frac{T_8^p - T_4^p}{l} + 2k\left(\frac{l}{2}\right)\frac{T_3^p - T_4^p}{l}\right] + (1-\emptyset)\left[2k\left(\frac{l}{2}\right)\frac{T_8^{p+1} - T_4^{p+1}}{l} + 2k\left(\frac{l}{2}\right)\frac{T_3^{p+1} - T_4^{p+!}}{l}\right]$$
$$= \rho C_p\left(\frac{l^2}{4}\right)\frac{T_4^{p+1} - T_4^p}{\Delta x}$$

Node 5:

$$\emptyset\left[\varepsilon\sigma(l)\left(T_\infty^4 - T_5^{p4}\right) + k\left(\frac{l}{2}\right)\frac{T_1^p - T_5^p}{l} + k\left(\frac{l}{2}\right)\frac{T_9^p - T_5^p}{l} + k(l)\frac{T_6^p - T_5^p}{l}\right]$$
$$+ (1-\emptyset)\left[\varepsilon\sigma(l)\left(T_\infty^4 - T_5^{(p+1)4}\right) + k\left(\frac{l}{2}\right)\frac{T_1^{p+1} - T_5^{p+1}}{l} + k\left(\frac{l}{2}\right)\frac{T_9^{p+1} - T_5^{p+1}}{l}\right.$$
$$\left. + k(l)\frac{T_6^{p+1} - T_5^{p+1}}{l}\right] = \rho C_p\left(\frac{l^2}{2}\right)\frac{T_5^{p+1} - T_5^p}{\Delta x}$$

Node 6 : $(inner\ node)$

$$\emptyset\left[k(l)\frac{T_2^p - T_6^p}{l} + k(l)\frac{T_5^p - T_6^p}{l} + k(l)\frac{T_{10}^p - T_6^p}{l} + k(l)\frac{T_7^p - T_6^p}{l}\right]$$
$$+ (1-\emptyset)\left[k(l)\frac{T_2^{p+1} - T_6^{p+1}}{l} + k(l)\frac{T_5^{p+1} - T_6^{p+1}}{l} + k(l)\frac{T_{10}^{p+1} - T_6^{p+1}}{l}\right.$$
$$\left. + k(l)\frac{T_7^{p+1} - T_6^{p+1}}{l}\right] = \rho C_p(l^2)\frac{T_6^{p+1} - T_6^p}{\Delta x}$$

Node 7 : $(inner\ node)$

$$\emptyset\left[k(l)\frac{T_3^p - T_7^p}{l} + k(l)\frac{T_6^p - T_7^p}{l} + k(l)\frac{T_8^p - T_7^p}{l} + k(l)\frac{T_{11}^p - T_7^p}{l}\right]$$
$$+ (1-\emptyset)\left[k(l)\frac{T_3^{p+1} - T_7^{p+1}}{l} + k(l)\frac{T_6^{p+1} - T_7^{p+1}}{l} + k(l)\frac{T_8^{p+1} - T_7^{p+1}}{l}\right.$$
$$\left. + k(l)\frac{T_{11}^{p+1} - T_7^{p+1}}{l}\right] = \rho C_p(l^2)\frac{T_7^{p+1} - T_7^p}{\Delta x}$$

Node 8 :

$$\emptyset\left[k\left(\frac{l}{2}\right)\frac{T_4^p-T_8^p}{l}+k\left(\frac{l}{2}\right)\frac{T_{12}^p-T_8^p}{l}+2k(l)\frac{T_7^p-T_8^p}{l}\right]$$
$$+(1-\emptyset)\left[k\left(\frac{l}{2}\right)\frac{T_4^{p+1}-T_8^{p+1}}{l}+k\left(\frac{l}{2}\right)\frac{T_{12}^{p+1}-T_8^{p+1}}{l}+2k(l)\frac{T_7^{p+1}-T_8^{p+1}}{l}\right]$$
$$=\rho C_p\left(\frac{l^2}{2}\right)\frac{T_8^{p+1}-T_8^p}{\Delta x}$$

Node 9:

$$\emptyset\left[\varepsilon\sigma(l)\left(T_\infty^4-T_9^{p4}\right)+k\left(\frac{l}{2}\right)\frac{T_5^p-T_9^p}{l}+k\left(\frac{l}{2}\right)\frac{T_{11}^p-T_9^p}{l}+k(l)\frac{T_{10}^p-T_9^p}{l}\right]$$
$$+(1-\emptyset)\left[\varepsilon\sigma(l)\left(T_\infty^4-T_9^{(p+1)4}\right)+k\left(\frac{l}{2}\right)\frac{T_5^{p+1}-T_9^{p+1}}{l}+k\left(\frac{l}{2}\right)\frac{T_{11}^{p+1}-T_9^{p+1}}{l}\right.$$
$$\left.+k(l)\frac{T_{10}^{p+1}-T_9^{p+1}}{l}\right]=\rho C_p\left(\frac{l^2}{2}\right)\frac{T_9^{p+1}-T_9^p}{\Delta x}$$

Node 10 : $(inner\ node)$

$$\emptyset\left[k(l)\frac{T_6^p-T_{10}^p}{l}+k(l)\frac{T_9^p-T_{10}^p}{l}+k(l)\frac{T_{11}^p-T_{10}^p}{l}+k(l)\frac{T_{14}^p-T_{10}^p}{l}\right]$$
$$+(1-\emptyset)\left[k(l)\frac{T_6^p-T_{10}^p}{l}+k(l)\frac{T_9^{p+1}-T_{10}^{p+1}}{l}+k(l)\frac{T_{11}^{p+1}-T_{10}^{p+1}}{l}\right.$$
$$\left.+k(l)\frac{T_{14}^{p+1}-T_{10}^{p+1}}{l}\right]=\rho C_p(l^2)\frac{T_{10}^{p+1}-T_{10}^p}{\Delta x}$$

Node 11 : $(inner\ node)$

$$\emptyset\left[k(l)\frac{T_7^p-T_{11}^p}{l}+k(l)\frac{T_{10}^p-T_{11}^p}{l}+k(l)\frac{T_{12}^p-T_{11}^p}{l}+k(l)\frac{T_{15}^p-T_{11}^p}{l}\right]$$
$$+(1-\emptyset)\left[k(l)\frac{T_7^{p+1}-T_{11}^{p+1}}{l}+k(l)\frac{T_{10}^{p+1}-T_{11}^{p+1}}{l}+k(l)\frac{T_{12}^{p+1}-T_{11}^{p+1}}{l}\right.$$
$$\left.+k(l)\frac{T_{15}^{p+1}-T_{11}^{p+1}}{l}\right]=\rho C_p(l^2)\frac{T_{11}^{p+1}-T_{11}^p}{\Delta x}$$

Node 12 :

$$\emptyset\left[k\left(\frac{l}{2}\right)\frac{T_8^p-T_{12}^p}{l}+k\left(\frac{l}{2}\right)\frac{T_{16}^p-T_{12}^p}{l}+2k(l)\frac{T_{11}^p-T_{12}^p}{l}\right]$$
$$+(1-\emptyset)\left[k\left(\frac{l}{2}\right)\frac{T_8^{p+1}-T_{12}^{p+1}}{l}+k\left(\frac{l}{2}\right)\frac{T_{16}^{p+1}-T_{12}^{p+1}}{l}+2k(l)\frac{T_{11}^{p+1}-T_{12}^{p+1}}{l}\right]$$
$$=\rho C_p\left(\frac{l^2}{2}\right)\frac{T_{12}^{p+1}-T_{12}^p}{\Delta x}$$

Node 13 :

$$\emptyset\left[\varepsilon\sigma\left(\frac{l}{2}\right)\left(T_{\infty}^{4}-T_{13}^{p4}\right)+h\left(\frac{l}{2}\right)\left(T_{\infty}-T_{13}^{p}\right)+k\left(\frac{l}{2}\right)\frac{T_{9}^{p}-T_{13}^{p}}{l}+k\left(\frac{l}{2}\right)\frac{T_{14}^{p}-T_{13}^{p}}{l}\right]$$
$$+(1-\emptyset)\left[\varepsilon\sigma\left(\frac{l}{2}\right)\left(T_{\infty}^{4}-T_{13}^{(p+1)4}\right)+h\left(\frac{l}{2}\right)\left(T_{\infty}-T_{13}^{p+1}\right)+k\left(\frac{l}{2}\right)\frac{T_{9}^{p+1}-T_{13}^{p+1}}{l}\right.$$
$$\left.+k\left(\frac{l}{2}\right)\frac{T_{14}^{p+1}-T_{13}^{p+1}}{l}\right]=\rho C_{p}\left(\frac{l^{2}}{4}\right)\frac{T_{13}^{p+1}-T_{13}^{p}}{\Delta x}$$

Node 14 :

$$\emptyset\left[h(l)\left(T_{\infty}-T_{13}^{p}\right)+k\left(\frac{l}{2}\right)\frac{T_{13}^{p}-T_{14}^{p}}{l}+k\left(\frac{l}{2}\right)\frac{T_{15}^{p}-T_{14}^{p}}{l}+k(l)\frac{T_{10}^{p}-T_{14}^{p}}{l}\right]$$
$$+(1-\emptyset)\left[h(l)\left(T_{\infty}-T_{13}^{p+1}\right)+k\left(\frac{l}{2}\right)\frac{T_{13}^{p+1}-T_{14}^{p+1}}{l}+k\left(\frac{l}{2}\right)\frac{T_{15}^{p+1}-T_{14}^{p+1}}{l}\right.$$
$$\left.+k(l)\frac{T_{10}^{p+1}-T_{14}^{p+1}}{l}\right]=\rho C_{p}\left(\frac{l^{2}}{2}\right)\frac{T_{14}^{p+1}-T_{14}^{p}}{\Delta x}$$

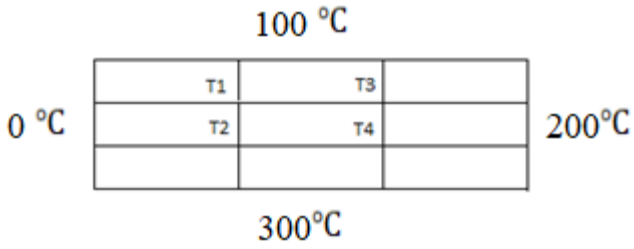
Node 15 :

$$\emptyset\left[h(l)\left(T_{\infty}-T_{15}^{p}\right)+k\left(\frac{l}{2}\right)\frac{T_{14}^{p}-T_{15}^{p}}{l}+k\left(\frac{l}{2}\right)\frac{T_{16}^{p}-T_{15}^{p}}{l}+k(l)\frac{T_{11}^{p}-T_{15}^{p}}{l}\right]$$
$$+(1-\emptyset)\left[h(l)\left(T_{\infty}-T_{15}^{p+1}\right)+k\left(\frac{l}{2}\right)\frac{T_{14}^{p+1}-T_{15}^{p+1}}{l}+k\left(\frac{l}{2}\right)\frac{T_{16}^{p+1}-T_{15}^{p+1}}{l}\right.$$
$$\left.+k(l)\frac{T_{11}^{p+1}-T_{15}^{p+1}}{l}\right]=\rho C_{p}\left(\frac{l^{2}}{2}\right)\frac{T_{15}^{p+1}-T_{15}^{p}}{\Delta x}$$

Node 16 :

$$\emptyset\left[h\left(\frac{l}{2}\right)\left(T_{\infty}-T_{16}^{p}\right)+2k\left(\frac{l}{2}\right)\frac{T_{15}^{p}-T_{16}^{p}}{l}+k\left(\frac{l}{2}\right)\frac{T_{12}^{p}-T_{16}^{p}}{l}\right]$$
$$+(1-\emptyset)\left[h\left(\frac{l}{2}\right)\left(T_{\infty}-T_{16}^{p+1}\right)+2k\left(\frac{l}{2}\right)\frac{T_{15}^{p+1}-T_{16}^{p+1}}{l}+k\left(\frac{l}{2}\right)\frac{T_{12}^{p+1}-T_{16}^{p+1}}{l}\right]$$
$$=\rho C_{p}\left(\frac{l^{2}}{4}\right)\frac{T_{16}^{p+1}-T_{16}^{p}}{\Delta x}$$

PROBLEM: A 0.3 m x 0.3 m square shape is given with outside temperatures are 0 C, 100 C, 200 C and 100 C as shown in the figure. If Δx=0.1 m and Δy=0.1 m, find temperatures T1,T2,T3 and T4 by using finite difference formulation.



$$100 + 0 - 4T_1 + T_3 + T_2 = 0$$

$$0 + 300 - 4T_2 + T_1 + T_4 = 0$$

$$100 + 200 - 4T_3 + T_2 + T_4 = 0$$

$$200 + 300 - 4T_4 + T_2 + T_3 = 0$$

GAUSS ELIMINATION 4 UNKNOWN

	-4	1	1	0	T1	-100
	1	-4	0	1	T2	-300
	0	1	-4	1	T3	-300
	0	1	1	-4	T4	-500

	-4	1	1	0	T1	-100
-0.25	0	-3.75	0.25	1	T2	-325
0	0	1	-4	1	T3	-300
0	0	1	1	-4	T4	-500

	-4	1	1	0	T1	-100
	0	-3.75	0.25	1	T2	-325
-0.26667	0	0	-3.93333	1.266667	T3	-386.667
-0.26667	0	0	1.066667	-3.73333	T4	-586.667

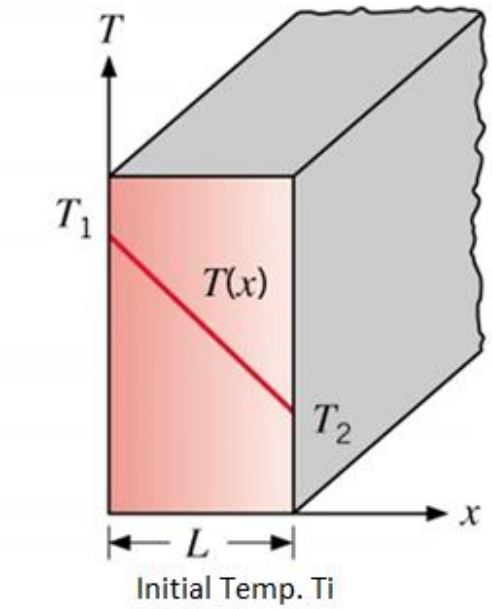
	-4	1	1	0	T1	-100
	0	-3.75	0.25	1	T2	-325
	0	0	-3.93333	1.266667	T3	-386.667
-0.27119	0	0	0	-3.38983	T4	-691.525

T3	**104**
T4	**152**
T3	**164**

4.2 METHOD OF LINES: A METHOD COMBINE FINITE DIFFERENCE AND ONE DIMENSIONAL NUMERICAL D:E: SOLUTIONS

Method of lines, in general a method to combine finite difference method with one dimensional numerical differential eqaution solving methods such as Runge Kutta Method. In order to explain the methods let us consider two dimensional time dependent T(x,t) wall problem

Time dependent wall problem (conduction heat transfer)



Differential equation to be solved:

$$\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T(x,t)}{\partial t}$$

Boundary conditions:

At x=0 t>0 $T(x,t) = d_0(t)$

At x=L t>0 $T(x,t) = d_1(t)$

At $0 \le x \le L$ $t = 0$ $T(x,t) = f(x)$

$$\alpha\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{\partial T(x,t)}{\partial t}$$

$\frac{\partial^2 T(x,t)}{\partial x^2}$ term will be approximated by using finite difference

$$\frac{\partial^2 T(x,t)}{\partial x^2} \cong \frac{T(x_{n+1},t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2} \quad n = 1..N-1$$

$$\alpha \frac{T(x_{n+1},t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2} \cong \frac{\partial T(x_n,t)}{\partial t}$$

$$x_n = n\Delta x$$

Applying boundary condition yields:

$$n = 1 \quad \frac{T(x_{n+1},t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2} = \frac{T(x_{n+1},t) - 2T(x_n,t) + d_0(t)}{\Delta x^2}$$

$$n = N - 1 \quad \frac{T(x_{n+1},t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2} = \frac{d_1(t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2}$$

At $0 \le x \le L\ t = 0\ \ T(x_n,t) = f(x_n)\ \ n = 1..N-1$

This creates vector of ordinary differential equation. Set of ordinary differential equation can be solved by using a numerical method such as Runge-Kutta method.

RUNGE KUTTA METHODS

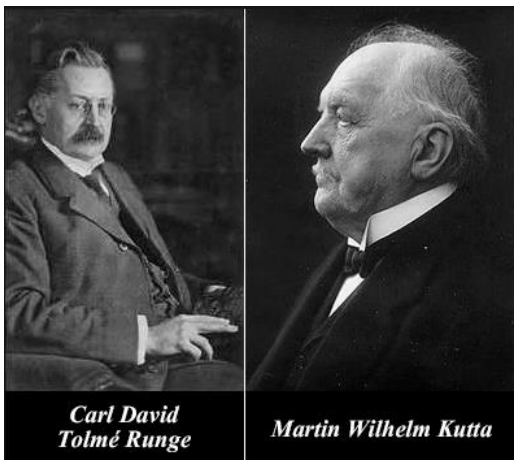


Figure 4.2.1 Carl Davis Tolme Runge and Martin Wilhelm Kutta

Runge-Kutta method as in most general form can be given as:
For differential form

$$\frac{dy}{dx} = f(x,y)$$

$$y_{n+1} = y_n + h\phi(x_n, y_n, h)$$

Where h=Δx is step size and

$$\phi(x_n, y_n, h) = \sum_{i=1}^{m} b_i k_i$$

Where k_i is m dimensional vector calculated as

$$x_i = x_n + c_i h \quad i = 1..m$$

$$y_i = y_n + h\sum_{j=1}^{i-1} a_{ij}\, k_j$$

$$k_i = f(x_i, y_i) = f\left(x_n + c_i h\ , y_n + h\sum_{j=1}^{i-1} a_{ij}\, k_j\right) \quad i = 1..m$$

or in open form

$$k_1 = f(x_i + c_1 h, y_i)$$
$$k_2 = f(x_i + c_2 h, y_i + a_{21} k_1 h)$$
$$k_3 = f(x_i + c_3 h, y_i + a_{31} k_1 h + a_{32} k_2 h)$$
$$\dots\dots$$
$$k_m = f\left(x_i + c_m h, y_i + a_{m1} k_1 h + a_{m2} k_2 h + .. + a_{m,m-1} k_{m-1} h\right)$$

In this equation a is constant values depends on polynamial degree of Runge-Kutta equation and coefficients
In numerical analysis books, coefficients are usually shown as Buthcher tableu which puts the coefficients of the method in a table as follows:

$$\begin{array}{c|cccccc} c_1 & a_{11} & a_{12} & a_{13} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & a_{23} & \dots & a_{2s} \\ c_3 & a_{31} & a_{32} & a_{33} & \dots & a_{3s} \\ \vdots & & \vdots & \vdots & \dots & \vdots \\ c_s & a_{s1} & a_{s2} & a_{s3} & \dots & a_{s4} \\ \hline & b_1 & b_2 & b_3 & \dots & b_s \end{array}$$

As it is seen from the equation the definition is very general. Depends on polynamial degree of Runge-Kutta equation accuracy will improve. For example Runge-Kutta equation derived by using **a second degree polynomial (RK2)** has the following form

$$y_{n+1} = y_n + h(0.5k_1 + 0.5k_2)$$
$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + h, y_i + k_1 h)$$

This equation can be given as Buthcher tableu as:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & 0.5 & 0.5 \end{array}$$

Second degree Ralston method:

$$y_{n+1} = y_n + h\left(\frac{1}{3}k_1 + \frac{2}{3}k_2\right)$$
$$k_1 = f(x_i, y_i)$$
$$k_2 = f\left(x_i + \frac{3}{4}h, y_i + \frac{3}{4}k_1 h\right)$$

This equation can be given as Buthcher tableu as:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 3/4 & 3/4 & 0 \\ \hline & 1/3 & 2/3 \end{array}$$

Runge-Kutta with third degree polynomial solution RK3:

$$y_{n+1} = y_n + h\left(\frac{1}{6}k_1 + \frac{4}{6}k_2 + \frac{1}{6}k_3\right)$$
$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + 0.5h, y_i + 0.5k_1 h)$$
$$k_3 = f(x_i + h, y_i - k_1 h + 2k_2 h)$$

This equation can be given as Buthcher tableu as:

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 1 & -1 & 2 & 0 \\ \hline & 1/6 & 4/6 & 1/6 \end{array}$$

Runge-Kutta with fourth degree polynomial solution RK4:

$$y_{n+1} = y_n + h(\frac{1}{6}k_1 + \frac{2}{6}k_2 + \frac{2}{6}k_3 + \frac{1}{6}k_4)$$
$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + 0.5h, y_i + 0.5k_1h)$$
$$k_3 = f(x_i + 0.5h, y_i + 0.5k_2h)$$
$$k_4 = f(x_i + h, y_i + k_3h)$$

This equation can be given as Buthcher tableu as:

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 2/6 & 2/6 & 1/6 \end{array}$$

Computer codes for Runge-Kutta solutions

```
public class Buthcher
{public double a[][];
 public double b[][];
 public double c[];
 public Buthcher(double ai[][],double bi[][],double ci[])
 {set(ai,bi,ci); }

 public void set(double ai[][],double bi[][],double ci[])
 {a=ai;b=bi;c=ci;}

 public Buthcher(String isim)
 {if(isim.equals("RK2"))
   {double a2[][]={{0,0},{1,0}};
   double b2[][]={{0.5,0.5}};
   double c2[]={0,1};
   set(a2,b2,c2);
   }
  else if(isim.equals("Raltston"))
  {double a2[][]={{0,0},{3.0/4.0,0}};
   double b2[][]={{1.0/3.0,2.0/3.0}};
   double c2[]={1.0/6.0,4.0/6.0,1.0/6.0};
   set(a2,b2,c2);
  }
  else if(isim.equals("RK3"))
  {double a2[][]={{0,0,0},{0.5,0,0},{-1,2,0}};
   double b2[][]={{1.0/6.0,4.0/6.0,1.0/6.0}};
   double c2[]={0,0.5,1};
   set(a2,b2,c2);
  }
  else if(isim.equals("RK4"))
  {double a2[][]={{0,0,0,0},{0.5,0,0,0},{0,0.5,0,0},{0,0,1,0}};
   double b2[][]={{1.0/6.0,2.0/6.0,2.0/6.0,1.0/6.0}};
   double c2[]={0,0.5,0.5,1};
   set(a2,b2,c2);
  }
  else if(isim.equals("RK6"))
  {double a2[][]={{0,0,0,0,0,0},{0.25,0,0,0,0,0},{0.125,0.125,0,0,0,0},{0,-0.5,1,0,0,0},{3.0/16.0,0,0,9.0/16.0,0,0},{-
3.0/7.0,2.0/7.0,12.0/7.0,-12.0/7.0,8.0/7.0,0}};
   double b2[][]={{7.0/90.0,0,32.0/90.0,12.0/90.0,32.0/90.0,7.0/90.0}};
   double c2[]={0,0.25,0.25,0.5,0.75,1.0};
   set(a2,b2,c2);
  }
  else if(isim.equals("RKV9"))
  { //Runge-Kutta-Verner 9uncu derece sabit adımlı metodu
    int s=10;
    double c[]=new double[s];
    double a[][]=new double[s][s];
    double bh[]=new double[s];
    double bh1[]=new double[s];
c[0]=0;
c[1] =  0.;
c[2] =  .6e-1;
c[3] =  .95933333333333333333333333333333333333333e-1;
```

```
c[4] =  .1439;
c[5] =  .4973;
c[6] =  .9725;
c[7] =  .9995;
c[8] =  1.;
c[9] =  1.;
a[2][1] =  .6e-1;
a[3][1] =  .19239962962962
```

```
c[4] =  0.6;
c[5] =  1;
c[6] = 7.0/8.0;

a[1][1] = 0;

a[2][1] =  0.2;

a[3][1] =  3.0/40.0;
a[3][2] =  9.0/40.0;

a[4][1] =  0.3;
a[4][2] =  -9.0/10.0;
a[4][3] =  6.0/5.0;

a[5][1] =  -11.0/54.0;
a[5][2] =  0.25;
a[5][3] =  -70.0/27.0;
a[5][4] =  35.0/27.0;

bh1[1] =  37.0/378.0;
bh1[2] =  250.0/621.0;
bh1[3] =  125.0/591.0;
bh1[4] =  512.0/1771.0;
bh1[5] =  0.0;

bh2[1] =  2825.0/27648.0;
bh2[2] =  18575.0/48384.0;
bh2[3] =  13525.0/55296.0;
bh2[4] =  277.0/14336.0;
bh2[5] =  1.0/4.0;
double b[][]={bh1,bh2};
set(a,b,c);
  }
  else if(isim.equals("RKDP45"))
  {// Runge-Kutta Dormant-Prince order 4-5
  int s=8;
double dydx=1.0;
double c[]=new double[s];
double a[][]=new double[s][s];
double bh1[]=new double[s];
double bh2[]=new double[s];
c[1]=0.0e0;
     c[2]=0.2;
     c[3]=0.3;
     c[4]=0.8;
     c[5]=8.0/9.0;
     c[6]=1;
     c[7]=1;
     a[2][1]=0.2;
     a[3][1]=3.0/40.0;
     a[3][2]=9.0/40.0;
     a[4][1]=44.0/45.0;
     a[4][2]=-56.0/15.0;
     a[4][3]=32.0/9.0;
     a[5][1]=19372.0/6561.0;
     a[5][2]=-25360.0/2187.0;
     a[5][3]=64448.0/6561.0;
     a[5][4]=-212.0/729.0;
     a[6][1]=9017.0/3168.0;
     a[6][2]=-355.0/33.0;
     a[6][3]=46732.0/5247.0;
     a[6][4]=49.0/176.0;
     a[6][5]=-5103.0/18656.0;
     a[7][1]=35.0/384.0;
     a[7][3]=500.0/1113.0;
     a[7][4]=125.0/192.0;
     a[7][5]=-2187.0/6784.0;
     a[7][6]=11.0/84.0;
     bh1[1]=5179.0/57600.0;
     bh1[2]=0;
     bh1[3]=7571.0/16695.0;
     bh1[4]=393.0/640.0;
     bh1[5]=-92097.0/339200.0;
     bh1[6]=187.0/2100.0;
     bh1[7]=1.0/40.0;
     bh2[1]=35.0/384.0;
```

```
      bh2[2]=0;
      bh2[3]=500.0/1113.0;
      bh2[4]=125.0/192.0;
      bh2[5]=-2187.0/6784.0;
      bh2[6]=11.0/84.0;
      bh2[7]=0.0;
      double b[][]={bh1,bh2};
      set(a,b,c);
    }
    else if(isim.equals("RKV13"))
    { //Runge-Kutta-Verner 13üncü derece sabit adımlı metodu
  int s=14;
  double c[]=new double[s];
  double a[][]=new double[s][s];
  double bh[]=new double[s];
  c[0]=0;
  c[1] =  0.;
  c[2] =  .5e-1;
  a[2][1] =  .5e-1;
  c[3] =  .1065625;
  a[3][1] = -.69931640625e-2;
  a[3][2] =  .1135556640625;
  c[4] =  .15984375;
  a[4][1] =  .399609375e-1;
  a[4][2] =  0.;
  a[4][3] =  .1198828125;
  c[5] =  .39;
  a[5][1] =  .36139756280045751240529407211840283451 29;
  a[5][2] =  0.;
  a[5][3] = -1.34152406670049277181998778820271583491 7;
  a[5][4] =  1.37012650390003525941469371608431300040 4;
  c[6] =  .465;
  a[6][1] =  .49047202797202797202797202797202797202 80e-1;
  a[6][2] =  0.;
  a[6][3] =  0.;
  a[6][4] =  .23509720422144047398629883354934271431 22;
  a[6][5] =  .18085559298135672881090396365345448848 50;
  //
  c[7] =  .155;
  a[7][1] =  .61692890442890442890442890442890442890 44e-1;
  a[7][2] =  0.;
  a[7][3] =  0.;
  a[7][4] =  .11236568314640276622625570351301500154 43303;
  a[7][5] = -.38850460714513667670490481081112445674 56e-1;
  a[7][6] =  .19791887125220458553791887125220458553 79e-1;
  c[8] =  .943;
  a[8][1] = -1.76763024022232687573559711957214558671 4;
  a[8][2] =  0.;
  a[8][3] =  0.;
  a[8][4] = -62.5;
  a[8][5] = -6.06188937737666910082136145965933199975 8;
  a[8][6] =  5.65082319822276313856129803060084017420 1;
  a[8][7] =  65.62169641937623283799566054863063741227;
  c[9] =  .90180204173585695825970794067837214995 60;
  a[9][1] = -1.18094506655497079982511628262829795788 2;
  a[9][2] =  0.;
  a[9][3] =  0.;
  a[9][4] = -41.50473441114320841606641502701994225874;
  a[9][5] = -4.43443831910372501122516922984610021177 6;
  a[9][6] =  4.26040818858613302481219371074469324076 1;
  a[9][7] =  43.75364022446171584987676829438379303004;
  a[9][8] =  .78714254899123106874464750442263075508 60e-2;
  c[10] =  .909;
  a[10][1] = -1.28140599944148840545951029118205424626 6;
  a[10][2] =  0.;
  a[10][3] =  0.;
  a[10][4] = -45.04713996013986630220754257136007322267;
  a[10][5] = -4.73136206944957647731146426549128281094 3;
  a[10][6] =  4.51496701659380784118585158459724099621 4;
  a[10][7] =  47.44909557172985134869022392235929015114;
  a[10][8] =  .10592282971116611356873939555165428752 28e-1;
  a[10][9] = -.57468422638446162544323184782862962320 21e-2;
  c[11] =  .94;
  a[11][1] = -1.72447013426248519175670981748448186173 1;
  a[11][2] =  0.;
  a[11][3] =  0.;
  a[11][4] = -60.92349008483054016518434619253765246063;
```

```
a[11][5] = -5.9515183762223924552028327670618548682 90;
a[11][6] =  5.5565237306984562359797916508435924968 39;
a[11][7] =  63.9830119803330533683753637863599593928 1;
a[11][8] =  .1464202825041496159275921391759452676003e-1;
a[11][9] =  .6460408772358203603621865144977650714892e-1;
a[11][10] = -.7930323169008878984024452548693373291447e-1;
c[12] =  1.;
a[12][1] = -3.3016226677470790163539947897909836255 69;
a[12][2] =  0.;
a[12][3] =  0.;
a[12][4] = -118.0112723597525085666923303957898868510;
a[12][5] = -10.1414223884561124864278391603451089759 5;
a[12][6] =  9.1393113322320579235440122735568270006 19;
a[12][7] =  123.3759428284042683684847180986501894364;
a[12][8] =  4.6232443788745804748398076250676309247 92;
a[12][9] = -3.3832777380682019236525550971536811240814;
a[12][10] =  4.5275921003246181894512653393511290353 25;
a[12][11] = -5.8284954858116229631930880191629857037 55;
c[13] =  1.;
a[13][1] = -3.0395150337663090300401028518212002510 56;
a[13][2] =  0.;
a[13][3] =  0.;
a[13][4] = -109.2608680894176254686444192322164623352;
a[13][5] = -9.2906424974002934497176655426568975491 58;
a[13][6] =  8.4305049817649111421342992538361678034 54;
a[13][7] =  114.2010010378331313557424041095523427476;
a[13][8] = -.9637271342145479358162375658987901652762;
a[13][9] = -5.0348840888021897911986803361833323231 18;
a[13][10] =  5.9581308240029231775404021653881720727 94;
a[13][11] =  0.;
a[13][12] =  0.;
bh[1] =  .4431261522908979212486436510209029764893e-1;
bh[2] =  0.;
bh[3] =  0.;
bh[4] =  0.;
bh[5] =  0.;
bh[6] =  .3546095642343226447863179350895055038855;
bh[7] =  .2478480431366653069619986721504458660016;
bh[8] =  4.4481347324757844927251283171596488713 12;
bh[9] =  19.8468863661187336993093239929768793529 1;
bh[10] = -23.5816233774656184196951796087039496508 5;
bh[11] =  0.;
bh[12] =  0.;
bh[13] = -.3601679437289775162124536737746202409110;
double b[][]={bh};
set(a,b,c);
  }
else if(isim.equals("RKF6"))
  { //Runge-Kutta-Fehlberg 6ıncı derece sabit adımlı metodu
int s=7;
double c[]=new double[s];
double a[][]=new double[s][s];
double bh[]=new double[s];
c[1] =  1;
c[2] =  0.2;
c[3] =  0.3;
c[4] =  0.6;
c[5] =  1;
c[5] = 7.0/8.0;

a[1][1] =  1;

a[2][1] =  0.2;

a[3][1] =  3.0/40.0;
a[3][2] =  9.0/40.0;

a[4][1] =  0.3;
a[4][2] =  -9.0/10.0;
a[4][3] =  6.0/5.0;

a[5][1] =  -11.0/54.0;
a[5][2] =  0.25;
a[5][3] =  -70.0/27.0;
a[5][4] =  35.0/27.0;

bh[1] =  2825.0/27648.0;
```

```
bh[2] =  18575.0/48384.0;
bh[3] =  13525.0/55296.0;
bh[4] =  277.0/14336.0;
bh[5] =  1.0/4.0;
double b[][]={bh};
set(a,b,c);
  }

}
}
```

```
import java.io.*;
import java.text.*;
import java.util.Locale;
public class Runge_Kutta
{
double a[][];
double b[][];
double c[];
double h;//step size
int n;//number of steps

public Runge_Kutta(double ai[][],double bi[][],double ci[])
{a=ai;b=bi;c=ci;}

public Runge_Kutta(Buthcher bu)
{a=bu.a;b=bu.b;c=bu.c;}
public Runge_Kutta(String isim)
{Buthcher bu=new Buthcher(isim);
 a=bu.a;b=bu.b;c=bu.c;
}
public double[][] hesapla(String ss,double x0,double xn,double y0,double h)
{return hesapla(ss,x0,xn,y0,h,0);}

public double[][] hesapla(f_xy f,double x0,double xn,double y0,double h)
{return hesapla(f,x0,xn,y0,h,0);}

public double[][] hesapla(fs f,double x0,double xn,double y0,double h)
{return hesapla(f,x0,xn,y0,h,0);}

public double[][] hesapla(f_xy f,double x0,double xn,double y0,double hi,int bk)
{ h=hi;
  n=(int)((xn-x0)/h);
  double x[]=new double[n+1];
  double y[]=new double[n+1];
  int s=c.length;
  double xi;
  double yi;
  double ki[]=new double[s];
  x[0]=x0;y[0]=y0;
  for(int ni=0;ni<n;ni++)
  {for(int i=0;i<s;i++)
   { xi=x[ni]+c[i]*h;
     yi=y[ni];
    for(int j=0;j<i;j++)
    {yi+=h*a[i][j]*ki[j];//System.out.println("xi(0)="+xi+"xi(j)="+yi+"i="+i+"j="+j+"ki="+ki[j]);
    }
    ki[i]=f.func(xi,yi);
   }
   y[ni+1]=y[ni];
   for(int i=0;i<s;i++)
    {y[ni+1]+=h*b[bk][i]*ki[i];}
   x[ni+1]=x[ni]+h;
  }
  double XY[][]=new double[2][n+1];
  XY[0]=x;
  XY[1]=y;
  return XY;
}

public double[][] hesapla(String ss,double x0,double xn,double y0,double hi,int bk)
{fs f=new fs(ss);
h=hi;
  n=(int)((xn-x0)/h);
  double x[]=new double[n+1];
  double y[]=new double[n+1];
  int s=c.length;
  double xi;
```

```
  double yi;
  double ki[]=new double[s];
  x[0]=x0;y[0]=y0;
  for(int ni=0;ni<n;ni++)
  {for(int i=0;i<s;i++)
   { xi=x[ni]+c[i]*h;
     yi=y[ni];
    for(int j=0;j<i;j++)
    {yi+=h*a[i][j]*ki[j];}
    ki[i]=f.func(xi,yi);
   }
   y[ni+1]=y[ni];
   for(int i=0;i<s;i++)
    {y[ni+1]+=h*b[bk][i]*ki[i];}
   x[ni+1]=x[ni]+h;
  }
  double XY[][]=new double[2][n+1];
  XY[0]=x;
  XY[1]=y;
  return XY;
}
public double[][] hesapla(fs f,double x0,double xn,double y0,double hi,int bk)
{ h=hi;
  n=(int)((xn-x0)/h);
  double x[]=new double[n+1];
  double y[]=new double[n+1];
  int s=c.length;
  double xi;
  double yi;
  double ki[]=new double[s];
  x[0]=x0;y[0]=y0;
  for(int ni=0;ni<n;ni++)
  {for(int i=0;i<s;i++)
   { xi=x[ni]+c[i]*h;
     yi=y[ni];
    for(int j=0;j<i;j++)
    {yi+=h*a[i][j]*ki[j];
    }
    ki[i]=f.func(xi,yi);
   }
   y[ni+1]=y[ni];
   for(int i=0;i<s;i++)
    {y[ni+1]+=h*b[bk][i]*ki[i];}
   x[ni+1]=x[ni]+h;
  }
  double XY[][]=new double[2][n+1];
  XY[0]=x;
  XY[1]=y;
  return XY;
}

public static double[] tasi(double x1,double y1[])
{
int n1=y1.length;
double z[]=new double[n1+1];
z[0]=x1;
for(int i=1;i<=n1;i++)
{z[i]=y1[i-1];}
return z;
}
//çok boyutlu RK girişler N(toplam veri sayısı) veya h(step boyu) üzeriinden verilebilmektedir.
public double[][] hesapla(f_xi f,double x0,double xn,double y0[],double h)
{return hesapla(f,x0,xn,y0,h,0);}

public double[][] hesapla(f_xi f,double x0,double xn,double y0[],int ni)
{   n=ni;
            h=(xn-x0)/ni;
    return hesapla(f,x0,xn,y0,h,0);
}

public double[][] hesapla(f_xi f,double x0,double xn,double y0[],int ni,int bk)
{   n=ni;
            h=(xn-x0)/ni;
    return hesapla(f,x0,xn,y0,h,bk);
}
public double[][] hesapla(f_xi f,double x0,double xn,double y0[],double hi,int bk)
{ // bk index of k if more than one equation is existed
```

```
  h=hi;
  n=(int)((xn-x0)/h+1e-30);
  int m=y0.length;
  double x[]=new double[n+1];
  double y[][]=new double[n+1][m];
  int s=c.length;
  double xi;
  double yi;
  double z[]=new double[m+1];
  double ki[]=new double[s];

  x[0]=x0;
  for(int ni=0;ni<=n;ni++)
  {x[ni]=x0+ni*h;}
  y[0]=y0;
  for(int ni=0;ni<n;ni++)
  { for(int k=0;k<m;k++)
                {
     for(int i=0;i<s;i++)
     { xi=x[ni]+c[i]*h;
       z=tasi(xi,y[ni]);
       for(int j=0;j<i;j++)
          {z[k+1]+=h*a[i][j]*ki[j];//System.out.println("xi(0)="+z[0]+"xi(j)="+z[k+1]+"i="+i+"j="+j+"ki="+ki[j]);
          }
       ki[i]=f.func(z,k);
     }
   y[ni+1][k]=y[ni][k];
   for(int i=0;i<s;i++)
    {y[ni+1][k]+=h*b[bk][i]*ki[i];}
  }
  }
  double XY[][]=new double[m+1][n+1];
  XY[0]=x;
  for(int k=0;k<m;k++)
   { for(int ni=0;ni<=n;ni++)
     {XY[k+1][ni]=y[ni][k];}
   }
  return XY;
}

public static String toString(double left, int w, int d)
// converts a double to a string with given width and decimals.
{
                NumberFormat df=NumberFormat.getInstance(Locale.US);
                df.setMaximumFractionDigits(d);
    df.setMinimumFractionDigits(d);
    df.setGroupingUsed(false);
    String s = df.format(left);
    while (s.length() < w)
      s = " " + s;
    if (s.length() > w)
    {
       s = "";
       for (int i=0; i<w; i++)
         s = s + "-";
    }
    return s;
}

public String toString()
 {int m=b[0].length;
  int n=c.length;
  int k=b.length;
  String s="Buthcher table n = "+n+"\n";
  for(int i=0;i<n;i++)
  {s+=toString(c[i],15,8)+" | ";
  for(int j=0;j<i;j++)
  {s+=toString(a[i][j],15,8)+" ";}
  s+="\n";
  }
  for(int j=0;j<m;j++)
  {s+="----------------";}
  s+="\n                ";
  for(int i=0;i<k;i++)
  {
  for(int j=0;j<m;j++)
  {s+=toString(b[i][j],15,8)+" ";}
```

```
   s+="\n                ";
  }
   return s;
 }

}
```

PROBLEM 4.13 METHOD OF LINES, HOMOGENEOUS, TIME DEPENDENT SOLUTION; CARTESIAN COORDINATES, TWO DIMENSIONS

$$\alpha \frac{T(x_{n+1},t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2} \cong \frac{\partial T(x_n,t)}{\partial t}$$

$$x_n = n\Delta x$$

Applying boundary condition yields:

$$n = 1 \quad \alpha \frac{T(x_{n+1},t) - 2T(x_n,t) + d_0(t)}{\Delta x^2} \cong \frac{\partial T(x_1,t)}{\partial t}$$

$$n = N-1 \quad \alpha \frac{d_1(t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2} \cong \frac{\partial T(x_{n-1},t)}{\partial t}$$

At $0 \le x \le L\ t = 0\ T(x_n,t) = f(x_n)\ n = 1..N-1$

L=1 m
at x=0 t>0 T_1=20℃
at x=L t>0 T_2=100℃
initial temperature (t=0) T_2=50℃

COMPUTER CODES

```
//=====================================================
// Numerical Analysis package in java
// Partial differential equation solution
// Method of lines combined with
// Runge-Kutta-Verner 13 constant step method with
// multible differential equations
// Dr. Turhan Coban
// =====================================================
// dy1/dx=f(x,y1,y2,..,yn)
// dy2/dx=f(x,y1,y2,..,yn)
// ......
// dyn/dx=f(x,y1,y2,..,yn)
// Method of lines
import java.io.*;

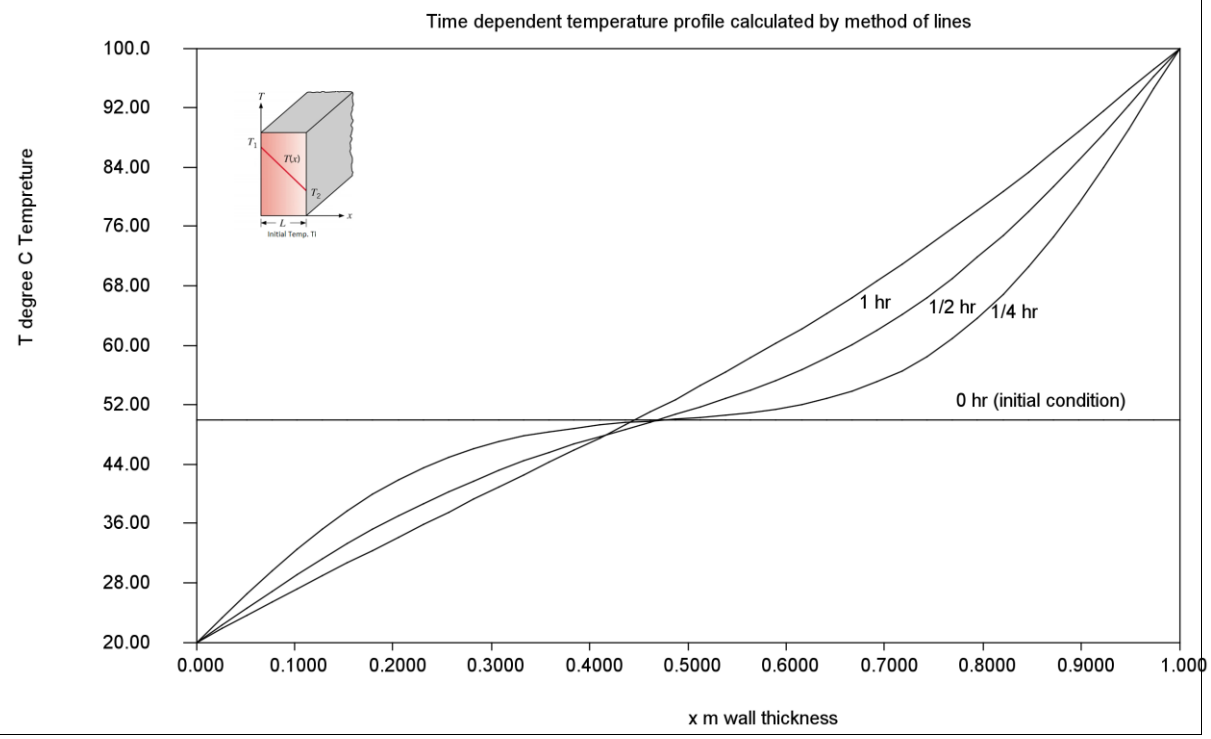
class fm2 extends f_xi
{ double L;
  int N;
  double alpha;
  double dx;
  double C;
  public fm2(double Li,int Ni,double alphai)
  {L=Li;N=Ni;alpha=alphai;dx=L/(N-1);C=alpha/(dx*dx);  }
//multivariable function
//lower limit x axis as a function of temperature
public double d0(double t)
```

```
{return 20.0;}
//upper limit x axis as a function of temperature
public double d1(double t)
{return 100.0;}
 //diferential equation set derived from finite difference formula
 double func(double T[],int n)
 {  double y=0;
   if(n==0)
   {y=C*(-2.0*T[1]+T[2])+C*d0(T[0]);}
   else if(n==(N-3) )
   {y=C*(-2.0*T[N-2]+T[N-3])+C*d1(T[0]);}
   else
    {y=C*(T[n]-2.0*T[n+1]+T[n+2]);}
    return y;
 }
}

class method_of_lines
{
public static void main(String args[]) throws IOException
  {
  double alpha=2e-5;
  double L=1.0;
 int N=40;
  double dx=L/(N-1);
  System.out.println("dx="+dx+"L="+L);
  fm2 b1=new fm2(L,N,alpha);
  double y[];
 double T[]=new double[N-2];
 double xx[]=new double[N];
 // initial (t=0) temperature gradient
  if_x f=x->40.0;
 //initial conditions
 // x axis
  for(int i=0;i<N;i++)
  {double x1=dx*i;
    xx[i]=x1;
  }
  //initial temperature profile
 for(int i=0;i<N-2;i++)
  {double x1=dx*(i+1);
    T[i]=f.func(x1);
  }
  //solution of partial differential equation
  Buthcher bb=new Buthcher("RKV13");
  Runge_Kutta rk=new Runge_Kutta(bb);
  double a[][]=rk.hesapla(b1,0.0,3600,T,1.0);

  double b[][]=Text.T(a);
  Text.printT(a[0]);
   int n=b.length;
  int m=b[0].length;
```

```
  System.out.println("n="+n+"m="+m);
  double c[][]=new double[n][m+1];
  for(int i=0;i<m+1;i++)
  {c[0][i]=f.func(xx[i]);}
  for(int j=1;j<n;j++)
  { c[j][0]=b1.d0(a[0][j]);
    c[j][m]=b1.d1(a[0][j]);
  }
  for(int i=1;i<m;i++)
  {for(int j=1;j<n;j++)
   {c[j][i]=b[j][i];}
  }
  double d[][]=Text.T(c);
  Text.print(c,"c");
  Text.print(c[0],"c0");
  Text.print(xx,"xx");
  Plot p=new Plot(xx,c[0]);
  p.addData(xx,c[900]);
  p.addData(xx,c[1800]);
  p.addData(xx,c[3600]);
  p.setPlabel("Heat Transfer wall T(x,t) method_of_lines solution");
  p.setXlabel("wall thickness x m");
   p.setYlabel("Temperature T degree C");
  p.plot();
 }
}
```



PROBLEM 4.14 METHOD OF LINES, HOMOGENEOUS, TIME DEPENDENT SOLUTION; CARTESIAN COORDINATES, TWO DIMENSIONS

$$\alpha \frac{T(x_{n+1},t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2} \cong \frac{\partial T(x_n,t)}{\partial t}$$

$$x_n = n\Delta x$$

Applying boundary condition yields:

$$n = 1 \quad \alpha \frac{T(x_{n+1},t) - 2T(x_n,t) + d_0(t)}{\Delta x^2} \cong \frac{\partial T(x_1,t)}{\partial t}$$

$$n = N-1 \quad \alpha \frac{d_1(t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2} \cong \frac{\partial T(x_{n-1},t)}{\partial t}$$

At $0 \le x \le L\ t = 0\ T(x_n,t) = f(x_n)\ n = 1..N-1$

Numerical Example 2
L=1 m
at x=0 t>0 $T_1(t)$=20℃ +20t/3600
at x=L t>0 $T_2(t)$=100℃ - 20t/3600
initial temperature (t=0) $T_2^{(X)}$=40℃ + $50sin(\frac{\pi}{2}x)$

```
//=======================================================
// Numerical Analysis package in java
// Partial differential equation solution
// Method of lines combined with
// Runge-Kutta-Verner 13 constant step method with
// multible differential equations
// Dr. Turhan Coban
// =======================================================
// dy1/dx=f(x,y1,y2,..,yn)
// dy2/dx=f(x,y1,y2,..,yn)
// ......
// dyn/dx=f(x,y1,y2,..,yn)
// Method of lines
import java.io.*;

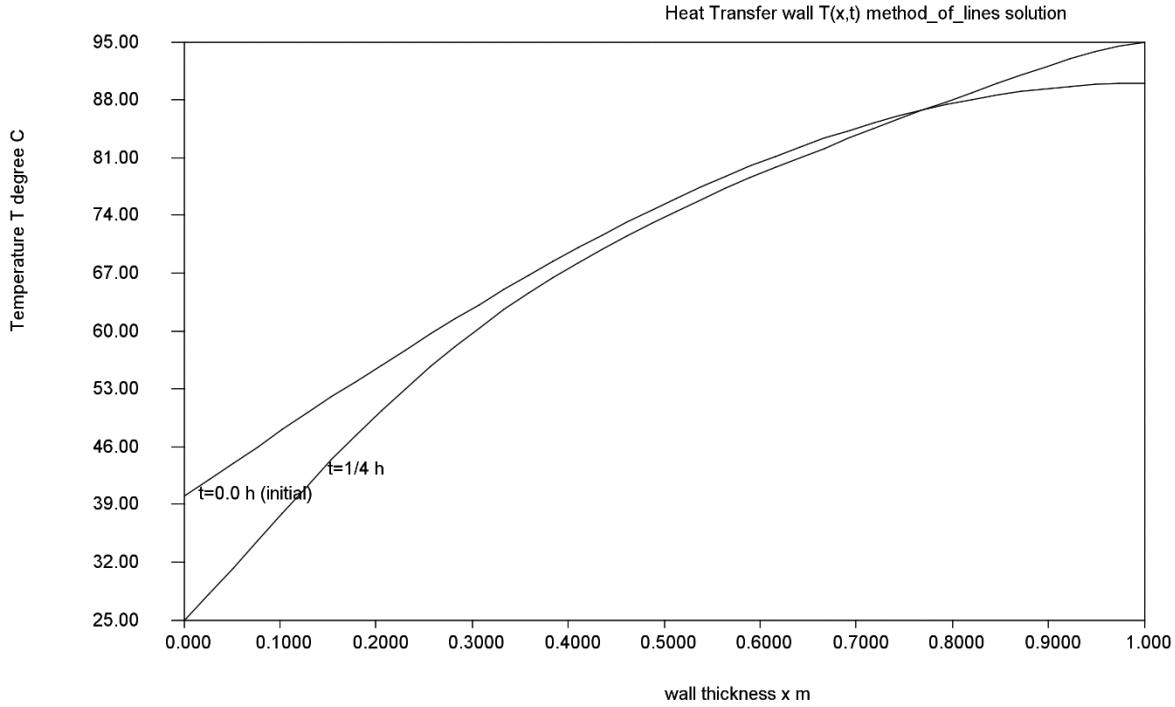
class fm2 extends f_xi
{ double L;
  int N;
  double alpha;
  double dx;
  double C;
  public fm2(double Li,int Ni,double alphai)
  {L=Li;N=Ni;alpha=alphai;dx=L/(N-1);C=alpha/(dx*dx);  }
//multivariable function
//lower limit x axis as a function of temperature
public double d0(double t)
{return 20.0+20.0*t/3600.0;}
//upper limit x axis as a function of temperature
public double d1(double t)
{return 100.0-20.0*t/3600.0;}
 //diferential equation set derived from finite difference formula
 double func(double T[],int n)
 {  double y=0;
   if(n==0)
   {y=C*(-2.0*T[1]+T[2])+C*d0(T[0]);}
   else if(n==(N-3) )
   {y=C*(-2.0*T[N-2]+T[N-3])+C*d1(T[0]);}
   else
```

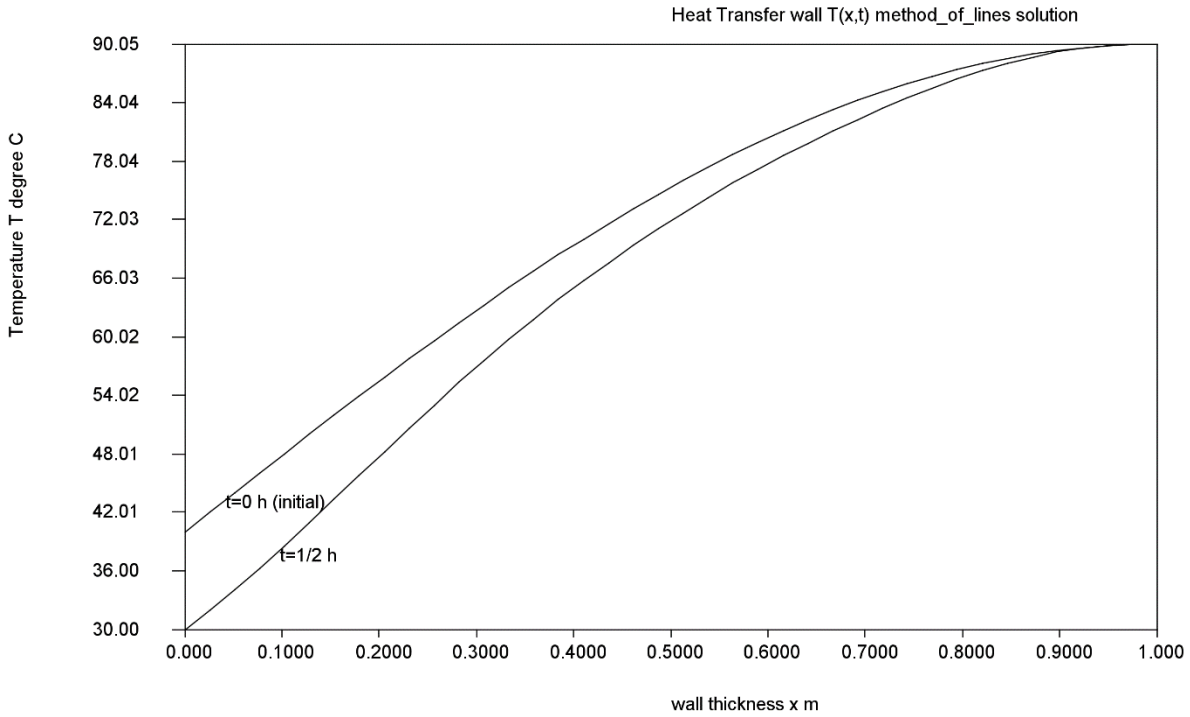
```
    {y=C*(T[n]-2.0*T[n+1]+T[n+2]);}
    return y;
 }
}

class method_of_lines
{
public static void main(String args[]) throws IOException
  {
  double alpha=2e-5;
  double L=1.0;
 int N=40;
  double dx=L/(N-1);
  System.out.println("dx="+dx+"L="+L);
  fm2 b1=new fm2(L,N,alpha);
  double y[];
 double T[]=new double[N-2];
 double xx[]=new double[N];
 // initial (t=0) temperature gradient
  if_x f=x->40.0+50*Math.sin(Math.PI*x/2.0);
 //initial conditions
 // x axis
  for(int i=0;i<N;i++)
  {double x1=dx*i;
    xx[i]=x1;
  }
  //initial temperature profile
 for(int i=0;i<N-2;i++)
  {double x1=dx*(i+1);
    T[i]=f.func(x1);
  }
  //solution of partial differential equation
  Buthcher bb=new Buthcher("RKV13");
  Runge_Kutta rk=new Runge_Kutta(bb);
  double a[][]=rk.hesapla(b1,0.0,3600,T,1.0);

  double b[][]=Text.T(a);
  Text.printT(a[0]);
   int n=b.length;
  int m=b[0].length;
  System.out.println("n="+n+"m="+m);
  double c[][]=new double[n][m+1];
  for(int i=0;i<m+1;i++)
  {c[0][i]=f.func(xx[i]);}
  for(int j=1;j<n;j++)
  { c[j][0]=b1.d0(a[0][j]);
     c[j][m]=b1.d1(a[0][j]);
  }
  for(int i=1;i<m;i++)
  {for(int j=1;j<n;j++)
    {c[j][i]=b[j][i];}
  }
  double d[][]=Text.T(c);
  Text.print(c,"c");
  Text.print(c[0],"c0");
  Text.print(xx,"xx");
  Plot p=new Plot(xx,c[0]);
  //p.addData(xx,c[900]);
  //p.addData(xx,c[1800]);
  p.addData(xx,c[3600]);
  p.setPlabel("Heat Transfer wall T(x,t) method_of_lines solution");
```

```
  p.setXlabel("wall thickness x m");
   p.setYlabel("Temperature T degree C");
  p.plot();
 }
}
```





5.0 FINITE ELEMENT METHODS

5.1 FINITE ELEMENT

The simplest finite element will be the linear one dimensional element. The linear temperature variation in the element is represented by:

$T(x) = \alpha_1 + \alpha_2 x$ where T is independent variable (tempearture if heat transfer problems are considered). Since there are two arbitrary constants in the linear representation, it requires only two nodes to determine the values of α_1, and α_2, namely,

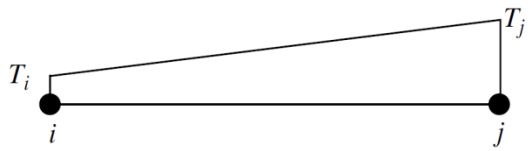


Figure 5.1 linear one dimensional finite element

$T_i = \alpha_1 + \alpha_2 x_i$

$T_j = \alpha_1 + \alpha_2 x_j$

From the above equation we get

$\alpha_1 = \frac{T_i x_j - T_j x_i}{x_j - x_i}$

$\alpha_2 = \frac{T_j - T_i}{x_j - x_i}$

On substituting the values α_1, and α_2, into the main equation, we get

$T(x) = \alpha_1 + \alpha_2 x = \frac{T_i x_j - T_j x_i}{x_j - x_i} + \frac{T_j - T_i}{x_j - x_i} x$ by rearranging the equation

$T(x) = T_i \left[\frac{x_j - x}{x_j - x_i}\right] + T_j \left[\frac{x - x_i}{x_j - x_i}\right]$ or

$T(x) = N_i T_i + N_j T_j = [N_i \quad N_j] \begin{Bmatrix} T_i \\ T_j \end{Bmatrix}$

Where N_i and N_j are called shape functions or interpolation functions. Equation can be rewritten as

$T = [N]\{T\}$ where $[N] = [N_i \quad N_j]$ and $\{T\} = \begin{Bmatrix} T_i \\ T_j \end{Bmatrix}$. This is basically linear interpolation formula that formulated in the shape function form. The derivative of the function can be easily formulated from this function

$$\frac{dT}{dx} = \frac{dN_i}{dx} T_i + \frac{dN_j}{dx} T_j = \left[-\frac{1}{x_j - x_i}\right] T_i + \left[\frac{1}{x_j - x_i}\right] T_j$$

$$l = x_j - x_i$$

$$g = \frac{dT}{dx} = \left[-\frac{1}{l}\right] T_i + \left[\frac{1}{l}\right] T_j = \begin{bmatrix} -\frac{1}{l} & \frac{1}{l} \end{bmatrix} \begin{Bmatrix} T_i \\ T_j \end{Bmatrix} = [B]\{T\}$$

If linear curve fitting representation will not be sufficient for the accuracy, degree of shape function can be increased. Instead of linear approximation, a quadratic approximation can be used:

$$T(x) = \alpha_1 + \alpha_2 x + \alpha_3 x^2$$

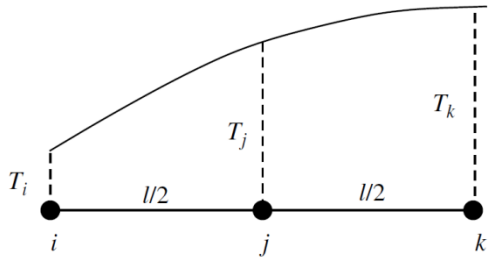


Figure 5.2 quadratic one dimensional finite element

$T_i = \alpha_1$

$$T_j = \alpha_1 + \alpha_2\left(\frac{l}{2}\right) + \alpha_3\left(\frac{l}{2}\right)^2$$
$$T_k = \alpha_1 + \alpha_2 l + \alpha_3 l^2$$

substituting these values back into the equation gives:

$$T = N_i T_i + N_j T_j + + N_k T_k = [N_i \quad N_j \quad N_k]\begin{Bmatrix} T_i \\ T_j \\ T_k \end{Bmatrix}$$

$$N_i = \left[1 - \frac{3x}{l} + \frac{2x^2}{l^2}\right]$$
$$N_j = \left[\frac{4x}{l} - \frac{4x^2}{l^2}\right]$$
$$N_k = \left[-\frac{x}{l} + \frac{2x^2}{l^2}\right]$$

The derivative of the function will be

$$g = \frac{dT}{dx} = \left[\left(\frac{4x}{l^2} - \frac{3}{l}\right) \quad \left(\frac{4}{l} - \frac{8x}{l^2}\right) \quad \left(-\frac{1}{l} + \frac{4x}{l^2}\right)\right]\begin{Bmatrix} T_i \\ T_j \\ T_k \end{Bmatrix} = [B]\{T\}$$

By using a linear or non-linear simplex elements a surface curve fitting can be achieved. The simplest geometric shape that can be employed to approximate irregular surfaces is the triangle and it is one of the popular elements currently used in finite element calculations. The two-dimensional linear triangular element, also known as a *simplex element*, is represented by

$$T(x, y) = \alpha_1 + \alpha_2 x + \alpha_3 y$$

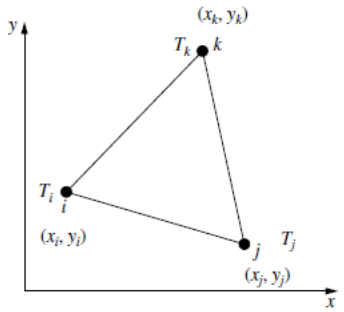


Figure 5.3 triangular finite element

where the polynomial is linear in *x* and *y* and contains three coefficients. Since a linear triangle has three nodes, the values of α_1, α_2 and α_3 are determined from

$T_i=\alpha_1+\alpha_2 x_i+\alpha_3 y_i$
$T_j=\alpha_1+\alpha_2 x_j+\alpha_3 y_j$
$T_k=\alpha_1+\alpha_2 x_k+\alpha_3 y_k$

which results in the following:

$$\alpha_1 = \frac{1}{2A}\left[(x_j y_k - x_k y_j)T_i + (x_k y_i - x_i y_k)T_j + (x_i y_j - x_j y_i)T_k\right]$$
$$\alpha_2 = \frac{1}{2A}\left[(y_j - y_k)T_i + (y_k - y_i)T_j + (y_i - y_j)T_k\right]$$
$$\alpha_3 = \frac{1}{2A}\left[(x_k - x_j)T_i + (x_i - x_k)T_j + (x_j - x_i)T_k\right]$$

where '*A*' is the area of the triangle given by

$$2A = det\begin{bmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{bmatrix} = (x_i y_j - x_j y_i) + (x_k y_j - x_i y_k) + (x_j y_k - x_k y_j)$$

Substituting the values of α_1, α_2 and α_3 into the equation and collating the coefficients of T_i, T_j and T_k, we get

$$T = N_i T_i + N_j T_j + N_k T_k = [N_i \quad N_j \quad N_k]\begin{Bmatrix} T_i \\ T_j \\ T_k \end{Bmatrix}$$

$$N_i = \frac{1}{2A}[a_i + b_i x + c_i y]$$

$$N_j = \frac{1}{2A}\left[a_j + b_j x + c_j y\right]$$
$$N_k = \frac{1}{2A}\left[a_k + b_k x + c_k y\right]$$

$a_i = (x_j y_k - x_k y_j)$ $a_j = (x_k y_i - x_i y_k)$ $a_k = (x_i y_j - x_j y_i)$
$b_i = (y_j - y_k)$ $b_j = (y_k - y_i)$ $b_k = (y_i - y_j)$
$c_i = (x_k - x_j)$ $c_j = (x_i - x_k)$ $c_k = (x_j - x_i)$

If we evaluate N_i at node i, where the coordinates are (x_i,y_i), then we obtain $N_i(x_i,y_i)=1$ and similarly $N_i(x_j,y_j)=0$, $N_i(x_k,y_k)=0$ Thus, we see that the shape functions have a value of unity at the designated vertex and zero at all other vertices. It is possible to show that $N_i+N_j+N_k=1$ everywhere in the element, including the boundaries. Linear simplex is given us a relatively simple way of interpolating any point in the surface (a linear interpolation). So the temperature profile of The simplex element can be calculated as:

$$T = N_i T_i + N_j T_j + + N_k T_k = [N_i \quad N_j \quad N_k]\begin{Bmatrix} T_i \\ T_j \\ T_k \end{Bmatrix} = [N]\{T\}$$

The derivative of the temperature field can be given as:

$$\{g\} = \begin{Bmatrix} \frac{\partial T}{\partial x} \\ \frac{\partial T}{\partial y} \end{Bmatrix} = \frac{1}{2A}\begin{bmatrix} b_i & b_j & b_k \\ c_i & c_j & c_k \end{bmatrix}\begin{Bmatrix} T_i \\ T_j \\ T_k \end{Bmatrix} = [B]\{T\}$$

Therefore heat transfer can be given as:

$$\begin{Bmatrix} q_x \\ q_y \end{Bmatrix} = -k[B]\{T\} = -\frac{k}{2A}\begin{bmatrix} b_i & b_j & b_k \\ c_i & c_j & c_k \end{bmatrix}\begin{Bmatrix} T_i \\ T_j \\ T_k \end{Bmatrix}$$

Integration of the finite element can be carried out as:
One dimensional integral:

$$\int_l N_i^a N_j^b N_k^c dl = \frac{a!\,b!\,c!}{(a+b+c+1)!} l$$

Two dimensional integral

$$\int_A N_i^a N_j^b N_k^c dA = \frac{a!\,b!\,c!}{(a+b+c+2)!} 2A$$

Different Finite elements can be defined the most common two dimensional element after tridigonal element is quadrilateral elements. Quadrilateral element is described by

$$T(x,y) = \alpha_1 + \alpha_2 x + \alpha_3 y + \alpha_4 xy$$

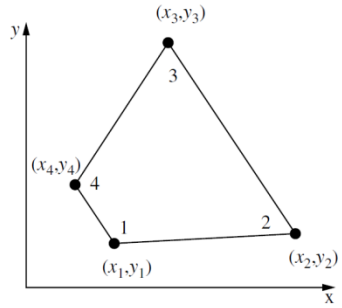


Figure 5.4 quadrilateral finite element
A special case of this element, rectangular element is used commonly.

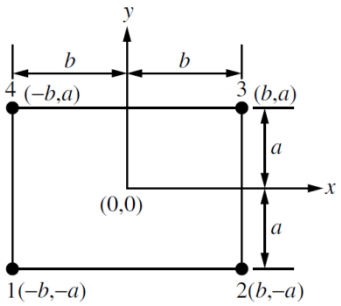


Figure 5.5 rectangular finite element

For this element shape function can be given as:

$$T(x,y) = N_1T_1 + N_2T_2 + N_3T_3 + N_4T_4$$

$$N_1 = \frac{1}{4ab}(b-x)(a-y)$$

$$N_2 = \frac{1}{4ab}(b+x)(a-y)$$

$$N_3 = \frac{1}{4ab}(b+x)(a+y)$$

$$N_4 = \frac{1}{4ab}(b-x)(a+y)$$

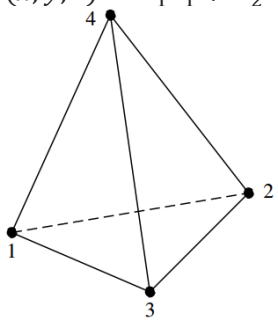
The derivatives can be expressed as follows:

$$\{g\} = \begin{Bmatrix} \frac{\partial T}{\partial x} \\ \frac{\partial T}{\partial y} \end{Bmatrix} = \begin{bmatrix} -(a-y) & (a-y) & (a+y) & -(a+y) \\ -(b-x) & -(b+x) & (b+x) & (b-x) \end{bmatrix} \begin{Bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{Bmatrix} = [B]\{T\}$$

As the last thing in this part, let us consider a three dimensional simplex element as well. The linear temperature representation of a tree-dimensional linear element (tetrahedron) is given by:

$T(x,y,z) = \alpha_1 + \alpha_2 x + \alpha_3 y + \alpha_4 z$ The constatnt of equation can be solved in a simialar manner of 2D simplex

$$T(x,y,z) = N_1T_1 + N_2T_2 + N_3T_3 + N_4T_4$$



Where

$$N_i = \frac{1}{6V}[a_i + b_i x + c_i y + d_i z]$$

The volume of the simplex element is:

$$6V = det \begin{bmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{bmatrix}$$

$$\{g\} = \begin{Bmatrix} \frac{\partial T}{\partial x} \\ \frac{\partial T}{\partial y} \\ \frac{\partial T}{\partial z} \end{Bmatrix} = \frac{1}{6V} \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{bmatrix} \begin{Bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{Bmatrix} = [B]\{T\}$$

Therefore heat transfer can be given as:

$$\begin{Bmatrix} q_x \\ q_y \\ q_z \end{Bmatrix} = -\frac{1}{6V} \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_z \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{bmatrix} \begin{Bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{Bmatrix} = -[D][B]\{T\}$$

Integration of the finite element can be carried out as:
Where:

$$b_1 = -det\begin{bmatrix}1 & y_2 & z_2\\ 1 & y_3 & z_3\\ 1 & y_4 & z_4\end{bmatrix} = (y_2 - y_4)(z_3 - z_4) - (y_3 - y_4)(z_2 - z_4)$$

$$b_2 = -det\begin{bmatrix}1 & y_1 & z_1\\ 1 & y_3 & z_3\\ 1 & y_4 & z_4\end{bmatrix} = (y_3 - y_4)(z_1 - z_4) - (y_1 - y_4)(z_3 - z_4)$$

$$b_3 = -det\begin{bmatrix}1 & y_1 & z_1\\ 1 & y_2 & z_2\\ 1 & y_4 & z_4\end{bmatrix} = (y_1 - y_4)(z_2 - z_4) - (y_2 - y_4)(z_1 - z_4)$$

$$b_4 = b_1 + b_2 + b_3$$

$$c_1 = -det\begin{bmatrix}x_2 & 1 & z_2\\ x_3 & 1 & z_3\\ x_4 & 1 & z_4\end{bmatrix} = (x_3 - x_4)(z_2 - z_4) - (x_2 - x_4)(z_3 - z_4)$$

$$c_2 = -det\begin{bmatrix}x_1 & 1 & z_1\\ x_3 & 1 & z_3\\ x_4 & 1 & z_4\end{bmatrix} = (x_1 - x_4)(z_3 - z_4) - (x_3 - x_4)(z_3 - z_4)$$

$$c_3 = -det\begin{bmatrix}x_1 & 1 & z_1\\ x_2 & 1 & z_2\\ x_4 & 1 & z_4\end{bmatrix} = (x_2 - x_4)(z_1 - z_4) - (x_1 - x_4)(z_2 - z_4)$$

$$c_4 = -(c_1 + c_2 + c_3)$$

$$d_1 = -det\begin{bmatrix}x_2 & y_2 & 1\\ x_3 & y_3 & 1\\ x_4 & y_4 & 1\end{bmatrix} = (x_2 - x_4)(y_3 - y_4) - (x_3 - x_4)(y_3 - y_4)$$

$$d_2 = -det\begin{bmatrix}x_1 & y_1 & 1\\ x_3 & y_3 & 1\\ x_4 & y_4 & 1\end{bmatrix} = (x_3 - x_4)(y_1 - y_4) - (x_1 - x_4)(y_3 - y_4)$$

$$d_3 = -det\begin{bmatrix}x_1 & y_1 & 1\\ x_2 & y_2 & 1\\ x_4 & y_4 & 1\end{bmatrix} = (x_1 - x_4)(y_2 - y_4) - (x_2 - x_4)(y_1 - y_4)$$

$$d_4 = -(d_1 + d_2 + d_3)$$

Three dimensional integral

$$\int_A N_1^a N_2^b N_3^c N_4^d dV = \frac{a!b!c!d!}{(a+b+c+d+3)!} 6V$$

5.2 SOLUTION OF HEAT CONDUCTION EQUATION BY USING FINITE ELEMENT METHOD

Now general steady state general heat conduction equation can be investigated and converted into an integral equation

$$\frac{\partial}{\partial x}\left(k_x \frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k_y \frac{\partial T}{\partial y}\right) + g = 0$$

Boundary conditions

T=T_b on surface S_1

$k_x \frac{\partial T}{\partial x}\vec{l} + k_y \frac{\partial T}{\partial y}\vec{m} + q = 0$ on surface S_2

$k_x \frac{\partial T}{\partial x}\vec{l} + k_y \frac{\partial T}{\partial y}\vec{m} + h(T - T_a) = 0$ on surface S_3

Now that Galarkin integration method will be applied to our differention equation . The method requires that the following expression will be satisfied

$$\int_{\Omega} w_k L(T) d\Omega = 0$$

where Ω is the volume of the element. Substituring steady state heat conduction equation and finite element node as weight factor into the galarkin integration equation gives:

$$\int_{\Omega} N_k \left[\frac{\partial}{\partial x}\left(k_x \frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k_y \frac{\partial T}{\partial y}\right) + g\right] d\Omega = 0$$

Note that for finite element $T = [N]\{T\}$ and integral can be converted by partial integration rule

$$\int udv = uv - \int vdu$$

$$\int_{\Omega} N_k \left[\frac{\partial}{\partial x}\left(k_x \frac{\partial T}{\partial x}\right)\right] d\Omega = \int_S N_k \left(k_x \frac{\partial T}{\partial x}\right) dS - \int_{\Omega} \frac{\partial N_k}{\partial x}(k_x) \frac{\partial N_m}{\partial x} T d\Omega$$

By applying these equation becomes

$$-\int_{\Omega} \left[k_x \frac{\partial N_k}{\partial x}\frac{\partial N_m}{\partial x} + k_y \frac{\partial N_k}{\partial y}\frac{\partial N_m}{\partial y}\right]\{T_m\} d\Omega + \int_{\Omega} g N_k d\Omega + \int_S N_k \left(k_x \frac{\partial T}{\partial x} + k_y \frac{\partial T}{\partial y}\right) dS = 0$$

From boundary conditions:

$$\int_S N_k \left(k_x \frac{\partial T}{\partial x} + k_y \frac{\partial T}{\partial y}\right) dS = -\int_S N_k (q + h(T - T_a) dS = -\int_S N_k \, q dS - \int_S N_k h \, N_m \{T_m\} dS + \int_S N_k h \, T_a dS$$

$$-\int_{\Omega} \left[k_x \frac{\partial N_k}{\partial x}\frac{\partial N_m}{\partial x} + k_y \frac{\partial N_k}{\partial y}\frac{\partial N_m}{\partial y}\right]\{T_m\} d\Omega + \int_{\Omega} g N_k d\Omega - \int_S N_k \, q dS - \int_S N_k h \, N_m \{T_m\} dS + \int_S N_k h \, T_a dS = 0$$

If this equation is written in matrix form:

$[K]\{T\} = \{f\}$ where

$$[K] = -\int_{\Omega} \left[k_x \frac{\partial N_k}{\partial x}\frac{\partial N_m}{\partial x} + k_y \frac{\partial N_k}{\partial y}\frac{\partial N_m}{\partial y}\right] d\Omega + \int_S N_k h \, N_m dS$$

$$\{f\} = \int_{\Omega} g N_k d\Omega - \int_S N_k \, q dS + \int_S N_k h \, T_a dS$$

If equation put into the matrix form:

$$[K] = \int_{\Omega} [B]^T [D][B] d\Omega + \int_S h \, [N]^T [N] dS$$

Where

$$[D] = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix}$$

$$\{f\} = \int_{\Omega} g[N]^T d\Omega - \int_S [N]^T \, q dS + \int_S h [N]^T \, T_a dS$$

$$\int_l N_i^a N_j^k dl = \frac{a!b!}{(a+b+1)!} l \qquad \int_l N_i^2 N_j^0 dl = \frac{2!0!}{(2+0+1)!} l = \frac{2l}{6} \qquad \int_l N_i^1 N_j^1 dl = \frac{1!1!}{(1+1+1)!} l = \frac{l}{6}$$

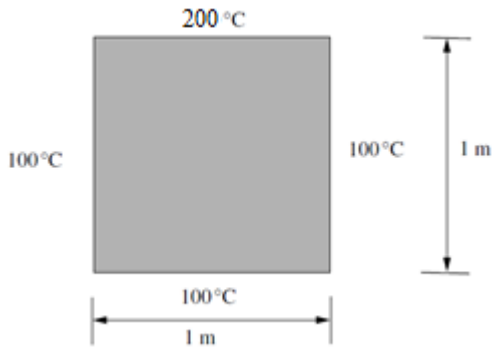
Matrix terms can be opened for two dimensional triangular element as:

$$[K] = \frac{t}{4A}\left\{k_x \begin{bmatrix} b_i^2 & b_i b_j & b_i b_k \\ b_i b_j & b_j^2 & b_j b_k \\ b_j b_k & b_j b_k & b_k^2 \end{bmatrix} + k_y \begin{bmatrix} c_i^2 & c_i c_j & c_i c_k \\ c_i c_j & c_j^2 & c_j c_k \\ c_j c_k & c_j c_k & c_k^2 \end{bmatrix}\right\} + \frac{htl_{ij}}{6}\begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \frac{htl_{jk}}{6}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} + \frac{htl_{ki}}{6}\begin{bmatrix} 2 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$

$$\{f\} = \frac{gAt}{3}\begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix} - \frac{qtl_{ij}}{2}\begin{Bmatrix} 1 \\ 1 \\ 0 \end{Bmatrix} - \frac{qtl_{jk}}{2}\begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} - \frac{qtl_{ik}}{2}\begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} + + \frac{hT_a tl_{jk}}{2}\begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} + + \frac{hT_a tl_{ik}}{2}\begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix}$$

PROBLEM 5.1 FINITE ELEMENT SOLUTION, NON-HOMOGENEOUS, STEADY STATE SOLUTION; CARTESIAN COORDINATES

A square plate with temperatures of 100 C in three sides and 200 C in the fourth side is given as 1 m. Calculate the temperature profile



Java program:

```
public class element_number
{int number;
 int ijk;
 public element_number(int numberi,int ijki)
 {
 number=numberi;
 if(ijki<=2) ijk=ijki;
 else {ijk=0;System.out.println("wrong value");}
 }
 public boolean equals(element_number en)
 {if(en.number==number && en.ijk==ijk) return true;
 else return false;
 }
 public int getNumber() {return number;}
 public int getIjk() {return ijk;}
 public int[] output()
 {int x[]={number,ijk};
  return x;
 }
}
```

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class setElement
{   ArrayList<element_number> en;
    int no[];
    int ij[];
                public setElement()
                {en=new ArrayList<element_number>();}
                public void add(int n1,int ij1)
                {element_number en=new element_number(n1,ij1);
                 add(en);
    }
                public void add(element_number s1)
                { //int n=en.size();
                  boolean b=false;
                  element_number s2;
                  Iterator n2=en.iterator();
      while(n2.hasNext()&& !b)
      { s2=(element_number)n2.next();
        b=s2.equals(s1);
      }
                if(!b) en.add(s1);
                }

  public int[][] output()
  {
    int n=en.size();
    int a[][]=new int[n][2];
    Iterator i=en.iterator();
    int j=0;
    while(i.hasNext()) {element_number el=(element_number)i.next();a[j]=el.output();j++;}
    return a;
  }
  public static void main(String arg[])
  {setElement se=new setElement();
```

```
   element_number en1=new element_number(11,0);
   element_number en2=new element_number(11,1);
   element_number en3=new element_number(13,2);
   element_number en4=new element_number(13,0);
   se.add(en1);
   se.add(en2);
   se.add(en3);
   se.add(en4);
   int a[][]=se.output();
   Text.print(a);
  }
}
```

```
 import javax.swing.*;
import static java.lang.Math.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class triangular_grid
{
public triangular_element te[];
public setElement se[];
public int N,M,K,K1,K2,Nm1,Mm1;
public double x[][];
public double y[][];
public double T[][];
public double Ti[];
public double xi[];
public double yi[];
public boolean bi[];
public double dx,dy;
public int Bx0[],BxM[],By0[],ByN[];
public double L,W;
public double min(double xi[][])
{ double min=1e200;
  for(int i=0;i<xi.length;i++)
   {for(int j=0;j<xi[0].length;j++)
     {if(xi[i][j]<min) min=xi[i][j];}
 }
 return min;
}
public double max(double xi[][])
{double max=-1e200;
 for(int i=0;i<xi.length;i++)
 {for(int j=0;j<xi[0].length;j++)
   {if(xi[i][j]<max) max=xi[i][j];}
 }
 return max;
}
public triangular_grid(double xi[][],double yi[][])
{x=xi;
 y=yi;
 N=x.length-1;
 M=x[0].length-1;
 T=new double[N+1][M+1];
 L=max(xi);
 W=max(yi);
 setElements();
}

public void generateTriangularGrid(double Li, double Wi,int Ni, int Mi)
{int i,j;
 N=Ni;M=Mi;
 L=Li;
 W=Wi;
 x=new double[N+1][M+1];
 y=new double[N+1][M+1];
 T=new double[N+1][M+1];
 setElements();
}

public void setElements()
{ int i,j;
 dx=L/M;
 dy=W/N;
```

```
 K=M*N;
 K1=K*2;
 K2=K1;
 //grid points
 xi=new double[K2];
 yi=new double[K2];
 Ti=new double[K2];
 bi=new boolean[K2];
 se=new setElement[K2];
 for(i=0;i<K2;i++) {se[i]=new setElement();}
 te=new triangular_element[K1];
 int k1=-1,k=0,n=0,m=0;
for(n=0;n<=N;n++)
{ for(m=0;m<=M;m++)
  {k=(N+1)*n+m;
   x[n][m]=m*dx;y[n][m]=n*dy;
   xi[k]=x[n][m];
   yi[k]=y[n][m];
   bi[k]=false;
   Ti[k]=0.0;
  }
}

for(n=0;n<N;n++)
{for(m=0;m<M;m++)
  { k=(N+1)*n+m;
              int kp1=(N+1)*(n+1)+m;
              int kp1p1= kp1+1;
              double xi1[]={x[n][m],x[n][m+1],x[n+1][m]};
              double yi1[]={y[n][m],y[n][m+1],y[n+1][m]};

              k1++;
              int n_point1[]={k,(k+1),kp1};
              se[k].add(k1,0);
              se[k+1].add(k1,1);
              se[kp1].add(k1,2);
              te[k1]=new triangular_element(k1,n_point1,xi1,yi1);
              double xi2[]={x[n][m+1],x[n+1][m+1],x[n+1][m]};
              double yi2[]={y[n][m+1],y[n+1][m+1],y[n+1][m]};
              k1++;
              int n_point2[]={k+1,kp1p1,kp1};
   se[k+1].add(k1,0);
              se[kp1p1].add(k1,1);
              se[kp1].add(k1,2);
              te[k1]=new triangular_element(k1,n_point2,xi2,yi2);
  }
}
}
public triangular_grid(double Li, double Wi,int Ni, int Mi)
{generateTriangularGrid(Li,Wi,Ni,Mi);}
public int find_element(double z[])
{ return find_element(z[0],z[1]);}
public int find_element(double x,double y)
{double R1=Math.sqrt(x*x+y*y);
 int kmin=0;
 double min=1.0e50;
 for(int k=0;k<K1;k++)
 {   te[k].setN(x,y);
     double r0=Math.sqrt((te[k].x[0]-x)*(te[k].x[0]-x)+(te[k].y[0]-y)*(te[k].y[0]-y));
     double r1=Math.sqrt((te[k].x[1]-x)*(te[k].x[1]-x)+(te[k].y[1]-y)*(te[k].y[1]-y));
     double r2=Math.sqrt((te[k].x[2]-x)*(te[k].x[2]-x)+(te[k].y[2]-y)*(te[k].y[2]-y));
     double dr=Math.sqrt(r0*r0+r1*r1+r2*r2);
               boolean b1=(x>=te[k].x[0]) && (x<=te[k].x[2]);
               boolean b2=(y>=te[k].y[0]) && (y<=te[k].y[1]);
               boolean b= b1 && b2;
     if(dr<min) {min=dr;kmin=k;}
 }
 return kmin;
}
public void setConvection(int cond,double L0,double L,double W0,double W, double hi, double tinfi)
{//cond 0: x==L0
 //cond 1: x==L
 //cond 2: y==W0
 //cond 3: y==W
 int j=0;
 for(int i=0;i<K1;i++)
 {  j=te[i].setI(cond,L0,L,W0,W);
```

```
                te[i].setConvection(j,hi,tinfi);
 }
}
public void setHeatFlux(int cond,double L0,double L,double W0,double W, double qi)
{//cond 0: x==L0
 //cond 1: x==L
 //cond 2: y==W0
 //cond 3: y==W
 int j=-1;
 for(int i=0;i<K1;i++)
 {  j=te[i].setI(cond,L0,L,W0,W);
                te[i].setHeatFlux(j,qi);
 }
}
/constant function temperature profile
public void setTemperature(int cond,double L0,double L,double W0,double W, double TTi)
{//cond 0: x==L0
 //cond 1: x==L
 //cond 2: y==W0
 //cond 3: y==W
 for(int i=0;i<K2;i++)
 {  if(cond==0)
    {if(xi[i]==L0) {Ti[i]=TTi;bi[i]=true;}}
    else if(cond==1)
    {if(xi[i]==L) {Ti[i]=TTi;bi[i]=true;}}
    else if(cond==2)
    {if(yi[i]==W0) {Ti[i]=TTi;bi[i]=true;}}
    else if(cond==3)
    {if(yi[i]==W) {Ti[i]=TTi;bi[i]=true;}}
 }
}
//variable function temperature profile
public void setTemperature(int cond,double L0,double L,double W0,double W, if_x Tx)
{//cond 0: x==L0
 //cond 1: x==L
 //cond 2: y==W0
 //cond 3: y==W
 for(int i=0;i<K2;i++)
 {  if(cond==0)
    {if(xi[i]==L0) {Ti[i]=Tx.func(yi[i]);bi[i]=true;}}
    else if(cond==1)
    {if(xi[i]==L) {Ti[i]=Tx.func(yi[i]);bi[i]=true;}}
    else if(cond==2)
    {if(yi[i]==W0) {Ti[i]=Tx.func(xi[i]);bi[i]=true;}}
    else if(cond==3)
    {if(yi[i]==W) {Ti[i]=Tx.func(xi[i]);bi[i]=true;}}
 }
} public void setHeatGeneration(int node[],double qgeni)
{  int n=node.length;
   for(int i=0;i<n;i++)
   {te[node[i]].setHeatGeneration(qgeni);}
}

public void solve(double ti,double k[])
{
double K[][]=new double[K2][K2];
double KK[][]=new double[K2][K2];
double ff[]=new double[K2];
double KN[][]=new double[3][3];
double f[]=new double[K2];
double fN[]=new double[3];
for(int i=0;i<K1;i++)
{KN=te[i].K(ti,k);
 fN=te[i].f(ti,k);
 int n1=te[i].n_element;
 int m[]=te[i].n_point;
 for(int j=0;j<3;j++)
   { for(int k1=0;k1<3;k1++)
      {K[m[j]][m[k1]]+=KN[j][k1];KK[m[j]][m[k1]]=K[m[j]][m[k1]];}
       f[m[j]]+=fN[j];
       ff[m[j]]=f[m[j]];
   }
}
//redesign K matrix and f vector if any known
//temperature boundary is given
//int j=0;
for(int j=0;j<K2;j++)
```

```
{
            for(int i=0;i<K2;i++)
    { if(bi[i])
      {
                if(i==j)
                {f[i]=Ti[i];K[i][j]=1.0;}
                else
                {
                K[i][j]=0.0;}
                //System.out.println("i="+i+"j="+j+"bi="+bi[i]+"Ti="+Ti[i]+"fi="+f[i]);
     }
      }//end of for
} //end of for
//System.out.println("K =\n"+Matrix.toString(K));
//System.out.println("f =\n"+Matrix.toStringT(f));
double TT[]=AXB(K,f);
int k1=0;
for(int n=0;n<=N;n++)
{ for(int m=0;m<=M;m++)
  {k1=(N+1)*n+m;
  T[n][m]=TT[k1];
  }
}
}

//LU decomposition method
public static double[][] LU(double c[][],int indx[],int d[])
{
//returns LU decomposition of matrix c and index indx
double a[][];
int n=c.length;
a=new double[n][n];
double vv[];
vv=new double[n];
double sum,dum,big,temp;
int i,j,k;
int imax;
int nmax=100;
double tiny=1.0e-40;
imax=0;
for(i=1;i<=n;i++)
{
  for(j=1;j<=n;j++)
     a[i-1][j-1]=c[i-1][j-1];
}
d[0]=1;
for(i=1;i<=n;i++)
  {
  big=0.0;
  for(j=1;j<=n;j++)
    {
    if(Math.abs(a[i-1][j-1])>big) big=Math.abs(a[i-1][j-1]);
    }
  if(big==0) {System.out.println("singular matrix");return a;}
  vv[i-1]=1.0/big;
  }
for(j=1;j<=n;j++)
{
  for(i=1;i<j;i++)
    {
      sum=a[i-1][j-1];
      for(k=1;k<i;k++)
                {
                  sum-=a[i-1][k-1]*a[k-1][j-1];
                }
    a[i-1][j-1]=sum;
    }
  big=0;
  for(i=j;i<=n;i++)
    {
    sum=a[i-1][j-1];
    for(k=1;k<j;k++)
      {
      sum-=a[i-1][k-1]*a[k-1][j-1];
      }
    a[i-1][j-1]=sum;
    dum=vv[i-1]*Math.abs(sum);
```

```
    if(dum>=big)
      {
      imax=i;
      big=dum;
      }
    } //end of i=0
if(j != imax)
    {
    for(k=1;k<=n;k++)
      {
      dum=a[imax-1][k-1];
      a[imax-1][k-1]=a[j-1][k-1];
      a[j-1][k-1]=dum;
      }
    d[0]=-d[0];
    vv[imax-1]=vv[j-1];
    } //end of if
indx[j-1]=imax;
if(a[j-1][j-1]==0) a[j-1][j-1]=tiny;
if(j!=n)
  {
  dum=1.0/a[j-1][j-1];
  for(i=j+1;i<=n;i++)
    a[i-1][j-1]*=dum;
  }//endif
} //end for j=
return a;
}
public static double[] LUaxb(double a[][],double x[],int indx[])
{
//solves AX=B system of linear equation of LU decomposed matrix a
//(calculated by method LU)
int ii=0;
int i,j,ll=0;
double sum=0;
int n=a.length;
double b[];
b=new double[n];
for(i=1;i<=n;i++)
{
    b[i-1]=x[i-1];
}
for(i=1;i<=n;i++)
  {
  ll=indx[i-1];
  sum=b[ll-1];
  b[ll-1]=b[i-1];
  if(ii!=0)
     {
     for(j=ii;j<=(i-1);j++)
       {
       sum-=a[i-1][j-1]*b[j-1];
       }
     }
  else if(sum!=0) ii=i;
  b[i-1]=sum;
  }
for(i=n;i>=1;i--)
  {
  sum=b[i-1];
  if(i<n)
    {
    for(j=(i+1);j<=n;j++)
      {
      sum-=a[i-1][j-1]*b[j-1];
      }
    }
  b[i-1]=sum/a[i-1][i-1];
  }
return b;
}
public void ContourPlot()
{ ContourPlot pp=new ContourPlot( x,y,T,16,true,"Finite element","x","y");
  pp.plot();
}

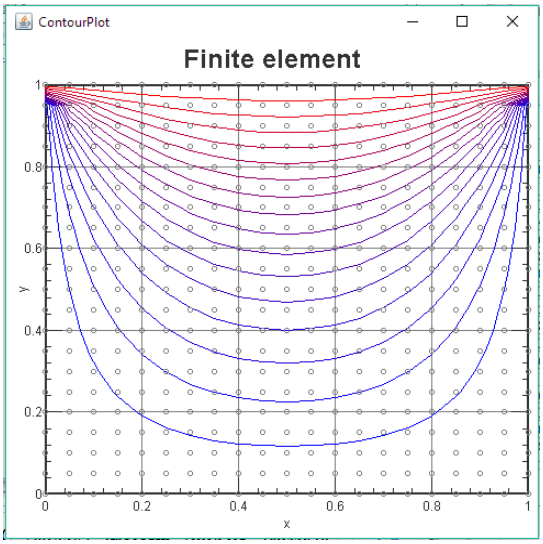
public void surfacePlot()
```

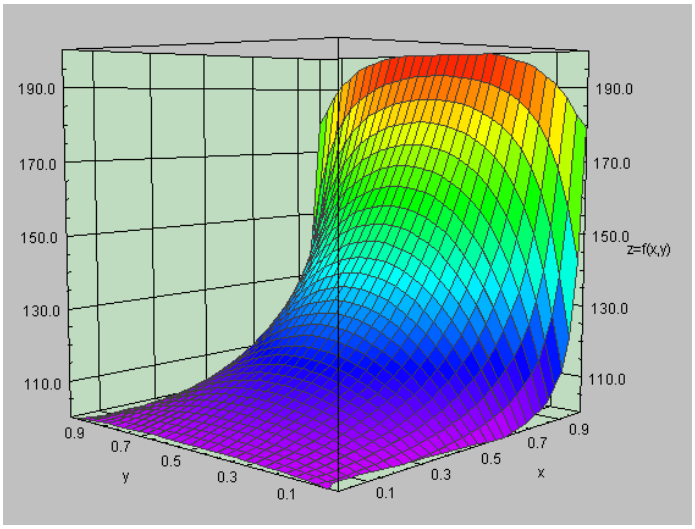
```
{    double b[][]=surfacePlot.dataformat(0,L,0,W,T);
     surfacePlot p1;
     p1=new surfacePlot(b);
     p1.plot();
}
public static double[] AXB(double a[][],double b[])
{
//Solution of system of linear equations by LU method
// note that the same calculation can be done by divide method.

Text.print(a,"K");
Text.print(b,"f");
int n=a.length;
double c[]=new double[n];
int d[]={1};
int indx[]=new int[n];
double e[][]=new double[n][n];
e=LU(a,indx,d);
c=LUaxb(e,b,indx);
return c;
} //end of AXB
public static void main(String arg[]) //throws RemoteException, VisADException
{double L=1.0;
 double W=1.0;
 int N=20,M=20;
 double hi=20.0;
 triangular_grid tg=new triangular_grid(L,W,M,N);
 int cond0=0;//x==0
 int cond1=1;//x==L
 int cond2=2;//y==0
 int cond3=3;//y==W
 double Ti0=100.0;
 double Ti1=100.0;
 double Ti2=100.0;
 double Ti3=200.0;
 tg.setTemperature(cond0,0.0,L,0.0,W,Ti0);
 tg.setTemperature(cond1,0.0,L,0.0,W,Ti1);
 tg.setTemperature(cond2,0.0,L,0.0,W,Ti2);
 tg.setTemperature(cond3,0.0,L,0.0,W,Ti3);
 //tg.setConvection(cond3,0.0,L,0.0,W,hi,Ti3);
 double k[]={50,50};
 tg.solve(1,k);
 Text.print(tg.T);
 tg.ContourPlot();
}}
```

data output(veri çıktısı) :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
100.0	100.2...	100.5...	100.79...	101.0...	101.23...	101.4...	101.55...	101.6...	101.71...	101.7...	101.71...	101.6...	101.55...	101.4...	101.23...	101.0...	100.79...	100.5...	100.27...	100.0
100.0	100.5...	101.0...	101.61...	102.0...	102.50...	102.8...	103.14...	103.3...	103.47...	103.5...	103.47...	103.3...	103.14...	102.8...	102.50...	102.0...	101.61...	101.0...	100.55...	100.0
100.0	100.8...	101.6...	102.46...	103.1...	103.83...	104.3...	104.81...	105.1...	105.32...	105.3...	105.32...	105.1...	104.81...	104.3...	103.83...	103.1...	102.46...	101.6...	100.85...	100.0
100.0	101.1...	102.3...	103.38...	104.3...	105.25...	106.0...	106.59...	107.0...	107.29...	107.3...	107.29...	107.0...	106.59...	106.0...	105.25...	104.3...	103.38...	102.3...	101.17...	100.0
100.0	101.5...	102.9...	104.39...	105.6...	106.81...	107.7...	108.54...	109.1...	109.44...	109.5...	109.44...	109.1...	108.54...	107.7...	106.81...	105.6...	104.39...	102.9...	101.51...	100.0
100.0	101.9...	103.7...	105.52...	107.1...	108.54...	109.7...	110.69...	111.3...	111.81...	111.9...	111.81...	111.3...	110.69...	109.7...	108.54...	107.1...	105.52...	103.7...	101.90...	100.0
100.0	102.3...	104.6...	106.79...	108.7...	110.49...	111.9...	113.11...	113.9...	114.46...	114.6...	114.46...	113.9...	113.11...	111.9...	110.49...	108.7...	106.79...	104.6...	102.35...	100.0
100.0	102.8...	105.6...	108.25...	110.6...	112.71...	114.4...	115.85...	116.8...	117.46...	117.6...	117.46...	116.8...	115.85...	114.4...	112.71...	110.6...	108.25...	105.6...	102.86...	100.0
100.0	103.4...	106.8...	109.94...	112.7...	115.27...	117.3...	118.97...	120.1...	120.85...	121.0...	120.85...	120.1...	118.97...	117.3...	115.27...	112.7...	109.94...	106.8...	103.46...	100.0
100.0	104.1...	108.1...	111.94...	115.3...	118.23...	120.6...	122.55...	123.9...	124.72...	124.9...	124.72...	123.9...	122.55...	120.6...	118.23...	115.3...	111.94...	108.1...	104.16...	100.0
100.0	105.0...	109.8...	114.30...	118.2...	121.69...	124.4...	126.67...	128.2...	129.14...	129.4...	129.14...	128.2...	126.67...	124.4...	121.69...	118.2...	114.30...	109.8...	105.01...	100.0
100.0	106.0...	111.8...	117.16...	121.8...	125.76...	128.9...	131.41...	133.1...	134.16...	134.5...	134.16...	133.1...	131.41...	128.9...	125.76...	121.8...	117.16...	111.8...	106.06...	100.0
100.0	107.3...	114.3...	120.66...	126.0...	130.58...	134.1...	136.88...	138.7...	139.88...	140.2...	139.88...	138.7...	136.88...	134.1...	130.58...	126.0...	120.66...	114.3...	107.38...	100.0
100.0	109.0...	117.5...	125.02...	131.2...	136.32...	140.2...	143.17...	145.1...	146.34...	146.7...	146.34...	145.1...	143.17...	140.2...	136.32...	131.2...	125.02...	117.5...	109.09...	100.0
100.0	111.4...	121.7...	130.58...	137.6...	143.18...	147.3...	150.37...	152.4...	153.59...	153.9...	153.59...	152.4...	150.37...	147.3...	143.18...	137.6...	130.58...	121.7...	111.42...	100.0
100.0	114.7...	127.6...	137.83...	145.6...	151.39...	155.5...	158.55...	160.5...	161.62...	161.9...	161.62...	160.5...	158.55...	155.5...	151.39...	145.6...	137.83...	127.6...	114.79...	100.0
100.0	120.1...	136.0...	147.53...	155.5...	161.19...	165.0...	167.73...	169.4...	170.42...	170.7...	170.42...	169.4...	167.73...	165.0...	161.19...	155.5...	147.53...	136.0...	120.13...	100.0
100.0	129.6...	148.9...	160.66...	167.9...	172.70...	175.8...	177.84...	179.1...	179.86...	180.0...	179.86...	179.1...	177.84...	175.8...	172.70...	167.9...	160.66...	148.9...	129.69...	100.0
100.0	149.7...	169.2...	178.22...	183.0...	185.82...	187.5...	188.71...	189.4...	189.80...	189.9...	189.80...	189.4...	188.71...	187.5...	185.82...	183.0...	178.22...	169.2...	149.72...	100.0
200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0

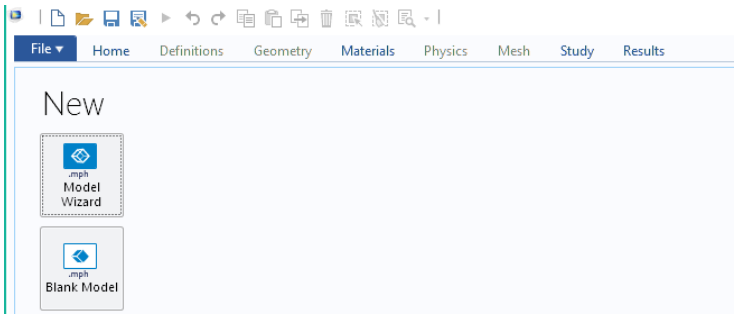




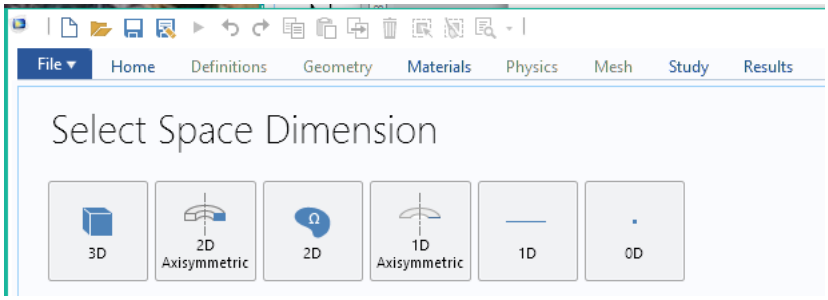
One of the advantages of finite element methods is integration of boundary values into the basic differential equation. It will allow to develop a general diferential equation solution packages, some commercial and public Access packages are available to solve heat transfer problems. Some of the package programs are:

Package	Web adres	Commercial/public licence	Company
COMSOL	https://www.comsol.com	Commercial licence	COMSOL.ltd
ABACUS	https://www.3ds.com/products-services/simulia/products/abaqus/	Commercial licence	
NASTRAN	https://www.autodesk.com/education/free-software/nastran	Commercial licence /public licence available for students and educators	AUTODESK
Open Source Field Operation and Manipulation (OpenFOAM)	https://openfoam.org/version/6/	Public licence	OpenCFD Ltd
ANSYS FLUENT	https://www.ansys.com/products/fluids/ansys-fluent	Comercial licence	ANSYS

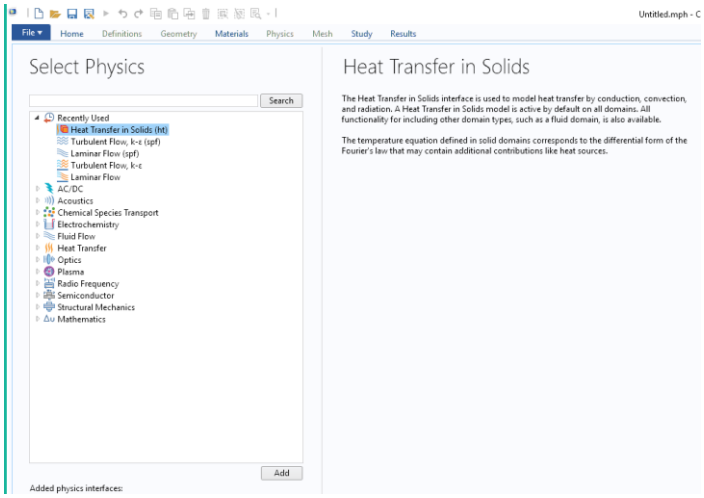
As an example, we will run comsol package to solve the previous problem

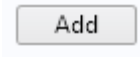


Model Wisard will be selected



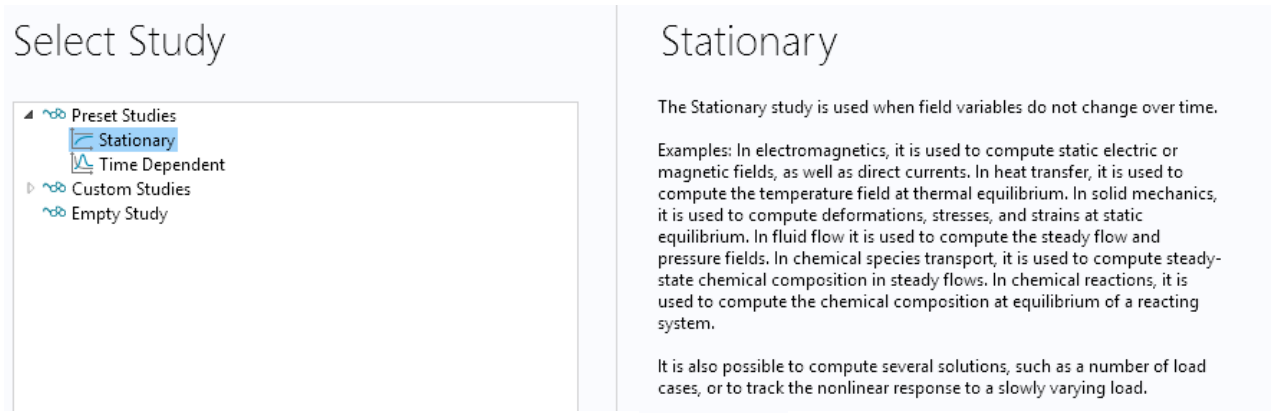
2D will be selected



Heat transfer in solids will be selected and then push 

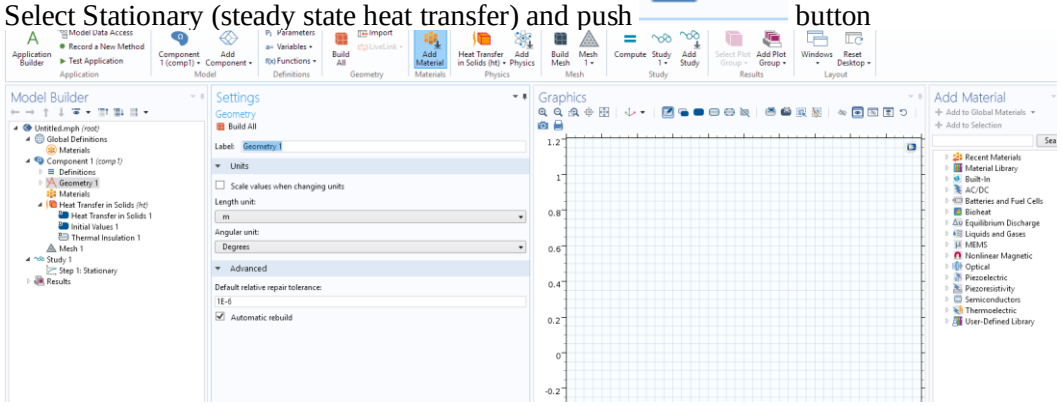
button then push study 

button

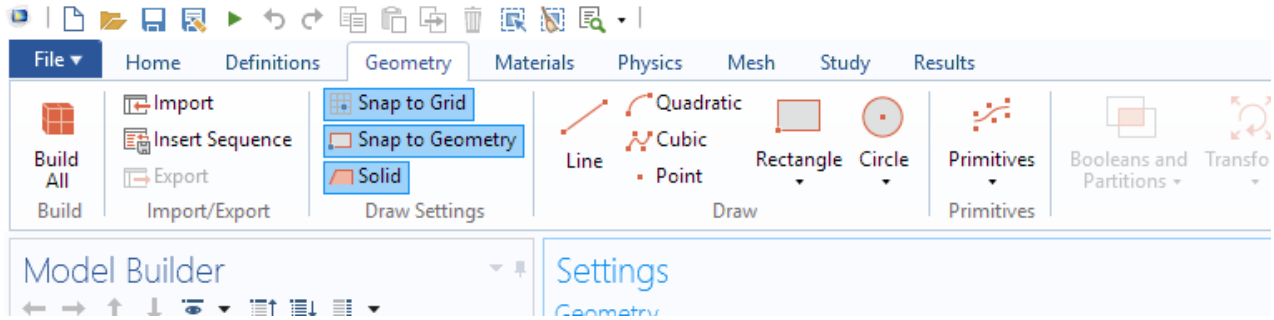


Select Stationary (steady state heat transfer) and push 

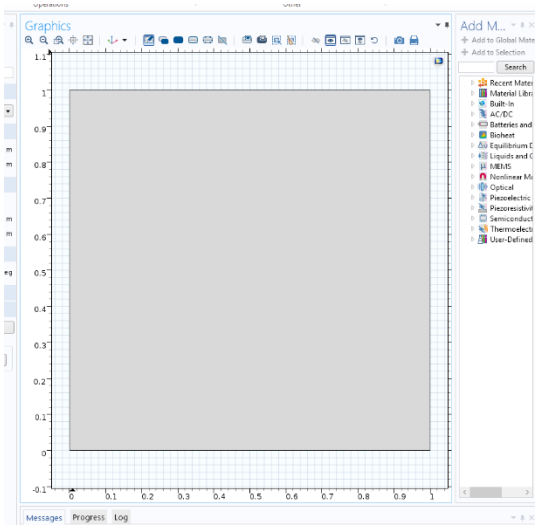
button



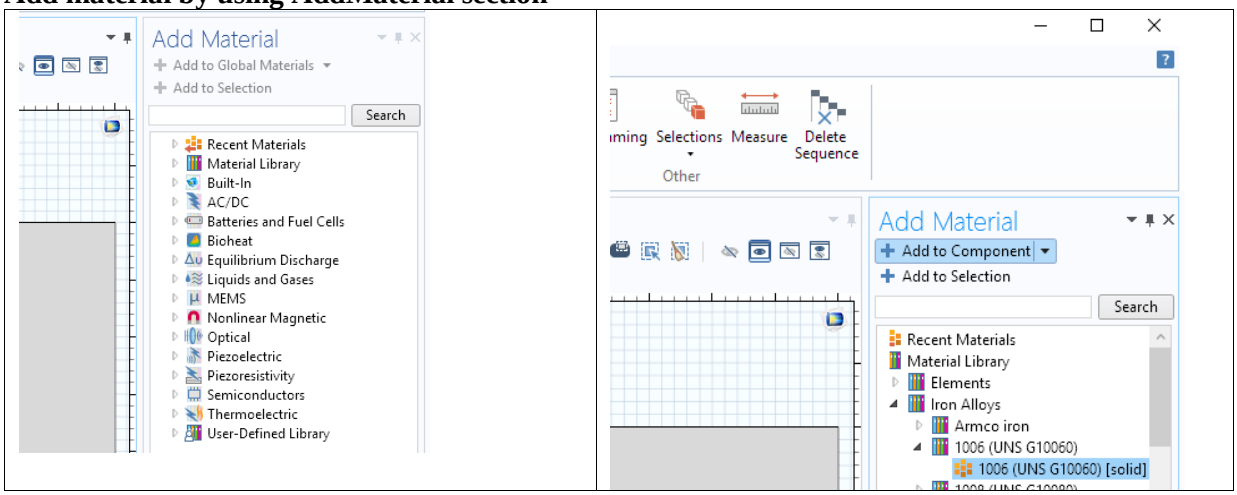
Page will open to draw the shape. By using Geometry and Rectangle draw a square 1 m x 1 m dimensions



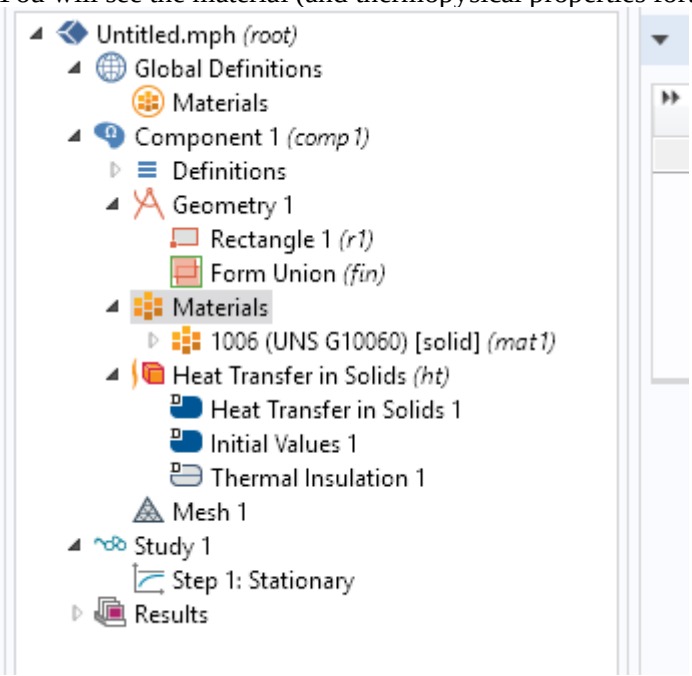
Zoom extent to put square at the center of your drawing screen



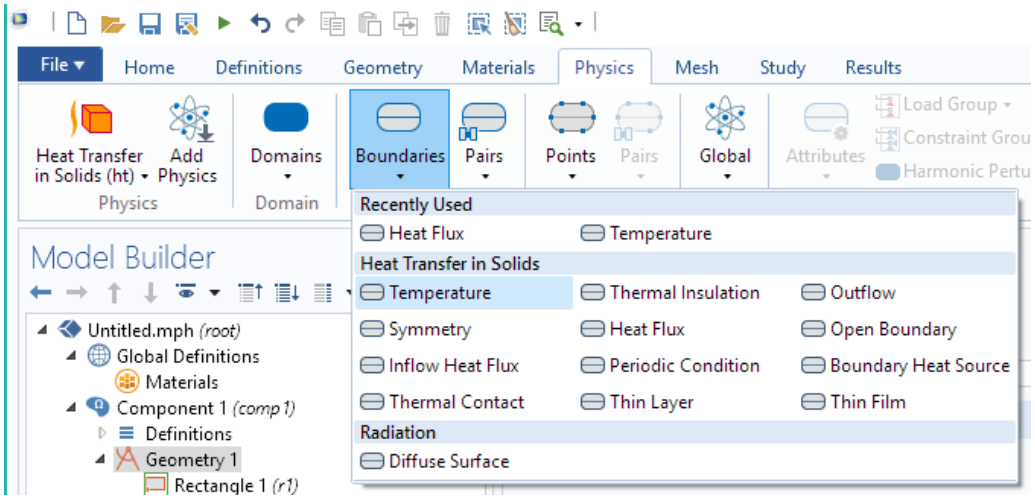
Add material by using AddMaterial section



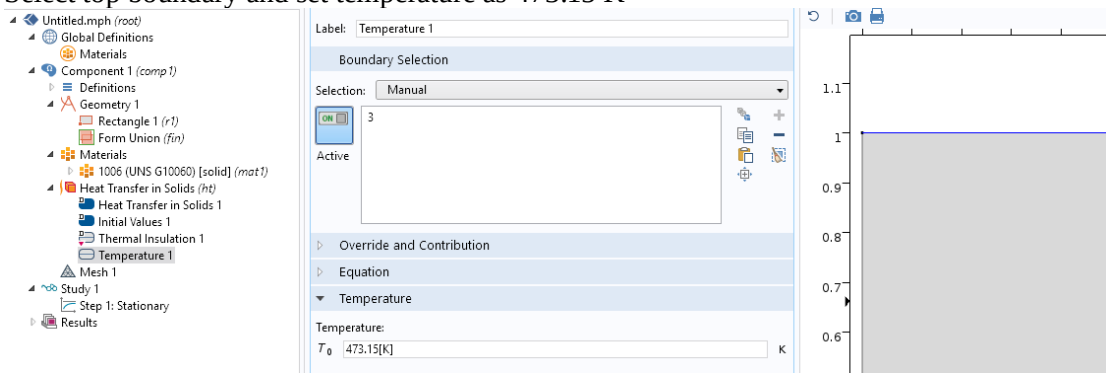
You will see the material (and thermopysical properties fort hat material) added into input section



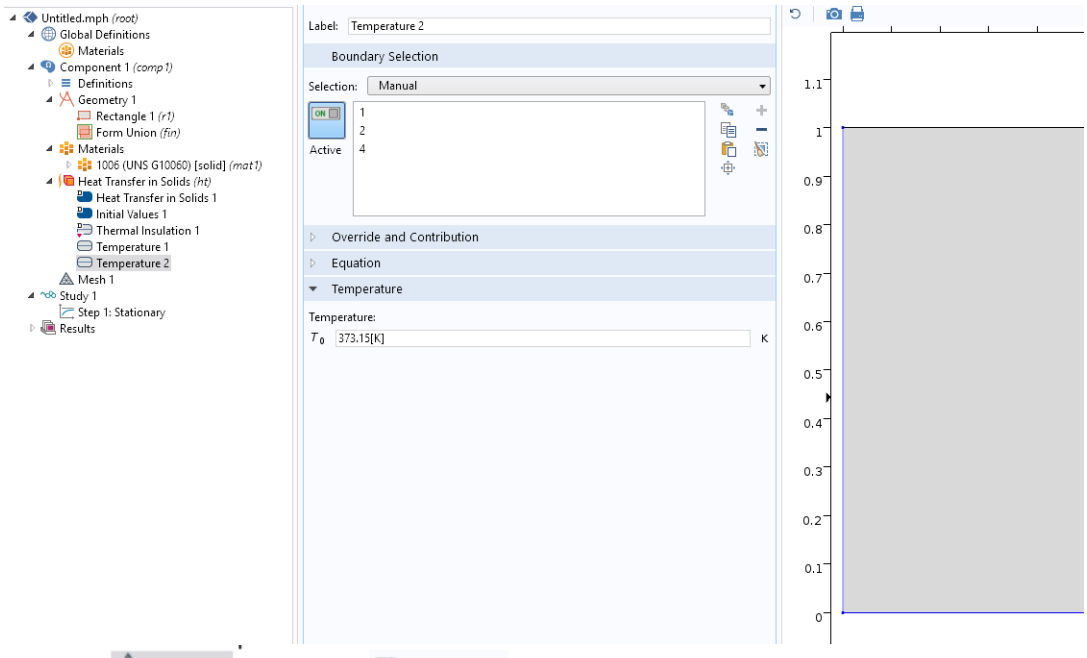
From physics and boundaries select temperature boundary



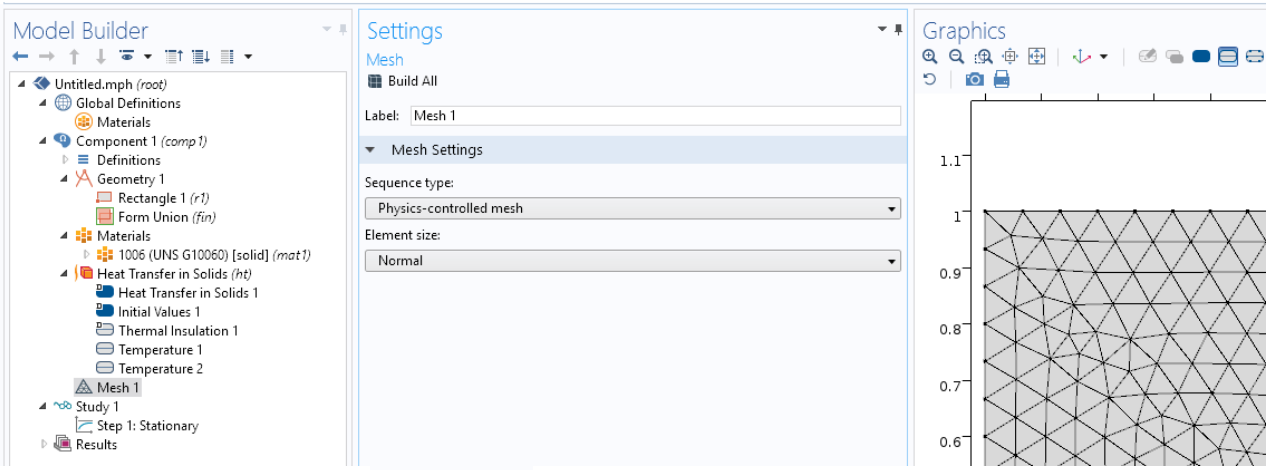
Select top boundary and set temperature as 473.15 K



Repeat the same process (add a new temperature profile select the other surfaces) and select temperature of other surfaces as 373.15 K

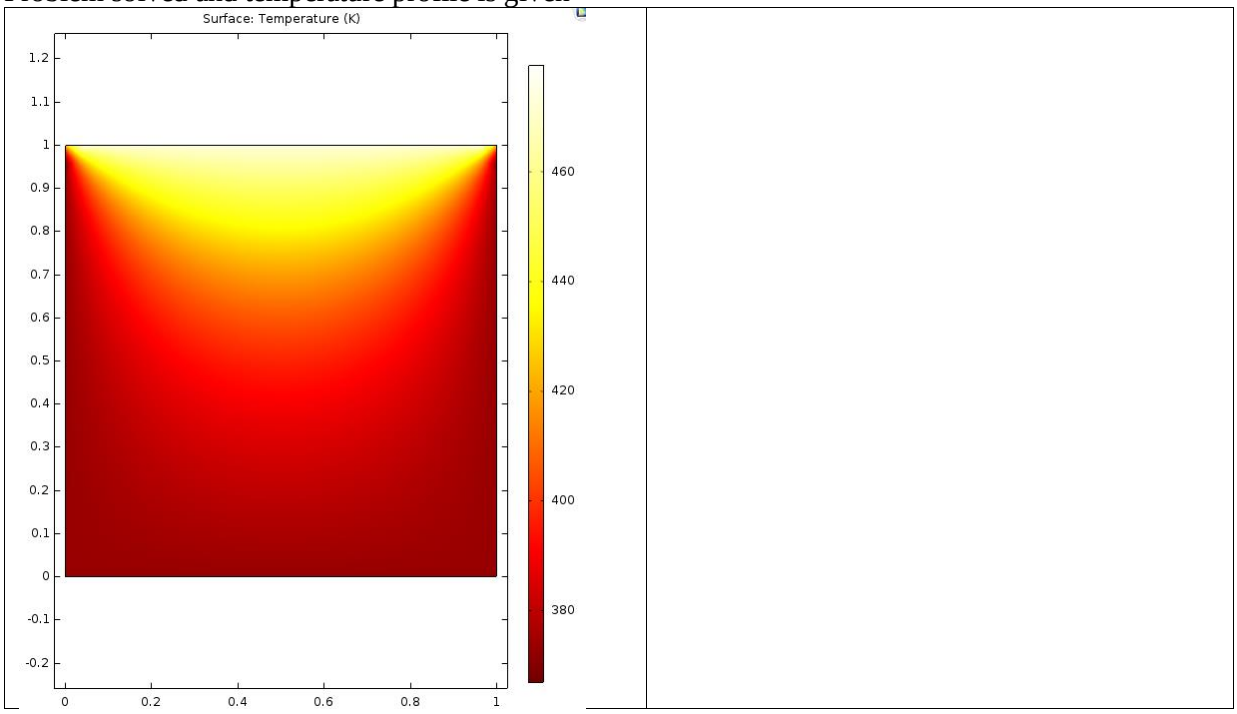


Select Mesh 1 and push Build All



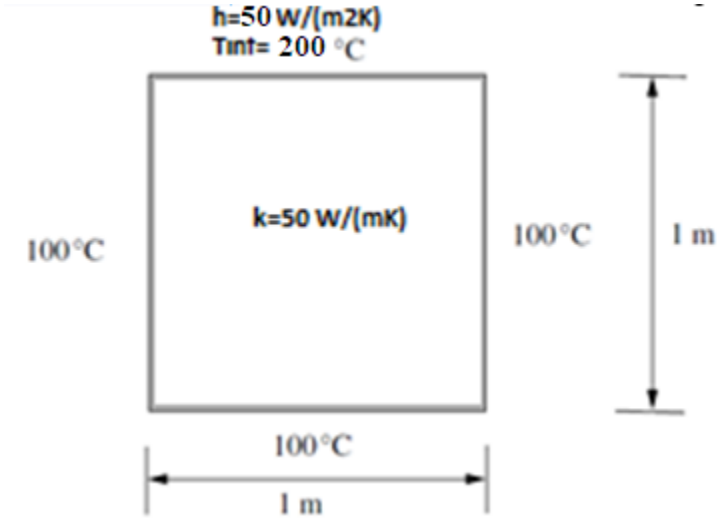
Select Study 1 and push Compute

Problem solved and temperature profile is given



PROBLEM 5.2 NON-HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CARTESIAN COORDINATES, FINITE ELEMENT SOLUTION

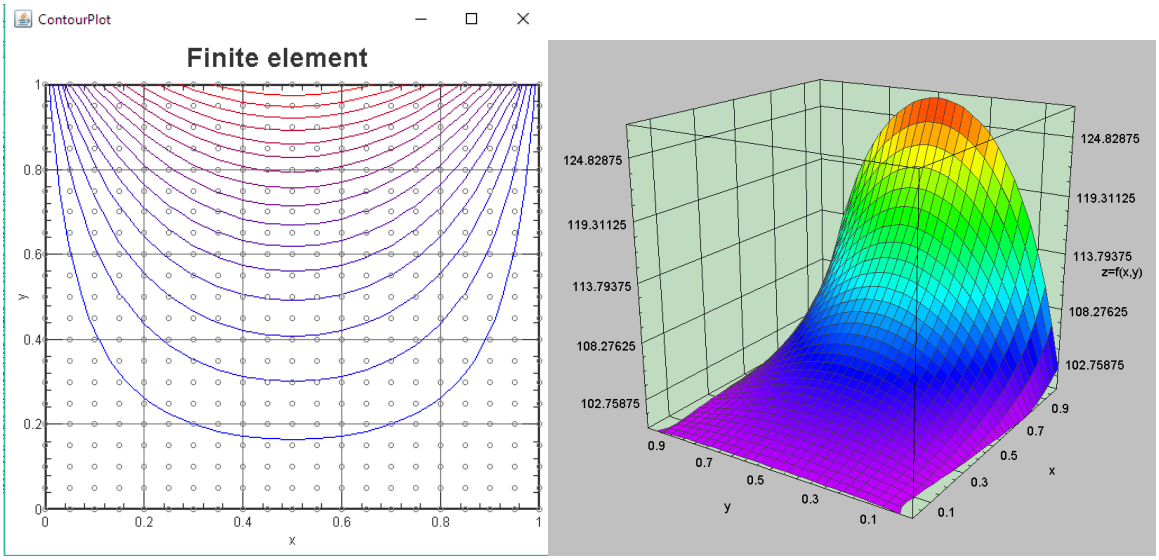
A second problem: In this one, three side temperatures specified as 100 C, and in the third side convective heat transfer is existed.



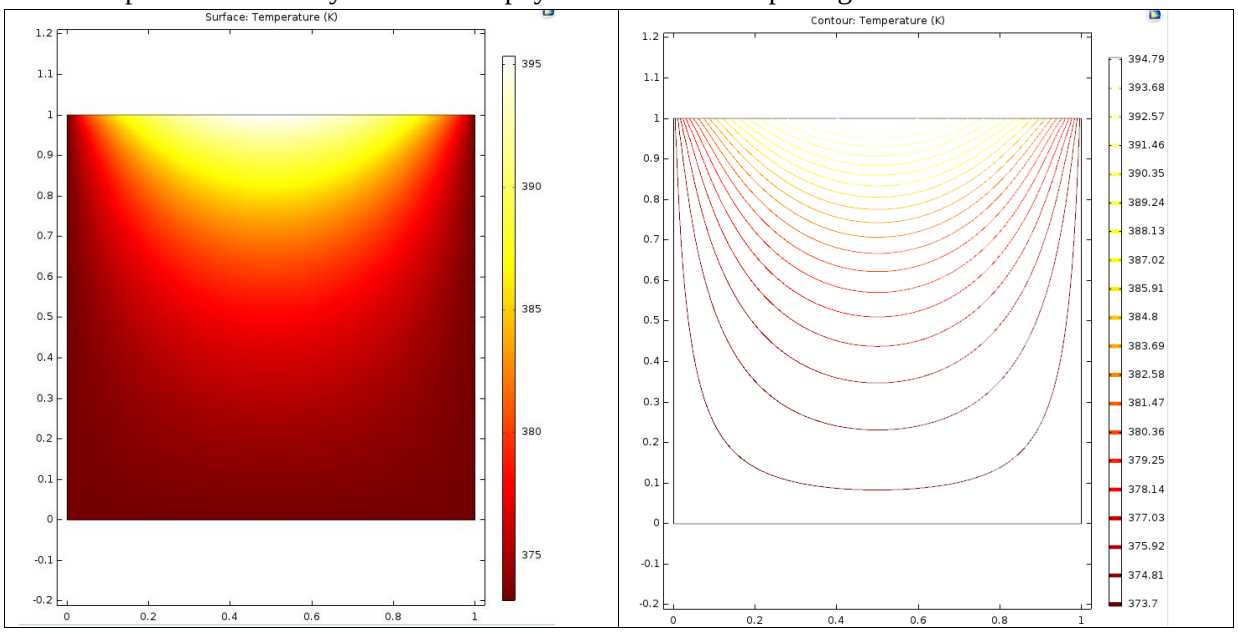

```
 public static void main(String arg[]) //throws RemoteException, VisADException
{double L=1.0;
 double W=1.0;
 int N=20,M=20; //Number of division in x and y side
 double hi=50.0;
 triangular_grid tg=new triangular_grid(L,W,M,N);
 int cond0=0;//x==0
 int cond1=1;//x==L
 int cond2=2;//y==0
 int cond3=3;//y==W
 double Ti0=100.0;
 double Ti1=100.0;
 double Ti2=100.0;
 double Ti3=200.0;
 tg.setTemperature(cond0,0.0,L,0.0,W,Ti0);
 tg.setTemperature(cond1,0.0,L,0.0,W,Ti1);
 tg.setTemperature(cond2,0.0,L,0.0,W,Ti2);
tg.setConvection(cond3,0.0,L,0.0,W,hi,Ti3);
 double k[]={50,50};
 tg.solve(1,k);
 Text.print(tg.T);
 tg.ContourPlot();
 tg.surfacePlot();
}
}
```

data output(veri çıktısı) :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
100.0	100.065...	100.129...	100.190...	100.246...	100.296...	100.339...	100.373...	100.398...	100.413...	100.419...	100.413...	100.398...	100.373...	100.339...	100.296...	100.246...	100.190...	100.129...	100.065...	100.0
100.0	100.133...	100.263...	100.386...	100.499...	100.601...	100.687...	100.756...	100.807...	100.837...	100.848...	100.837...	100.807...	100.756...	100.687...	100.601...	100.499...	100.386...	100.263...	100.133...	100.0
100.0	100.203...	100.402...	100.591...	100.765...	100.920...	101.051...	101.157...	101.235...	101.282...	101.298...	101.282...	101.235...	101.157...	101.051...	100.920...	100.765...	100.591...	100.402...	100.203...	100.0
100.0	100.279...	100.552...	100.811...	101.049...	101.262...	101.442...	101.587...	101.693...	101.758...	101.780...	101.758...	101.693...	101.587...	101.442...	101.262...	101.049...	100.811...	100.552...	100.279...	100.0
100.0	100.362...	100.716...	101.051...	101.360...	101.635...	101.869...	102.056...	102.193...	102.277...	102.305...	102.277...	102.193...	102.056...	101.869...	101.635...	101.360...	101.051...	100.716...	100.362...	100.0
100.0	100.455...	100.898...	101.319...	101.705...	102.049...	102.341...	102.576...	102.747...	102.851...	102.886...	102.851...	102.747...	102.576...	102.341...	102.049...	101.705...	101.319...	100.898...	100.455...	100.0
100.0	100.559...	101.103...	101.619...	102.094...	102.514...	102.872...	103.158...	103.367...	103.494...	103.537...	103.494...	103.367...	103.158...	102.872...	102.514...	102.094...	101.619...	101.103...	100.559...	100.0
100.0	100.678...	101.337...	101.962...	102.535...	103.043...	103.474...	103.818...	104.070...	104.222...	104.273...	104.222...	104.070...	103.818...	103.474...	103.043...	102.535...	101.962...	101.337...	100.678...	100.0
100.0	100.814...	101.607...	102.356...	103.042...	103.648...	104.162...	104.572...	104.870...	105.051...	105.112...	105.051...	104.870...	104.572...	104.162...	103.648...	103.042...	102.356...	101.607...	100.814...	100.0
100.0	100.974...	101.920...	102.812...	103.628...	104.347...	104.954...	105.437...	105.788...	106.001...	106.072...	106.001...	105.788...	105.437...	104.954...	104.347...	103.628...	102.812...	101.920...	100.974...	100.0
100.0	101.160...	102.287...	103.346...	104.310...	105.158...	105.870...	106.435...	106.844...	107.092...	107.175...	107.092...	106.844...	106.435...	105.870...	105.158...	104.310...	103.346...	102.287...	101.160...	100.0
100.0	101.382...	102.720...	103.974...	105.110...	106.103...	106.933...	107.589...	108.062...	108.348...	108.443...	108.348...	108.062...	107.589...	106.933...	106.103...	105.110...	103.974...	102.720...	101.382...	100.0
100.0	101.648...	103.238...	104.720...	106.054...	107.210...	108.171...	108.926...	109.467...	109.793...	109.902...	109.793...	109.467...	108.926...	108.171...	107.210...	106.054...	104.720...	103.238...	101.648...	100.0
100.0	101.972...	103.865...	105.615...	107.174...	108.513...	109.616...	110.475...	111.089...	111.456...	111.579...	111.456...	111.089...	110.475...	109.616...	108.513...	107.174...	105.615...	103.865...	101.972...	100.0
100.0	102.374...	104.636...	106.700...	108.514...	110.052...	111.303...	112.270...	112.956...	113.365...	113.501...	113.365...	112.956...	112.270...	111.303...	110.052...	108.514...	106.700...	104.636...	102.374...	100.0
100.0	102.888...	105.604...	108.036...	110.131...	111.876...	113.277...	114.346...	115.099...	115.546...	115.694...	115.546...	115.099...	114.346...	113.277...	111.876...	110.131...	108.036...	105.604...	102.888...	100.0
100.0	103.575...	106.857...	109.708...	112.098...	114.044...	115.581...	116.740...	117.549...	118.026...	118.184...	118.026...	117.549...	116.740...	115.581...	114.044...	112.098...	109.708...	106.857...	103.575...	100.0
100.0	104.554...	108.541...	111.842...	114.508...	116.623...	118.262...	119.483...	120.329...	120.826...	120.990...	120.826...	120.329...	119.483...	118.262...	116.623...	114.508...	111.842...	108.541...	104.554...	100.0
100.0	106.101...	110.912...	114.611...	117.469...	119.677...	121.361...	122.603...	123.458...	123.959...	124.124...	123.959...	123.458...	122.603...	121.361...	119.677...	117.469...	114.611...	110.912...	106.101...	100.0
100.0	108.940...	114.393...	118.222...	121.079...	123.254...	124.901...	126.111...	126.942...	127.427...	127.587...	127.427...	126.942...	126.111...	124.901...	123.254...	121.079...	118.222...	114.393...	108.940...	100.0

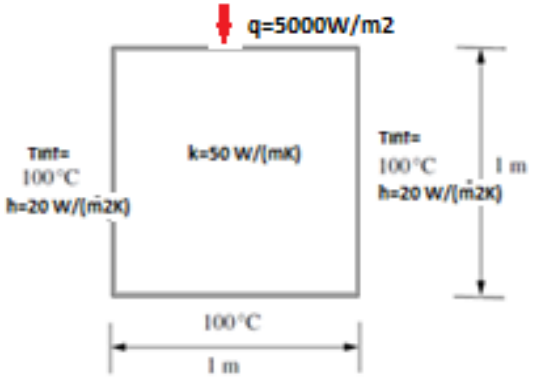


The same problem solved by comsol multiphysics finite element package:



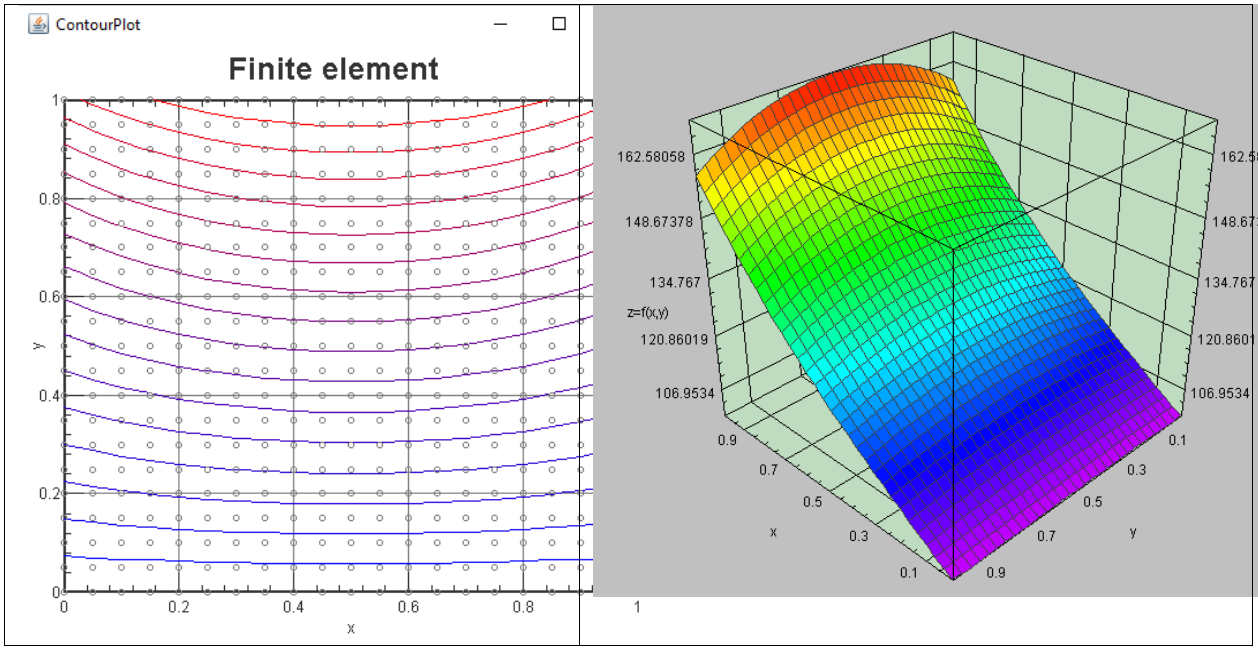
PROBLEM 5.3 NON-HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CARTESIAN COORDINATES, FINITE ELEMENT SOLUTION

A third problem: In this,one side temperatures specified as 100 C, and in second and third side convective heat transfer is existed. In the fourth side heat flux is given



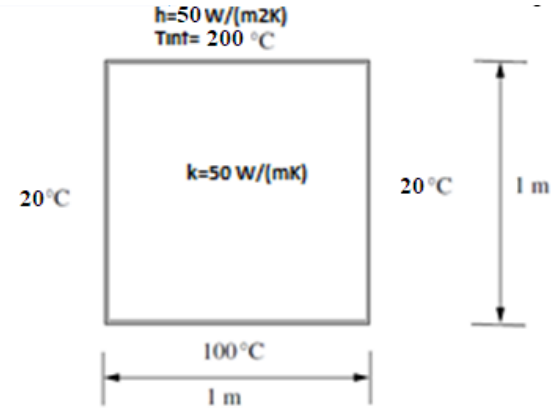
```
 public static void main(String arg[]) //throws RemoteException, VisADException
{double L=1.0;
 double W=1.0;
 int N=20,M=20;
 double hi=50.0;
 triangular_grid tg=new triangular_grid(L,W,M,N);
 int cond0=0;//x==0
 int cond1=1;//x==L
 int cond2=2;//y==0
 int cond3=3;//y==W
 double Ti0=100.0;
 double Ti1=100.0;
 double Ti2=100.0;
  double qi=-5000.0; //W/m2
 tg.setTemperature(cond0,0.0,L,0.0,W,Ti0);
 tg.setConvection(cond1,0.0,L,0.0,W,hi,Ti1);
 tg.setConvection(cond2,0.0,L,0.0,W,hi,Ti1);
 tg.setHeatFlux(cond3,0.0,L,0.0,W,qi);
 double k[]={50,50};
 tg.solve(1,k);
 Text.print(tg.T);
 tg.ContourPlot();
 tg.surfacePlot();
}
```

100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.
102.15	102.25	102.34	102.42	102.50	102.56	102.61	102.65	102.68	102.69	102.70	102.69	102.68	102.65	102.61	102.56	102.
104.30	104.51	104.69	104.86	105.00	105.13	105.23	105.31	105.37	105.40	105.41	105.40	105.37	105.31	105.23	105.13	105.
106.47	106.78	107.06	107.31	107.53	107.72	107.87	107.99	108.08	108.13	108.15	108.13	108.08	107.99	107.87	107.72	107.
108.68	109.09	109.47	109.80	110.09	110.34	110.55	110.71	110.82	110.89	110.91	110.89	110.82	110.71	110.55	110.34	110.
110.92	111.44	111.91	112.33	112.70	113.01	113.27	113.47	113.61	113.70	113.73	113.70	113.61	113.47	113.27	113.01	112.
113.20	113.83	114.41	114.91	115.36	115.74	116.05	116.29	116.46	116.57	116.60	116.57	116.46	116.29	116.05	115.74	115.
115.55	116.29	116.96	117.56	118.08	118.53	118.89	119.18	119.38	119.50	119.54	119.50	119.38	119.18	118.89	118.53	118.
117.96	118.82	119.60	120.29	120.89	121.40	121.82	122.14	122.38	122.52	122.57	122.52	122.38	122.14	121.82	121.40	120.
120.45	121.43	122.31	123.10	123.78	124.36	124.84	125.21	125.47	125.63	125.68	125.63	125.47	125.21	124.84	124.36	123.
123.04	124.14	125.13	126.01	126.78	127.43	127.96	128.37	128.67	128.85	128.91	128.85	128.67	128.37	127.96	127.43	126.
125.73	126.95	128.06	129.04	129.89	130.61	131.20	131.66	131.98	132.18	132.25	132.18	131.98	131.66	131.20	130.61	129.
128.53	129.89	131.12	132.20	133.13	133.93	134.57	135.07	135.43	135.65	135.72	135.65	135.43	135.07	134.57	133.93	133.
131.47	132.97	134.31	135.50	136.52	137.39	138.09	138.63	139.02	139.26	139.33	139.26	139.02	138.63	138.09	137.39	136.
134.56	136.20	137.67	138.96	140.07	141.01	141.76	142.35	142.77	143.02	143.10	143.02	142.77	142.35	141.76	141.01	140.
137.81	139.61	141.20	142.60	143.80	144.80	145.61	146.24	146.68	146.95	147.04	146.95	146.68	146.24	145.61	144.80	143.
141.26	143.22	144.94	146.44	147.72	148.78	149.65	150.31	150.78	151.06	151.15	151.06	150.78	150.31	149.65	148.78	147.
144.93	147.05	148.90	150.50	151.85	152.97	153.88	154.57	155.06	155.35	155.45	155.35	155.06	154.57	153.88	152.97	151.
148.87	151.15	153.12	154.80	156.22	157.38	158.32	159.04	159.54	159.84	159.94	159.84	159.54	159.04	158.32	157.38	156.
153.13	155.57	157.63	159.37	160.83	162.02	162.98	163.71	164.23	164.53	164.64	164.53	164.23	163.71	162.98	162.02	160.
157.84	160.36	162.46	164.22	165.69	166.90	167.87	168.60	169.12	169.43	169.53	169.43	169.12	168.60	167.87	166.90	165.



The same problem solved by comsol multiphysics finite element package:

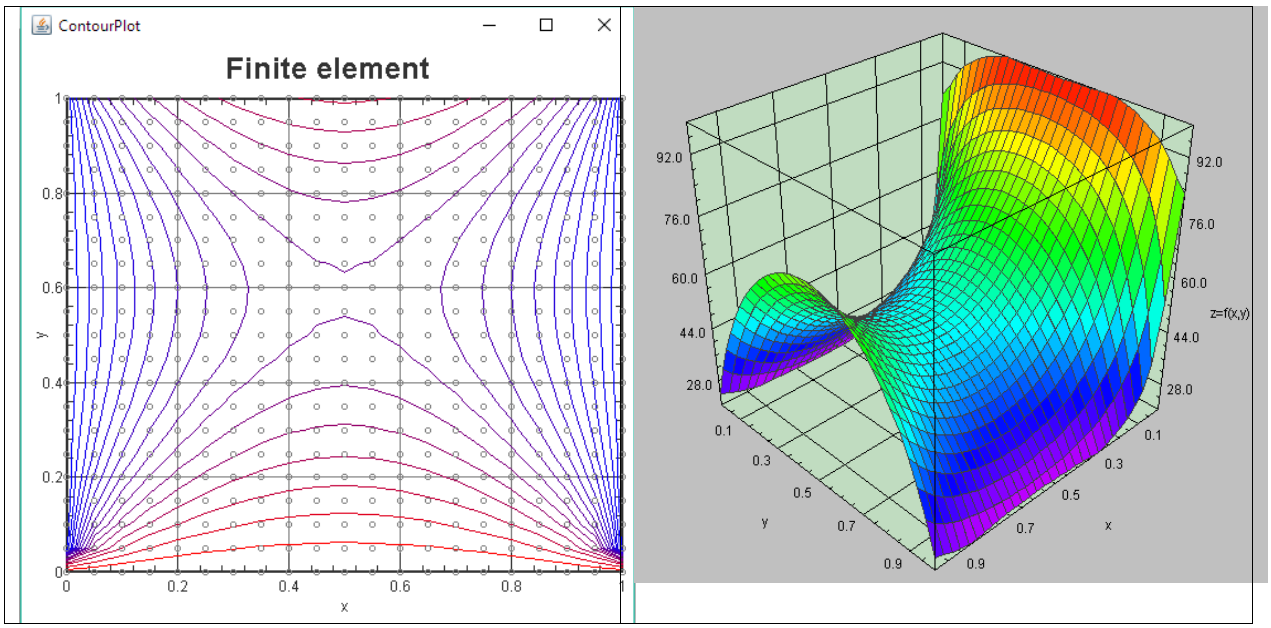
PROBLEM 5.4 NON-HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CARTESIAN COORDINATES, FINITE ELEMENT SOLUTION



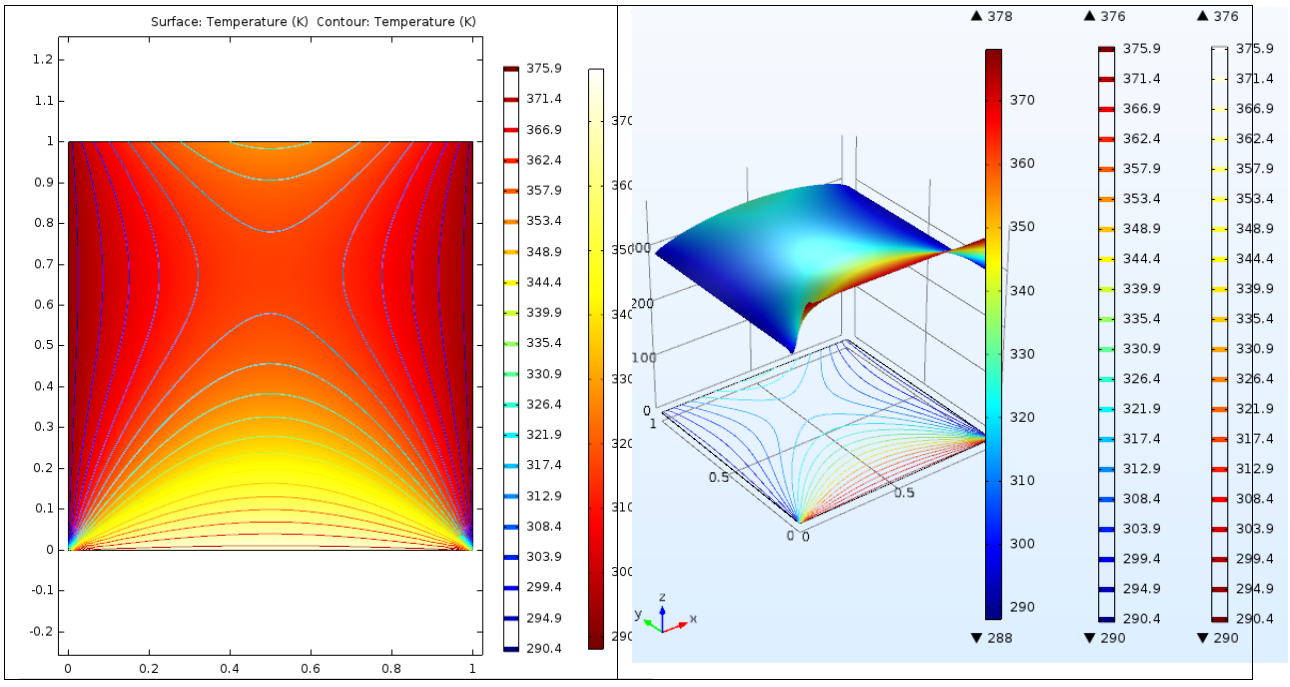

```
public static void main(String arg[]) //throws RemoteException, VisADException
{double L=1.0;
 double W=1.0;
 int N=20,M=20;
 double hi=50.0;
 triangular_grid tg=new triangular_grid(L,W,M,N);
 int cond0=0;//x==0
 int cond1=1;//x==L
 int cond2=2;//y==0
 int cond3=3;//y==W
 double Ti0=20.0;
 double Ti1=20.0;
 double Ti2=100.0;
 double Ti3=200.0;
 tg.setTemperature(cond0,0.0,L,0.0,W,Ti0);
 tg.setTemperature(cond1,0.0,L,0.0,W,Ti1);
 tg.setTemperature(cond2,0.0,L,0.0,W,Ti2);
 tg.setConvection(cond3,0.0,L,0.0,W,hi,Ti3);
 double k[]={50,50};
 tg.solve(1,k);
 Text.print(tg.T);
 tg.ContourPlot();
 tg.surfacePlot();
}
}
```

data output(veri çıktısı) :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
20.0	59.9132...	75.6319...	82.9615...	86.9052...	89.2586...	90.7537...	91.7246...	92.3372...	92.6772...	92.7864...	92.6772...	92.3372...	91.7246...	90.7537...	89.2586...	86.9052...	82.9615...	75.6319...	59.9132...	20.0
20.0	44.0210...	59.6531...	69.3089...	75.4006...	79.3756...	82.0318...	83.8074...	84.9469...	85.5854...	85.7913...	85.5854...	84.9469...	83.8074...	82.0318...	79.3756...	75.4006...	69.3089...	59.6531...	44.0210...	20.0
20.0	36.5178...	49.6506...	59.2204...	66.0126...	70.8116...	74.1903...	76.5263...	78.0575...	78.9263...	79.2080...	78.9263...	78.0575...	76.5263...	74.1903...	70.8116...	66.0126...	59.2204...	49.6506...	36.5178...	20.0
20.0	32.3996...	43.2112...	51.9096...	58.6177...	63.6680...	67.3914...	70.0501...	71.8306...	72.8542...	73.1881...	72.8542...	71.8306...	70.0501...	67.3914...	63.6680...	58.6177...	51.9096...	43.2112...	32.3996...	20.0
20.0	29.8695...	38.8850...	46.5889...	52.8807...	57.8514...	61.6572...	64.4521...	66.3608...	67.4716...	67.8362...	67.4716...	66.3608...	64.4521...	61.6572...	57.8514...	52.8807...	46.5889...	38.8850...	29.8695...	20.0
20.0	28.1935...	35.8704...	42.6805...	48.4646...	53.1997...	56.9339...	59.7401...	61.6889...	62.8352...	63.2135...	62.8352...	61.6889...	59.7401...	56.9339...	53.1997...	48.4646...	42.6805...	35.8704...	28.1935...	20.0
20.0	27.0341...	33.7226...	39.7980...	45.0976...	49.5488...	53.1387...	55.8857...	57.8193...	58.9669...	59.3472...	58.9669...	57.8193...	55.8857...	53.1387...	49.5488...	45.0976...	39.7980...	33.7226...	27.0341...	20.0
20.0	26.2202...	32.1880...	37.6913...	42.5790...	46.7592...	50.1863...	52.8448...	54.7357...	55.8659...	56.2417...	55.8659...	54.7357...	52.8448...	50.1863...	46.7592...	42.5790...	37.6913...	32.1880...	26.2202...	20.0
20.0	25.6588...	31.1177...	36.2003...	40.7680...	44.7227...	48.0024...	50.5716...	52.4129...	53.5191...	53.8879...	53.5191...	52.4129...	50.5716...	48.0024...	44.7227...	40.7680...	36.2003...	31.1177...	25.6588...	20.0
20.0	25.2972...	30.4238...	35.2243...	39.5698...	43.3613...	46.5289...	49.0263...	50.8251...	51.9096...	52.2719...	51.9096...	50.8251...	49.0263...	46.5289...	43.3613...	39.5698...	35.2243...	30.4238...	25.2972...	20.0
20.0	25.1064...	30.0559...	34.7034...	38.9255...	42.6239...	45.7256...	48.1794...	49.9518...	51.0224...	51.3803...	51.0224...	49.9518...	48.1794...	45.7256...	42.6239...	38.9255...	34.7034...	30.0559...	25.1064...	20.0
20.0	25.0727...	29.9898...	34.6079...	38.8049...	42.4833...	45.5702...	48.0141...	49.7803...	50.8477...	51.2047...	50.8477...	49.7803...	48.0141...	45.5702...	42.4833...	38.8049...	34.6079...	29.9898...	25.0727...	20.0
20.0	25.1946...	30.2227...	34.9336...	39.2028...	42.9341...	46.0579...	48.5262...	50.3078...	51.3834...	51.7430...	51.3834...	50.3078...	48.5262...	46.0579...	42.9341...	39.2028...	34.9336...	30.2227...	25.1946...	20.0
20.0	25.4832...	30.7728...	35.7012...	40.1385...	43.9924...	47.2011...	49.7252...	51.5411...	52.6352...	53.0005...	52.6352...	51.5411...	49.7252...	47.2011...	43.9924...	40.1385...	35.7012...	30.7728...	25.4832...	20.0
20.0	25.9653...	31.6842...	36.9597...	41.6576...	45.6959...	49.0287...	51.6325...	53.4963...	54.6156...	54.9888...	54.6156...	53.4963...	51.6325...	49.0287...	45.6959...	41.6576...	36.9597...	31.6842...	25.9653...	20.0
20.0	26.6939...	33.0390...	38.7959...	43.8363...	48.1048...	51.5854...	54.2799...	56.1961...	57.3421...	57.7234...	57.3421...	56.1961...	54.2799...	51.5854...	48.1048...	43.8363...	38.7959...	33.0390...	26.6939...	20.0
20.0	27.7715...	34.9818...	41.3487...	46.7869...	51.3016...	54.9281...	57.7055...	59.6661...	60.8332...	61.2208...	60.8332...	59.6661...	57.7055...	54.9281...	51.3016...	46.7869...	41.3487...	34.9818...	27.7715...	20.0
20.0	29.4102...	37.7681...	44.8303...	50.6610...	55.3864...	59.1201...	61.9478...	63.9295...	65.1040...	65.4931...	65.1040...	63.9295...	61.9478...	59.1201...	55.3864...	50.6610...	44.8303...	37.7681...	29.4102...	20.0
20.0	32.1013...	41.8501...	49.5435...	55.6403...	60.4631...	64.2182...	67.0362...	69.0000...	70.1601...	70.5439...	70.1601...	69.0000...	67.0362...	64.2182...	60.4631...	55.6403...	49.5435...	41.8501...	32.1013...	20.0
20.0	37.1451...	47.9873...	55.8532...	61.8935...	66.6075...	70.2532...	72.9790...	74.8742...	75.9924...	76.3622...	75.9924...	74.8742...	72.9790...	70.2532...	66.6075...	61.8935...	55.8532...	47.9873...	37.1451...	20.0

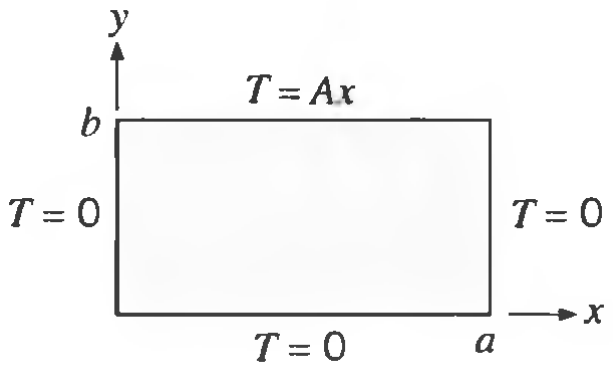


The same problem solved by comsol multiphysics finite element package:



PROBLEM 5.5 NON-HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CARTESIAN COORDINATES, FINITE ELEMENT SOLUTION

A two dimensional rectangular plate is subjected to the boundary conditions shown in the figure. Derive an expression for the steady-state temperature distribution T(x,y) by using finite difference method



Take: L=1 m W=1 m A=100 T=100x

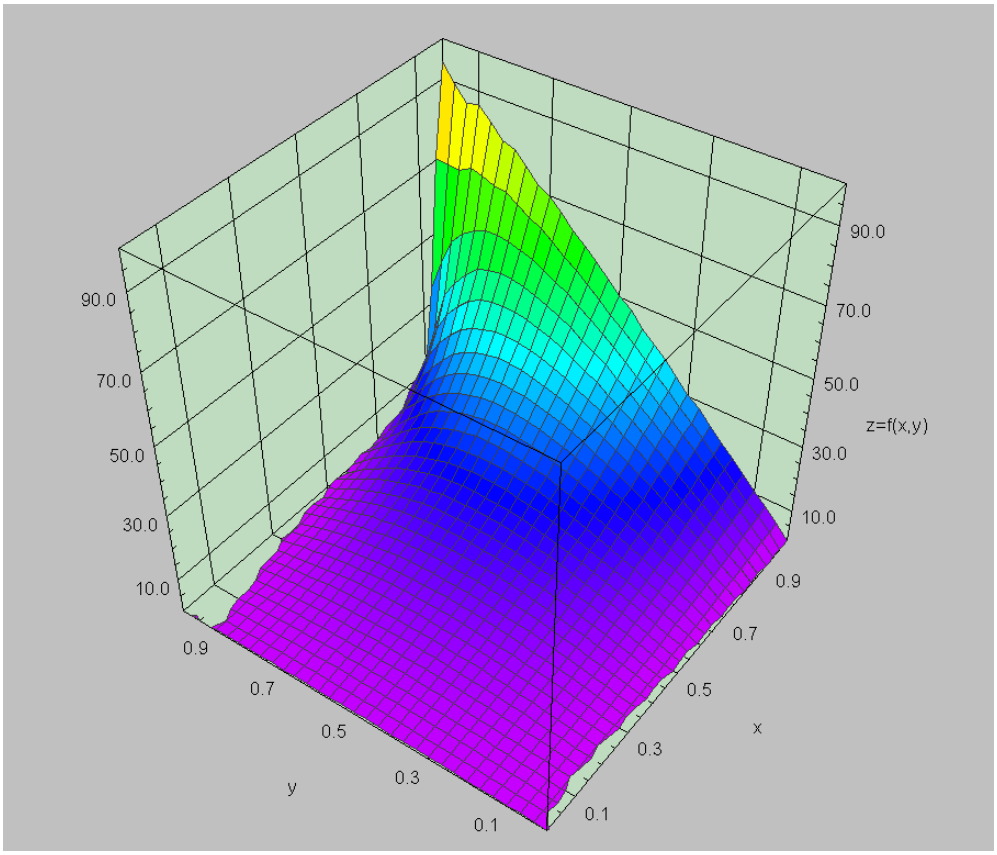
```
/variable function temperature profile
public void setTemperature(int cond,double L0,double L,double W0,double W, if_x Tx)
{//cond 0: x==L0
 //cond 1: x==L
 //cond 2: y==W0
 //cond 3: y==W
 for(int i=0;i<K2;i++)
 {  if(cond==0)
    {if(xi[i]==L0) {Ti[i]=Tx.func(yi[i]);bi[i]=true;}}
    else if(cond==1)
    {if(xi[i]==L) {Ti[i]=Tx.func(yi[i]);bi[i]=true;}}
    else if(cond==2)
    {if(yi[i]==W0) {Ti[i]=Tx.func(xi[i]);bi[i]=true;}}
    else if(cond==3)
    {if(yi[i]==W) {Ti[i]=Tx.func(xi[i]);bi[i]=true;}}
 }
}
```

```
public static void main(String arg[]) //throws RemoteException, VisADException
{double L=1.0;
 double W=1.0;
 int N=10,M=10;
 int K2=2*M*N;

 double hi=50.0;
 triangular_grid tg=new triangular_grid(L,W,M,N);
 int cond0=0;//x==0
 int cond1=1;//x==L
 int cond2=2;//y==0
 int cond3=3;//y==W
 double Ti0=0.0;
 double Ti1=0.0;
 double Ti2=0.0;
 double dx=W/K2;
if_x Ti3=x->100.0*x/L;
 tg.setTemperature(cond0,0.0,L,0.0,W,Ti0);
 tg.setTemperature(cond1,0.0,L,0.0,W,Ti1);
 tg.setTemperature(cond2,0.0,L,0.0,W,Ti2);
  tg.setTemperature(cond3,0.0,L,0.0,W,Ti3);
 //tg.setConvection(cond3,0.0,L,0.0,W,hi,Ti3);
 double k[]={50,50};
 tg.solve(1,k);
 Text.print(tg.T);
 tg.ContourPlot();
 tg.surfacePlot();
}
```

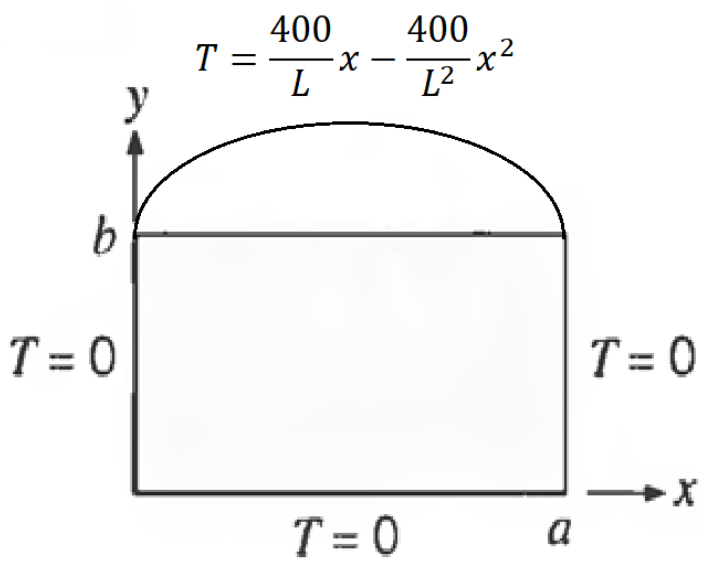
0	0	0	0	0	0	0	0	0	0	0

0	0.5	0.963	1.353	1.633	1.77	1.739	1.525	1.136	0.607	0
0	1.037	2	2.815	3.409	3.71	3.659	3.224	2.412	1.294	0
0	1.647	3.185	4.5	5.477	6.001	5.965	5.298	3.996	2.156	0
0	2.367	4.591	6.523	8	8.85	8.901	8.009	6.117	3.333	0
0	3.23	6.29	8.999	11.15	12.5	12.78	11.72	9.13	5.059	0
0	4.261	8.341	12.04	15.1	17.22	18	16.96	13.62	7.773	0
0	5.475	10.78	15.7	19.99	23.28	25.04	24.5	20.63	12.41	0
0	6.864	13.59	20	25.88	30.87	34.38	35.37	32	21.23	0
0	8.393	16.71	24.84	32.67	39.94	46.23	50.59	50.77	40.5	0
0	10	20	30	40	50	60	70	80	90	100



PROBLEM 5.6 NON-HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CARTESIAN COORDINATES, FINITE ELEMENT SOLUTION

A two dimensional rectangular plate is subjected to the boundary conditions shown in the figure. Derive an expression for the steady-state temperature distribution T(x,y) by using finite difference method



Take: L=1 m W=1 m

Boundary conditions :

At x=0 T=0

At x=L T=0

At y=0 T=0

at y=W $T = \frac{400}{L}x - \frac{400}{L^2}x^2$

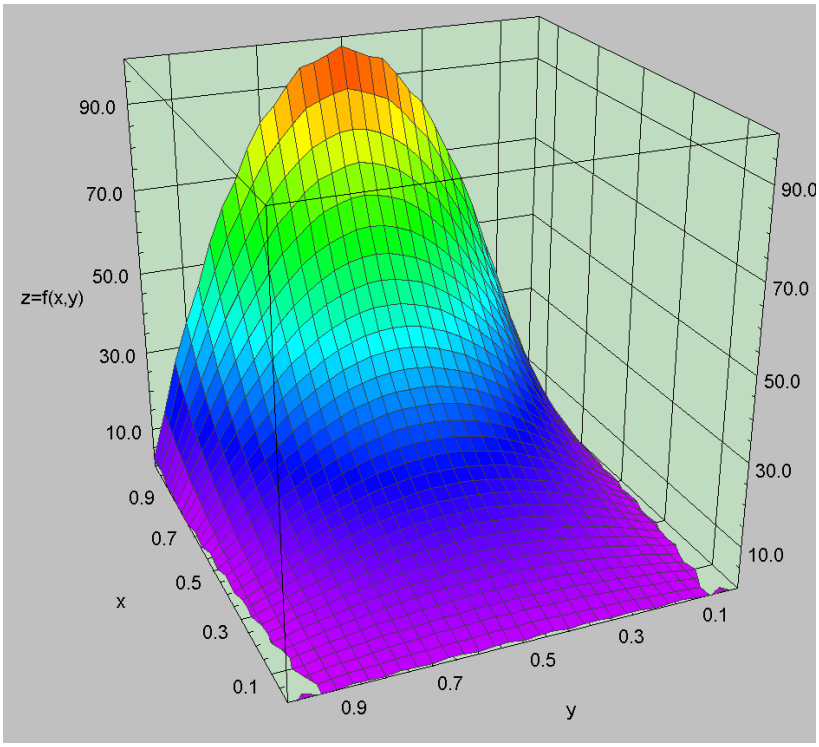
```
/variable function temperature profile
public void setTemperature(int cond,double L0,double L,double W0,double W, if_x Tx)
{//cond 0: x==L0
 //cond 1: x==L
 //cond 2: y==W0
 //cond 3: y==W
 for(int i=0;i<K2;i++)
 {  if(cond==0)
    {if(xi[i]==L0) {Ti[i]=Tx.func(yi[i]);bi[i]=true;}}
    else if(cond==1)
    {if(xi[i]==L) {Ti[i]=Tx.func(yi[i]);bi[i]=true;}}
    else if(cond==2)
    {if(yi[i]==W0) {Ti[i]=Tx.func(xi[i]);bi[i]=true;}}
    else if(cond==3)
    {if(yi[i]==W) {Ti[i]=Tx.func(xi[i]);bi[i]=true;}}
 }
}
```

```
public static void main(String arg[]) //throws RemoteException, VisADException
{double L=1.0;
 double W=1.0;
 int N=10,M=10;
 int K2=2*M*N;

 double hi=50.0;
 triangular_grid tg=new triangular_grid(L,W,M,N);
 int cond0=0;//x==0
 int cond1=1;//x==L
 int cond2=2;//y==0
 int cond3=3;//y==W
 double Ti0=0.0;
 double Ti1=0.0;
 double Ti2=0.0;
 double dx=W/K2;
if_x Ti3=x->100.0*x/L;
 tg.setTemperature(cond0,0.0,L,0.0,W,Ti0);
 tg.setTemperature(cond1,0.0,L,0.0,W,Ti1);
 tg.setTemperature(cond2,0.0,L,0.0,W,Ti2);
  tg.setTemperature(cond3,0.0,L,0.0,W,Ti3);
 //tg.setConvection(cond3,0.0,L,0.0,W,hi,Ti3);
 double k[]={50,50};
 tg.solve(1,k);
```

```
Text.print(tg.T);
tg.ContourPlot();
tg.surfacePlot();
}
```

0	0	0	0	0	0	0	0	0	0	0
0	0.898	1.708	2.349	2.761	2.902	2.761	2.349	1.708	0.898	0
0	1.885	3.583	4.929	5.791	6.088	5.791	4.929	3.583	1.885	0
0	3.058	5.812	7.991	9.386	9.866	9.386	7.991	5.812	3.058	0
0	4.535	8.615	11.84	13.9	14.61	13.9	11.84	8.615	4.535	0
0	6.469	12.27	16.85	19.76	20.76	19.76	16.85	12.27	6.469	0
0	9.065	17.17	23.51	27.54	28.92	27.54	23.51	17.17	9.065	0
0	12.62	23.82	32.5	37.96	39.83	37.96	32.5	23.82	12.62	0
0	17.61	32.98	44.7	51.99	54.46	51.99	44.7	32.98	17.61	0
0	24.85	45.79	61.33	70.84	74.04	70.84	61.33	45.79	24.85	0
0	36	64	84	96	100	96	84	64	36	0



Now **time dependent** general heat conduction equation can be investigated as a finite element equation

$$\frac{\partial}{\partial x}\left(k_x \frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k_y \frac{\partial T}{\partial y}\right) + g - \rho C_p \frac{\partial T}{\partial t} = 0$$

Boundary conditions

$T=T_b$ on surface S_1

$k_x \frac{\partial T}{\partial x}\vec{l} + k_y \frac{\partial T}{\partial y}\vec{m} + q = 0$ on surface S_2

$k_x \frac{\partial T}{\partial x}\vec{l} + k_y \frac{\partial T}{\partial y}\vec{m} + h(T - T_a) = 0$ on surface S_3

$T=T_0$ on x=0

Galarkin integration method will be applied to our differential equation . The method requires that the following expression will be satisfied

$\int_\Omega w_k L(T) d\Omega = 0$ where Ω is the volume of the element.

$\int_\Omega N_k \left[\frac{\partial}{\partial x}\left(k_x \frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k_y \frac{\partial T}{\partial y}\right) + g - \rho C_p \frac{\partial T}{\partial t}\right] d\Omega = 0$

Note that for finite element T=[N]{T} $\frac{\partial T}{\partial t} = [N]\frac{\partial \{T\}}{\partial t}$ and integral can be converted by partial integration rule

$$\int udv = uv - \int vdu$$

$$\int_\Omega \alpha_1 \left[\frac{\partial \alpha_2}{\partial x}\right] d\Omega = \int_S \alpha_1 \alpha_2 dS - \int_\Omega \left[\frac{\partial \alpha_1}{\partial x}\right] \alpha_2 d\Omega$$

$\int_\Omega N_k \left[\frac{\partial}{\partial x}\left(k_x \frac{\partial T}{\partial x}\right)\right] d\Omega = \int_S N_k \left(k_x \frac{\partial T}{\partial x}\right) dS - \int_\Omega \frac{\partial N_k}{\partial x}(k_x)\frac{\partial N_m}{\partial x} T d\Omega$

By applying these equation becomes

$$-\int_\Omega \left[k_x \frac{\partial N_k}{\partial x}\frac{\partial N_m}{\partial x} + k_y \frac{\partial N_k}{\partial y}\frac{\partial N_m}{\partial y}\right]\{T_m\} d\Omega - \int_\Omega \rho C_p N_k N_m \left\{\frac{\partial T_m}{\partial t}\right\} d\Omega + \int_\Omega g N_k d\Omega + \int_S N_k \left(k_x \frac{\partial T}{\partial x} + k_y \frac{\partial T}{\partial y}\right) dS = 0$$

From boundary conditions:

$$\int_S N_k \left(k_x \frac{\partial T}{\partial x} + k_y \frac{\partial T}{\partial y}\right) dS = -\int_S N_k (q + h(T - T_a) dS = -\int_S N_k\, q dS - \int_S N_k h\, N_m \{T_m\} dS + \int_S N_k h\, T_a dS$$

$$-\int_\Omega \left[k_x \frac{\partial N_k}{\partial x}\frac{\partial N_m}{\partial x} + k_y \frac{\partial N_k}{\partial y}\frac{\partial N_m}{\partial y}\right]\{T_m\} d\Omega - \int_\Omega \rho C_p N_k N_m \left\{\frac{\partial T_m}{\partial t}\right\} d\Omega + \int_\Omega g N_k d\Omega - \int_S N_k q dS - \int_S N_k h\, N_m \{T_m\} dS + \int_S N_k h\, T_a dS = 0$$

If this equation is written in matrix form :

$[C]\left\{\frac{dT}{dt}\right\} + [K]\{T\} = \{f\}$ where

$[C] = \int_\Omega \rho C_p N_k N_m d\Omega$

$[K] = -\int_\Omega \left[k_x \frac{\partial N_k}{\partial x}\frac{\partial N_m}{\partial x} + k_y \frac{\partial N_k}{\partial y}\frac{\partial N_m}{\partial y}\right] d\Omega + \int_S N_k h\, N_m dS$

$\{f\} = \int_\Omega g N_k d\Omega - \int_S N_k\, q dS + \int_S N_k h\, T_a dS$

$[C] = \int_\Omega \rho C_p [N]^T [N] d\Omega$

$[K] = -\int_\Omega [B]^T [D][B] d\Omega + \int_S h [N]^T [N]\, dS$ where

$[D] = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix}$

$\{f\} = \int_\Omega g [N]^T d\Omega - \int_S [N]^T q dS + \int_S [N]^T h\, T_a dS$

Line integrals

$\int_l N_i^a N_j^b dl = \frac{a!b!}{(a+b+1)!} l \quad \int_l N_i^2 N_j^0 dl = \frac{2!0!}{(2+o+1)!} l = \frac{2l}{6} \quad \int_l N_i N_j dl = \frac{1!1!}{(1+1+1)!} l = \frac{l}{6}$

Area integrals

$\int_A N_i^a N_j^b N_k^c dA = \frac{a!b!c!}{(a+b+c+2)!} 2A$

$\int_A N_i^2 N_j^0 N_k^0 dA = \frac{2!0!0!}{(2+0+0+2)!} 2A = \frac{2A}{12} = \frac{A}{6} \qquad \int_A N_i^1 N_j^1 N_k^0 dA = \frac{1!1!0!}{(1+1+0+2)!} 2A = \frac{A}{12}$

The basic equation is still a differential equation in time, therefore should be solved by using finite difference method.

$[C]\left\{\frac{\{T\}^{p+1} - \{T\}^p}{\Delta t}\right\} + \phi[K]\{T\}^p + (1-\phi)[K]\{T\}^{p+1} = \{f\}$

where $t = p\Delta t$ and ϕ is implicit explicit-scheme. If ϕ=1 It is fully explicit scheme, If ϕ=0, it is fully implicit scheme. ϕ=0.5 is called Crank-Nicolson method. Note that properties k,h,g, ρ , C_p and q can be changed in each step as a function of temperature, but at each single step calculations, they can be considered constant. When equation is arranged for p+1 time step (unknown) temperatures, it becomes

$([C] + (1-\phi)\Delta t[K])\{T\}^{p+1} = ([C] - \phi\Delta t[K])\{T\}^p + \Delta t\{f\}$

Matrix terms can be opened for **two dimensional triangular element** as:

$$[C] = \frac{\rho C_p A t}{12}\begin{bmatrix} 4 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 4 \end{bmatrix}$$

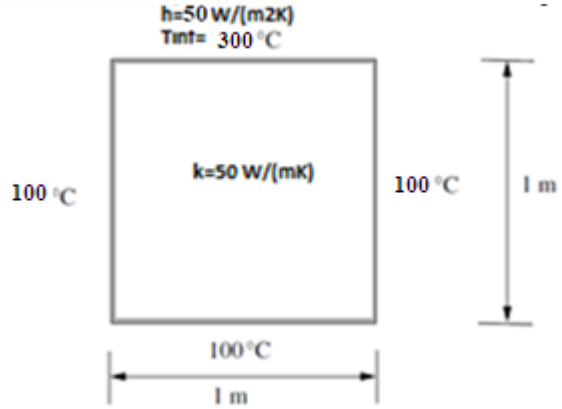
$$[K]=\frac{t}{4A}\left\{k_x\begin{bmatrix} b_i^2 & b_ib_j & b_ib_k \\ b_ib_j & b_j^2 & b_jb_k \\ b_jb_k & b_jb_k & b_k^2 \end{bmatrix}+k_y\begin{bmatrix} c_i^2 & c_ic_j & c_ic_k \\ c_ic_j & c_j^2 & c_jc_k \\ c_jc_k & c_jc_k & c_k^2 \end{bmatrix}\right\}+\frac{htl_{ij}}{6}\begin{bmatrix}2&1&0\\1&2&0\\0&0&0\end{bmatrix}+\frac{htl_{jk}}{6}\begin{bmatrix}0&1&0\\0&2&1\\0&1&0\end{bmatrix}+\frac{htl_{ki}}{6}\begin{bmatrix}2&1&0\\0&0&0\\0&1&2\end{bmatrix}$$

$$\{f\}=\frac{gAt}{3}\begin{Bmatrix}1\\1\\1\end{Bmatrix}-\frac{qtl_{ij}}{2}\begin{Bmatrix}1\\1\\0\end{Bmatrix}-\frac{qtl_{jk}}{2}\begin{Bmatrix}0\\1\\1\end{Bmatrix}-\frac{qtl_{ik}}{2}\begin{Bmatrix}1\\0\\1\end{Bmatrix}++\frac{hT_atl_{jk}}{2}\begin{Bmatrix}0\\1\\1\end{Bmatrix}++\frac{hT_atl_{ik}}{2}\begin{Bmatrix}1\\0\\1\end{Bmatrix}$$

Initial temperature profile will be defined and loaded into the program, and then Matrix will be solved and temperature profiles for each time step will be calculated.

PROBLEM 5.7 NON-HOMOGENEOUS, TIME DEPENDENT SOLUTION; CARTESIAN COORDINATES, FINITE ELEMENT SOLUTION

Plate initial temperature is T_i=50 ℃. (t=0). The plate exposed to 100 C temperature at 3 surfaces as shown in the figure and convectve heat flowwith temperature T_∞=300 ℃ and thermal conductivity h=50 W/(m^{2}K) (t>0). Thermal conductivty of plate is k=51 W/(mK), specific heat Cp=490 J/(kgK) and density ρ=7800 kg/m^3 Calculate temperature profile as a function of time.




```
public class triangular_element2
{
public double t; //Thickness of two dimensional element
public double a[],b[],c[];
public double x[],y[];//coordinates of data points
public double xc,yc; //center point
public double A; //Area
public double N[],B[][]; //Shape factor and derivative
public double Ti[];      //Temperature nodes
public int n_point[]; //point numbers
public int n_element; //element number
public double l[];
public boolean isconv[];
public boolean isheatflux[];
public boolean isheatgen;
public double h[];
public double Tinf[];
public double qflux[];
public double qgen;
public triangular_element2(int n_elementi,int n_pointi[],double xi[],double yi[])
{x=new double[3];
y=new double[3];
Ti=new double[3];
l=new double[3];//0:lij,1:ljk,2:lik
isconv=new boolean[3];
isheatflux=new boolean[3];
n_point=new int[3];
xc=0;yc=0;
isheatgen=false;
h=new double[3];
Tinf=new double[3];
qflux=new double[3];
n_element=n_elementi;
for(int i=0;i<3;i++)
{n_point[i]=n_pointi[i];
```

```
x[i]=xi[i];y[i]=yi[i];
xc+=x[i];yc+=y[i];
isconv[i]=false;
isheatflux[i]=false;
h[i]=0.0;
Tinf[i]=0.0;
qflux[i]=0.0;
qgen=0.0;
}
xc/=3.0;
yc/=3.0;
a=new double[3];
b=new double[3];
c=new double[3];
N=new double[3];
B=new double[2][3];
a[0]=x[1]*y[2]-x[2]*y[1];
a[1]=x[2]*y[0]-x[0]*y[2];
a[2]=x[0]*y[1]-x[1]*y[0];
b[0]=y[1]-y[2];
b[1]=y[2]-y[0];
b[2]=y[0]-y[1];
c[0]=x[2]-x[1];
c[1]=x[0]-x[2];
c[2]=x[1]-x[0];
A=0.5*((x[0]*y[1]-x[1]*y[0])+(x[2]*y[0]-x[0]*y[2])+(x[1]*y[2]-x[2]*y[1]));
for(int i=0;i<3;i++)
{ B[0][i]=1.0/(2.0*A)*b[i];
B[1][i]=1.0/(2.0*A)*c[i];
}
l[0]=Math.sqrt(b[2]*b[2]+c[2]*c[2]);
l[1]=Math.sqrt(b[0]*b[0]+c[0]*c[0]);
l[2]=Math.sqrt(b[1]*b[1]+c[1]*c[1]);
}

public double[][] K(double ti,double k[])
{t=ti;
double a0=t/(4.0*A)*k[0];
double a1=t/(4.0*A)*k[1];
double b0=t*h[0]*l[0]/6.0;
double b1=t*h[1]*l[1]/6.0;
double b2=t*h[2]*l[2]/6.0;
double K1[][]={{(a0*b[0]*b[0]+a1*c[0]*c[0]+2*b0+2.0*b2),
(a0*b[0]*b[1]+a1*c[0]*c[1]+b0),
(a0*b[0]*b[2]+a1*c[0]*c[2]+b2)},
{(a0*b[1]*b[0]+a1*c[1]*c[0]+b0),
(a0*b[1]*b[1]+a1*c[1]*c[1]+2*b0+2*b1),
(a0*b[1]*b[2]+a1*c[1]*c[2]+b1)},
{(a0*b[2]*b[0]+a1*c[2]*c[0]+b2),
(a0*b[2]*b[1]+a1*c[2]*c[1]+b1),
(a0*b[2]*b[2]+a1*c[2]*c[2]+2*b1+2*b2)}
};
return K1;
}
public double[] f(double ti,double k[])
{
t=ti;
double a0=qgen*A*t/3.0;
double b0=-qflux[0]*t*l[0]/2.0;
double b1=-qflux[1]*t*l[1]/2.0;
double b2=-qflux[2]*t*l[2]/2.0;
double c0= h[0]* Tinf[0]*t*l[0]/2.0;
double c1= h[1]* Tinf[1]*t*l[1]/2.0;
double c2= h[2]* Tinf[2]*t*l[2]/2.0;
double f1[]={(a0+b0+b2+c0+c2),(a0+b0+b1+c0+c1),(a0+b1+b2+c1+c2)};
return f1;
}
public double[][] C(double ti,double Cp,double ro)
{   t=ti;
            double x=ro*Cp*A*t/12.0;
            double cc[][]={{4.0*x,x,x},{x,4.0*x,x},{x,x,4.0*x}};
   return cc;
}
public int setI(int cond,double L0,double L,double W0,double W)
{
int i=100;
if(cond==0) //x=0
```

```
{if(x[0]==L0 && x[1]==L0) i=0;
else if(x[1]==L0 && x[2]==L0) i=1;
else if(x[0]==L0 && x[2]==L0) i=2;
else i=-1;
}
else if(cond==1) //x=L
{if(x[0]==L && x[1]==L) i=0;
else if(x[1]==L && x[2]==L) i=1;
else if(x[0]==L && x[2]==L) i=2;
else i=-1;
}
else if(cond==2) //y=0
{if(y[0]==W0 && y[1]==W0) i=0;
else if(y[1]==W0 && y[2]==W0) i=1;
else if(y[0]==W0 && y[2]==W0) i=2;
else i=-1;
}
else if(cond==3) //y=W
{if(y[0]==W && y[1]==W) i=0;
else if(y[1]==W && y[2]==W) i=1;
else if(y[0]==W && y[2]==W) i=2;
else i=-1;
}
return i;
}

public void setConvection(int i, double hi, double tinfi)
{if(i>=0 && i<=2)
{
h[i]=hi;Tinf[i]=tinfi;isconv[i]=true;
}
}

public void setHeatFlux(int i, double qi)
{if(i>=0 && i<=2 ){qflux[i]=qi;isheatflux[i]=true;}     }

public void setHeatGeneration(double qgeni)
{qgen=qgeni;isheatgen=true;}
public void setN(double x,double y)
{for(int i=0;i<3;i++)
{N[i]=1.0/(2.0*A)*(a[i]+b[i]*x+c[i]*y);}
}
public double[]  q(double k)
{ double qi[]=new double[2];
for(int i=0;i<3;i++)
{qi[0]+=-k*B[0][i]*Ti[i];
qi[1]+=-k*B[1][i]*Ti[i];
}
return qi;
}
public double[]  q(double k,double Ti[])
{  //heat transfer
setT(Ti);
double qi[]=new double[2];
for(int i=0;i<3;i++)
{qi[0]+=-k*B[0][i]*Ti[i];
qi[1]+=-k*B[1][i]*Ti[i];
}
return qi;
}

public void setN(double z[])
{setN(z[0],z[1]);}
public void setT(double TT[])
{for(int i=0;i<3;i++)
{Ti[i]=TT[i];}
}
public double T(double x,double y)
{ //interpolation
double TT=0;
setN(x,y);
for(int i=0;i<3;i++)
{TT+=N[i]*Ti[i];}
return TT;
}

public double T(double T[],double x,double y)
```

```
{ //set grid point temperatures and get interpolated value
setT(T);
setN(x,y);
double TT=0;
for(int i=0;i<3;i++)
{TT+=N[i]*Ti[i];}
return TT;
}
public double T(double T[],double z[])
{ return T(T,z[0],z[1]);}
public static void main(String arg[])
{
                /*double L=1.0;
double W=1.0;
int N=16,M=16;
double hi=20.0;
triangular_grid tg=new triangular_grid(L,W,M,N);
int cond0=0;//x==0
int cond1=1;//x==L
int cond2=2;//y==0
int cond3=3;//y==W
double Ti0=100.0;
double Ti1=100.0;
double Ti2=100.0;
double Ti3=500.0;
double qi=-5000.0;;
tg.setTemperature(cond0,0.0,L,0.0,W,Ti0);
tg.setTemperature(cond1,0.0,L,0.0,W,Ti1);
tg.setTemperature(cond2,0.0,L,0.0,W,Ti2);
tg.setConvection(cond3,0.0,L,0.0,W,hi,Ti3);
double k[]={50,50};
tg.solve(1,k);
Text.print(tg.T);
tg.ContourPlot();
//tg.Plot3D(0.0,L,0.0,W);
//tg.GridPlot(0.0,L,0.0,W);
tg.surfacePlot();
*/
}}
```

```
 import javax.swing.*;
import static java.lang.Math.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class triangular_grid2
{
public triangular_element2 te[];
public setElement se[];
public int N,M,K,K1,K2,Nm1,Mm1;
public double x[][];
public double y[][];
public double T[][];
public double TT[][][];
public double Ti[];
public double xi[];
public double yi[];
public boolean bi[];
public double dx,dy,dt,time;
int Np;
//In this version it is assumed that Cp,k, and ro is constant
public int Bx0[],BxM[],By0[],ByN[];
public double L,W;
public double min(double xi[][])
{ double min=1e200;
  for(int i=0;i<xi.length;i++)
   {for(int j=0;j<xi[0].length;j++)
     {if(xi[i][j]<min) min=xi[i][j];}
 }
 return min;
}
public double max(double xi[][])
{double max=-1e200;
 for(int i=0;i<xi.length;i++)
 {for(int j=0;j<xi[0].length;j++)
   {if(xi[i][j]<max) max=xi[i][j];}
```

```
 }
 return max;
}
public triangular_grid2(double xi[][],double yi[][],double TTi[][],double dti,double timei)
{x=xi;
 y=yi;
 T=TTi;
 N=x.length-1;
 M=x[0].length-1;
 T=new double[N+1][M+1];
 L=max(xi);
 W=max(yi);
 setElements();
 time=timei;
 dt=dti;
 Np=(int)((timei+0.00001)/dti);
 TT=new double[Np+1][N+1][N+1];
}

public void generateTriangularGrid(double Li, double Wi,int Ni, int Mi)
{int i,j;
 N=Ni;M=Mi;
 L=Li;
 W=Wi;
 x=new double[N+1][M+1];
 y=new double[N+1][M+1];
 T=new double[N+1][M+1];
 setElements();
}

public void setElements()
{ int i,j;
 dx=L/M;
 dy=W/N;
 K=M*N;
 K1=K*2;
 K2=K1;
 //grid points
 xi=new double[K2];
 yi=new double[K2];
 Ti=new double[K2];
 bi=new boolean[K2];
 se=new setElement[K2];
 for(i=0;i<K2;i++) {se[i]=new setElement();}
 te=new triangular_element2[K1];
 int k1=-1,k=0,n=0,m=0;
for(n=0;n<=N;n++)
{ for(m=0;m<=M;m++)
  {k=(N+1)*n+m;
   x[n][m]=m*dx;y[n][m]=n*dy;
   xi[k]=x[n][m];
   yi[k]=y[n][m];
   bi[k]=false;
   Ti[k]=0.0;
  }
}

for(n=0;n<N;n++)
{for(m=0;m<M;m++)
  { k=(N+1)*n+m;
              int kp1=(N+1)*(n+1)+m;
              int kp1p1= kp1+1;
              double xi1[]={x[n][m],x[n][m+1],x[n+1][m]};
              double yi1[]={y[n][m],y[n][m+1],y[n+1][m]};

              k1++;
              int n_point1[]={k,(k+1),kp1};
              se[k].add(k1,0);
              se[k+1].add(k1,1);
              se[kp1].add(k1,2);
              te[k1]=new triangular_element2(k1,n_point1,xi1,yi1);
              double xi2[]={x[n][m+1],x[n+1][m+1],x[n+1][m]};
              double yi2[]={y[n][m+1],y[n+1][m+1],y[n+1][m]};
              k1++;
              int n_point2[]={k+1,kp1p1,kp1};
   se[k+1].add(k1,0);
              se[kp1p1].add(k1,1);
```

```
                se[kp1].add(k1,2);
                te[k1]=new triangular_element2(k1,n_point2,xi2,yi2);
  }
}
}
public triangular_grid2(double Li, double Wi,int Ni, int Mi,double TTi[][],double dti,double timei)
{generateTriangularGrid(Li,Wi,Ni,Mi);
 T=TTi;
 time=timei;
 dt=dti;
  Np=(int)((timei+0.00001)/dti);
 TT=new double[Np+1][N+1][N+1];
}
public int find_element(double z[])
{ return find_element(z[0],z[1]);}
public int find_element(double x,double y)
{double R1=Math.sqrt(x*x+y*y);
 int kmin=0;
 double min=1.0e50;
 for(int k=0;k<K1;k++)
 {   te[k].setN(x,y);
     double r0=Math.sqrt((te[k].x[0]-x)*(te[k].x[0]-x)+(te[k].y[0]-y)*(te[k].y[0]-y));
     double r1=Math.sqrt((te[k].x[1]-x)*(te[k].x[1]-x)+(te[k].y[1]-y)*(te[k].y[1]-y));
     double r2=Math.sqrt((te[k].x[2]-x)*(te[k].x[2]-x)+(te[k].y[2]-y)*(te[k].y[2]-y));
     double dr=Math.sqrt(r0*r0+r1*r1+r2*r2);
                 boolean b1=(x>=te[k].x[0]) && (x<=te[k].x[2]);
                 boolean b2=(y>=te[k].y[0]) && (y<=te[k].y[1]);
                 boolean b= b1 && b2;
     if(dr<min) {min=dr;kmin=k;}
 }
 return kmin;
}
public void setConvection(int cond,double L0,double L,double W0,double W, double hi, double tinfi)
{//cond 0: x==L0
 //cond 1: x==L
 //cond 2: y==W0
 //cond 3: y==W
 int j=0;
 for(int i=0;i<K1;i++)
 {  j=te[i].setI(cond,L0,L,W0,W);
                te[i].setConvection(j,hi,tinfi);
 }
}
public void setHeatFlux(int cond,double L0,double L,double W0,double W, double qi)
{//cond 0: x==L0
 //cond 1: x==L
 //cond 2: y==W0
 //cond 3: y==W
 int j=-1;
 for(int i=0;i<K1;i++)
 {  j=te[i].setI(cond,L0,L,W0,W);
                te[i].setHeatFlux(j,qi);
 }
}
public void setTemperature(int cond,double L0,double L,double W0,double W, double TTi)
{//cond 0: x==L0
 //cond 1: x==L
 //cond 2: y==W0
 //cond 3: y==W
 for(int i=0;i<K2;i++)
 {  if(cond==0)
    {if(xi[i]==L0) {Ti[i]=TTi;bi[i]=true;}}
    else if(cond==1)
    {if(xi[i]==L) {Ti[i]=TTi;bi[i]=true;}}
    else if(cond==2)
    {if(yi[i]==W0) {Ti[i]=TTi;bi[i]=true;}}
    else if(cond==3)
    {if(yi[i]==W) {Ti[i]=TTi;bi[i]=true;}}
 }
}
public void setHeatGeneration(int node[],double qgeni)
{  int n=node.length;
   for(int i=0;i<n;i++)
   {te[node[i]].setHeatGeneration(qgeni);}
}

public void solve(double ti,double k[],double Cp,double ro,double fi)
```

```
{
double TT1[]=new double[K2];
double C[][]=new double[K2][K2];
double K[][]=new double[K2][K2];
double AA[][]=new double[K2][K2];
double AA1[][]=new double[K2][K2];
double BB[]=new double[K2];
double CK[][]=new double[K2][K2];
double KK[][]=new double[K2][K2];
double ff[]=new double[K2];
double CN[][]=new double[3][3];
double KN[][]=new double[3][3];
double f[]=new double[K2];
double fN[]=new double[3];
int k1=0;
for(int n=0;n<=N;n++)
{ for(int m=0;m<=M;m++)
  {k1=(N+1)*n+m;
   TT1[k1]=T[n][m];
   TT[0][n][m]=T[n][m];
  }
}
for(int ip=0;ip<Np;ip++)
{  //time steps
for(int i=0;i<K1;i++)
{KN=te[i].K(ti,k);
 CN=te[i].C(ti,Cp,ro);
 fN=te[i].f(ti,k);
 int n1=te[i].n_element;
 int m[]=te[i].n_point;
 for(int j=0;j<3;j++)
   { for(k1=0;k1<3;k1++)
      {K[m[j]][m[k1]]+=KN[j][k1];KK[m[j]][m[k1]]=K[m[j]][m[k1]];
       C[m[j]][m[k1]]+=CN[j][k1];CK[m[j]][m[k1]]=C[m[j]][m[k1]];
      }
       f[m[j]]+=fN[j];
       ff[m[j]]=f[m[j]];
   }
}
for(int i=0;i<K2;i++)
{for(int j=0;j<K2;j++)
 {AA[i][j]=C[i][j]+(1.0-fi)*dt*K[i][j];AA1[i][j]=C[i][j]-fi*dt*K[i][j];}
  BB[i]=f[i]*dt;
}
BB=add(multiply(AA1,TT1),BB);
//redesign AA matrix and BB vector if any known
//temperature boundary is given
//int j=0;
for(int j=0;j<K2;j++)
{
            for(int i=0;i<K2;i++)
    { if(bi[i])
      {
               if(i==j)
               {BB[i]=Ti[i];AA[i][j]=1.0;}
               else
               {
               AA[i][j]=0.0;}
               //System.out.println("i="+i+"j="+j+"bi="+bi[i]+"Ti="+Ti[i]+"fi="+f[i]);
      }
      }//end of for
} //end of for
//Text.print(K,"K");
//Text.print(C,"C");
//Text.print(f,"f");
TT1=AXB(AA,BB);
for(int n=0;n<=N;n++)
{ for(int m=0;m<=M;m++)
  {k1=(N+1)*n+m;
  T[n][m]=TT1[k1];
  TT[ip+1][n][m]=TT1[k1];
  }
}

} //end of time steps
}
public static double[][] multiply(double[][] left,double[][] right)
```

```
 {
//multiplication of two matrices
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right[0].length;
int n2=right.length;
//System.out.println(n1+"x"+m1+"  "+n2+"x"+m2);
double b[][];
b=new double[n1][m2];
 if(m1 != n2)
 {
 System.out.println("inner matrix dimensions must agree");
 for(ii=0;ii<n1;ii++)
   {
   for(jj=0;jj<m2;jj++)
     b[ii][jj]=0;
   }
   return b;
 }
 for(i=0;i<n1;i++)
   {
   for(j=0;j<m2;j++)
     {
     for(k=0;k<m1;k++){
        //System.out.println(i+"  "+j+"   "+k);
                b[i][j]+=left[i][k]*right[k][j];
                //System.out.println(b[i][j]);
    }
     }
   }
return b;
//end of multiply of two matrices
}
public static double[] multiply(double[][] left,double[] right)
{
//multiplication of one matrix with one vector
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right.length;
double[] b;
b=new double[m2];
 if(n1 != m2)
 {
 System.out.println("inner matrix dimensions must agree");
 for(ii=0;ii<n1;ii++)
   {
     b[ii]=0;
   }
   return b;
 }
 for(i=0;i<m1;i++)
   {
     b[i]=0;
     for(k=0;k<n1;k++)
            b[i]+=left[i][k]*right[k];
   }
return b;
//end of multiply of a matrix and a vector
}
public static double[] add(double[] left,double[] right)
{
//addition of two vectors
int n1=left.length;
int n2=right.length;
int nMax;
int i;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[];
b=new double[nMax];
for(i=0;i<n1;i++)
  {
    b[i]=b[i]+left[i];
  }
for(i=0;i<n2;i++)
```

```
  {
   b[i]=b[i]+right[i];
  }
return b;
//end of vector addition method
}
//LU decomposition method
public static double[][] LU(double c[][],int indx[],int d[])
{
//returns LU decomposition of matrix c and index indx
double a[][];
int n=c.length;
a=new double[n][n];
double vv[];
vv=new double[n];
double sum,dum,big,temp;
int i,j,k;
int imax;
int nmax=100;
double tiny=1.0e-40;
imax=0;
for(i=1;i<=n;i++)
{
 for(j=1;j<=n;j++)
    a[i-1][j-1]=c[i-1][j-1];
}
d[0]=1;
for(i=1;i<=n;i++)
 {
 big=0.0;
 for(j=1;j<=n;j++)
   {
   if(Math.abs(a[i-1][j-1])>big) big=Math.abs(a[i-1][j-1]);
   }
 if(big==0) {System.out.println("singular matrix");return a;}
 vv[i-1]=1.0/big;
 }
for(j=1;j<=n;j++)
{
 for(i=1;i<j;i++)
   {
    sum=a[i-1][j-1];
    for(k=1;k<i;k++)
              {
                sum-=a[i-1][k-1]*a[k-1][j-1];
              }
   a[i-1][j-1]=sum;
   }
 big=0;
 for(i=j;i<=n;i++)
   {
   sum=a[i-1][j-1];
   for(k=1;k<j;k++)
     {
     sum-=a[i-1][k-1]*a[k-1][j-1];
     }
   a[i-1][j-1]=sum;
   dum=vv[i-1]*Math.abs(sum);
   if(dum>=big)
     {
     imax=i;
     big=dum;
     }
   } //end of i=0
if(j != imax)
   {
   for(k=1;k<=n;k++)
     {
     dum=a[imax-1][k-1];
     a[imax-1][k-1]=a[j-1][k-1];
     a[j-1][k-1]=dum;
     }
   d[0]=-d[0];
   vv[imax-1]=vv[j-1];
   } //end of if
indx[j-1]=imax;
if(a[j-1][j-1]==0) a[j-1][j-1]=tiny;
```

```
if(j!=n)
  {
  dum=1.0/a[j-1][j-1];
  for(i=j+1;i<=n;i++)
    a[i-1][j-1]*=dum;
  }//endif
} //end for j=
return a;
}
public static double[] LUaxb(double a[][],double x[],int indx[])
{
//solves AX=B system of linear equation of LU decomposed matrix a
//(calculated by method LU)
int ii=0;
int i,j,ll=0;
double sum=0;
int n=a.length;
double b[];
b=new double[n];
for(i=1;i<=n;i++)
{
   b[i-1]=x[i-1];
}
for(i=1;i<=n;i++)
  {
  ll=indx[i-1];
  sum=b[ll-1];
  b[ll-1]=b[i-1];
  if(ii!=0)
     {
     for(j=ii;j<=(i-1);j++)
       {
       sum-=a[i-1][j-1]*b[j-1];
       }
     }
  else if(sum!=0) ii=i;
  b[i-1]=sum;
  }
for(i=n;i>=1;i--)
  {
  sum=b[i-1];
  if(i<n)
    {
    for(j=(i+1);j<=n;j++)
      {
      sum-=a[i-1][j-1]*b[j-1];
      }
    }
  b[i-1]=sum/a[i-1][i-1];
  }
return b;
}
public void ContourPlot(double Tx[][], String ss)
{ ContourPlot pp=new ContourPlot( x,y,Tx,16,true,ss,"x","y");
  pp.plot();
}

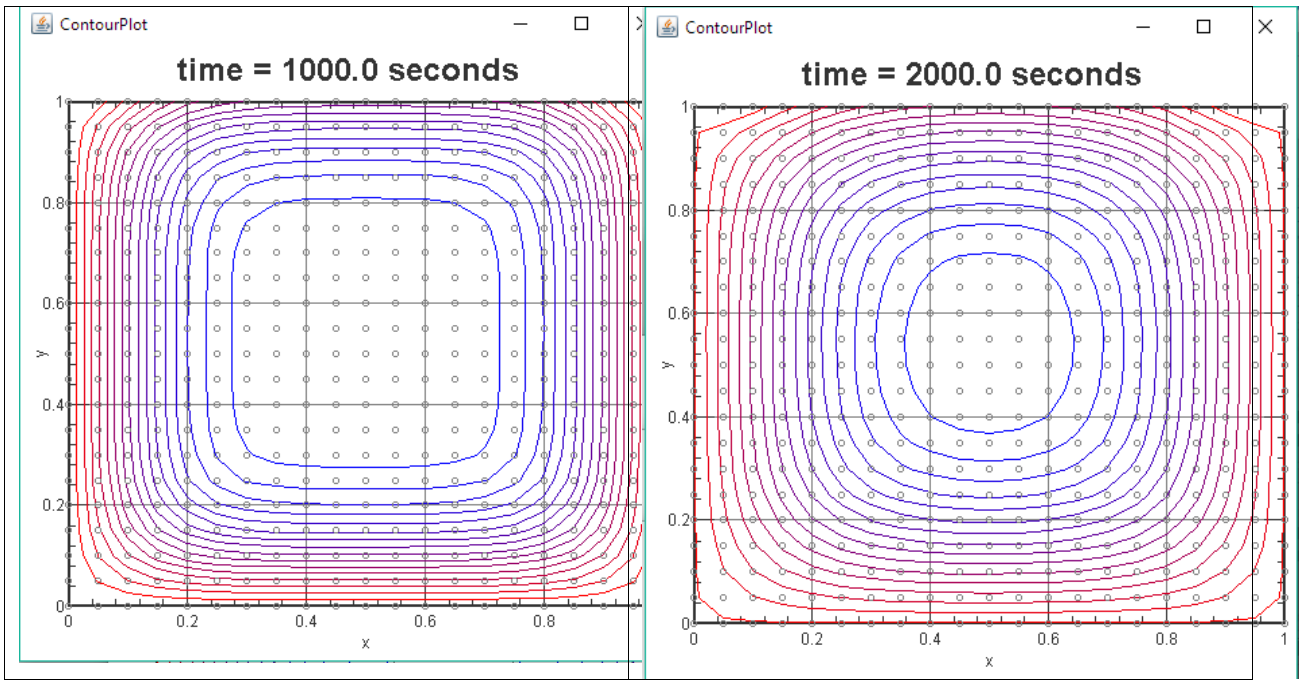
public void surfacePlot(double Tx[][])
{    double b[][]=surfacePlot.dataformat(0,L,0,W,Tx);
     surfacePlot p1;
     p1=new surfacePlot(b);
     p1.plot();
}
public static double[] AXB(double a[][],double b[])
{
//Solution of system of linear equations by LU method
// note that the same calculation can be done by divide method.
//Text.print(a,"K");
//Text.print(b,"f");
int n=a.length;
double c[]=new double[n];
int d[]={1};
int indx[]=new int[n];
double e[][]=new double[n][n];
e=LU(a,indx,d);
c=LUaxb(e,b,indx);
return c;
```

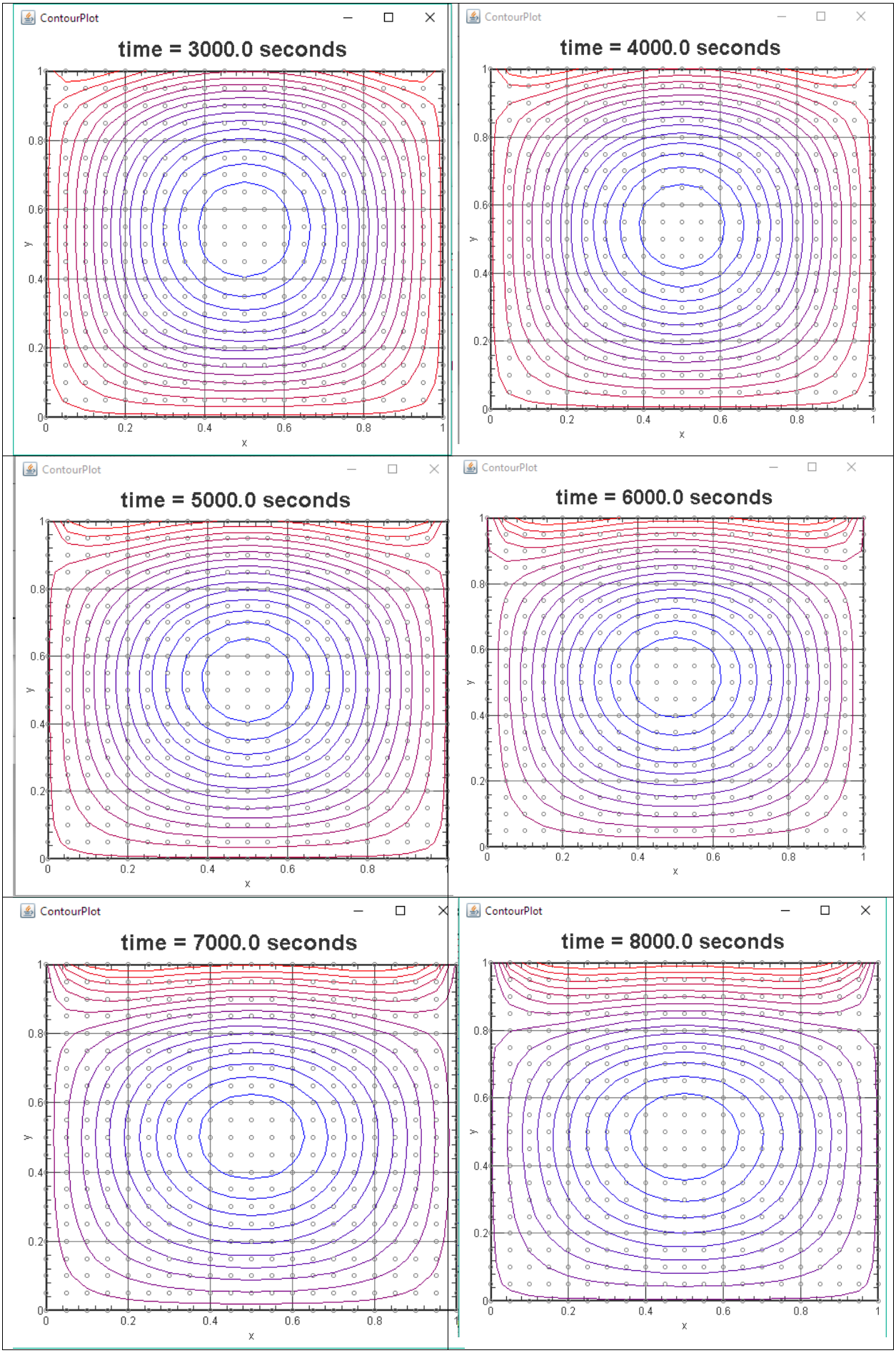
```
 } //end of AXB

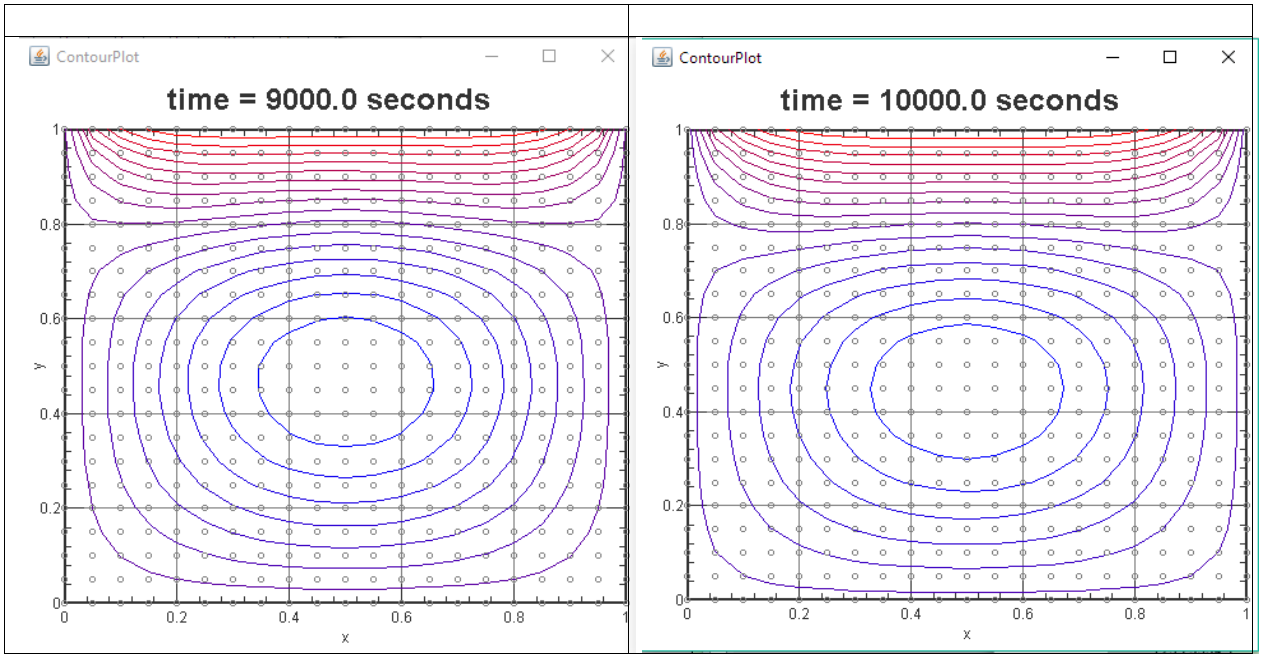
public static void main(String arg[]) //throws RemoteException, VisADException
{double L=1.0;
 double W=1.0;
 int N=20,M=20;
 double hi=50.0;
 double Cp=490.0;
 double ro=7800.0;

 int cond0=0;//x==0
 int cond1=1;//x==L
 int cond2=2;//y==0
 int cond3=3;//y==W
 double Ti0=100.0;
 double Ti1=100.0;
 double Ti2=100.0;
 double Ti3=300.0;
 double TT[][]=new double[N+1][M+1];
 for(int i=0;i<=N;i++)
 {for(int j=0;j<=M;j++)
   TT[i][j]=50.0;
 }
 triangular_grid2 tg=new triangular_grid2(L,W,M,N,TT,10.0,10000.0);
 tg.setTemperature(cond0,0.0,L,0.0,W,Ti0);
 tg.setTemperature(cond1,0.0,L,0.0,W,Ti1);
 tg.setTemperature(cond2,0.0,L,0.0,W,Ti2);
 //tg.setTemperature(cond3,0.0,L,0.0,W,Ti3);
 tg.setConvection(cond3,0.0,L,0.0,W,hi,Ti3);
 double k[]={51,51};
 double fi=0.5;
 tg.solve(1,k,Cp,ro,fi);

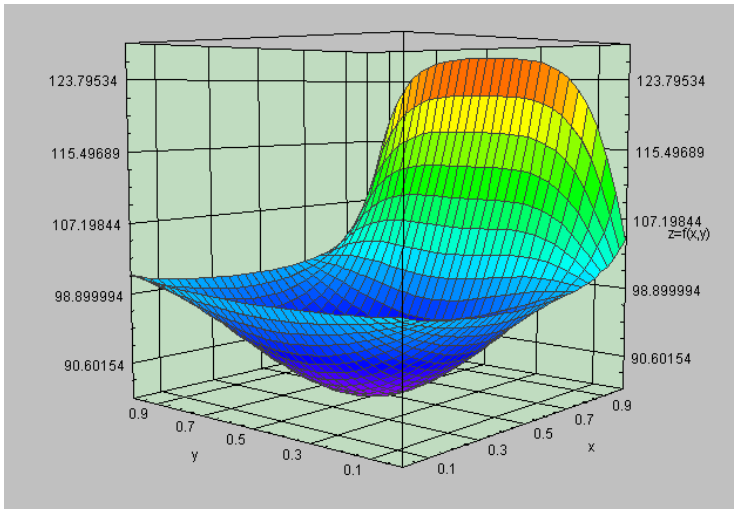
 //Take an output at every 2000 seconds
 for(int ip=0;ip<=1000;ip++)
 {if((ip%100)==0)
   { String s="time = "+ip*tg.dt+" seconds";
                System.out.println(s);
   Text.print(tg.TT[ip],s);
   tg.ContourPlot(tg.TT[ip],s);
    }
 }
 //temperature profile t=10 s
 tg.surfacePlot(tg.TT[1000]);
}}
```




ContourPlot
time = 3000.0 seconds
ContourPlot
time = 4000.0 seconds
ContourPlot
time = 5000.0 seconds
ContourPlot
time = 6000.0 seconds
ContourPlot
time = 7000.0 seconds
ContourPlot
time = 8000.0 seconds
x
y
0
0.2
0.4
0.6
0.8
1



time=10000 seconds



time = 2000.0 seconds

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
100.0	97.84433...	95.81717...	94.04260...	92.59541...	91.49635...	90.72019...	90.21272...	89.91028...	89.75646...	89.71330...	89.76716...	89.93015...	90.23826...	90.74565...	91.51431...	92.59866...	94.02739...	95.78702...	97.81388...	100.0
100.0	95.81716...	91.89572...	88.46848...	85.67641...	83.55736...	82.06123...	81.08273...	80.49888...	80.20099...	80.11623...	80.21864...	80.53215...	81.12691...	82.10824...	83.59643...	85.69670...	88.46367...	91.86938...	95.78702...	100.0
100.0	94.04255...	88.46840...	83.60347...	79.64424...	76.64158...	74.52248...	73.13650...	72.30885...	71.88553...	71.76356...	71.90661...	72.34928...	73.19193...	74.58506...	76.70021...	79.68702...	83.62218...	88.46364...	94.02737...	100.0
100.0	92.59521...	85.67607...	79.64390...	74.73921...	71.02223...	68.40032...	66.68573...	65.66135...	65.13633...	64.98332...	65.15776...	65.70316...	66.74485...	68.47052...	71.09395...	74.80121...	79.68681...	85.69651...	92.59857...	100.0
100.0	91.49569...	83.55619...	76.64013...	71.02097...	66.76528...	63.76486...	61.80321...	60.63089...	60.02901...	59.85178...	60.04843...	60.66939...	61.85919...	63.83419...	66.84073...	71.09299...	76.69921...	83.59570...	91.51395...	100.0
100.0	90.71828...	82.05774...	74.51787...	68.39529...	63.76077...	60.49468...	58.35989...	57.08386...	56.42781...	56.23282...	56.44375...	57.11594...	58.40770...	60.55599...	63.83071...	68.46646...	74.58157...	82.10580...	90.74443...	100.0
100.0	90.20760...	81.07319...	73.12354...	66.67065...	61.78794...	58.34812...	56.10026...	54.75649...	54.06480...	53.85758...	54.07671...	54.78078...	56.13725...	58.39695...	61.84568...	66.73202...	73.18143...	81.11960...	90.23458...	100.0
100.0	89.89749...	80.47478...	72.27561...	65.62161...	60.58790...	57.04248...	54.72594...	53.34098...	52.62744...	52.41227...	52.63547...	53.35759...	54.75170...	57.07729...	60.63020...	65.66799...	72.32086...	80.51236...	89.92011...	100.0
100.0	89.72674...	80.14451...	71.80679...	65.04080...	59.92284...	56.31841...	53.96342...	52.55530...	51.82929...	51.60925...	51.83404...	52.56521...	53.97901...	56.33984...	59.94940...	65.07051...	71.83637...	80.16958...	89.74212...	100.0
100.0	89.64897...	79.99307...	71.59048...	64.77134...	59.61285...	55.97965...	53.60566...	52.18590...	51.45342...	51.23055...	51.45551...	52.19027...	53.61258...	55.98922...	59.62477...	64.78473...	71.60386...	80.00446...	89.65599...	100.0
100.0	89.63735...	79.96864...	71.55292...	64.72139...	59.55216...	55.91041...	53.53015...	52.10615...	51.37099...	51.14656...	51.37083...	52.10578...	53.52948...	55.90930...	59.55052...	64.71920...	71.55037...	79.96618...	89.63570...	100.0
100.0	89.68621...	80.05980...	71.67693...	64.86828...	59.71332...	56.07944...	53.70290...	52.28022...	51.54509...	51.31993...	51.54275...	52.27522...	53.69478...	56.06779...	59.69820...	64.85050...	71.65831...	80.04330...	89.67572...	100.0
100.0	89.81206...	80.29714...	72.00341...	65.25951...	60.14750...	56.53969...	54.17755...	52.76190...	52.02945...	51.80421...	52.02462...	52.75170...	54.16116...	56.51655...	60.11794...	65.22528...	71.96810...	80.26622...	89.79259...	100.0
100.0	90.05593...	80.75673...	72.63456...	66.01480...	60.98524...	57.42791...	55.09419...	53.69297...	52.96652...	52.74195...	52.95862...	53.67646...	55.06808...	57.39172...	60.93986...	65.96323...	72.58228...	80.71163...	90.02785...	100.0
100.0	90.48828...	81.56675...	73.73919...	67.32876...	62.43644...	58.96265...	56.67605...	55.29906...	54.58302...	54.36014...	54.57135...	55.27497...	56.63868...	58.91202...	62.37457...	67.26027...	73.67149...	81.50962...	90.45331...	100.0
100.0	91.21867...	82.91834...	75.55863...	69.47032...	64.78446...	61.43472...	59.21788...	57.87692...	57.17667...	56.95677...	57.16052...	57.84408...	59.16810...	61.36926...	64.70717...	69.38796...	75.48035...	82.85466...	91.18080...	100.0
100.0	92.41756...	85.08459...	78.41141...	72.77374...	68.36772...	65.18339...	63.05912...	61.76622...	61.08733...	60.87177...	61.06608...	61.72375...	62.99647...	65.10403...	68.27839...	72.68396...	78.33169...	85.02423...	92.38382...	100.0
100.0	94.37446...	88.45859...	82.69546...	77.61578...	73.54365...	70.55404...	68.53899...	67.30339...	66.65033...	66.44023...	66.62343...	67.25061...	68.46353...	70.46281...	73.44759...	77.52808...	82.62769...	88.41637...	94.35578...	100.0
100.0	97.67173...	93.62836...	88.87995...	84.37123...	80.63245...	77.83921...	75.93707...	74.76238...	74.13740...	73.93329...	74.10426...	74.69854...	75.84883...	77.73827...	80.53559...	84.29672...	88.84081...	93.62456...	97.68532...	100.0
100.0	103.7495...	101.4781...	97.44751...	93.33174...	89.84264...	87.21202...	85.41157...	84.29521...	83.69815...	83.49989...	83.65800...	84.21916...	85.30983...	87.10239...	89.74947...	93.28042...	97.45460...	101.5393...	103.8272...	100.0

time = 4000.0 seconds

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
100.0	98.90729...	97.84887...	96.86162...	95.97540...	95.21183...	94.58407...	94.09790...	93.75378...	93.54939...	93.48201...	93.55025...	93.75477...	94.09776...	94.58134...	95.20529...	95.96457...	96.84722...	97.83331...	98.89529...	100.0
100.0	97.84816...	95.76674...	93.82647...	92.08540...	90.58563...	89.35279...	88.39810...	87.72241...	87.32115...	87.18902...	87.32344...	87.72579...	88.40045...	89.35166...	90.57894...	92.07225...	93.80789...	95.74633...	97.83269...	100.0
100.0	96.85864...	93.82267...	90.99406...	88.45672...	86.27157...	84.47562...	83.08502...	82.10086...	81.51649...	81.32427...	81.52054...	82.10749...	83.09162...	84.47913...	86.26932...	88.44737...	90.97843...	93.80440...	96.84453...	100.0
100.0	95.96748...	92.07246...	88.44496...	85.19198...	82.39113...	80.08948...	78.30746...	77.04636...	76.29759...	76.05145...	76.30339...	77.05638...	78.31891...	80.09900...	82.39569...	85.18980...	88.43629...	92.06019...	95.95728...	100.0
100.0	95.19474...	90.55514...	86.23554...	82.36282...	79.02892...	76.28956...	74.16875...	72.66791...	71.77677...	71.48387...	71.78394...	72.68072...	74.18444...	76.30474...	79.04043...	82.36849...	86.23496...	90.55009...	95.18926...	100.0
100.0	94.55149...	89.29221...	84.39669...	80.00833...	76.23093...	73.12732...	70.72452...	69.02400...	68.01417...	67.68212...	68.02205...	69.03836...	70.74281...	73.14637...	76.24765...	80.02035...	84.40303...	89.29356...	94.55027...	100.0
100.0	94.04094...	88.28973...	82.93688...	78.13885...	74.00889...	70.61550...	67.98814...	66.12842...	65.02374...	64.66015...	65.03148...	66.14268...	68.00673...	70.63566...	74.02780...	78.15420...	82.94735...	88.29523...	94.04263...	100.0
100.0	93.66062...	87.54267...	81.84842...	76.74404...	72.34989...	68.73884...	65.94239...	63.96246...	62.78588...	62.39804...	62.79256...	63.97486...	65.95876...	68.75694...	72.36740...	76.75898...	81.85947...	87.54939...	93.66345...	100.0
100.0	93.40548...	87.04089...	81.11609...	75.80371...	71.22907...	67.46837...	64.55491...	62.49124...	61.26415...	60.85885...	61.26895...	62.50014...	64.56667...	67.48139...	71.24168...	75.81450...	81.12412...	87.04583...	93.40761...	100.0
100.0	93.27124...	86.77566...	80.72648...	75.29962...	70.62346...	66.77673...	63.79462...	61.68089...	60.42302...	60.00654...	60.42528...	61.68501...	63.79988...	66.78223...	70.62832...	75.30321...	80.72851...	86.77632...	93.27113...	100.0
100.0	93.25771...	86.74609...	80.67705...	75.22667...	70.52475...	66.65226...	63.64679...	61.51427...	60.24381...	59.82201...	60.24314...	61.51281...	63.64437...	66.64870...	70.52005...	75.22107...	80.67114...	86.74081...	93.25434...	100.0
100.0	93.37192...	86.96486...	80.98397...	75.60221...	70.94970...	67.11004...	64.12449...	62.00265...	60.73666...	60.31508...	60.73295...	61.99539...	64.11404...	67.09699...	70.93490...	75.58682...	80.96949...	86.95310...	93.36492...	100.0
100.0	93.63095...	87.46335...	81.68850...	76.47297...	71.94707...	68.19872...	65.27525...	63.19231...	61.94693...	61.53087...	61.94035...	63.17960...	65.25731...	68.17685...	71.92296...	76.44864...	81.66625...	87.44574...	93.62070...	100.0
100.0	94.06526...	88.29678...	82.86233...	77.91955...	73.60091...	70.00262...	67.18226...	65.16504...	63.95546...	63.54996...	63.94645...	65.14777...	67.15816...	69.97368...	73.56958...	77.88854...	82.83452...	88.27514...	94.05286...	100.0
100.0	94.72377...	89.55101...	84.61305...	80.05893...	76.03007...	72.63889...	69.95993...	68.03278...	66.87258...	66.48220...	66.86181...	68.01230...	69.93169...	72.60558...	75.99475...	80.02480...	84.58318...	89.52828...	94.71095...	100.0
100.0	95.68413...	91.35470...	87.09166...	83.04584...	79.38436...	76.25008...	73.74418...	71.92665...	70.82657...	70.45500...	70.81486...	71.90460...	73.71433...	76.21576...	79.34917...	83.01318...	87.06432...	91.33478...	95.67328...	100.0
100.0	97.07723...	93.90332...	90.50453...	87.07279...	83.83663...	80.99175...	78.67796...	76.98140...	75.94775...	75.59727...	75.93597...	76.95958...	78.64926...	80.96017...	83.80624...	87.04696...	90.48526...	93.89108...	97.07136...	100.0
100.0	99.15084...	97.50787...	95.13077...	92.36639...	89.56898...	87.01495...	84.89261...	83.31681...	82.34989...	82.02093...	82.33886...	83.29692...	84.86774...	86.98986...	89.54822...	92.35314...	95.12585...	97.50922...	99.15395...	100.0
100.0	102.4518...	102.6850...	101.3357...	99.17021...	96.74848...	94.44258...	92.48642...	91.01788...	90.11148...	89.80248...	90.10188...	91.00133...	92.46764...	94.42725...	96.74178...	99.17524...	101.3521...	102.7075...	102.4699...	100.0
100.0	108.4010...	110.2747...	109.5360...	107.6921...	105.4862...	103.3401...	101.5026...	100.1169...	99.25969...	98.96749...	99.25191...	100.1045...	101.4914...	103.3368...	105.4970...	107.7205...	109.5808...	110.3283...	108.4461...	100.0

time = 6000.0 seconds

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
100.0	99.286699...	98.590806...	97.931804...	97.327133...	96.791841...	96.338320...	95.976241...	95.712685...	95.552388...	95.498026...	95.550436...	95.708717...	95.970175...	96.330139...	96.781708...	97.315519...	97.919629...	98.579621...	99.278999...	100.0
100.0	98.589515...	97.214781...	95.913435...	94.719624...	93.662934...	92.767768...	92.053189...	91.533162...	91.217037...	91.110104...	91.214026...	91.527026...	92.043782...	92.755054...	93.647187...	94.701640...	95.894757...	97.197939...	98.578325...	100.0
100.0	97.926650...	95.906992...	93.995739...	92.242739...	90.691275...	89.377093...	88.328144...	87.564919...	87.101156...	86.944635...	87.097811...	87.558078...	88.317603...	89.362783...	90.673503...	92.222448...	93.974762...	95.888274...	97.914444...	100.0
100.0	97.314309...	94.699097...	92.224796...	89.955664...	87.947566...	86.246686...	84.889174...	83.901561...	83.301658...	83.099570...	83.298493...	83.895050...	84.879066...	86.232863...	87.930300...	89.935891...	92.204363...	94.680949...	97.302583...	100.0
100.0	96.766443...	93.618455...	90.640474...	87.909593...	85.492902...	83.445930...	81.812199...	80.623701...	79.901949...	79.659170...	79.899247...	80.618100...	81.803415...	83.433794...	85.477612...	87.891980...	90.622223...	93.602253...	96.756013...	100.0
100.0	96.294671...	92.687855...	89.275957...	86.147065...	83.377941...	81.032223...	79.159892...	77.797770...	76.970676...	76.692753...	76.968507...	77.793233...	79.152694...	81.022162...	83.365142...	86.132218...	89.260508...	92.674121...	96.285839...	100.0
100.0	95.908512...	91.925943...	88.158314...	84.702645...	81.643656...	79.051782...	76.982514...	75.476874...	74.562607...	74.255577...	74.560864...	75.473208...	76.976654...	79.043534...	81.633104...	84.690357...	88.145506...	91.914555...	95.901199...	100.0
100.0	95.615967...	91.348328...	87.310071...	83.604863...	80.323574...	77.542095...	75.320518...	73.703498...	72.721393...	72.391639...	72.719855...	73.700281...	75.315414...	77.534965...	80.314520...	83.594394...	87.299229...	91.338747...	95.609850...	100.0
100.0	95.424410...	90.969300...	86.751619...	82.879199...	79.447186...	76.535650...	74.208500...	72.513606...	71.483750...	71.137882...	71.482153...	72.510327...	74.203431...	76.528764...	79.438668...	82.869572...	86.741841...	90.960799...	95.419055...	100.0
100.0	95.341729...	90.804005...	86.504257...	82.551770...	79.044106...	76.064410...	73.679883...	71.941452...	70.884348...	70.529106...	70.882449...	71.937645...	73.674191...	76.056960...	79.035225...	82.542067...	86.494688...	90.795890...	95.336723...	100.0
100.0	95.377709...	90.871017...	86.593660...	82.653405...	79.148503...	76.164445...	73.771648...	72.024355...	70.960593...	70.602756...	70.958240...	72.019723...	73.764909...	76.155910...	79.138675...	82.643022...	86.583730...	90.862820...	95.372764...	100.0
100.0	95.545676...	91.195312...	87.053690...	83.223849...	79.803396...	76.880150...	74.528427...	72.806679...	71.756540...	71.402796...	71.753709...	72.801177...	74.520579...	76.870454...	79.792531...	83.212686...	87.043293...	91.186930...	95.540722...	100.0
100.0	95.864660...	91.811919...	87.930686...	84.316001...	81.064475...	78.267559...	76.005359...	74.342362...	73.325214...	72.981968...	73.322039...	74.336251...	75.996775...	78.257163...	81.053089...	84.304579...	87.920291...	91.803710...	95.859888...	100.0
100.0	96.362617...	92.770985...	89.288722...	86.000317...	83.003276...	80.396367...	78.269246...	76.695485...	75.728867...	75.401931...	75.725638...	76.689332...	78.260743...	80.386287...	82.992513...	85.989811...	89.279412...	92.763800...	96.358509...	100.0
100.0	97.082123...	94.145837...	91.216836...	88.369662...	85.709536...	83.350318...	81.397608...	79.938541...	79.036832...	78.730991...	79.033978...	79.933187...	81.390398...	83.342075...	85.701126...	88.361872...	91.210295...	94.141020...	97.079450...	100.0
100.0	98.091853...	96.047312...	93.839845...	91.544881...	89.292357...	87.225462...	85.475290...	84.148210...	83.320933...	83.039397...	83.318983...	84.144694...	85.470884...	87.220971...	89.288512...	91.542151...	93.838316...	96.046698...	98.091686...	100.0
100.0	99.512561...	98.650871...	97.334712...	95.680480...	93.879000...	92.125413...	90.588225...	89.398614...	88.648625...	88.392473...	88.648155...	89.398076...	90.588298...	92.126830...	93.882263...	95.685586...	97.340956...	98.656820...	99.516339...	100.0
100.0	101.58249...	102.24854...	101.95265...	100.96701...	99.608392...	98.152077...	96.813913...	95.752283...	95.074418...	94.842204...	95.075991...	95.755846...	96.820140...	98.161608...	99.621459...	100.98305...	101.96995...	102.26411...	101.59227...	100.0
100.0	104.84127...	107.34235...	108.03723...	107.61979...	106.61377...	105.38989...	104.20853...	103.24951...	102.63054...	102.41833...	102.63465...	103.25816...	104.22241...	105.40957...	106.63927...	107.65000...	108.06944...	107.37172...	104.86047...	100.0
100.0	110.70661...	114.76425...	115.99278...	115.83091...	114.98732...	113.88284...	112.79187...	111.89772...	111.31846...	111.12049...	111.32545...	111.91219...	112.81455...	113.91431...	115.02750...	115.87841...	116.04423...	114.81343...	110.74242...	100.0

time = 8000.0 seconds

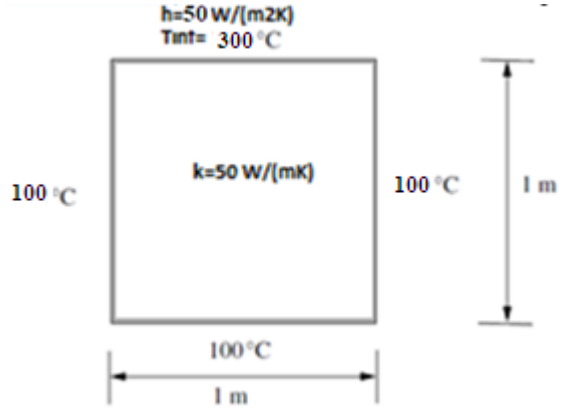
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
100.0	99.50216...	99.01553...	98.55288...	98.12599...	97.74552...	97.42081...	97.15969...	96.96835...	96.85127...	96.81117...	96.84897...	96.96383...	97.15312...	97.41248...	97.73585...	98.11559...	98.54261...	99.00660...	99.49629...	100.0
100.0	99.01482...	98.05267...	97.13821...	96.29457...	95.54276...	94.90118...	94.38531...	94.00739...	93.77627...	93.69732...	93.77239...	93.99977...	94.37427...	94.88722...	95.52665...	96.27741...	97.12149...	98.03842...	99.00581...	100.0
100.0	98.55017...	97.13490...	95.79011...	94.54961...	93.44420...	92.50092...	91.74253...	91.18704...	90.84750...	90.73180...	90.84266...	91.17756...	91.72880...	92.48362...	93.42432...	94.52857...	95.76982...	97.11787...	98.53963...	100.0
100.0	98.11969...	96.28470...	94.54135...	92.93327...	91.50031...	90.27752...	89.29442...	88.57443...	88.13451...	87.98493...	88.12921...	88.56406...	89.27943...	90.25868...	91.47875...	92.91057...	94.51962...	96.26665...	98.10866...	100.0
100.0	97.73419...	95.52334...	93.42298...	91.48551...	89.75887...	88.28534...	87.10057...	86.23290...	85.70289...	85.52301...	85.69751...	86.22238...	87.08539...	88.26630...	89.73716...	91.46275...	93.40132...	95.50548...	97.72337...	100.0
100.0	97.40390...	94.87091...	92.46432...	90.24400...	88.26484...	86.57541...	85.21677...	84.22168...	83.61393...	83.40797...	83.60873...	84.21152...	85.20213...	86.55709...	88.24401...	90.22224...	92.44371...	94.85400...	97.39372...	100.0
100.0	97.13857...	94.34655...	91.69323...	89.24440...	87.06061...	85.19567...	83.69530...	82.59611...	81.92478...	81.69754...	81.91992...	82.58663...	83.68167...	85.17864...	87.04129...	89.22429...	91.67425...	94.33104...	97.12928...	100.0
100.0	96.94774...	93.96893...	91.13678...	88.52117...	86.18688...	84.19187...	82.58575...	81.40847...	80.68928...	80.44606...	80.68486...	81.39984...	82.57336...	84.17643...	86.16941...	88.50304...	91.11972...	93.95504...	96.93946...	100.0
100.0	96.84117...	93.75705...	90.82229...	88.10877...	85.68394...	83.60884...	81.93630...	80.70921...	79.95921...	79.70570...	79.95527...	80.70152...	81.92528...	83.59514...	85.66847...	88.09277...	90.80728...	93.74488...	96.83394...	100.0
100.0	96.82944...	93.73136...	90.77879...	88.04330...	85.59337...	83.49220...	81.79536...	80.54854...	79.78575...	79.52797...	79.78231...	80.54184...	81.78578...	83.48032...	85.58000...	88.02951...	90.76589...	93.72093...	96.82326...	100.0
100.0	96.92475...	93.91520...	91.03896...	88.36459...	85.96011...	83.89025...	82.21329...	80.97798...	80.22100...	79.96513...	80.21811...	80.97235...	82.20525...	83.88031...	85.94896...	88.35312...	91.02827...	93.90658...	96.91964...	100.0
100.0	97.14217...	94.33697...	91.64161...	89.11873...	86.83472...	84.85592...	83.24404...	82.05175...	81.31916...	81.07135...	81.31688...	82.04732...	83.23774...	84.84816...	86.82604...	89.10984...	91.63334...	94.33030...	97.13823...	100.0
100.0	97.50149...	95.03314...	92.63517...	90.36125...	88.27596...	86.44866...	84.94641...	83.82759...	83.13714...	82.90326...	83.13559...	83.82460...	84.94217...	86.44348...	88.27020...	90.35537...	92.62972...	95.02873...	97.49887...	100.0
100.0	98.03034...	96.05318...	94.08266...	92.16116...	90.35355...	88.73600...	87.38470...	86.36684...	85.73425...	85.51948...	85.73362...	86.36563...	87.38303...	88.73400...	90.35139...	92.15899...	94.08064...	96.05151...	98.02929...	100.0
100.0	98.76996...	97.46777...	96.06918...	94.60619...	93.15098...	91.79445...	90.62842...	89.73336...	89.17082...	88.97914...	89.17133...	89.73438...	90.62996...	91.79645...	93.15332...	94.60867...	96.07148...	97.46953...	98.77088...	100.0
100.0	99.78700...	99.38372...	98.71359...	97.80940...	96.76792...	95.70922...	94.75041...	93.99098...	93.50527...	93.33890...	93.50719...	93.99477...	94.75593...	95.71620...	96.77592...	97.81774...	98.72137...	99.38986...	99.79045...	100.0
100.0	101.1998...	101.9718...	102.1859...	101.9163...	101.3208...	100.5714...	99.82287...	99.19913...	98.78978...	98.64859...	98.79342...	99.20627...	99.83325...	100.5845...	101.3357...	101.9320...	102.2007...	101.9836...	101.2066...	100.0
100.0	103.2446...	105.5197...	106.7309...	107.1090...	106.9383...	106.4714...	105.9103...	105.4065...	105.0645...	104.9457...	105.0702...	105.4176...	105.9265...	106.4918...	106.9617...	107.1338...	106.7546...	105.5390...	103.2559...	100.0
100.0	106.4593...	110.5262...	112.6873...	113.5965...	113.7460...	113.4847...	113.0594...	112.6432...	112.3511...	112.2492...	112.3590...	112.6588...	113.0821...	113.5135...	113.7795...	113.6323...	112.7223...	110.5558...	106.4776...	100.0
100.0	112.2615...	117.8220...	120.4572...	121.5678...	121.8326...	121.6513...	121.2854...	120.9119...	120.6460...	120.5538...	120.6564...	120.9323...	121.3153...	121.6895...	121.8774...	121.6167...	120.5064...	117.8660...	112.2918...	100.0

time = 10000.0 seconds

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
100.0	99.65025...	99.30814...	98.98250...	98.68153...	98.41278...	98.18293...	97.99769...	97.86168...	97.77827...	97.74955...	97.77623...	97.85770...	97.99197...	98.17577...	98.40461...	98.67291...	98.97415...	99.30101...	99.64565...	100.0
100.0	99.30855...	98.63280...	97.98977...	97.39555...	96.86496...	96.41121...	96.04559...	95.77718...	95.61268...	95.55619...	95.60916...	95.77031...	96.03574...	96.39894...	96.85103...	97.38098...	97.97584...	98.62117...	99.30133...	100.0
100.0	98.98429...	97.99210...	97.04814...	96.17591...	95.39709...	94.73107...	94.19442...	93.80054...	93.55925...	93.47660...	93.55472...	93.79173...	94.18182...	94.71542...	95.37942...	96.15756...	97.03079...	97.97781...	98.97561...	100.0
100.0	98.68657...	97.40387...	96.18364...	95.05609...	94.04924...	93.18814...	92.49429...	91.98506...	91.67323...	91.56667...	91.66812...	91.97513...	92.48011...	93.17058...	94.02950...	95.03570...	96.16449...	97.38827...	98.67721...	100.0
100.0	98.42433...	96.88571...	95.42195...	94.06919...	92.86097...	91.82744...	90.99451...	90.38317...	90.00892...	89.88128...	90.00357...	90.37278...	90.97969...	91.80914...	92.84046...	94.04809...	95.40226...	96.86977...	98.41486...	100.0
100.0	98.20648...	96.45509...	94.78859...	93.24795...	91.87138...	90.69334...	89.74361...	89.04639...	88.61961...	88.47431...	88.61430...	89.03608...	89.72891...	90.67522...	91.85113...	93.22720...	94.76930...	96.43956...	98.19730...	100.0
100.0	98.04195...	96.12960...	94.30917...	92.62518...	91.11943...	89.82989...	88.78960...	88.02555...	87.55781...	87.39878...	87.55274...	88.01572...	88.77561...	89.81267...	91.10022...	92.60554...	94.29097...	96.11500...	98.03336...	100.0
100.0	97.94000...	95.92732...	94.00986...	92.23418...	90.64447...	89.28133...	88.18042...	87.37117...	86.87557...	86.70725...	86.87092...	87.36215...	88.16759...	89.26555...	90.62690...	92.21624...	93.99327...	95.91404...	97.93221...	100.0
100.0	97.91042...	95.86733...	93.91814...	92.10962...	90.48705...	89.09278...	87.96462...	87.13411...	86.62507...	86.45231...	86.62096...	87.12615...	87.95330...	89.07888...	90.47157...	92.09384...	93.90356...	95.85568...	97.90359...	100.0
100.0	97.96404...	95.97061...	94.06382...	92.28862...	90.68999...	89.31128...	88.19213...	87.36621...	86.85919...	86.68718...	86.85574...	87.35952...	88.18262...	89.29959...	90.67698...	92.27536...	94.05157...	95.96082...	97.95830...	100.0
100.0	98.11344...	96.26119...	94.48057...	92.81229...	91.29973...	89.98685...	88.91524...	88.12102...	87.63214...	87.46625...	87.62944...	88.11579...	88.90780...	89.97770...	91.28954...	92.80188...	94.47094...	96.25348...	98.10891...	100.0
100.0	98.37397...	96.76803...	95.20804...	93.72778...	92.36811...	91.17387...	90.18944...	89.45440...	88.99980...	88.84538...	88.99795...	89.45082...	90.18434...	91.16759...	92.36109...	93.72057...	95.20134...	96.76262...	98.37076...	100.0
100.0	98.76556...	97.52789...	96.29506...	95.09112...	93.95449...	92.93244...	92.07424...	91.42482...	91.01980...	90.88189...	91.01891...	91.42310...	92.07178...	92.92939...	93.95104...	95.08752...	96.29163...	97.52505...	98.76382...	100.0
100.0	99.31579...	98.59013...	97.80448...	96.97110...	96.12828...	95.32971...	94.63351...	94.09314...	93.75095...	93.63388...	93.75116...	94.09354...	94.63404...	95.33032...	96.12888...	96.97158...	97.80476...	98.59019...	99.31570...	100.0
100.0	100.0656...	100.0249...	99.82071...	99.45454...	98.97184...	98.44090...	97.93528...	97.52163...	97.25189...	97.15875...	97.25333...	97.52441...	97.93921...	98.44567...	98.97706...	99.45970...	99.82527...	100.0283...	100.0674...	100.0
100.0	101.0815...	101.9385...	102.4615...	102.6530...	102.5831...	102.3491...	102.0501...	101.7716...	101.5785...	101.5106...	101.5813...	101.7771...	102.0579...	102.3587...	102.5936...	102.6636...	102.4710...	101.9458...	101.0855...	100.0
100.0	102.4813...	104.5007...	105.8957...	106.7106...	107.0766...	107.1435...	107.0479...	106.9001...	106.7803...	106.7365...	106.7847...	106.9086...	107.0600...	107.1585...	107.0932...	106.7275...	105.9112...	104.5128...	102.4881...	100.0
100.0	104.5008...	107.9986...	110.3672...	111.8079...	112.5794...	112.9123...	112.9910...	112.9535...	112.8951...	112.8716...	112.9012...	112.9654...	113.0080...	112.9333...	112.6030...	111.8323...	110.3898...	108.0167...	104.5112...	100.0
100.0	107.6781...	112.9307...	116.2140...	118.1534...	119.2163...	119.7303...	119.9251...	119.9608...	119.9434...	119.9337...	119.9514...	119.9763...	119.9474...	119.7581...	119.2479...	118.1865...	116.2455...	112.9568...	107.6939...	100.0
100.0	113.4310...	120.1281...	123.8388...	125.9365...	127.0763...	127.6377...	127.8659...	127.9251...	127.9224...	127.9182...	127.9323...	127.9445...	127.8938...	127.6729...	127.1168...	125.9797...	123.8812...	120.1653...	113.4560...	100.0

PROBLEM 5.8 NON-HOMOGENEOUS, TIME DEPENDENT SOLUTION; CARTESIAN COORDINATES, FINITE ELEMENT SOLUTION

Plate initial temperature is $\boldsymbol{T_i = F(x,y) = 50 + 30\sin(2\pi x)\cos(2\pi y)}$ ℃. (t=0). The plate exposed to 100 C temperature at 3 surfaces as shown in the figure and convectve heat flowwith temperature T_∞=300 ℃ and thermal conductivity h=50 W/(m^{2}K) (t>0). Thermal conductivty of plate is k=51 W/(mK), specific heat Cp=490 J/(kgK) and density ρ=7800 kg/m^3
Calculate temperature profile as a function of time.

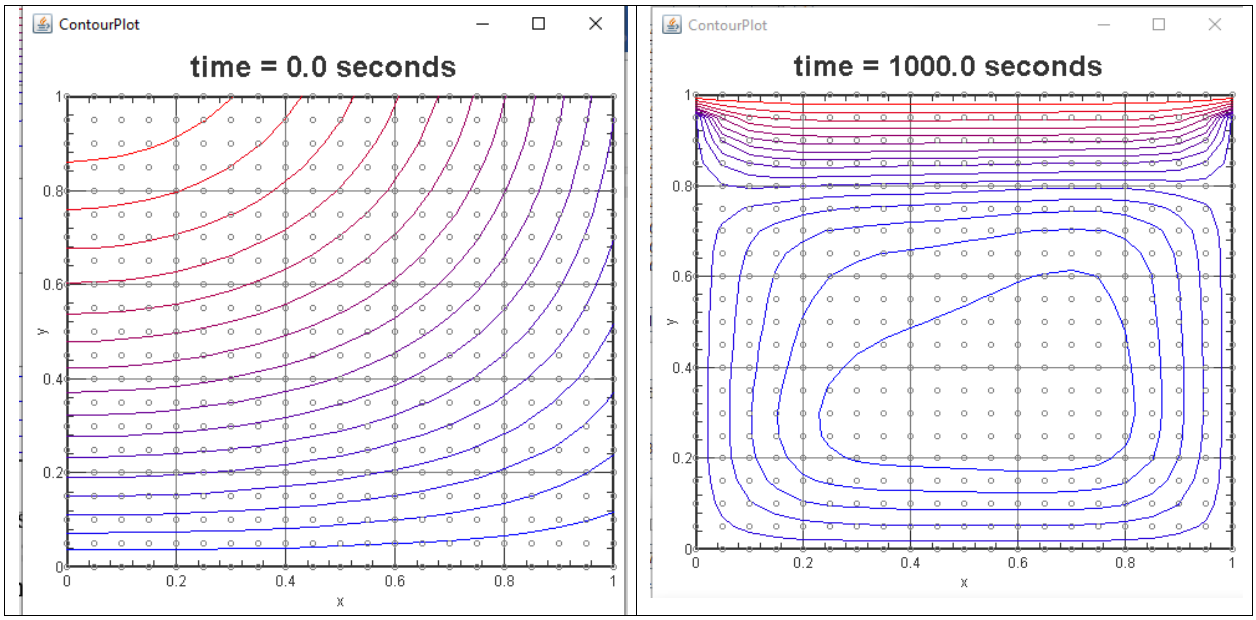


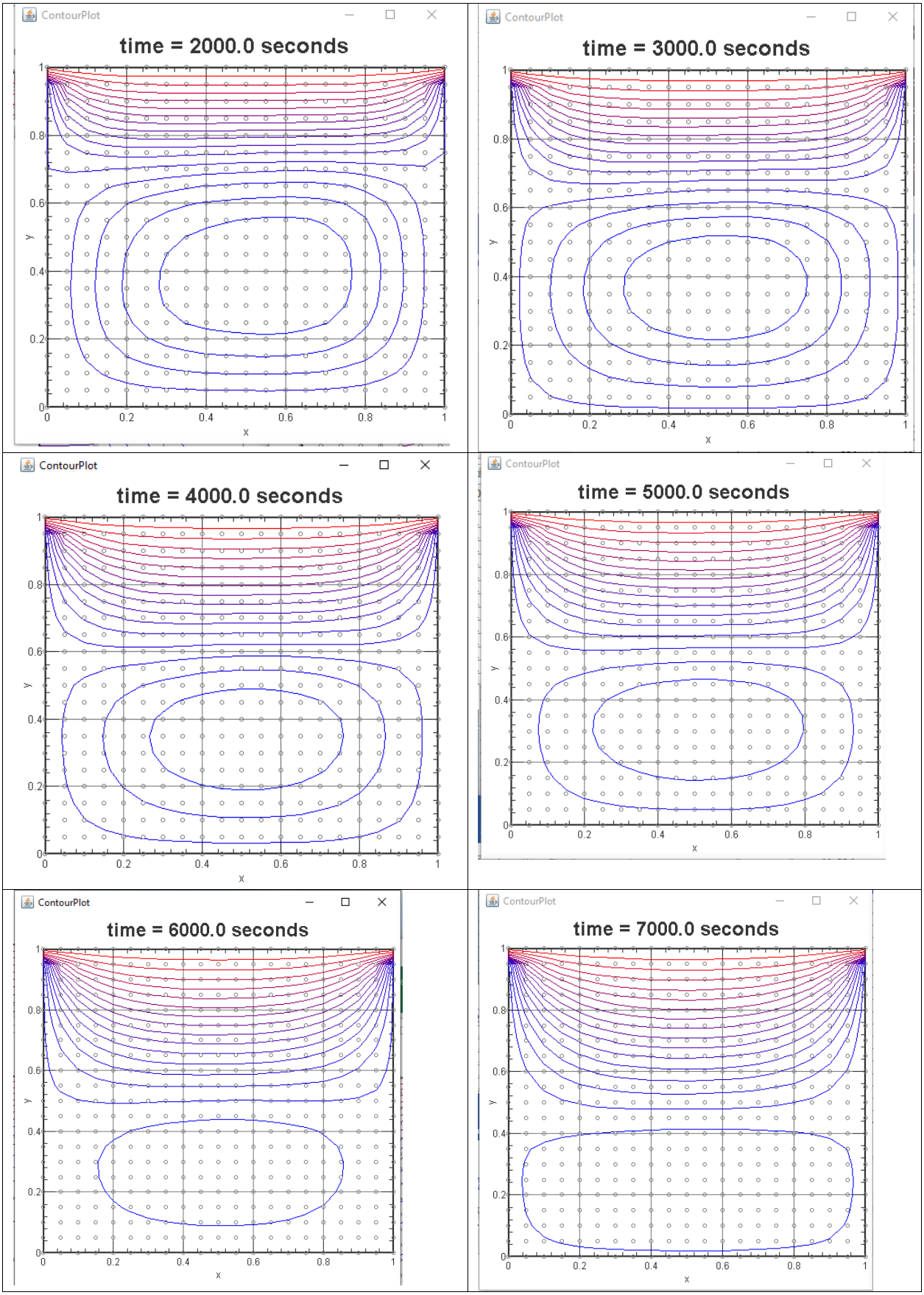

```
public static void main(String arg[]) //throws RemoteException, VisADException
{double L=1.0;
 double W=1.0;
 int N=20,M=20;
 double hi=50.0;
 double Cp=490.0;
 double ro=7800.0;

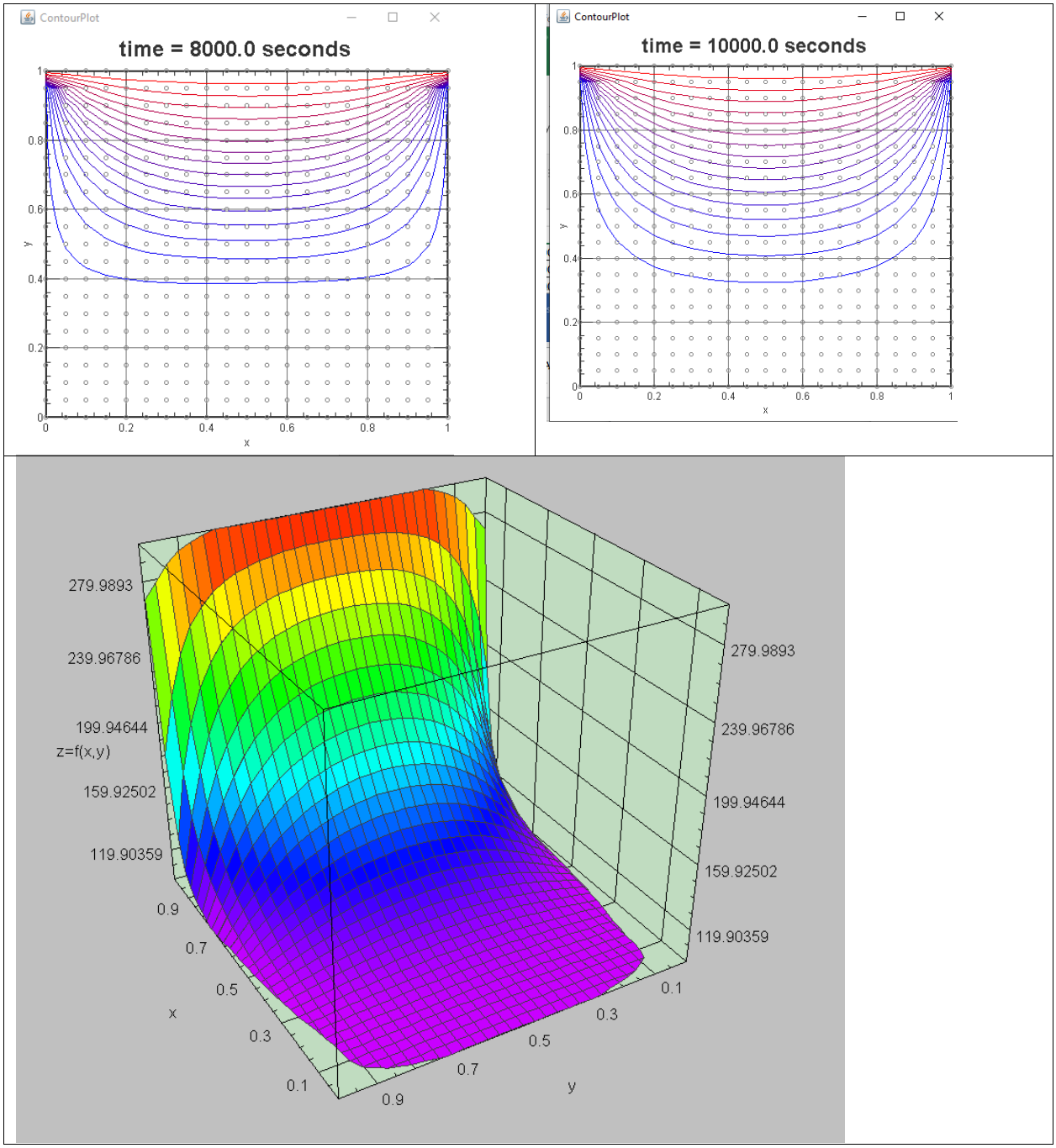
 int cond0=0;//x==0
 int cond1=1;//x==L
 int cond2=2;//y==0
 int cond3=3;//y==W
 double Ti0=100.0;
 double Ti1=100.0;
 double Ti2=100.0;
 double Ti3=300.0;
 double TT[][]=new double[N+1][M+1];
 for(int i=0;i<=N;i++)
 {for(int j=0;j<=M;j++)
   TT[i][j]=50.0+30.0*Math.sin(2.0*Math.PI*i*L/100.0)*Math.cos(2.0*Math.PI*j/100.0);
 }
```

```
triangular_grid2 tg=new triangular_grid2(L,W,M,N,TT,10.0,10000.0);
tg.setTemperature(cond0,0.0,L,0.0,W,Ti0);
tg.setTemperature(cond1,0.0,L,0.0,W,Ti1);
tg.setTemperature(cond2,0.0,L,0.0,W,Ti2);
 tg.setTemperature(cond3,0.0,L,0.0,W,Ti3);
//tg.setConvection(cond3,0.0,L,0.0,W,hi,Ti3);
double k[]={51,51};
double fi=0.5;
tg.solve(1,k,Cp,ro,fi);

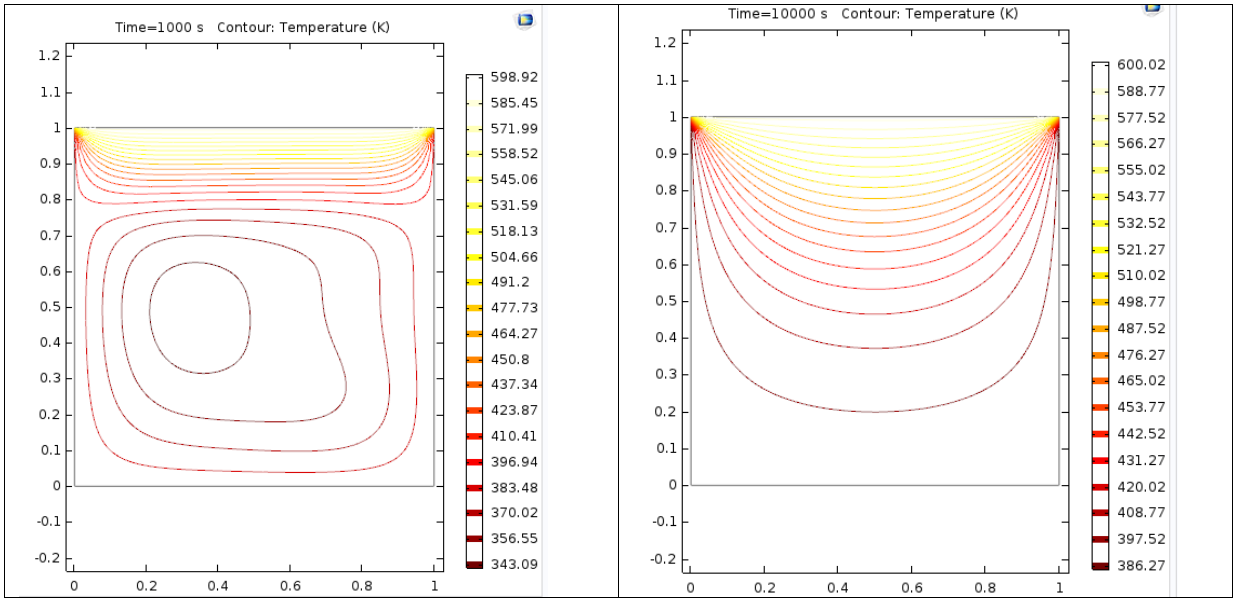
//Take an output at every 2000 seconds
for(int ip=0;ip<=1000;ip++)
{if((ip%100)==0)
  { String s="time = "+ip*tg.dt+" seconds";
            System.out.println(s);
  Text.print(tg.TT[ip],s);
  tg.ContourPlot(tg.TT[ip],s);
  }
 }
//temperature profile t=10 s
tg.surfacePlot(tg.TT[1000]);
}
```




ContourPlot
time = 2000.0 seconds
ContourPlot
time = 3000.0 seconds
ContourPlot
time = 4000.0 seconds
ContourPlot
time = 5000.0 seconds
ContourPlot
time = 6000.0 seconds
ContourPlot
time = 7000.0 seconds
x
y
0
0.2
0.4
0.6
0.8
1



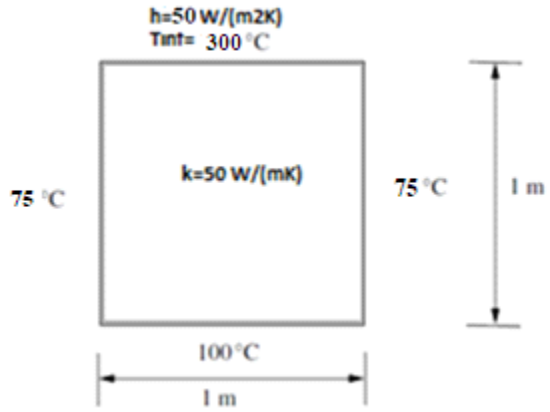
Comsol solutions:



PROBLEM 5.9 NON-HOMOGENEOUS, TIME DEPENDENT SOLUTION; CARTESIAN COORDINATES, FINITE ELEMENT SOLUTION

Plate initial temperature is T_i=50 ℃. (t=0). The plate exposed to 100 C temperature at the bottom surface and 75 ℃ at the side surfaces as shown in the figure and convectve heat flow with temperature T_∞=300 ℃ and thermal conductivity h=50 W/(m^{2}K) (t>0). Thermal conductivty of plate is k=51 W/(mK), specific heat Cp=490 J/(kgK) and density ρ=7800 kg/m^3

Calculate temperature profile as a function of time.

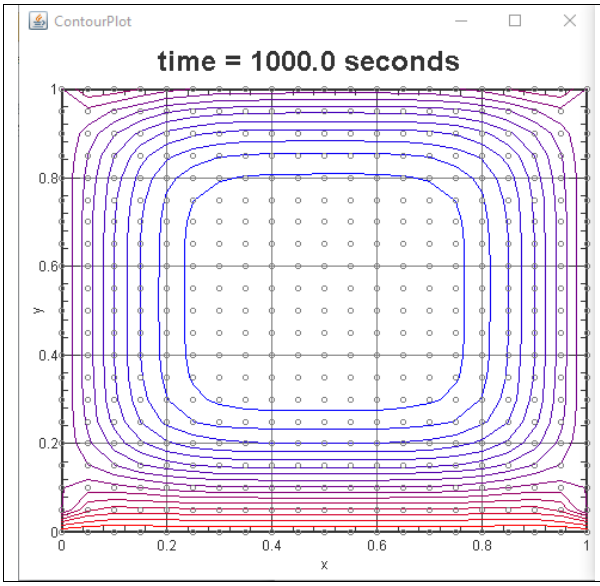


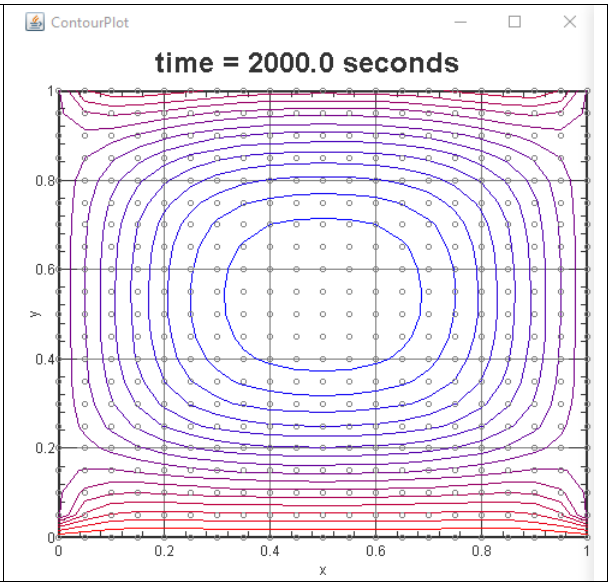

```
public static void main(String arg[])
{double L=1.0;
 double W=1.0;
 int N=20,M=20;
 double hi=50.0;
 double Cp=490.0;
 double ro=7800.0;

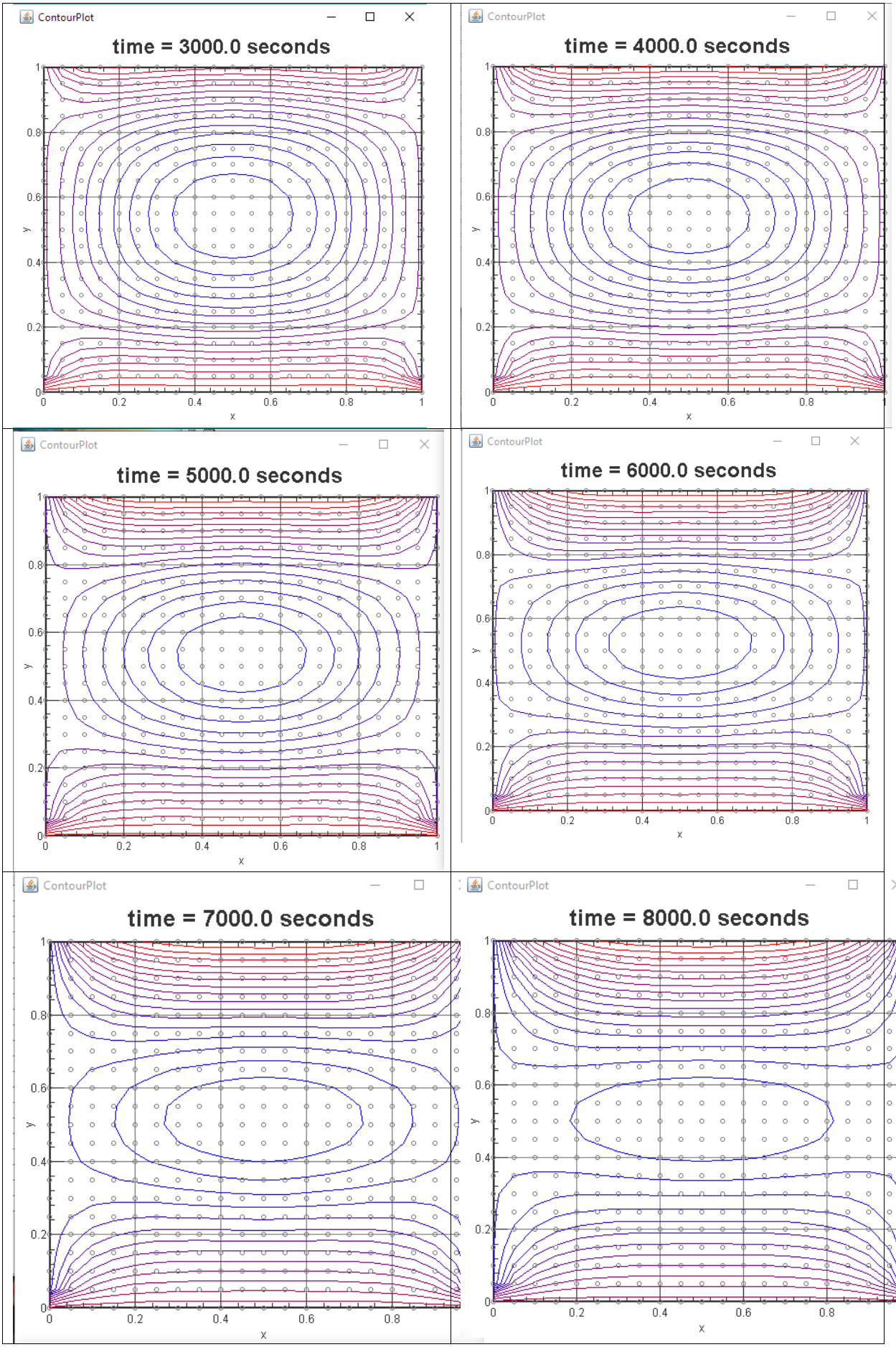
 int cond0=0;//x==0
 int cond1=1;//x==L
 int cond2=2;//y==0
 int cond3=3;//y==W
 double Ti0=75.0;
 double Ti1=75.0;
 double Ti2=100.0;
 double Ti3=300.0;
 double TT[][]=new double[N+1][M+1];
 for(int i=0;i<=N;i++)
 {for(int j=0;j<=M;j++)
```

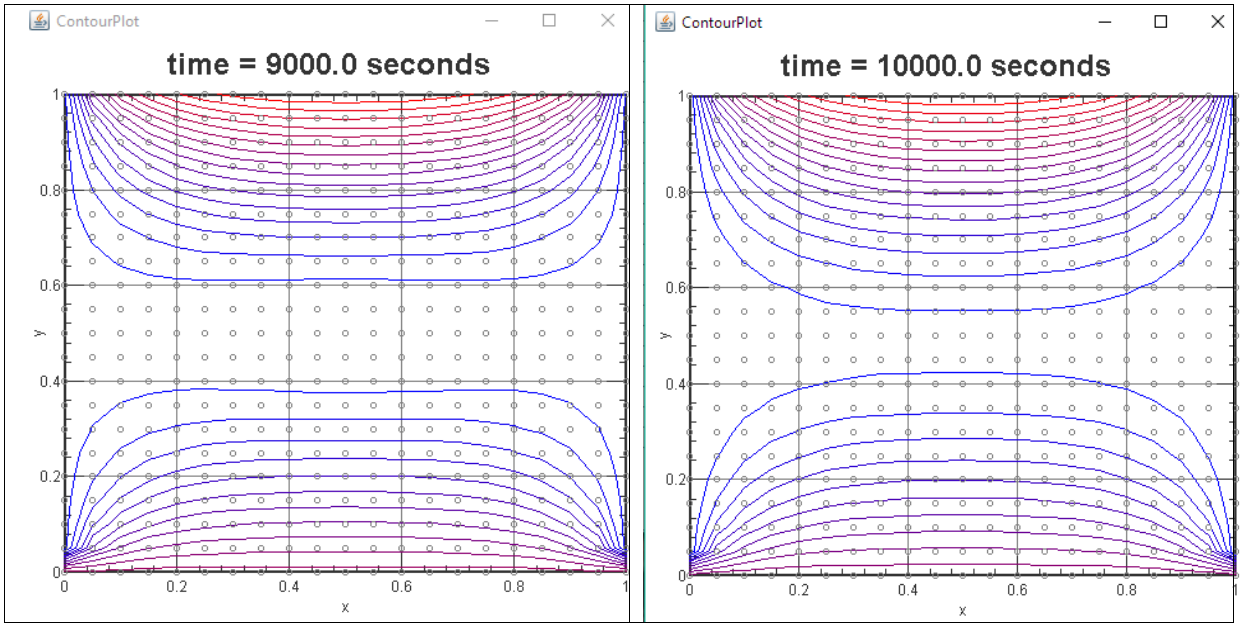
```
  TT[i][j]=50.0;
 }
 triangular_grid2 tg=new triangular_grid2(L,W,M,N,TT,10.0,10000.0);
 tg.setTemperature(cond0,0.0,L,0.0,W,Ti0);
 tg.setTemperature(cond1,0.0,L,0.0,W,Ti1);
 tg.setTemperature(cond2,0.0,L,0.0,W,Ti2);
 tg.setConvection(cond3,0.0,L,0.0,W,hi,Ti3);
 double k[]={51,51};
 double fi=0.5;
 tg.solve(1,k,Cp,ro,fi);

 //Take an output at every 2000 seconds
 for(int ip=0;ip<=1000;ip++)
 {if((ip%100)==0)
   { String s="time = "+ip*tg.dt+" seconds";
              System.out.println(s);
   Text.print(tg.TT[ip],s);
   tg.ContourPlot(tg.TT[ip],s);
   }
 }
 //temperature profile t=10 s
 tg.surfacePlot(tg.TT[1000]);
}
```






ContourPlot
time = 3000.0 seconds
ContourPlot
time = 4000.0 seconds
ContourPlot
time = 5000.0 seconds
ContourPlot
time = 6000.0 seconds
ContourPlot
time = 7000.0 seconds
ContourPlot
time = 8000.0 seconds
x
y

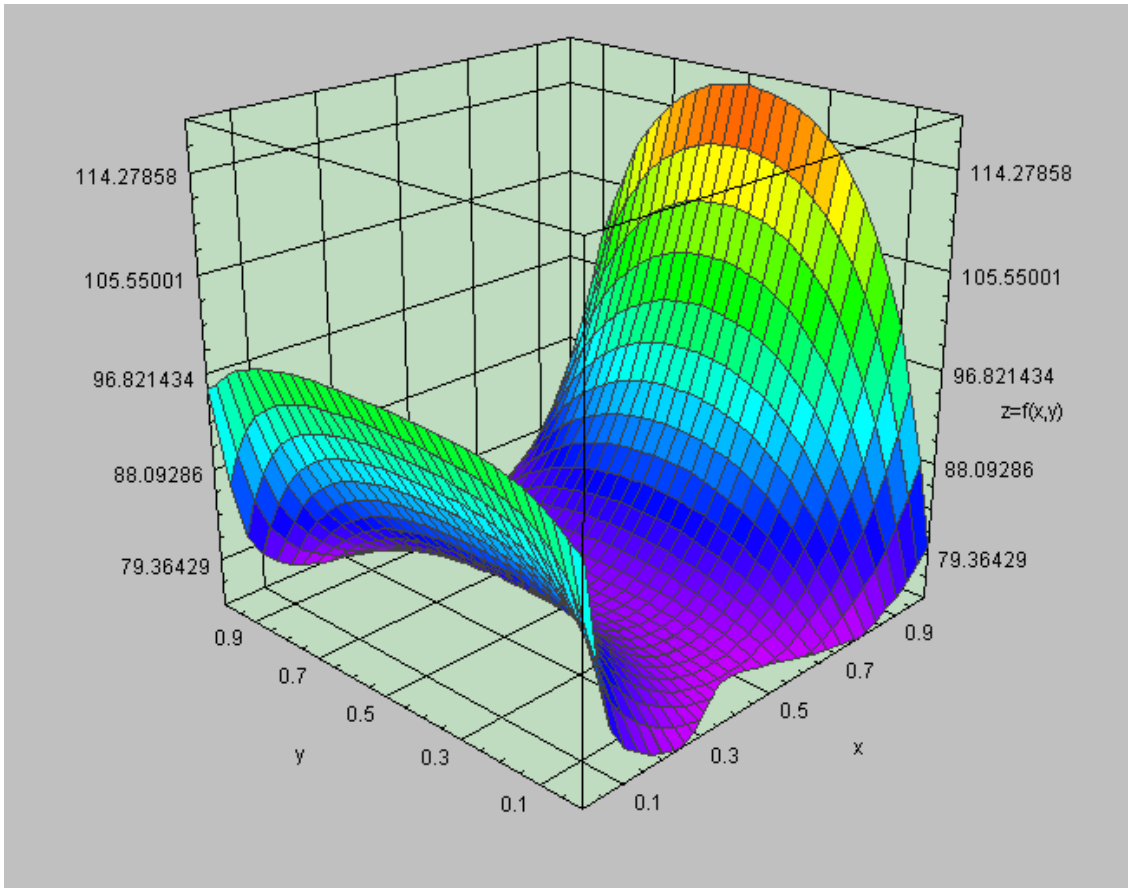


Time=10000 seconds

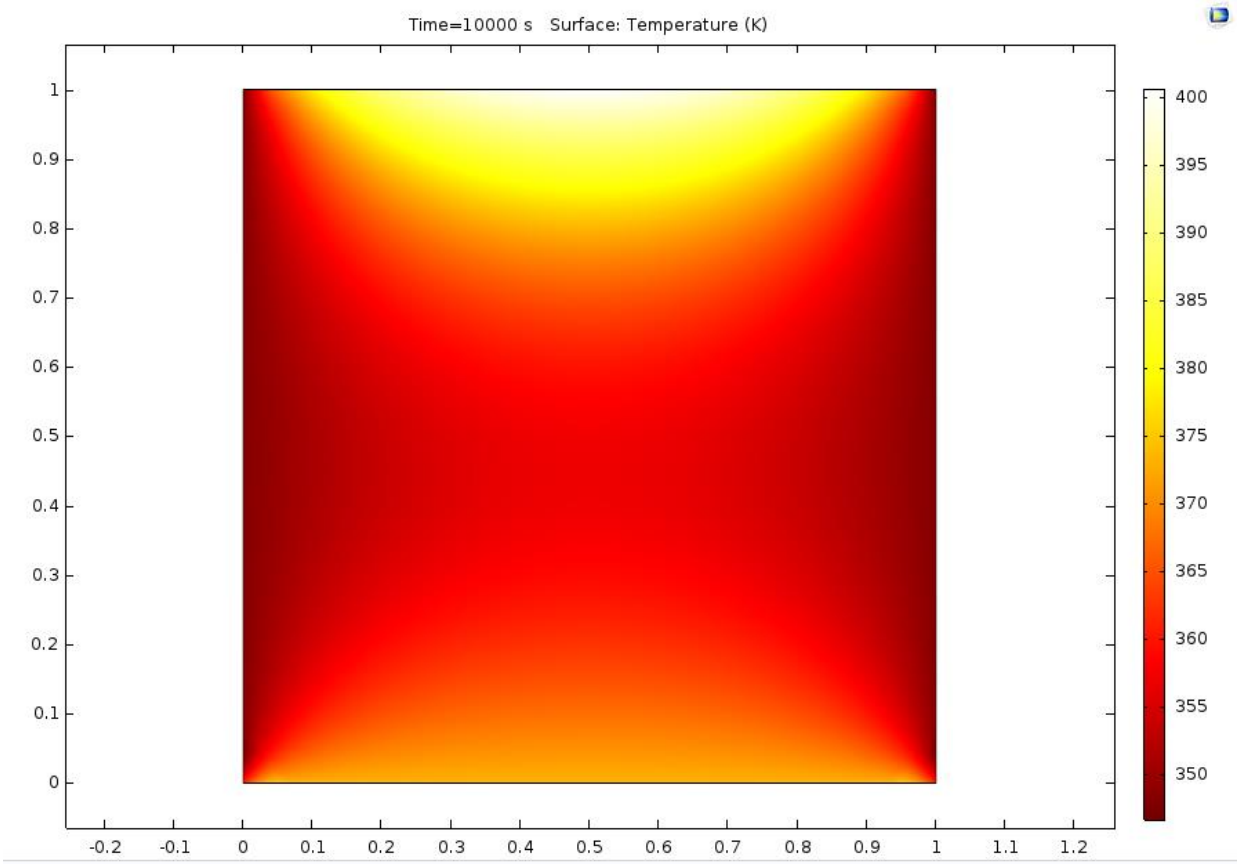
time = 10000.0 seconds

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
75.0	87.229309...	91.903096...	93.966679...	94.989316...	95.537405...	95.844329...	96.018494...	96.115020...	96.163070...	96.177116...	96.161579...	96.112110...	96.014311...	95.839098...	95.531436...	94.983018...	93.960580...	91.897889...	87.225955...	75.0
75.0	82.025628...	86.440531...	89.010491...	90.500524...	91.373299...	91.887297...	92.187439...	92.356468...	92.441325...	92.466239...	92.438748...	92.351447...	92.180239...	91.878328...	91.363124...	90.489882...	89.000323...	86.432034...	82.020350...	75.0
75.0	79.456902...	82.872185...	85.207340...	86.724110...	87.682734...	88.275685...	88.632611...	88.837184...	88.940799...	88.971283...	88.937494...	88.830750...	88.623406...	88.264257...	87.669834...	86.710712...	85.194669...	82.861752...	79.450560...	75.0
75.0	77.966402...	80.457632...	82.331470...	83.647255...	84.528255...	85.095278...	85.445269...	85.648647...	85.752226...	85.782615...	85.748494...	85.641390...	85.434907...	85.082450...	84.513833...	83.632361...	82.317492...	80.446240...	77.959570...	75.0
75.0	76.999504...	78.757407...	80.156686...	81.190741...	81.911211...	82.387722...	82.686399...	82.860879...	82.949550...	82.975259...	82.945639...	82.853282...	82.675568...	82.374345...	81.896222...	81.175329...	80.142302...	78.745768...	76.992585...	75.0
75.0	76.333749...	77.534600...	78.522206...	79.274804...	79.811372...	80.170718...	80.396195...	80.526665...	80.591849...	80.610181...	80.587956...	80.519109...	80.385436...	80.157456...	79.796550...	79.259614...	78.508089...	77.523236...	76.327035...	75.0
75.0	75.870725...	76.664043...	77.327440...	77.840183...	78.208006...	78.452874...	78.603306...	78.687174...	78.727039...	78.737404...	78.723318...	78.679954...	78.593037...	78.440234...	78.193907...	77.825770...	77.314086...	76.653332...	75.864422...	75.0
75.0	75.564312...	76.080819...	76.515000...	76.850620...	77.088708...	77.242597...	77.331918...	77.377247...	77.396018...	77.399770...	77.392586...	77.370591...	77.322457...	77.230965...	77.075751...	76.837397...	76.502774...	76.071036...	75.558570...	75.0
75.0	75.393318...	75.753827...	76.056956...	76.289946...	76.452226...	76.552864...	76.606596...	76.629704...	76.636470...	76.636555...	76.633419...	76.623790...	76.598194...	76.542540...	76.440736...	76.278231...	76.046137...	75.745180...	75.388250...	75.0
75.0	75.350164...	75.672652...	75.946166...	76.158827...	76.309018...	76.403737...	76.455470...	76.478582...	76.485978...	76.486518...	76.483388...	76.473561...	76.448338...	76.394974...	76.299266...	76.148885...	75.936982...	75.665312...	75.345861...	75.0
75.0	75.436128...	75.841436...	76.191703...	76.472748...	76.681398...	76.823836...	76.912336...	76.961427...	76.984335...	76.990192...	76.982280...	76.957442...	76.906673...	76.816874...	76.673641...	76.464829...	76.184374...	75.835563...	75.432676...	75.0
75.0	75.659758...	76.276794...	76.817378...	77.261569...	77.604041...	77.851461...	78.018056...	78.120777...	78.175017...	78.191356...	78.173575...	78.117980...	78.014074...	77.846554...	77.598556...	77.255944...	76.812141...	76.272567...	75.657252...	75.0
75.0	76.037342...	77.008689...	77.862856...	78.570745...	79.125152...	79.535613...	79.821418...	80.004867...	80.105779...	80.137588...	80.105039...	80.003426...	79.819358...	79.533055...	79.122259...	78.567732...	77.859995...	77.006323...	76.035899...	75.0
75.0	76.595519...	78.084379...	79.387551...	80.464323...	81.308364...	81.936952...	82.379408...	82.667470...	82.828282...	82.879820...	82.828345...	82.667584...	82.379551...	81.937087...	81.308447...	80.464311...	79.387417...	78.084145...	76.595288...	75.0
75.0	77.377274...	79.576839...	81.478078...	83.029919...	84.235207...	85.128383...	85.756623...	86.166678...	86.396566...	86.470730...	86.397548...	86.168572...	85.759292...	85.131617...	84.238727...	83.033386...	81.481118...	79.579089...	77.378454...	75.0
75.0	78.454908...	81.601057...	84.260826...	86.385687...	88.007512...	89.194594...	90.023302...	90.562265...	90.864179...	90.961810...	90.866204...	90.566187...	90.028861...	89.201392...	88.015012...	86.393216...	84.267605...	81.606257...	78.457771...	75.0
75.0	79.959627...	84.345255...	87.921485...	90.688292...	92.748089...	94.229187...	95.251155...	95.911239...	96.279782...	96.399026...	96.282980...	95.917448...	95.259990...	94.240060...	92.760190...	90.700587...	87.932729...	84.354050...	79.964572...	75.0
75.0	82.155690...	88.130388...	92.731455...	96.137622...	98.595930...	100.32720...	101.50598...	102.26130...	102.68130...	102.81722...	102.68579...	102.27004...	101.51848...	100.34267...	98.613317...	96.155519...	92.748113...	88.143713...	82.163375...	75.0
75.0	85.647960...	93.516339...	99.069745...	102.96608...	105.68916...	107.57027...	108.83641...	109.64237...	110.08909...	110.23381...	110.09499...	109.65387...	108.85292...	107.59086...	105.71254...	102.99054...	99.093054...	93.535643...	85.659646...	75.0
75.0	92.031750...	101.43781...	107.38867...	111.38625...	114.12725...	116.00313...	117.25959...	118.05740...	118.49929...	118.64287...	118.50664...	118.07179...	117.28035...	116.02923...	114.15726...	111.41827...	107.42017...	101.46540...	92.050322...	75.0

T=10000 s 3-D temperature plot

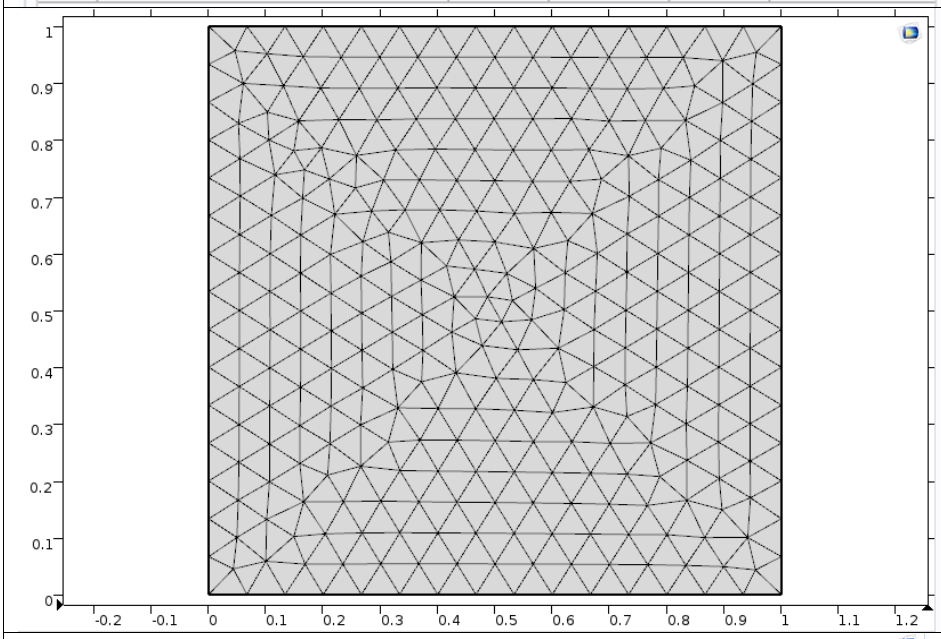


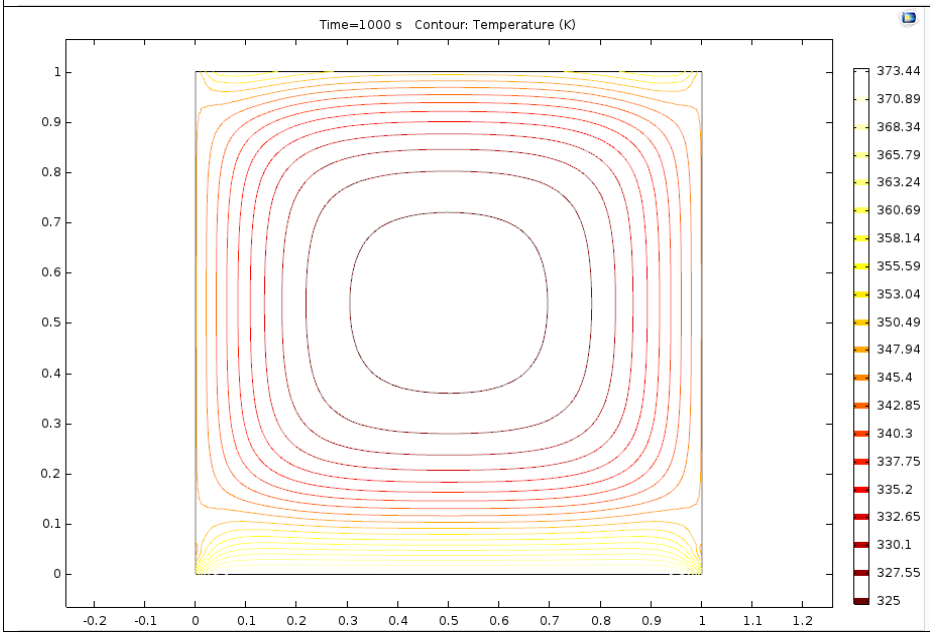
Comsol finite element package results of this problem:

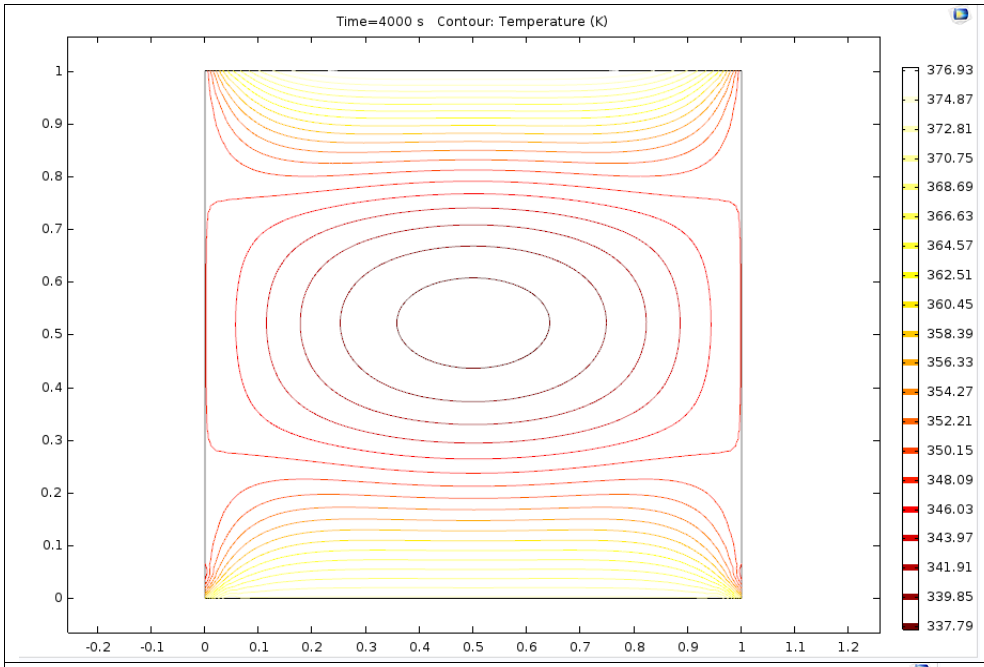


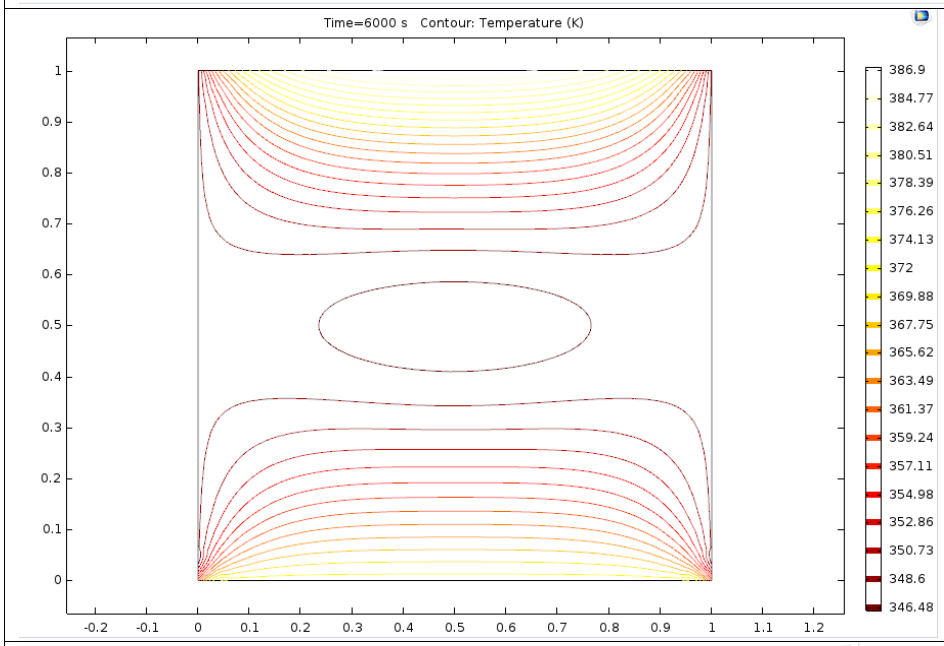
Material Contents

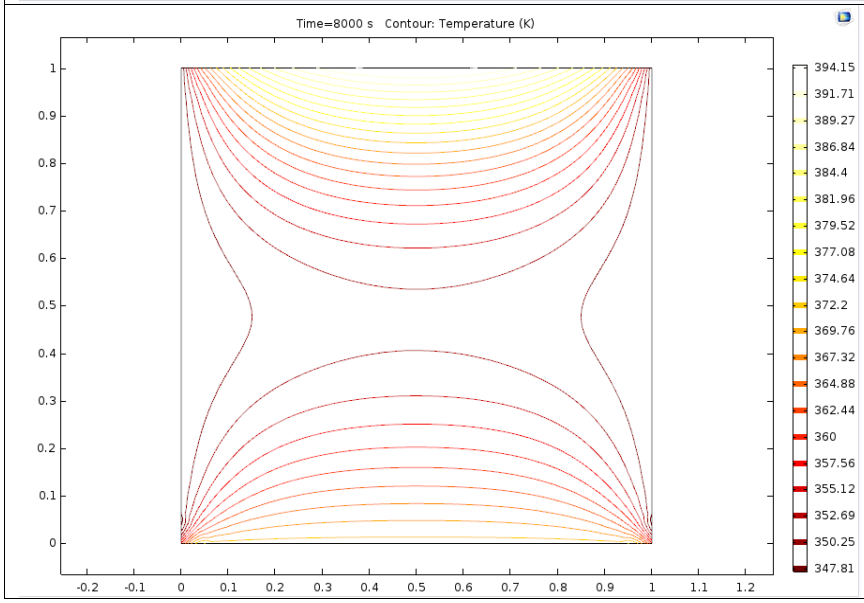
Property	Name	Value	Unit	Property group
Thermal conductivity	k	50	W/(m·...	Basic
Density	rho	490	kg/m³	Basic
Heat capacity at constant pres...	Cp	7800	J/(kg·K)	Basic

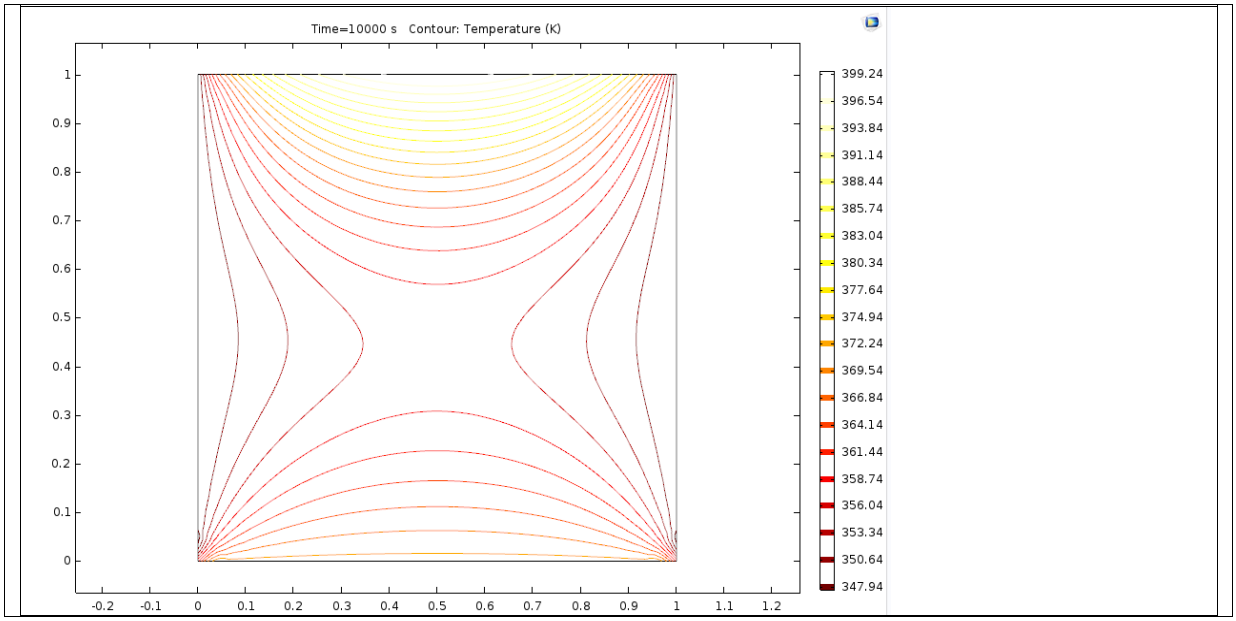





Time=4000 s Contour: Temperature (K)
376.93
374.87
372.81
370.75
368.69
366.63
364.57
362.51
360.45
358.39
356.33
354.27
352.21
350.15
348.09
346.03
343.97
341.91
339.85
337.79


Time=6000 s Contour: Temperature (K)
386.9
384.77
382.64
380.51
378.39
376.26
374.13
372
369.88
367.75
365.62
363.49
361.37
359.24
357.11
354.98
352.86
350.73
348.6
346.48


Time=8000 s Contour: Temperature (K)
394.15
391.71
389.27
386.84
384.4
381.96
379.52
377.08
374.64
372.2
369.76
367.32
364.88
362.44
360
357.56
355.12
352.69
350.25
347.81



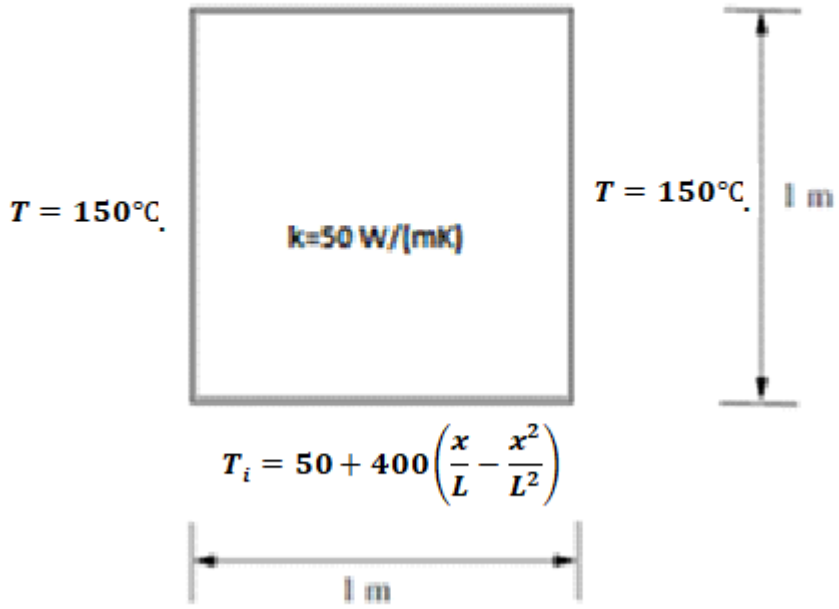
PROBLEM 5.10 NON-HOMOGENEOUS, TIME DEPENDENT SOLUTION; CARTESIAN COORDINATES, FINITE ELEMENT SOLUTION

Plate initial temperature is. (t=0). The plate exposed to $T_i = 150 + 400\left(\frac{x}{L} - \frac{x^2}{L^2}\right)$ ℃. temperature at the bottom surface and 150 ℃ at the side surfaces as shown in the figure and convectve heat flow with temperature T_∞=300 ℃ and thermal conductivity h=50 W/(m^{2}K) (t>0). Thermal conductivty of plate is k=50 W/(mK), specific heat Cp=490 J/(kgK) and density ρ=7800 kg/m^3

Calculate temperature profile as a function of time.

. $$T_i = 50 + 400\left(\frac{x}{r} - \frac{x^2}{L^2}\right)\left(\frac{y}{W} - \frac{y^2}{W}\right)\ ℃..$$

$$h = 50\ \frac{W}{m^2K}\quad T_\infty = 300℃$$



$$T_i = 50 + 400\left(\frac{x}{L} - \frac{x^2}{L^2}\right)$$

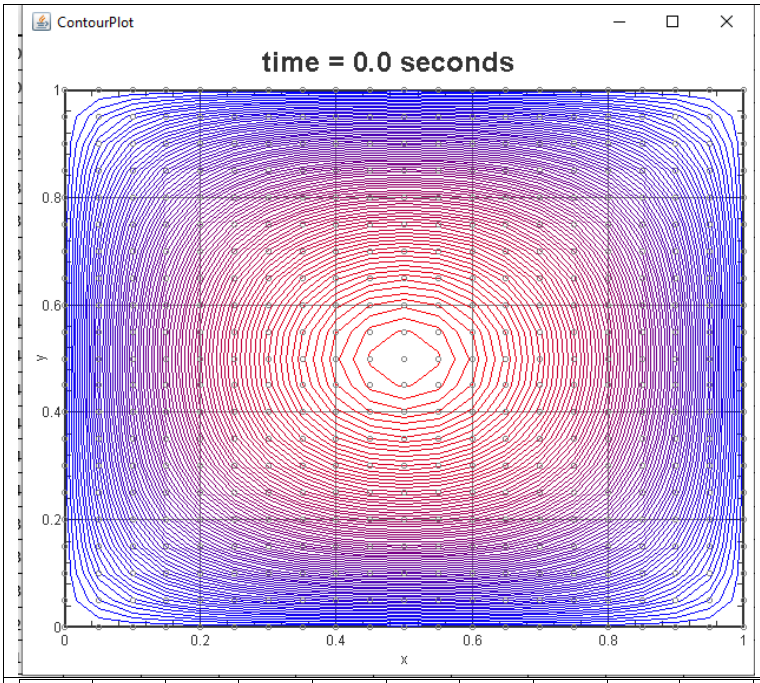

```
public static void main(String arg[]) //throws RemoteException, VisADException
{double L=1.0;
 double W=1.0;
 int N=20,M=20;
 double hi=50.0;
 double Cp=490.0;
```

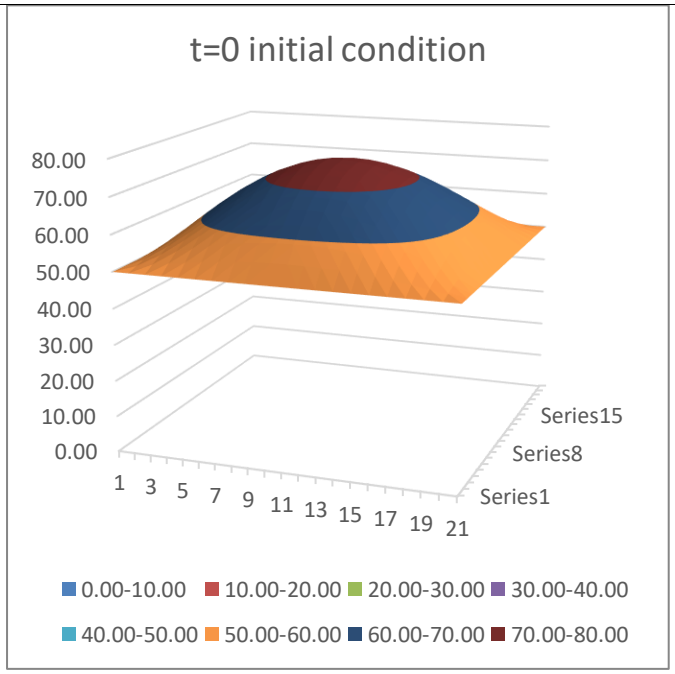
```
double ro=7800.0;

int cond0=0;//x==0
int cond1=1;//x==L
int cond2=2;//y==0
int cond3=3;//y==W
double Ti0=150.0;
double Ti1=150.0;
 if_x Ti2=x->150.0+400.0/L*x-400/(L*L)*x*x;
double Ti3=300.0;
double TT[][]=new double[N+1][M+1];
double dx=L/N;
double dy=W/M;
double x,y;
for(int i=0;i<=N;i++)
{  x=i*dx;
            for(int j=0;j<=M;j++)
            {y=j*dy;
            TT[i][j]=50.0+400*(x/L-x*x/(L*L))*(y/W-y*y/(W*W));
            }
}
triangular_grid2 tg=new triangular_grid2(L,W,M,N,TT,10.0,10000.0);
tg.setTemperature(cond0,0.0,L,0.0,W,Ti0);
tg.setTemperature(cond1,0.0,L,0.0,W,Ti1);
tg.setTemperature(cond2,0.0,L,0.0,W,Ti2);
tg.setConvection(cond3,0.0,L,0.0,W,hi,Ti3);
double k[]={51,51};
double fi=0.5;
tg.solve(1,k,Cp,ro,fi);

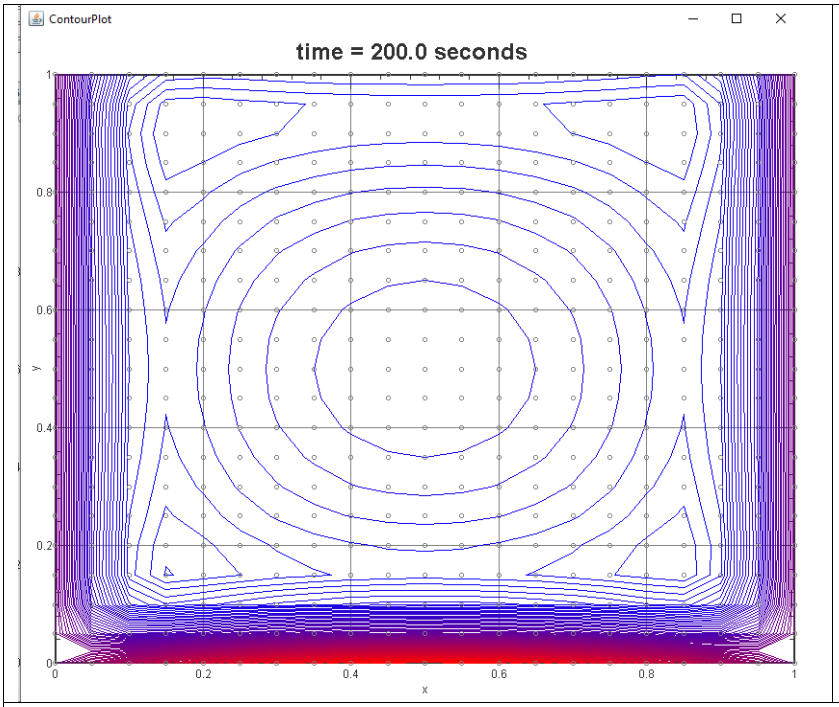
//Take an output at every 2000 seconds
for(int ip=0;ip<=100;ip++)
{if((ip%2)==0)
  { String s="time = "+ip*tg.dt+" seconds";
            System.out.println(s);
  Text.print(tg.TT[ip],s);
  tg.ContourPlot(tg.TT[ip],s);
  }
```

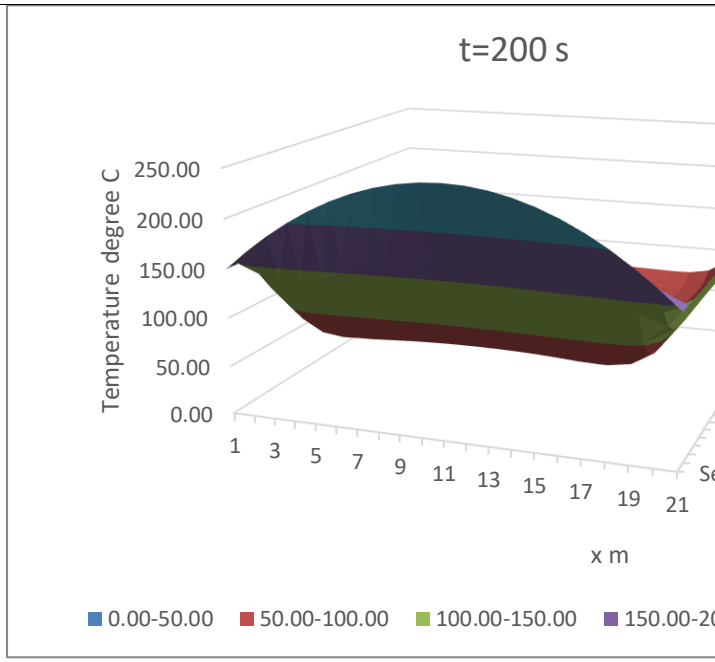
50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00
50.00	50.90	51.71	52.42	53.04	53.56	53.99	54.32	54.56	54.70	54.75	54.70	54.56	54.32	53.99	53.56	53.04	52.42	51.71	50.90	50.00
50.00	51.71	53.24	54.59	55.76	56.75	57.56	58.19	58.64	58.91	59.00	58.91	58.64	58.19	57.56	56.75	55.76	54.59	53.24	51.71	50.00
50.00	52.42	54.59	56.50	58.16	59.56	60.71	61.60	62.24	62.62	62.75	62.62	62.24	61.60	60.71	59.56	58.16	56.50	54.59	52.42	50.00
50.00	53.04	55.76	58.16	60.24	62.00	63.44	64.56	65.36	65.84	66.00	65.84	65.36	64.56	63.44	62.00	60.24	58.16	55.76	53.04	50.00
50.00	53.56	56.75	59.56	62.00	64.06	65.75	67.06	68.00	68.56	68.75	68.56	68.00	67.06	65.75	64.06	62.00	59.56	56.75	53.56	50.00
50.00	53.99	57.56	60.71	63.44	65.75	67.64	69.11	70.16	70.79	71.00	70.79	70.16	69.11	67.64	65.75	63.44	60.71	57.56	53.99	50.00
50.00	54.32	58.19	61.60	64.56	67.06	69.11	70.70	71.84	72.52	72.75	72.52	71.84	70.70	69.11	67.06	64.56	61.60	58.19	54.32	50.00
50.00	54.56	58.64	62.24	65.36	68.00	70.16	71.84	73.04	73.76	74.00	73.76	73.04	71.84	70.16	68.00	65.36	62.24	58.64	54.56	50.00
50.00	54.70	58.91	62.62	65.84	68.56	70.79	72.52	73.76	74.50	74.75	74.50	73.76	72.52	70.79	68.56	65.84	62.62	58.91	54.70	50.00
50.00	54.75	59.00	62.75	66.00	68.75	71.00	72.75	74.00	74.75	75.00	74.75	74.00	72.75	71.00	68.75	66.00	62.75	59.00	54.75	50.00
50.00	54.70	58.91	62.62	65.84	68.56	70.79	72.52	73.76	74.50	74.75	74.50	73.76	72.52	70.79	68.56	65.84	62.62	58.91	54.70	50.00
50.00	54.56	58.64	62.24	65.36	68.00	70.16	71.84	73.04	73.76	74.00	73.76	73.04	71.84	70.16	68.00	65.36	62.24	58.64	54.56	50.00
50.00	54.32	58.19	61.60	64.56	67.06	69.11	70.70	71.84	72.52	72.75	72.52	71.84	70.70	69.11	67.06	64.56	61.60	58.19	54.32	50.00
50.00	53.99	57.56	60.71	63.44	65.75	67.64	69.11	70.16	70.79	71.00	70.79	70.16	69.11	67.64	65.75	63.44	60.71	57.56	53.99	50.00
50.00	53.56	56.75	59.56	62.00	64.06	65.75	67.06	68.00	68.56	68.75	68.56	68.00	67.06	65.75	64.06	62.00	59.56	56.75	53.56	50.00
50.00	53.04	55.76	58.16	60.24	62.00	63.44	64.56	65.36	65.84	66.00	65.84	65.36	64.56	63.44	62.00	60.24	58.16	55.76	53.04	50.00
50.00	52.42	54.59	56.50	58.16	59.56	60.71	61.60	62.24	62.62	62.75	62.62	62.24	61.60	60.71	59.56	58.16	56.50	54.59	52.42	50.00
50.00	51.71	53.24	54.59	55.76	56.75	57.56	58.19	58.64	58.91	59.00	58.91	58.64	58.19	57.56	56.75	55.76	54.59	53.24	51.71	50.00
50.00	50.90	51.71	52.42	53.04	53.56	53.99	54.32	54.56	54.70	54.75	54.70	54.56	54.32	53.99	53.56	53.04	52.42	51.71	50.90	50.00
50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00





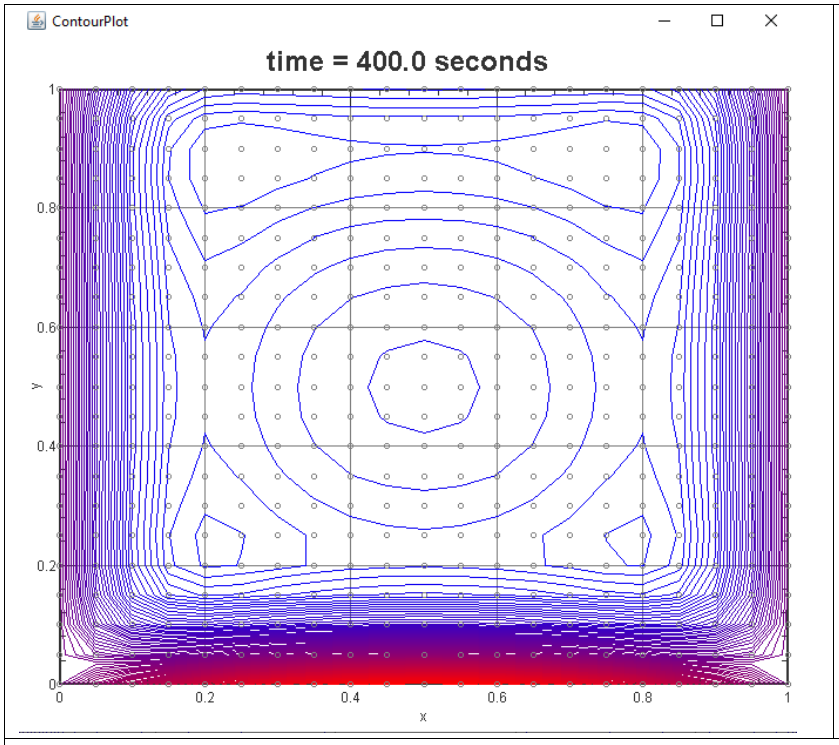
150.00	169.00	186.00	201.00	214.00	225.00	234.00	241.00	246.00	249.00	250.00	249.00	246.00	241.00	234.00	225.00	214.00	201.00	186.00	169.00	150.00
150.00	141.35	136.89	138.23	143.26	149.19	154.49	158.68	161.67	163.47	164.08	163.50	161.73	158.77	154.64	149.47	143.75	138.88	137.30	141.23	150.00
150.00	121.61	102.24	94.29	93.90	96.36	99.12	101.36	102.98	103.97	104.32	104.04	103.12	101.58	99.42	96.84	94.68	95.39	103.31	122.16	150.00
150.00	111.77	85.19	72.91	70.08	70.99	72.60	73.98	74.97	75.58	75.81	75.65	75.11	74.18	72.89	71.40	70.73	73.87	86.29	112.57	150.00
150.00	108.67	79.89	66.43	63.01	63.59	64.96	66.15	67.01	67.53	67.72	67.57	67.09	66.27	65.11	63.80	63.33	66.92	80.53	109.20	150.00
150.00	108.35	79.40	65.96	62.68	63.40	64.89	66.17	67.09	67.64	67.83	67.65	67.11	66.21	64.94	63.46	62.77	66.11	79.62	108.56	150.00
150.00	108.67	80.01	66.85	63.83	64.78	66.45	67.87	68.89	69.50	69.70	69.50	68.89	67.88	66.46	64.78	63.84	66.88	80.05	108.72	150.00
150.00	108.98	80.60	67.71	64.91	66.05	67.87	69.42	70.52	71.18	71.40	71.18	70.52	69.42	67.87	66.05	64.91	67.71	80.61	108.99	150.00
150.00	109.20	81.02	68.31	65.68	66.95	68.89	70.52	71.69	72.39	72.62	72.39	71.69	70.52	68.89	66.95	65.68	68.31	81.02	109.20	150.00
150.00	109.32	81.28	68.67	66.14	67.49	69.50	71.19	72.39	73.11	73.35	73.11	72.39	71.19	69.50	67.49	66.14	68.68	81.28	109.33	150.00
150.00	109.37	81.36	68.80	66.29	67.67	69.70	71.41	72.62	73.35	73.59	73.35	72.62	71.41	69.70	67.67	66.29	68.80	81.36	109.37	150.00
150.00	109.33	81.28	68.68	66.14	67.49	69.50	71.18	72.39	73.11	73.35	73.11	72.39	71.18	69.50	67.49	66.14	68.67	81.28	109.32	150.00
150.00	109.20	81.03	68.31	65.68	66.95	68.89	70.52	71.69	72.39	72.62	72.39	71.69	70.52	68.89	66.95	65.68	68.31	81.02	109.20	150.00
150.00	108.98	80.61	67.71	64.91	66.05	67.88	69.42	70.53	71.19	71.41	71.19	70.53	69.42	67.88	66.05	64.91	67.71	80.61	108.98	150.00
150.00	108.69	80.02	66.87	63.84	64.79	66.46	67.88	68.90	69.50	69.71	69.50	68.90	67.88	66.46	64.79	63.84	66.87	80.02	108.68	150.00
150.00	108.31	79.29	65.79	62.47	63.17	64.64	65.90	66.81	67.35	67.53	67.35	66.81	65.90	64.64	63.17	62.47	65.78	79.27	108.29	150.00
150.00	107.92	78.47	64.57	60.88	61.27	62.50	63.57	64.34	64.80	64.95	64.80	64.34	63.57	62.49	61.27	60.87	64.52	78.40	107.84	150.00
150.00	107.67	77.88	63.55	59.44	59.47	60.41	61.27	61.89	62.25	62.37	62.25	61.88	61.26	60.40	59.45	59.37	63.39	77.65	107.49	150.00
150.00	108.07	78.32	63.72	59.21	58.86	59.51	60.14	60.60	60.87	60.96	60.87	60.59	60.13	59.48	58.79	59.01	63.32	77.82	107.72	150.00
150.00	110.08	81.29	66.87	62.14	61.50	61.89	62.34	62.68	62.88	62.94	62.87	62.67	62.33	61.85	61.34	61.71	66.11	80.50	109.68	150.00
150.00	115.31	88.66	75.05	70.43	69.64	69.90	70.26	70.54	70.71	70.76	70.71	70.54	70.25	69.83	69.35	69.65	73.84	87.74	115.35	150.00

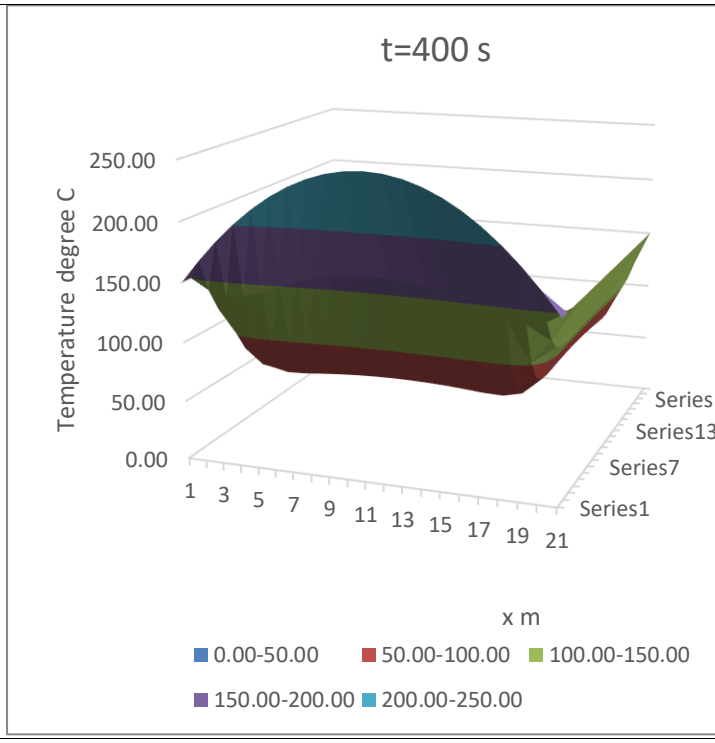


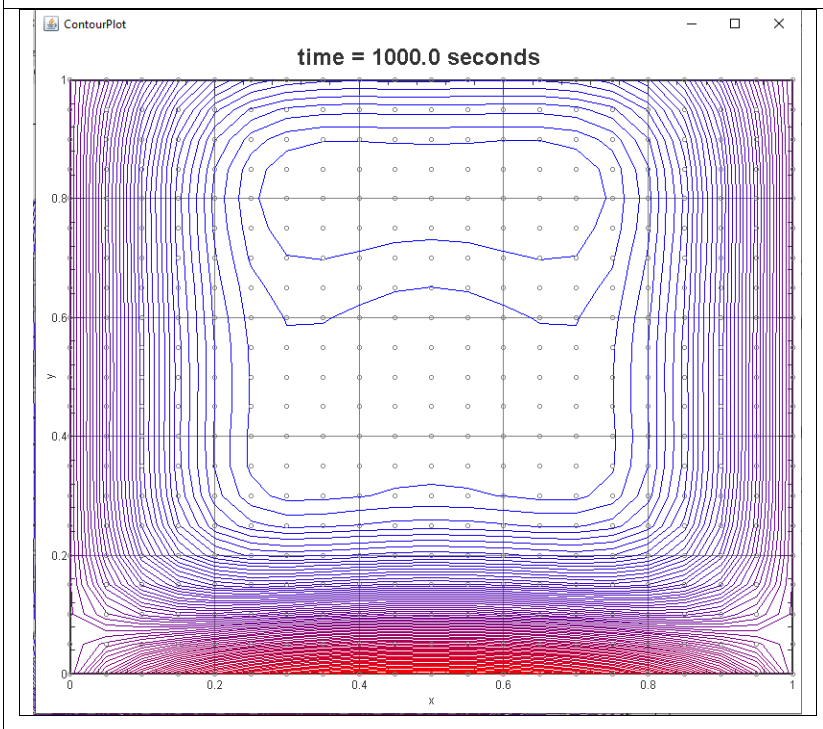


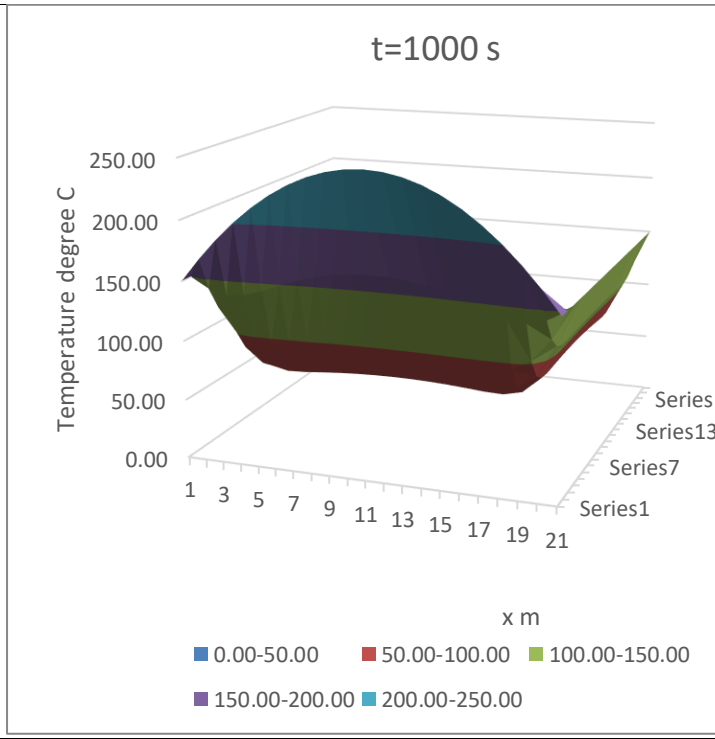
T=400 s

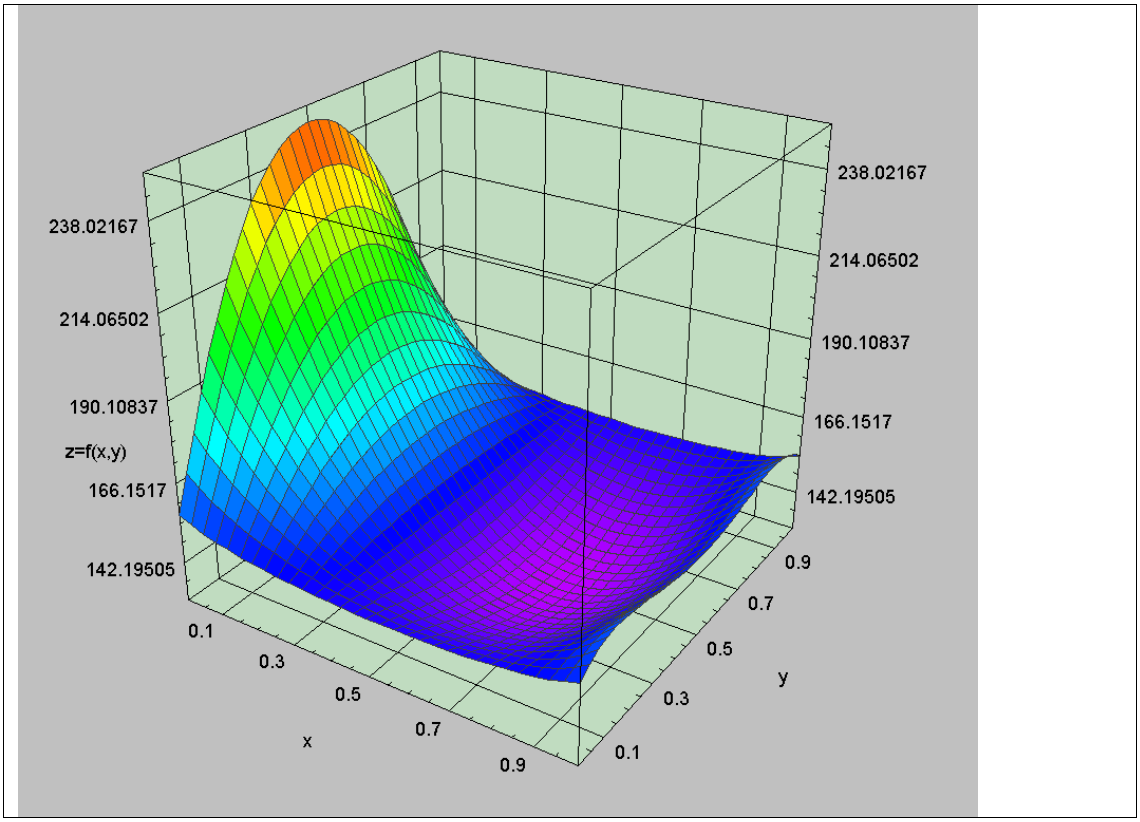
150.00	169.00	186.00	201.00	214.00	225.00	234.00	241.00	246.00	249.00	250.00	249.00	246.00	241.00	234.00	225.00	214.00	201.00	186.00	169.00	150.00
150.00	141.35	136.89	138.23	143.26	149.19	154.49	158.68	161.67	163.47	164.08	163.50	161.73	158.77	154.64	149.47	143.75	138.88	137.30	141.23	150.00
150.00	121.61	102.24	94.29	93.90	96.36	99.12	101.36	102.98	103.97	104.32	104.04	103.12	101.58	99.42	96.84	94.68	95.39	103.31	122.16	150.00
150.00	111.77	85.19	72.91	70.08	70.99	72.60	73.98	74.97	75.58	75.81	75.65	75.11	74.18	72.89	71.40	70.73	73.87	86.29	112.57	150.00
150.00	108.67	79.89	66.43	63.01	63.59	64.96	66.15	67.01	67.53	67.72	67.57	67.09	66.27	65.11	63.80	63.33	66.92	80.53	109.20	150.00
150.00	108.35	79.40	65.96	62.68	63.40	64.89	66.17	67.09	67.64	67.83	67.65	67.11	66.21	64.94	63.46	62.77	66.11	79.62	108.56	150.00
150.00	108.67	80.01	66.85	63.83	64.78	66.45	67.87	68.89	69.50	69.70	69.50	68.89	67.88	66.46	64.78	63.84	66.88	80.05	108.72	150.00
150.00	108.98	80.60	67.71	64.91	66.05	67.87	69.42	70.52	71.18	71.40	71.18	70.52	69.42	67.87	66.05	64.91	67.71	80.61	108.99	150.00
150.00	109.20	81.02	68.31	65.68	66.95	68.89	70.52	71.69	72.39	72.62	72.39	71.69	70.52	68.89	66.95	65.68	68.31	81.02	109.20	150.00
150.00	109.32	81.28	68.67	66.14	67.49	69.50	71.19	72.39	73.11	73.35	73.11	72.39	71.19	69.50	67.49	66.14	68.68	81.28	109.33	150.00
150.00	109.37	81.36	68.80	66.29	67.67	69.70	71.41	72.62	73.35	73.59	73.35	72.62	71.41	69.70	67.67	66.29	68.80	81.36	109.37	150.00
150.00	109.33	81.28	68.68	66.14	67.49	69.50	71.18	72.39	73.11	73.35	73.11	72.39	71.18	69.50	67.49	66.14	68.67	81.28	109.32	150.00
150.00	109.20	81.03	68.31	65.68	66.95	68.89	70.52	71.69	72.39	72.62	72.39	71.69	70.52	68.89	66.95	65.68	68.31	81.02	109.20	150.00
150.00	108.98	80.61	67.71	64.91	66.05	67.88	69.42	70.53	71.19	71.41	71.19	70.53	69.42	67.88	66.05	64.91	67.71	80.61	108.98	150.00
150.00	108.69	80.02	66.87	63.84	64.79	66.46	67.88	68.90	69.50	69.71	69.50	68.90	67.88	66.46	64.79	63.84	66.87	80.02	108.68	150.00
150.00	108.31	79.29	65.79	62.47	63.17	64.64	65.90	66.81	67.35	67.53	67.35	66.81	65.90	64.64	63.17	62.47	65.78	79.27	108.29	150.00
150.00	107.92	78.47	64.57	60.88	61.27	62.50	63.57	64.34	64.80	64.95	64.80	64.34	63.57	62.49	61.27	60.87	64.52	78.40	107.84	150.00
150.00	107.67	77.88	63.55	59.44	59.47	60.41	61.27	61.89	62.25	62.37	62.25	61.88	61.26	60.40	59.45	59.37	63.39	77.65	107.49	150.00
150.00	108.07	78.32	63.72	59.21	58.86	59.51	60.14	60.60	60.87	60.96	60.87	60.59	60.13	59.48	58.79	59.01	63.32	77.82	107.72	150.00
150.00	110.08	81.29	66.87	62.14	61.50	61.89	62.34	62.68	62.88	62.94	62.87	62.67	62.33	61.85	61.34	61.71	66.11	80.50	109.68	150.00
150.00	115.31	88.66	75.05	70.43	69.64	69.90	70.26	70.54	70.71	70.76	70.71	70.54	70.25	69.83	69.35	69.65	73.84	87.74	115.35	150.00


ContourPlot
time = 400.0 seconds
y
x


t=400 s
Temperature degree C
250.00
200.00
150.00
100.00
50.00
0.00
1 3 5 7 9 11 13 15 17 19 21
Series
Series13
Series7
Series1
x m
0.00-50.00
50.00-100.00
100.00-150.00
150.00-200.00
200.00-250.00


ContourPlot
time = 1000.0 seconds
y
x


t=1000 s
Temperature degree C
250.00
200.00
150.00
100.00
50.00
0.00
1 3 5 7 9 11 13 15 17 19 21
Series
Series13
Series7
Series1
x m
0.00-50.00
50.00-100.00
100.00-150.00
150.00-200.00
200.00-250.00



6.0 FURTHER ANALYTICAL SOLUTION METHODS:

6.1 DUHAMEL'S THEOREM

Consider the three dimensional, nonhomogeneous heat conduction problem in a region R with time dependent boundary condition function and heat generation in the form

$\nabla^2 T(r,t) + \frac{1}{k} g(r,t) = \frac{1}{\alpha}\frac{\partial T(r,t)}{\partial t}$ in region R t>0

$k_i \frac{\partial T(r,t)}{\partial n_i} + h_i T(r,t) = f_i(r,t)$ on boundary S_i

$T(r,t) = F(r)$ for t=0

This problem can not be solved with separation of variables methods described in previous problems because of the non-homegeneous terms g(r,t) and f_i(r,t). Therefore instead of solving this problem directly, problem will be expressed in terms of solution of a simpler auxilary problem as follows: Let $\Phi(r,t,\tau)$ be solution of the above problem on the assumption that the nonhomogeneous terms $g(r,\tau)$ and $f_i(r,\tau)$ do not depend on time; namely , the variable τ is merely a parameter but not a time variable. Then $\Phi(r,t,\tau)$ is the solution of the following auxilary problem

$\nabla^2 \Phi(r,t,\tau) + \frac{1}{k} g(r,\tau) = \frac{1}{\alpha}\frac{\partial \Phi(r,t,\tau)}{\partial t}$ in region R t>0

$k_i \frac{\partial \Phi(r,t,\tau)}{\partial n_i} + h_i \Phi(r,t,\tau) = f_i(r,\tau)$ on boundary S_i

$\Phi(r,t,\tau) = F(r)$ for t=0

This problem can be solved by using methods such as separation of variables. Then Duhamel's theorem relates the solution $\Phi(r,t,\tau)$ and $T(r,t)$ by the following integral equation

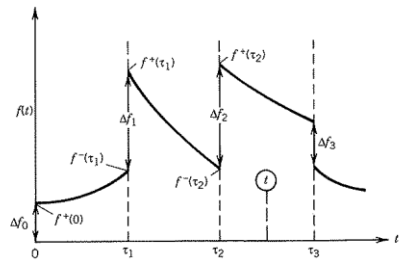
$T(r,t) = \frac{\partial}{\partial t}\int_{\tau=0}^{t} \Phi(r,t-\tau,\tau)\,d\tau$

This result can also be expressed in the alternative form by performing the differential. Since

$\Phi(r,t-\tau,\tau)|_{\tau=t} = \Phi(r,0,\tau) = F(r)$

$T(r,t) = F(r) + \int_{\tau=0}^{t} \frac{\partial}{\partial t}\Phi(r,t-\tau,\tau)\,d\tau$

If the time function has an F(r) function with discontinuities such as:



Equation becomes:

$T(r,t) = F(r) + \int_{\tau=0}^{t} \frac{\partial}{\partial t}\Phi(\mathrm{r,t\text{-}\tau,\tau})\mathrm{d}\tau + \sum_{j=0}^{N-1} \Phi(\mathrm{r,t\text{-}}\tau_j)\,\Delta f_j$

After these initial definitions, now A heat transfer problem can be investigated

A slab of thickness L is initially at zero temperature. For times t>0 boundary surface at x=0 kept at zero temperature, while the surface at x=L is subjected to a time varying temperature f(t) defined by

$$f(t) = \begin{cases} F(t) \ for\ 0 \le t < \tau_1 \\ 0 \ for \ t > \tau_1 \end{cases}$$

Equation

$\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T(x,t)}{\partial t}$ with boundary condition:

$T(x,t) = 0$ x=0 t>0

$T(x,t) = f(t)$ x=L t>0

$T(x,t) = 0 \quad t = 0 \ \ 0 \le x \le L$

Becomes

$\frac{\partial^2 \Phi(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial \Phi(x,t)}{\partial t}$

$\Phi(x,t) = 0$ x=0 t>0

$\Phi(x,t) = 1$ x=L t>0

$\Phi(x,t) = 0 \quad t = 0 \ \ 0 \le x \le L$

Problem divided into two part

$\Phi(x,t) = \Phi_H(x,t) + \Phi_{SS}(x)$

Where

$\frac{\partial^2 \Phi_{SS}(x)}{\partial x^2} = 0$

$\Phi_{SS}(x) = 0$ x=0

$\Phi_{SS}(x) = 1$ xL

$\frac{\partial \Phi_{SS}(x)}{\partial x} = C_1 \quad \Phi_{SS}(x) = C_1 x + C_2 \quad \Phi_{SS}(x) = 0$ x=0 ➔C_2=0

$\Phi_{SS}(x) = 1$ x=L ➔$C_1=\frac{1}{L}$

$\Phi_{SS}(x) = \frac{x}{L}$

$\frac{\partial^2 \Phi_H(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial \Phi_H(x,t)}{\partial t}$

$\Phi_H(x,t) = 0$ x=0 t>0

$\Phi_H(x,t) = 0$ x=L t>0

$\Phi_H(x,t) = -\Phi_{SS}(x) \quad 0 \le x \le L$ t=0

Separation of variables:

$\Phi_H(x,t) = X(x)*\Gamma(t)$

$\Gamma(t)\frac{\partial^2 X(x)}{\partial x^2} = \frac{X(x)}{\alpha}\frac{\partial \Gamma(t)}{\partial t}$

$\frac{1}{X(x)}\frac{\partial^2 X(x)}{\partial x^2} = \frac{1}{\alpha\Gamma(t)}\frac{\partial \Gamma(t)}{\partial t} = -\beta^2$

$\frac{\partial^2 X(x)}{\partial x^2} + \beta^2 X(x) = 0$

$\frac{\partial \Gamma(t)}{\partial t} + \beta^2 \alpha \Gamma(t) = 0$

Boundary conditions

$X(x) = 0$ x=0 t>0

$X(x) = 0$ x=L t>0

$X(x) = C_1 \cos(\beta x) + C_2 \sin(\beta x)$

$\Gamma(t) = C_3 \exp(-\beta^2 \alpha t)$

$\Phi_H(x,t) = (C_1 \cos(\beta x) + C_2 \sin(\beta x)) \exp(-\beta^2 \alpha t)$

Applications of boundary conditions:

$X(x) = C_1 \cos(\beta 0) + C_2 \sin(\beta 0) = 0$ $C_1=0$

$X(x) = C_2 \sin(\beta L) = 0$ $\beta L = m\pi$ $\beta_m = \frac{m\pi}{L}$

$\Phi_H(x,t) = \sum_{n=0}^{\infty} C_2 \sin(\beta x) \exp(-\beta^2 \alpha t)$

$\Phi_H(x,t) = -\Phi_{SS}(x)$ at t=0

$-\Phi_{SS}(x) = \sum_{n=0}^{\infty} C_n \sin(\beta_n x) \exp(-\beta_n^2 \alpha 0) = \sum_{n=0}^{\infty} C_n \sin(\beta_n x)$

$-\frac{x}{L} = \sum_{n=0}^{\infty} C_n \sin(\beta_n x)$ from ortogonality condition

$-\int_{x=0}^{L} \frac{x}{L} \sin(\beta_n x)\, dx = C_n \int_{x=0}^{L} \sin^2(\beta_n x)\, dx$

$\cos^2(\beta_n x) + \sin^2(\beta_n x) = 1$

$\cos^2(\beta_n x) - \sin^2(\beta_n x) = \cos(2\beta_n x)$ $\sin^2(\beta_n x) = \frac{1-\cos(2\beta_n x)}{2}$

$u = \frac{x}{L}$ $dv = sin(\beta_n x)dx$ $du = \frac{dx}{L}$ $v = -\frac{sin(\beta_n x)dx}{\beta_n}$

$\int udv = uv - \int vdu$

$\frac{x}{L}\left(\frac{1}{\beta_n}\cos(\beta_n x)\right)\Big|_{x=0}^{L} + \int_{x=0}^{L} \frac{1}{L\beta_n}\cos(\beta_n x)dx = C_n \int_{x=0}^{L} \frac{1-\cos(2\beta_n x)}{2} dx$

$\frac{L}{L}\left(\frac{1}{\beta_n}\cos(m\pi)\right) + \frac{L}{\beta_n^2}[\sin(\beta_n L) - \sin(\beta_n 0)] = C_n\left(\frac{L}{2} - \frac{sin(2\beta_n x)}{4\beta_n}\right)$

$C_n = \frac{2}{L\beta_n}(-1)^n$

$\Phi_H(x,t) = \frac{2}{L}\sum_{n=0}^{\infty} \frac{(-1)^n}{\beta_n} C_2 \sin(\beta x) \exp(-\beta^2 \alpha t)$

$\Phi(x,t) = \Phi_H(x,t) + \Phi_{SS}(x) = \frac{x}{L} + \frac{2}{L}\sum_{n=0}^{\infty} \frac{(-1)^n}{\beta_n} C_2 \sin(\beta x) \exp(-\beta^2 \alpha t)$

The function $\Phi(x, t-\tau)$is obtained by replacing t by $t-\tau$ in equation

$\Phi(x,t) = \frac{x}{L} + \frac{2}{L}\sum_{n=0}^{\infty} \frac{(-1)^n}{\beta_n} C_2 \sin(\beta x) \exp(-\beta^2 \alpha (t-\tau))$

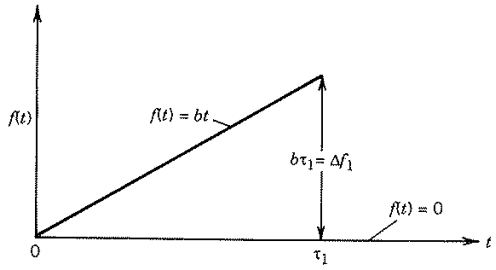
Duhamel theorem now can be applied to solve the problem. Remember that function to be considered was

$$f(t) = \begin{cases} f(t) \ for\ 0 \le t < \tau_1 \\ 0 \ for \ t > \tau_1 \end{cases}$$

For boundary condition T(x,t) = f(t) x=L t > 0

The solution for times $t < \tau_1$

$T(x,t) = \int_{\tau=0}^{t} \Phi(x, t-\tau) \frac{\partial f(\tau)}{\partial \tau} d\tau$ for a specific function $(t) = \begin{cases} F(t) = bt \ for\ 0 \le t < \tau_1 \\ 0 \ for \ t > \tau_1 \end{cases}$



$T(x,t) = \int_{\tau=0}^{t} \left(\frac{x}{L} + \frac{2}{L}\sum_{n=0}^{\infty} \frac{(-1)^n}{\beta_n} C_2 \sin(\beta_n x) * \exp(-\beta^2 \alpha (t-\tau))\right) b d\tau$

The integration is performed to give

$$T(x,t) = b\frac{xt}{L} + b\frac{2}{L}\sum_{n=0}^{\infty} \frac{(-1)^n}{\alpha\beta_n^3} \sin(\beta_n x) * [1 - \exp(-\alpha\beta_n^2 t)]$$

The solution for times $t > \tau_1$

Java solution for this problem is given as follows:

```
 import java.util.*;
class fc extends f_xxi
{
public double func(double t,int i)
{
  double ff=0;
  if(i==0)
  {
              if(t<=10)ff=2.0*t;
              df=20.0;
  }
  else     ff=0.0;
  return ff;
}
}

public class HT_duhamel1
{

//integration function
public static double intfr(double to,double t,double x,double L,double alpha)
{ double fi=x/L;
  double bm=0;
  int plusminus=-1;
  for(int m=1;m<20;m++)
  {  bm=m*Math.PI/L;
               fi+=2.0/L*Math.exp(-alpha*bm*bm*(t-to))*plusminus/bm*Math.sin(bm*x);
               plusminus*=-1;
  }
              //double T=x/Math.sqrt(4.0*Math.PI*alpha)*f(to)/Math.pow((t-to),1.5)*Math.exp(-x*x/(4.0*alpha*(t-to)));
              return fi;
}
// analytic solution function
public static double intfr1(double t,double x,double L,double alpha)
{ double fi=2*x/L*t;
  double bm=0;
  int plusminus=-1;
  for(int m=1;m<20;m++)
  {  bm=m*Math.PI/L;
               fi+=4.0/L*plusminus/(bm*bm*bm*alpha)*(1.0-Math.exp(-alpha*bm*bm*t))*Math.sin(bm*x);
               plusminus*=-1;
  }
              //double T=x/Math.sqrt(4.0*Math.PI*alpha)*f(to)/Math.pow((t-to),1.5)*Math.exp(-x*x/(4.0*alpha*(t-to)));
              return fi;
}
//integral
//Gauss Kronrod integration formula
public static double integral(f_xxi f,int ii,double a,double b,double t,double x,double L,double alpha)
   {
                //eps hata miktari
                int n=50;
                double eps=1e-10;
      int Aused=0;
                boolean sonuc;
      double integral=0;
      double A[][];
      int Aw = 0;
      double toplamhata = 0;
      double[] c;
      double[] xg;
      double[] xk;
      int nn = 0;
      int ng = 0;
      int i = 0;
      int j = 0;
      int h = 0;
      double v = 0;
      double k1 = 0;
      double k2 = 0;
      double intg = 0;
      double intk = 0;
      double ta = 0;
      double tb = 0;
```

```
Aw = 4;
A = new double[n-1+1][Aw-1+1];
nn = 61;
ng = 15;
c = new double[nn];
xk = new double[nn];
xg = new double[nn];
//Gauss katsayilari
xg[0] = 0.00796819249616660561546588347467 4;
xg[1] = 0.01846646831109095914230213191204 7;
xg[2] = 0.02878470788332336934971917961129 2;
xg[3] = 0.03879919256962704959680193644634 8;
xg[4] = 0.04840267283059405290293814042280 8;
xg[5] = 0.05749315621761906648172168940205 6;
xg[6] = 0.06597422988218049512812851511596 2;
xg[7] = 0.07375597473770520626824385002219 1;
xg[8] = 0.08075589522942021535469493846053 0;
xg[9] = 0.08689978720108297980238753071512 6;
xg[10] = 0.09212252223778612871763270708761 9;
xg[11] = 0.09636873717464425963946862635181 0;
xg[12] = 0.09959342058679526706278028210356 9;
xg[13] = 0.10176238974840550459642895216855 4;
xg[14] = 0.10285265289355884034128563670541 5;
c[0] = 0.99948441005049063757132589570581 1;
c[1] = 0.99689348407464954027163005091869 5;
c[2] = 0.99163099687040459485862836610948 6;
c[3] = 0.98366812327974720997003258160566 3;
c[4] = 0.97311632250112626837469386842370 7;
c[5] = 0.96002186496830751221687102558179 8;
c[6] = 0.94437444474855997941583132403743 9;
c[7] = 0.92620004742927432587932427708047 4;
c[8] = 0.90557330769990779854652255892595 8;
c[9] = 0.88256053579205268154311646253022 6;
c[10] = 0.85720523354606109895865851065894 4;
c[11] = 0.82956576238276839744289811973250 2;
c[12] = 0.79972783582183908301366894232268 3;
c[13] = 0.76777743210482619491797734097450 3;
c[14] = 0.73379006245322680472617113136952 8;
c[15] = 0.69785049479331579693229238802664 0;
c[16] = 0.66006106412662696137005366814927 1;
c[17] = 0.62052618298924286114047755643118 9;
c[18] = 0.57934523582636169175602493217254 0;
c[19] = 0.53662414814201989926416979331107 3;
c[20] = 0.49248046786177857499369306120770 9;
c[21] = 0.44703376953808917678060990032285 4;
c[22] = 0.40040125483039439253547621154266 1;
c[23] = 0.35270472553087811347103720708937 4;
c[24] = 0.30407320227362507737267710719925 7;
c[25] = 0.25463692616788984643980512981780 5;
c[26] = 0.20452511668230989143895767100202 5;
c[27] = 0.15386991360858354696379467274325 6;
c[28] = 0.10280693796673703014709675131800 1;
c[29] = 0.05147184255531769583302521316672 3;
c[30] = 0.00000000000000000000000000000000 0;
//Kronrod katsayilari
xk[0] = 0.00138901369867700762455159122676 0;
xk[1] = 0.00389046112709988405126720184451 6;
xk[2] = 0.00663070391593129217331982636975 0;
xk[3] = 0.00927327965951776342844114689202 4;
xk[4] = 0.01182301525349634174223289885325 1;
xk[5] = 0.01436972950704580481245143244358 0;
xk[6] = 0.01692088918905327262757228942032 2;
xk[7] = 0.01941414119394238117340895105012 8;
xk[8] = 0.02182803582160919229716748573833 9;
xk[9] = 0.02419116207808060136568637072523 2;
xk[10] = 0.02650995488233310161060170933507 5;
xk[11] = 0.02875404876504129284397878535433 4;
xk[12] = 0.03090725756238776247288425294309 2;
xk[13] = 0.03298144705748372603181419101685 4;
xk[14] = 0.03497933802806002417349967073146 8;
xk[15] = 0.03688236465182122922391106561713 6;
xk[16] = 0.03867894562472759295034865153228 1;
xk[17] = 0.04037453895153595911199527975246 8;
xk[18] = 0.04196981021516424614714754128597 0;
xk[19] = 0.04345253970135606931683172811707 3;
xk[20] = 0.04481480013316266319235555161672 3;
xk[21] = 0.04605923827100698811627173555937 4;
```

```
xk[22] = 0.047185546569299153945261478181099;
xk[23] = 0.048185861757087129140779492298305;
xk[24] = 0.049055434555029778887528165367238;
xk[25] = 0.049795683427074206357811569379942;
xk[26] = 0.050405921402782346840893085653585;
xk[27] = 0.050881795898749606492297473049805;
xk[28] = 0.051221547849258772170656282604944;
xk[29] = 0.051426128537459025933862879215781;
xk[30] = 0.051494729429451567558340433647099;
for(i=nn-1; i>=nn/2; i--)
{c[i] = -c[nn-1-i];}
for(i=nn-1; i>=nn/2; i--)
{ xk[i] = xk[nn-1-i]; }
for(i=ng-1; i>=0; i--)
{ xg[nn-2-2*i] = xg[i];
  xg[1+2*i] = xg[i];
}
for(i=0; i<=nn/2; i++)
{ xg[2*i] = 0; }
k1 = 0.5*(b-a);
k2 = 0.5*(b+a);
intg = 0;
intk = 0;
for(i=0; i<=nn-1; i++)
{  double tt=(k1*c[i]+k2);
                v = intfr(tt,t,x,L,alpha)*f.dfunc(tt,ii);
  intk = intk+v*xk[i];
  if( i%2==1 )
  {intg = intg+v*xg[i];}
}
intk = intk*(b-a)*0.5;
intg = intg*(b-a)*0.5;
A[0][0] = Math.abs(intg-intk);
A[0][1] = intk;
A[0][2] = a;
A[0][3] = b;
toplamhata = A[0][0];
if( toplamhata<eps )
{
   sonuc = true;
   integral = intk;
   Aused = 1;
   return integral;
}
Aused = 1;
for(h=1; h<=n-1; h++)
{
   Aused = h+1;
   gir(A, h, Aw);
   toplamhata = toplamhata-A[h-1][0];
   ta = A[h-1][2];
   tb = A[h-1][3];
   A[h-1][2] = ta;
   A[h-1][3] = 0.5*(ta+tb);
   A[h][2] = 0.5*(ta+tb);
   A[h][3] = tb;
   for(j=h-1; j<=h; j++)
   {
      k1 = 0.5*(A[j][3]-A[j][2]);
      k2 = 0.5*(A[j][3]+A[j][2]);
      intg = 0;
      intk = 0;
      for(i=0; i<=nn-1; i++)
      {   double tt=(k1*c[i]+k2);
                  v = intfr(tt,t,x,L,alpha)*f.dfunc(tt,ii);
         intk = intk+v*xk[i];
         if( i%2==1 )
         { intg = intg+v*xg[i];}
      }
      intk = intk*(A[j][3]-A[j][2])*0.5;
      intg = intg*(A[j][3]-A[j][2])*0.5;
      A[j][0] = Math.abs(intg-intk);
      A[j][1] = intk;
      toplamhata = toplamhata+A[j][0];
   }
   cik(A, h-1, Aw);
   cik(A, h, Aw);
```

```
        if( toplamhata<eps )
        {break;}
    }
    sonuc = toplamhata<eps;
    integral = 0;
    for(j=0; j<=Aused-1; j++)
    { integral = integral+A[j][1];}
    return integral;
}
public static void gir(double A[][],int n,int Awidth)
{
    int i = 0;
    int p = 0;
    double t = 0;
    int maxcp = 0;
    if( n==1 )
    { return; }
    for(i=0; i<=Awidth-1; i++)
    {   t = A[n-1][i];
        A[n-1][i] = A[0][i];
        A[0][i] = t;
    }//end of for
    p = 0;
    while( 2*p+1<n-1 )
    {   maxcp = 2*p+1;
        if( 2*p+2<n-1 )
        {   if( A[2*p+2][0]>A[2*p+1][0] )
            { maxcp = 2*p+2; }
        }
        if( A[p][0]<A[maxcp][0] )
        {   for(i=0; i<=Awidth-1; i++)
            {   t = A[p][i];
                A[p][i] = A[maxcp][i];
                A[maxcp][i] = t;
            }
            p = maxcp;
        }
        else
        { break; }
    } //end of while
}

public static void cik(double A[][],int n,int Awidth)
{
    int i = 0;
    int p = 0;
    double t = 0;
    int kk = 0;
    if( n==0 )
    {return;}
    p = n;
    while( p!=0 )
    {   kk = (p-1)/2;
        if( A[p][0]>A[kk][0] )
        { for(i=0; i<=Awidth-1; i++)
            { t = A[p][i];
              A[p][i] = A[kk][i];
              A[kk][i] = t;
            }
            p = kk;
        }
        else
        {break;}
    }
}
public static double Tplus(double T1,double x,double t,double L,double alpha)
{ double to1=9.0;
  double to2=20.0;
  ////to>to1
  fc f=new fc();
              double TT=T1+integral(f,0,0.0,to1,t,x,L,alpha)+
              integral(f,0,to1,to2,t,x,L,alpha)+
              intfr(to1,t,x,L,alpha)*f.df;
              return TT;
             }
             public static double T(double T1,double x,double to,double t,double L,double alpha)
{ double to1=9.0;
```

```
      double to2=20.0;
      //to<to1
      fc f=new fc();
                  double TT=T1+integral(f,0,0.0,to,t,x,L,alpha);
                  return TT;
                }
     public static void main (String args[])
     {
       double alpha=4.0e-5;
       double x=0.4; //m
       double t=9.0;//s
       double L=0.5; //m
       double T1=20.0;// degree C;
       double T=T(T1,x,t,t,L,alpha);
       double T2=T1+intfr1(t,x,L,alpha);
       double T3=Tplus(T1,x,t,L,alpha);
       System.out.println("T="+T+"T2="+T2);
       System.out.println("T3="+T3);
     }
}
```

```
import static java.lang.Math.*;

abstract  class f_xxi extends Mathd
{
  // single function single independent variable
  // example f=x*x
  // Reference :"Generation of Finite Difference Formulas on Arbitrary Spaced Grids",
  // Bength Fornberg, Mathematics of Computation, Volume 51, Number 184, October 1988
  // pages 699-706
double df;
double c[][][];
double alpha[];
double h;
int M;
public double a[];
//constructors
   public f_xxi(double ai[])
   {setA(ai);
   derivative_coef();
   df=0;
   }

   public f_xxi()
   { double ai[]=new double[0];
                setA(ai);
                derivative_coef();
                df=0;
   }

  public void setA(double ai[])
  {int n=ai.length;
   a=new double[n];
   for(int i=0;i<n;i++)
   {a[i]=ai[i];}
  }
// abstract function func
  abstract public double func(double x,int i);

// 1st order derivative

public double dfunc(double x,int i)
{double h=1.0e-3;
 int n=1;
 return dfunc(x,i,n,h);
}
// nth order derivative
public double dfunc(double x,int i,int n)
{double h=1.0e-2;
 return dfunc(x,i,n,h);
}

double dfunc(double x,int k,int N,double hi)
{// order of the maximum derivative
 // N order of derivative
  setH(hi);
```

```
  double deriv=0;
  double h1=1/h;
  double h2=1;
  for(int j=0;j<N;j++)
  {h2*=h1;}
  for(int i=0;i<c[0].length;i++)
  {deriv+=c[M][i][N]*func(x+alpha[i]*h,k);}
  return deriv*h2;
}


public void setH(double hi)
{h=hi;}
public void coef_out(int N)
{  double a[][]=new double[alpha.length][2];
   for(int i=0;i<alpha.length;i++)
   {a[i][0]=alpha[i];
    a[i][1]=c[M][i][N];
   }
   String s1="Derivative coefficients for order of the highest derivative M = "+M;
   String s[]=new String[2];
   s[0]=""+'\u0394'+"x";
   if(N==1){s[1]="df(x)/dx";}
   else if(N==2) {s[1]="d"+'\u00B2'+"f(x)/dx"+'\u00B2';}
   else if(N==3) {s[1]="d"+'\u00B3'+"f(x)/dx"+'\u00B3';}
   else  {s[1]="d("+N+")f(x)/dx"+N;}
              Text.print(a,s,s1);
}
public void coef_out()
{  double a[][]=new double[alpha.length][M+1];
   String s[]=new String[M+1];
   s[0]=""+'\u0394'+"x";
   for(int j=1;j<=M;j++)
   {
   if(j==1){s[1]="df(x)/dx";}
   else if(j==2) {s[j]="d"+'\u00B2'+"f(x)/dx"+'\u00B2';}
   else if(j==3) {s[j]="d"+'\u00B3'+"f(x)/dx"+'\u00B3';}
   else  {s[j]="d("+j+")f(x)/dx"+j;}
   }
   for(int i=0;i<alpha.length;i++)
   {a[i][0]=alpha[i];
    for(int j=1;j<=M;j++)
    a[i][j]=c[M][i][j];
   }
    String s1="Derivative coefficients for order of the highest derivative M = "+M;
              Text.print(a,s,s1);
}
public void setOrder(int Mi)
{M=Mi;}

public  void derivative_coef()
{
h=0.0001;
double x0=0;
M=60;
double alphai[]={0,1,-1,2,-2,3,-3,4,-4,5,-5,6,-6,7,-7,8,-8,9,-9,10,-10,11,-11,12,-12,13,-13,14,-14,15,
-15,16,-16,17,-17,18,-18,19,-19,20,-20,21,-21,22,-22,23,-23,24,-24,25,-25,26,-26,27,-27,28,-28,29,-29,30,-30,31,-31,32,-32,33,-33,34,-
34,35,-35,36,-36,37,-37,38,-38,39,-39,30,-40,41,-41};
alpha=alphai;
int N=alpha.length-1;
// M   degree of highest derivative
// N+1 number of coefficients
double delta[][][]=new double[N+1][N+1][M+1];
double c1,c2,c3;
delta[0][0][0]=1.0;
c1=1.0;
//for(int i=0;i<alpha.length;i++)
//{System.out.println(i+" "+alpha[i]);}
for(int n=1;n<=N;n++)
{  c2=1;
   for(int nu=0;nu<=(n-1);nu++)
   {c3=alpha[n]-alpha[nu];
    c2=c2*c3;
    if(n<=M) delta[n-1][nu][n]=0.0;
    for(int m=0;m<=Math.min(n,M);m++)
    {   //System.out.println("n="+n+"m="+m+"nu="+nu);
       if(m==0)
```

```
      { delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m])/c3;}
      else
      {delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m]-m*delta[n-1][nu][m-1])/c3;}
   }//next m
  }//next nu
  for(int m=0;m<=Math.min(n,M);m++)
  {  if(m==0)
               {delta[n][n][m]=c1/c2*(-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
               else
               {delta[n][n][m]=c1/c2*(m*delta[n-1][n-1][m-1]-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
  }//next m
  c1=c2;
}//next n
c=delta;
}

public double dfunc_n(double x,int i,int n,double h)
{
double hh=1/h;
double df=0;
if(n==0) df=func(x,i);
else if(n==1)
{df=(3.0*func(x-4.0*h,i)-32.0*func(x-3.0*h,i)+168.0*func(x-2.0*h,i)-672.0*func(x-h,i)+672.0*func(x+h,i)-
168.0*func(x+2.0*h,i)+32.0*func(x+3.0*h,i)-3.0*func(x+4.0*h,i))/840.0*hh;}
else if(n==2)
{df=(-14350.0*func(x,i)-9.0*func(x-4*h,i)+128*func(x-3*h,i)-1008*func(x-2*h,i)+8064*func(x-h,i)+8064.0*func(x+h,i)-
1008.0*func(x+2.0*h,i)+128.0*func(x+3.0*h,i)-9.0*func(x+4.0*h,i))/5040.0*hh*hh;}
else if(n==3)
{df=(-7.0*func(x-4.0*h,i)+72.0*func(x-3.0*h,i)-338.0*func(x-2.0*h,i)+488.0*func(x-h,i)-488.0*func(x+h,i)+338.0*func(x+2.0*h,i)-
72.0*func(x+3.0*h,i)+7.0*func(x+4.0*h,i))/240.0*hh*hh*hh;}
else if(n==4)
{df=(2730.0*func(x,i)+7.0*func(x-4.0*h,i)-96.0*func(x-3.0*h,i)+676.0*func(x-2*h,i)-1952.0*func(x-h,i)-
1952.0*func(x+h,i)+676.0*func(x+2.0*h,i)-96.0*func(x+3.0*h,i)+7.0*func(x+4.0*h,i))/240.0*hh*hh*hh*hh;}
else if(n==5)
{df=(func(x-4.0*h,i)-9.0*func(x-3.0*h,i)+26.0*func(x-2.0*h,i)-29.0*func(x-h,i)+29.0*func(x+h,i)-
26.0*func(x+2.0*h,i)+9.0*func(x+3.0*h,i)-func(x+4.0*h,i))/6.0*hh*hh*hh*hh*hh;}
else if(n==6)
{df=(-150.0*func(x,i)-func(x-4.0*h,i)+12.0*func(x-3.0*h,i)-52.0*func(x-2.0*h,i)+116.0*func(x-h,i)+116.0*func(x+h,i)-
52.0*func(x+2.0*h,i)+12.0*func(x+3.0*h,i)-func(x+4.0*h,i))/4.0*hh*hh*hh*hh*hh*hh;}
else if(n==7)
{df=(-func(x-4.0*h,i)+6.0*func(x-3.0*h,i)-14.0*func(x-2.0*h,i)+14.0*func(x-h,i)-14.0*func(x+h,i)+14.0*func(x+2.0*h,i)-
6.0*func(x+3.0*h,i)+func(x+4.0*h,i))/2.0*hh*hh*hh*hh*hh*hh*hh;}
else if(n==8)
{df=(70.0*func(x,i)+func(x-4.0*h,i)-8.0*func(x-3.0*h,i)+28.0*func(x-2.0*h,i)-56.0*func(x-h,i)-56.0*func(x+h,i)+28.0*func(x+2.0*h,i)-
8.0*func(x+3.0*h,i)+func(x+4.0*h,i))*hh*hh*hh*hh*hh*hh*hh*hh;}
else df=0;
return df;
}

}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" HT_duhamel1
T=20.144848295310858T2=20.14484829531157
T3=19.428225405496892

> Terminated with exit code 0.
```

6.2 GREEN FUNCTIONS

Consider the following three-dimensional nonhomogeneous boundary value problem of heat conduction:

$\nabla^2 T(r,t) + \frac{1}{k} g(r,t) = \frac{1}{\alpha} \frac{\partial T(r,t)}{\partial t}$ in region R t>0

With boundary conditions

$k_i \frac{\partial T(r,t)}{\partial n_i} + h_i T(r,t) i = f_i(r,t)$ on S_i t>0

T(r,t)=F(r) for t=0

Where $\frac{\partial T(r,t)}{\partial n_i}$ denotes differentiation along the outward-drawn normal to the boundary surface S_i, i=1,2,3..N

N is the continous boundary surfaces in the region. For generality it is assumed that the generation term g(r,t) and boundary condition function $f_i(r,t)$ vary both position and time. In this equation k_i and h_i considered to be constant. To solve the problem we will considered the following auxilary problem for the same region R

$\nabla^2 G(r,t|r',\tau) + \frac{1}{k}\delta(r-r')\delta(t-\tau) = \frac{1}{\alpha}\frac{\partial G(r,t|r',\tau)}{\partial t}$ in region R t>τ

With boundary conditions

$k_i \frac{\partial G(r,t|r',\tau)}{\partial n_i} + h_i G(r,t|r',\tau) = 0$ on S_i t>0

In this equation G is called Green's function. Obeying the casuality requirements the Green's function G be zero for $t < \tau$. The source in equation is a unit impulsive source for the three-dimensional problem considered here. The delta function , $\delta(r-r')$ represent a point heat source located at r', while the $\delta(t-\tau)$ indicates that its an intantaneous heat source releasing its energy spontaneously at time t=τ. In the case of three-dimensional transient non-homogeneous heat conduction problem given by the equation above T(r,t) is expressed in terms of three-dimensional Green's function $G(r,t|r',\tau)$ as

Where R refers to entire volume of the region considered where S_i refers boundary surface of the region R, i=1,2,3..N. dv' and ds' refers differential volume and surface elements. If the same equation is written for two dimensional problem:

$$T(r,t) = \int_R G(r,t|r',\tau)|_{\tau=0} F(r')dA' + \frac{\alpha}{k}\int_{\tau=0}^{t} d\tau \int_R G(r,t|r',\tau)g(r',\tau)dA' + \alpha \int_{\tau=0}^{t} d\tau \sum_{i=1}^{N} \int_{l_i} G(r,t|r',\tau)|_{r'=r_i} \frac{1}{k_i} f_i dl_i'$$

Where A is the area under consideration, l_i is the length of the boundary i for the region A

And for one dimensional problem:

$$T(x,t) = \int_R x'^P G(x,t|x',\tau)|_{\tau=0} F(x')dx' + \frac{\alpha}{k}\int_{\tau=0}^{t} d\tau \int_R x'^P G(x,t|x',\tau)g(x',\tau)dx' + \alpha \int_{\tau=0}^{t} d\tau \sum_{i=1}^{2} x'^P G(x,t|x',\tau)|_{x'=x_i} \frac{1}{k_i} f_i$$

Where x'^P is the Sturm-Liouville weight function such that

$P = \begin{Bmatrix} 0 & slab \\ 1 & cylinder \\ 2 & sphere \end{Bmatrix}$ Here L refers to the thickness or radius of one dimensional region.

It should be note that these equations are given for the boundary conditions of the third kind i.e.

$k_i \frac{\partial T(r,t)}{\partial n_i} + h_i T(r,t) i = f_i(r,t)$ on S_i t>0

If the boundary condition is heat flux (boundary condition of the second kind)

$k_i \frac{\partial T(r,t)}{\partial n_i} = f_i(r,t)$ equation will remain the same but $f_i(r,t)$ indicates the heat flux

If the boundary condition is the temperature (boundary condition of the first kind)

In this case $k_i = 0 \; and \; h_i = 1$ should be the desired conditions, but considering $\frac{G}{k_i}$ term in the equation will go the infinity. Therefore and equivalent term will be used and $\frac{G}{k_i}$ will be replaced by $-\frac{1}{h_i}\frac{\partial G}{\partial n_i}$ also considering condition that $h_i = 1$, it will be $-\frac{\partial G}{\partial n_i}$ instead of $\frac{G}{k_i}$ in the equation.

In order to solve the problem by using Green function approach requires to find the green function representation. To achive that we can use the homogeneous part solution of the equation. Suppose solution of the homogeneous part of a heat transfer problem is in the form of:

$T(r,t) = \int_R K(r,r',\tau)F(r')dv'$ and General green function solution of the first part is given as

$T(r,t) = \int_R G(r,t|r',t)|_{t=0} F(r')dv'$ That implies that

$G(r,t|r',t)|_{t=0} = K(r,r',t)$

Let us consider an example now:

A slab, $0 \le x \le L$, is initially at temperature F(x). For times t>0 the boundaries at x=0 and x=l are maintained at temperatures $f_1(t)$ and $f_2(t)$ respectively, whereas heat is generated in the medium at a rate of g(x,t) W/m^3. Obtain an expression and solve the equation

$\frac{\partial^2 T(x,t)}{\partial x^2} + \frac{1}{k} g(x,t) = \frac{1}{\alpha}\frac{\partial T(x,t)}{\partial t}$ in $0 \le x \le L$ t>0

$T(x,t) = 0$ at x=0 t>0

$T(x,t) = 0$ at x=L t>0

$T(x,t) = F(x)$ at $0 \le x \le L$ t=0

To determine appropriate Green's function, we consider the homogeneous version of this problem as

$\frac{\partial^2 \varphi(x,t)}{\partial x^2} + \frac{1}{k} g(x,t) = \frac{1}{\alpha}\frac{\partial \varphi(x,t)}{\partial t}$ in $0 \le x \le L$ t>0

$\varphi(x,t) = f_1(t)$ at x=0 t>0

$\varphi(x,t) = f_2(t)$ at x=L t>0

$\varphi(x,t) = F(x)$ at $0 \le x \le L$ t=0

Separation of variables:

$\varphi(x,t) = X(x)\Gamma(t)$

$\Gamma(t)\frac{\partial^2 X(x)}{\partial x^2} = \frac{X(x)}{\alpha}\frac{\partial \Gamma(t)}{\partial t}$

$\frac{1}{X(x)}\frac{\partial^2 X(x)}{\partial x^2} = \frac{1}{\alpha\Gamma(t)}\frac{\partial \Gamma(t)}{\partial t} = -\beta^2$

$\frac{\partial^2 X(x)}{\partial x^2} + \beta^2 X(x) = 0$

$\frac{\partial \Gamma(t)}{\partial t} + \beta^2 \alpha \Gamma(t) = 0$

Boundary conditions

$X(x) = 0$ x=0 t>0

$X(x) = 0$ x=L t>0

$X(x) = C_1 \cos(\beta x) + C_2 \sin(\beta x)$

$\Gamma(t) = C_3 \exp(-\beta^2 \alpha t)$

$\varphi(x,t) = [C_1 \cos(\beta x) + C_2 \sin(\beta x)]\exp(-\beta^2 \alpha t)$

Applications of boundary conditions:

$X(x) = C_1 \cos(\beta 0) + C_2 \sin(\beta 0) = 0$ $C_1=0$

$X(x) = C_1 \sin(\beta L) = 0$

$\varphi(x,t) = [C_1 \cos(\beta x) + C_2 \sin(\beta x)]\exp(-\beta^2 \alpha t)$

$\varphi_H(x,t) = \sum_{n=0}^{\infty} C_n \sin(\beta_n x) \exp(-\beta_n^2 \alpha t)$

t=0 $\varphi_H(x,t) = 0$ $\sin(\beta_n x) = 0$

$\beta_n = \frac{n\pi}{L}$

$\psi(x,t) = F(x)$ in $0 \le x \le L$ for t=0

$$F(x) = \sum_{n=0}^{\infty} C_n \sin(\beta_m x)$$

from ortogonality condition

$$\int_{x=0}^{L} F(x)\sin(\beta_m x)dx = C_m \int_{x=0}^{L} \sin^2(\beta_m x)dx$$

$$\cos^2(\beta_m x) + \sin^2(\beta_m x) = 1$$

$$\cos^2(\beta_m x) - \sin^2(\beta_m x) = \cos(2\beta_m x) \rightarrow \sin^2(\beta_m x) = \frac{1-\cos(2\beta_m x)}{2}$$

$$\int_{x=0}^{L} F(x)\sin(\beta_m x)dx = C_m \int_{x=0}^{L} \frac{1-\cos(2\beta_m x)}{2}dx$$

$$\int_{x=0}^{L} F(x)\sin(\beta_m x)dx = C_m\left(\frac{L}{2} - \frac{\sin(2\beta_m x)}{4\beta_m}\right) = C_m\left(\frac{L}{2} - \frac{\sin(2m\pi)}{4\beta_m}\right) = C_m\frac{L}{2}$$

$$C_m = \frac{2}{L}\int_{x=0}^{L} F(x)\sin(\beta_m x)dx$$

$$\psi(x,t) = \left[\frac{2}{L}\sum_{m=1}^{\infty}\sin(\beta_m x) * \exp(-\beta_m^2 \alpha t)\right]\int_{x'=0}^{L}\sin(\beta_m x')F(x')dx'$$

$$\psi(x,t) = \int_{x'=0}^{L}\left[\frac{2}{L}\sum_{m=1}^{\infty}\sin(\beta_m x) * \exp(-\beta_m^2 \alpha t)\right]\sin(\beta_m x')F(x')dx' \text{ where } \beta_m = \frac{m\pi}{L} \text{ m=1,2,3,..}$$

the solution of problem in terms of Green's function is given as:

$$T(x,t) = \int_L G(x,t|x',\tau)\Big|_{\tau=0} F(x')dx' \text{ A comparison of the equations yields}$$

$$G(x,t|x',\tau)\big|_{\tau=0} = \frac{2}{L}\sum_{n=0}^{\infty}\sin(\beta_m x)*\sin(\beta_m x')\exp(-\beta_m^2\alpha t)$$

The desired Green function is obtained by replacing t with $t-\tau$ in the equation so that

$$G(x,t|x',\tau) = \frac{2}{L}\sum_{n=0}^{\infty}\sin(\beta_m x)*\sin(\beta_m x')\exp\left[-\beta_m^2\alpha(t-\tau)\right]$$

Then the solution of the actual problem is given in terms of Green function will be

$$T(x,t) = \int_L G(x,t|x',\tau)\Big|_{\tau=0} F(x')dx' +$$

$$\frac{\alpha}{k}\int_{\tau=0}^{t} d\tau \int_L x'^{P}\, G(x,t|x',\tau)g(x',\tau)dx' + \alpha\int_{\tau=0}^{t}\frac{\partial G(x,t|x',\tau)}{\partial x'}\bigg|_{x'=0} f_1(\tau)d\tau - \int_{\tau=0}^{t}\frac{\partial G(x,t|x',\tau)}{\partial x'}\bigg|_{x'=L} f_2(\tau)d\tau$$

It should be note that in the original problem given boundary condition was second(derivative) kind. In this equation it is both the first kind(direct equation), therefore $G(x,t|x',\tau)\big|_{x'=0}$ replaced with $\left.\frac{\partial G(x,t|x',\tau)}{\partial x'}\right|_{x'=0}$ for terms involving $f_1(\tau)$ and $G(x,t|x',\tau)\big|_{x'=L}$ replaced with $\left.\frac{\partial G(x,t|x',\tau)}{\partial x'}\right|_{x'=L}$ for terms involving $f_2(\tau)$.

Introducing Green's function solution for these equation will become:

$$T(x,t) = \left[\frac{2}{L}\sum_{m=1}^{\infty}\sin(\beta_m x)*\exp(-\beta_m^2\alpha t)\right]\int_{x'=0}^{L}\sin(\beta_m x')F(x')dx'$$

$$+\frac{\alpha}{k}\left[\frac{2}{L}\sum_{m=1}^{\infty}\sin(\beta_m x)*\exp(-\beta_m^2\alpha t)\right]\int_{\tau=0}^{t}\exp(-\beta_m^2\alpha\tau)d\tau\int_{x'=0}^{L}\sin(\beta_m x')g(x')dx'$$

$$+\alpha\left[\frac{2}{L}\sum_{m=1}^{\infty}\beta_m*\sin(\beta_m x)*\exp(-\beta_m^2\alpha t)\right]\int_{\tau=0}^{t}\exp(-\beta_m^2\alpha\tau)f_1(\tau)d\tau$$

$$-\alpha\left[\frac{2}{L}\sum_{m=1}^{\infty}(-1)^m\beta_m\sin(\beta_m x)*\exp(-\beta_m^2\alpha t)\right]\int_{\tau=0}^{t}\exp(-\beta_m^2\alpha\tau)f_2(\tau)d\tau$$

Numerical example: The following internal and external Wall temperature profiles as a function of time and initial wall temperature profile as a function of distance is given. Wall thisckness is L=0.2 m and thermal diffusivity coefficient is $\alpha=5.06\text{x}10^{-7}$ m^2/s (Brick Wall, source : indian standard 3792-1978) . Heat is generated in in slub with a constant rate q=1000 W/m^3. Find the temperature profile

t hours	t seconds	f1(t)	f2(t)
0	0	0	20
1	3600	2.58819	20.51764
2	7200	5	21
3	10800	7.071068	21.41421
4	14400	8.660254	21.73205
5	18000	9.659258	21.93185
6	21600	10	22
7	25200	9.659258	21.93185
8	28800	8.660254	21.73205
9	32400	7.071068	21.41421
10	36000	5	21
11	39600	2.58819	20.51764
12	43200	-3.2E-15	20
13	46800	-2.58819	19.48236
14	50400	-5	19
15	54000	-7.07107	18.58579
16	57600	-8.66025	18.26795
17	61200	-9.65926	18.06815

x	F(x)
0	0
0.02	2
0.04	4
0.06	6
0.08	8
0.1	10
0.12	12
0.14	14
0.16	16
0.18	18
0.2	20

18	64800	-10	18
19	68400	-9.65926	18.06815
20	72000	-8.66025	18.26795
21	75600	-7.07107	18.58579
22	79200	-5	19
23	82800	-2.58819	19.48236
24	86400	6.43E-15	20

```
//Green function heat conduction with cubic spline curve fitting
import java.io.*;
import javax.swing.*;
import java.lang.*;
import java.util.Locale;

class HT_Green_Function1
{
  public static double [] thomas(double f[],double e[],double g[],double r[])
  {
  // 3 band matrix system of equation solving algorithm
  int n=f.length;
  double x[]=new double[n];
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
     }
  for(int k=1;k<n;k++)
     {r[k]=r[k]-e[k]*r[k-1];
     }
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
     {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
  return x;
  }


  public static double [][] cubic_spline(double xi[],double yi[],double c0,double cn)
  {
  int n=xi.length;
  double h[]=new double[n];
  double w[]=new double[n];
  double f[]=new double[n];
  double e[]=new double[n];
  double g[]=new double[n];
  double d[]=new double[n];
  double x[]=new double[n];
  double S[][]=new double[4][n];
  int k;
  for(k=0;k<(n-1);k++)
     {h[k]=xi[k+1]-xi[k];
      w[k]=(yi[k+1]-yi[k])/h[k];
                }
  d[0]=c0;
  d[n-1]=cn;
  for(k=1;k<(n-1);k++)
     {d[k]=6.0*(w[k]-w[k-1]);}
  f[0]=1.0;
  f[n-1]=1.0;
  g[0]=0.0;
  g[n-1]=0.0;
  e[0]=0.0;
  e[n-1]=0.0;
  for(k=1;k<(n-1);k++)
  {f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];}
  S[2]=thomas(f,e,g,d);
  S[3]=xi;
  for(k=0;k<(n-1);k++)
    {S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
     S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
    }
  return S;
  }

  public static double funcSpline(double S[][],double x)
  {
  int n=S[0].length;
```

```
  double xx1=0;
  double xx2=0;
  double y=0;
  double hk=0;
  for(int k=0;k<(n-1);k++)
     {if(S[3][k]<=x && x<=S[3][k+1])
      {hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
       break;
       }
     }
       if(y==0 && S[3][n-2]<=x )
       {
                  int k=n-2;
                  hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
       }
  return y;
  }
 public static double[][]  funcSpline(double xi[],double yi[],int numberofmidpoints,double x0,double xn)
  {
               //numberofmidpoints :  in x--o--o--x--o--o--x chain if x's are esxperimental points
               // numberofmidpoints is 2
               int n=xi.length;
               int nn=(n-1)*(numberofmidpoints+1)+1;
               double z[][]=new double[2][nn];
    double S[][]=cubic_spline(xi,yi,x0,xn);
    double dx=0;
    int k=0;
    int i;
    for(i=0;i<(n-1);i++)
       {  z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);k++;
          for(int j=0;j<numberofmidpoints;j++)
             {dx=(xi[i+1]-xi[i])/((double)numberofmidpoints+1.0);
          z[0][k]=z[0][k-1]+dx;z[1][k]=funcSpline(S,z[0][k]);k++;}
       }
       z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);
    return z;
 }
 public static double[][]  funcSpline(double xi[],double yi[],int numberofmidpoints)
 {
               //numberofmidpoints :  in x--o--o--x--o--o--x chain if x's are esxperimental points
               // numberofmidpoints is 2
               int n=xi.length;
               int nn=(n-1)*(numberofmidpoints+1)+1;
               double z[][]=new double[2][nn];
    double S[][]=cubic_spline(xi,yi,0,0);
    double dx=0;
    int k=0;
    int i;
    for(i=0;i<(n-1);i++)
       {  z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);k++;
          for(int j=0;j<numberofmidpoints;j++)
             {dx=(xi[i+1]-xi[i])/((double)numberofmidpoints+1.0);
          z[0][k]=z[0][k-1]+dx;z[1][k]=funcSpline(S,z[0][k]);k++;}
       }
       z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);
    return z;
 }
  // Türev formülleri
// =================
  public static double dfSpline(double xi[],double yi[],double c0,double cn,double x)
  { //kübik şerit türev formülü
               double S[][]=cubic_spline(xi,yi,c0,cn);
    return dfSpline(S,x);
  }

  public static double dfSpline(double S[][],double x)
  {
    //kübik şerit türev formülü
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
```

```
 double y=0;
 double hk=0;
 for(int k=0;k<(n-1);k++)
    {if(S[3][k]<=x && x<=S[3][k+1])
     {hk=(S[3][k+1]-S[3][k]);
      xx1=(x-S[3][k]);
      xx2=(S[3][k+1]-x);
      y=S[0][k]-S[1][k]+(xx1*xx1*S[2][k+1]-xx2*xx2*S[2][k])/(2.0*hk);
      break;
      }
    }
      if(y==0 && S[3][n-2]<=x )
      {
                int k=n-2;
                hk=(S[3][k+1]-S[3][k]);
      xx1=(x-S[3][k]);
      xx2=(S[3][k+1]-x);
      y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
      }
 return y;
 }

public static double  dfSpline(double xi[],double yi[],double x)
{
   //doğal kübik şerit türev formülü
            double S[][]=cubic_spline(xi,yi,0,0);
   return dfSpline(S,x);
}

 public static double intSpline(double S[][],double a,double b)
 {
   //kübik şerit integral[S(x)]dx  formülü
 int n=S[0].length;
 double xx1=0;
 double xx2=0;
 double y1,y2;
 double hk=0;
 double toplam=0;
 y1=0;y2=0;
 for(int k=0;k<(n-1);k++)
    { hk=(S[3][k+1]-S[3][k]);
     if(a>S[3][k+1])
     {toplam=0;
     }
              else if(S[3][k]<=a && a<=S[3][k+1] && b>S[3][k+1])//şart 2
      {
      xx1=(a-S[3][k]);
      xx2=(S[3][k+1]-a);
      y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      xx1=hk;
      xx2=0;
      y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      toplam+=(y2-y1);
      }
      else if(S[3][k]<=a && a<=S[3][k+1] && S[3][k]<=b && b<=S[3][k+1])//şart 3
      {
      xx1=(a-S[3][k]);
      xx2=(S[3][k+1]-a);
      y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      xx1=(b-S[3][k]);
      xx2=(S[3][k+1]-b);
      y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      toplam+=(y2-y1);
      }
      else if(a<S[3][k] && b>=S[3][k+1]) //şart 4
      {
      xx1=0;
      xx2=hk;
      y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      xx1=hk;
      xx2=0;
      y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      toplam+=(y2-y1);
      }
      else if(S[3][k]<=b && b<=S[3][k+1])//şart 5
      {xx1=0;
       xx2=hk;
```

```
        y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
        xx1=(b-S[3][k]);
        xx2=(S[3][k+1]-b);
        y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
        toplam+=(y2-y1);
        }
        else break;
     }
  return toplam;
  }


public static double golden(double S[][],double a,double b)
{
//  find the minimum of the function
//  note  maximum f(x) = minimum (-f(x))
    double epsilon=1.0e-10;
    double delta=1.0e-8;
    int print=0;
    double r1 = (Math.sqrt(5.0)-1.0)/2.0; // golden ratio
    double r2 = r1*r1;
    double h = b - a;
    double ya = -funcSpline(S,a);
    double yb = -funcSpline(S,b);
    double c = a + r2*h;
    double d = a + r1*h;
    double yc = -funcSpline(S,c);
    double yd = -funcSpline(S,d);
    int k = 1;
    double dp,dy,p,yp;
    while ((Math.abs(yb-ya)>epsilon) || (h>delta))
    {
      k++;
      if (yc<yd)
        {
        b = d;
        yb = yd;
        d = c;
        yd = yc;
        h = b - a;
        c = a + r2 * h;
        yc = -funcSpline(S,c);
        }
      else
       {
         a = c;
         ya = yc;
         c = d;
         yc = yd;
         h = b - a;
         d = a + r1 * h;
         yd = -funcSpline(S,d);
       }//end of if
    }//end of while
    dp = Math.abs(b-a);
    dy = Math.abs(yb-ya);
    p = a;
    yp = ya;
    if (yb<ya)
    {
      p = b;
      yp = yb;
    }
    if(print==1)
    {System.out.println("x min = "+p+"ymin = "+yp+"error of x ="+dp+"error of y"+dy); }
    return p;
}
public static void  printData(double c[][])
  {       String s3[]={"x[i]","y[i]"};
       Text.print(Text.T(c),s3,"Input data");
  }
  public static void  printOutputData(double c[][],int xarray[],int yarray[],int ngroup)
  {
                int n=c[0].length;
                int m=xarray.length;
                double z[][];
    double E[][];
```

```
  double xi[]=new double[n];
  double yi[]=new double[n];
  double dx=0;
  int k=0;
  int i;
  String s="";
  String s1[]={"x","y"};
  for(i=0;i<m;i++)
  {
            s="data set no : "+i+" x ref : "+xarray[i]+" y ref : "+yarray[i];
            xi=c[xarray[i]];
            yi=c[yarray[i]];
  z=funcSpline(xi,yi,ngroup);
  Text.printT(z,s1,s);
  }

}

public static void printCoefficients(int datano,double c[][],int x1array[],int y1array[],String s)
{    int n=c[0].length;
     String s2[]={"dataset number : k","a[k]","b[k]","c[k]","x[k]"};
     double d[][]=Text.T(cubic_spline(c[x1array[datano]],c[y1array[datano]],0,0));
     double e[][]=new double[d.length][d[0].length+1];
     for(int i=0;i<e.length;i++)
     {for(int j=0;j<e[0].length;j++)
      {if(j==0) e[i][j]=i;
       else e[i][j]=d[i][j-1];
      }
     }
     int i=1;
     Text.print(e,s2,s);
}
public static void  funcEKKPlot(double c[][],int xarray[],int yarray[],int ngroup)
{ int sp[]={20,22,24,25,26,27,32,33,34,35,21,23};
  int sp1[]={0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9};
            int n=c[0].length;
            int m=xarray.length;
            double z[][];
  double E[][];
  double xi[]=new double[n];
  double yi[]=new double[n];
  double dx=0;
  int k=0;
  int i;
  String s="";
  String s1[]={"x","y"};
  xi=c[xarray[0]];
            yi=c[yarray[0]];
  Plot pp=new Plot(xi,yi);
  pp.setPlotType(0,sp[0]);
  for(i=0;i<m;i++)
  {
            s="data set no : "+i+" x ref : "+xarray[i]+" y ref : "+yarray[i];
            xi=c[xarray[i]];
            yi=c[yarray[i]];
  z=funcSpline(xi,yi,ngroup);
  if(i!=0) {pp.addData(xi,yi,sp[i]);}
  pp.addData(z[0],z[1],sp1[i]);
  }
  pp.setPlabel("y=f(x)");
  pp.setXlabel("x");
  pp.setYlabel("y");
  pp.plot();
}
public static void  errorEKKPlot(double c[][],int xarray[],int yarray[],int ngroup)
{ int sp[]={1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9};
            int n=c[0].length;
            int m=xarray.length;
            double z[][];
  double E[][];
  double xi[]=new double[n];
  double yi[]=new double[n];
  double dx=0;
  int k=0;
  int i;
  String s="";
  String s1[]={"x","y"};
```

```
    xi=c[xarray[0]];
                yi=c[yarray[0]];
    Plot pp=new Plot(xi,yi);
    i=0;
    do
    {
                s="data set no : "+i+" x ref : "+xarray[i]+" y ref : "+yarray[i];
                xi=c[xarray[i]];
                yi=c[yarray[i]];
    z=funcSpline(xi,yi,ngroup);
    if(i==0) {pp=new Plot(z[0],z[1]);pp.setPlotType(i,sp[i]);}
    else     {pp.addData(z[0],z[1],sp[i]);}
    i++;
    } while(i<m);
    pp.setPlabel("Error=y-yi");
    pp.setXlabel("x");
    pp.setYlabel("dy");
    pp.plot();
  }
    public static String toString(double z[][])
  { String s="";
    for(int i=0;i<z.length;i++)
    {  for(int j=0;j<z[0].length;j++)
                  {s+=String.format("%f ",z[i][j]);}
                  s+="\n";
    }
    return s;
  }
  public static double T(double L,double alpha,double g,double k,double t,double x)
  {//Cubic spline curve fitting
     double f1[][]=Text.readDoubleT("f1.txt");
     double f2[][]=Text.readDoubleT("f2.txt");
     double F[][]=Text.readDoubleT("F.txt");
     double af1[][]=cubic_spline(f1[0],f1[1],0.0,0.0);
     double af2[][]=cubic_spline(f2[0],f2[1],0.0,0.0);
     double aF[][]=cubic_spline(F[0],F[1],0.0,0.0);
     double PI=Math.PI;
     double T=0;
     double t1=0,t2=0,t3=0,t4=0;
     int plus=-1;
     for(int m=1;m<20;m++)
     { double beta=m*PI/L;
                if_x y1=(x1)->Math.sin(beta*x1)*funcSpline(aF,x1);
      if_x y2=(x1)->Math.sin(beta*x1)*g;
      if_x yf1=(to)->Math.exp(-beta*beta*alpha*to)*funcSpline(af1,to);
      if_x yf2=(to)->Math.exp(-beta*beta*alpha*to)*funcSpline(af1,to);
      if_x yf=(to)->Math.exp(-beta*beta*alpha*to);
      t1+=2.0/L*Math.sin(beta*x)*Math.exp(-beta*beta*alpha*t)*y1.integral(0,L,10);
      t2+=alpha/k*2.0/L*Math.sin(beta*x)*yf.integral(0,t,10)*y2.integral(0,L,10);
      t3+=alpha*2.0/L*Math.sin(beta*x)*Math.exp(-beta*beta*alpha*t)*yf1.integral(0,t,10);
      t4+=alpha*2.0/L*plus*beta*Math.sin(beta*x)*Math.exp(-beta*beta*alpha*t)*yf1.integral(0,t,10);
      plus*=-1;
      }
      T=t1+t2+t3-t4;
      return T;
    }
  public static String toStringT(double z[][])
  { String s="";
    for(int i=0;i<z[0].length;i++)
    {  for(int j=0;j<z.length;j++)
                  {s+=String.format(Locale.US,"%f ",z[j][i]);}
                  s+="\n";
    }
    return s;
  }
  public static void main(String args[]) throws IOException
  {   double L=0.2;
                double alpha=5.06e-7;
                double g=1000;//  W/m^3 heat generation
                double k=0.8; //  W/mK
                double t=3600;
                double x=0.1;
                double T=T(L,alpha,g,k,t,x);
      System.out.println("t="+t+"x="+x+"T="+T);
  }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" HT_Green_Function1
t=3600.0x=0.1T=10.321184843506975

> Terminated with exit code 0.
```

A second problem utilising Green functions :

$\frac{\partial^2 T(x,t)}{\partial x^2} + \frac{1}{k} g(x,t) = \frac{1}{\alpha} \frac{\partial T(x,t)}{\partial t}$ Boundary conditions $\frac{\partial T(x,t)}{\partial x} = f_1(t)$ x=0 t >0 $\frac{\partial T(x,t)}{\partial x} + HT(x,t) = f_2(t)$ x=L t>0 $H = \frac{h}{k}$ $T(x,t) = F(x)$ t=0 $0 \le x \le L$	Homogeneous problem $\frac{\partial^2 \varphi(x,t)}{\partial x^2} = \frac{1}{\alpha} \frac{\partial \varphi(x,t)}{\partial t}$ Boundary conditions $\frac{\partial \varphi(x,t)}{\partial x} = 0$ x=0 t >0 by arranging $\frac{\partial \varphi(x,t)}{\partial x} + H\varphi(x,t) = 0$ $\varphi(x,t) = F(x)$ t=0 $0 \le x \le L$

Separation of variables:

$\varphi(x,t) = X(x)\Gamma(t)$

$\Gamma(t)\frac{\partial^2 X(x)}{\partial x^2} = \frac{X(x)}{\alpha}\frac{\partial \Gamma(t)}{\partial t}$

$\frac{1}{X(x)}\frac{\partial^2 X(x)}{\partial x^2} = \frac{1}{\alpha\Gamma(t)}\frac{\partial \Gamma(t)}{\partial t} = -\beta^2$

$\frac{\partial^2 X(x)}{\partial x^2} + \beta^2 X(x) = 0$

$\frac{\partial \Gamma(t)}{\partial t} + \beta^2 \alpha \Gamma(t) = 0$

Boundary conditions

$\frac{\partial X(x)}{\partial x} = 0$ x=0 t >0

$\frac{\partial X(x)}{\partial x} + HX(x) = 0$ x=L t>0 $H = \frac{h}{k}$

$T(x,t) = F(x)$ t=0 $0 \le x \le L$

Solution set:

$X(x) = C_1 \cos(\beta x) + C_2 \sin(\beta x)$

$\Gamma(t) = C_3 \exp(-\beta^2 \alpha t)$

Application of boundary conditions:

$\frac{\partial X(x)}{\partial x} = -C_1 \beta \sin(\beta x) + C_2 \beta \cos(\beta x)$

$\frac{\partial X(x)}{\partial x} = 0$ x=0 t >0

$\frac{\partial X(x)}{\partial x} = -C_1 \beta \sin(\beta 0) + C_2 \beta \cos(\beta 0) = -C_1 * 0 + C_2 \beta * 1 = 0 \xrightarrow{yields} C_2 = 0$

$\frac{\partial X(x)}{\partial x} + HX(x) = 0$ x=L t>0

$-C_1 \beta \sin(\beta L) + C_1 H \cos(\beta L) = 0 \xrightarrow{because} C_1 \neq 0$

$\beta_m \sin(\beta_m L) - H \cos(\beta_m L) = 0$ or

$\beta_m \tan(\beta_m L) = H$

$\beta_m L \sin(\beta_m L) - HL \cos(\beta_m L) = 0$

$\beta_m \mathrm{L} \tan(\beta_m L) = HL$ $\xi_m = \beta_m \mathrm{L}$ $Bi = HL = \frac{hL}{k} = Biot\ number$

$\xi_m \sin(\xi_m) + Bi \cos(\xi_m) = 0$ or $\xi_m \tan(\xi_m) = Bi$

These equations has more than one solution. It can be solved only by numerical root finding methods. If $\xi_m \tan(\xi_m) = Bi$ form is taken into account, a caution should be recmmended due to fact that $\tan(\xi_n)$ is nıt a continious function. Temperature profile becomes.

$$\varphi(x,t) = \sum_{m=1}^{\infty} C_m \cos(\beta_m x) \exp(-\beta_m^2 \alpha t)$$

We still do not know the C_n coefficients we will use the remaining boundary conditions to find it.

$\varphi(x,t) = F(x)$ t=0 $0 \le x \le L$

$$F(x) = \sum_{m=1}^{\infty} C_m \cos(\beta_n x)$$

This equation has n coefficients. In order to solve these coefficients, we will use properties of orthogonal functions.

$$\int_0^L X(\beta_n x) X(\beta_m x) dx = \begin{cases} 0 & m \neq n \\ N(\beta_m) & m = n \end{cases}$$

$$\int_0^L F(x) \cos(\beta_m x) = C_m \int_0^L cos^2(\beta_m x) dx$$

$$N(\beta_m) = \int_0^L cos^2(\beta_m x) dx = \int_0^L \frac{1 + cos(2\beta_m x)}{2} dx = \left[\frac{x}{2} + \frac{\sin(2\beta_m x)}{4\beta_m}\right]_{x=0}^{L} = \frac{L}{2} + \frac{\sin(2\beta_m L)}{4\beta_m}$$

To get an alternative formula for $N(\beta_m)$ eigenvalue function

$\beta_m \sin(\beta_n L) = H \cos(\beta_m L)$ can be utilised

$\sin(\beta_m L) = \frac{H}{\beta_m} \cos(\beta_m L)$ $\sin(2\beta_m L) = 2\sin(\beta_m L)\cos(\beta_m L)$ $\sin(2\beta_m L) = 2\frac{H}{\beta_m}\cos^2(\beta_m L)$

On the other hand taking square of eigenvalue function:

$\sin^2(\beta_m L) = \left[\frac{H}{\beta_m}\right]^2 \cos^2(\beta_m L)$ $\sin^2(\beta_m L) + \cos^2(\beta_m L) = 1$ $\sin^2(\beta_m L) = 1 - \cos^2(\beta_m L)$

$1 - \cos^2(\beta_m L) = \frac{H^2}{\beta_m^2}\cos^2(\beta_m L)$ $1 = \left[\frac{H^2}{\beta_m^2} + 1\right]\cos^2(\beta_m L)$ $\cos^2(\beta_m L) = \frac{\beta_m^2}{H^2 + \beta_m^2}$

Substituting into previous equation

$$\sin(2\beta_m L) = 2\frac{H}{\beta_m}\cos^2(\beta_m L) = 2\frac{H}{\beta_m}\frac{\beta_m^2}{H^2 + \beta_m^2}$$

$$N(\beta_m) = \int_0^L cos^2(\beta_m x') dx' = \frac{L}{2} + \frac{\sin(2\beta_m L)}{4\beta_m} = \frac{L}{2} + \frac{2\frac{H}{\beta_m}\frac{\beta_m^2}{H^2 + \beta_m^2}}{4\beta_m} = \frac{L(\beta_m^2 + H^2) + H}{2(\beta_m^2 + H^2)}$$

$$C_m = \frac{1}{N(\beta_m)} \int_0^L F(x') \cos(\beta_m x') dx'$$

$$\varphi(x,t) = \int_0^L \left[\frac{2(\beta_m^2 + H^2)}{L(\beta_m^2 + H^2) + H} \cos(\beta_m x)\cos(\beta_m x')\exp(-\beta_m^2 \alpha t)\right] F(x') dx'$$

From this solution iy can be observed that

$$G(x,t|x',0) = \left[\frac{2(\beta_m^2 + H^2)}{L(\beta_m^2 + H^2) + H} \cos(\beta_m x)\cos(\beta_m x')\exp(-\beta_m^2 \alpha t)\right]$$

Now $G(x,t|x',\tau)$ can be obtained by replacing t with $t - \tau$

$$G(x,t|x',\tau) = \left[\frac{2(\beta_m^2 + H^2)}{L(\beta_m^2 + H^2) + H} \cos(\beta_m x)\cos(\beta_m x')\exp(-\beta_m^2 \alpha (t - \tau))\right]$$

Now that Green function constructed solution for the problem can be written as

$$T(x,t)=\int_0^L [G(x,t|x',\tau)]_{\tau=0} F(x')dx'$$
$$+\frac{\alpha}{k}\int_{\tau=0}^{t} d\tau \int_{x'=0}^{L} G(x,t|x',\tau)g(x',\tau)dx' + \frac{\alpha}{k}\int_{\tau=0}^{t} G(x,t|x',\tau)|_{x'=0} f_1(\tau)d\tau + \frac{\alpha}{k}\int_{\tau=0}^{t} G(x,t|x',\tau)|_{x'=L} f_2(\tau)d\tau$$

$$T(x,t)=\int_0^L \left[\frac{2\left(\beta_m^2+H^2\right)}{L\left(\beta_m^2+H^2\right)+H}\cos(\beta_m x)\cos(\beta_m x')\exp\left(-\beta_m^2\alpha(t)\right)\right]F(x')dx'$$
$$+\frac{\alpha}{k}\int_{\tau=0}^{t} d\tau \int_{x'=0}^{L} G(x,t|x',\tau)g(x',\tau)dx' + \frac{\alpha}{k}\int_{\tau=0}^{t} G(x,t|x',\tau)|_{x'=0} f_1(\tau)d\tau + \frac{\alpha}{k}\int_{\tau=0}^{t} G(x,t|x',\tau)|_{x'=L} f_2(\tau)d\tau$$

6.3 LAPLACE TRANSFORMS

The Laplace transform and inversion formula of function F(t) is defined by:

$$L[F(t)]=F(s)=\int_{t=0}^{\infty} e^{-st}F(t)dt$$

Inverse Laplace transform can be written as:

$$F(t)=\frac{1}{2\pi i}\int_{s=\gamma-i\infty}^{\gamma+i\infty} e^{st}F(s)ds$$

where s is the Laplace transform variable, $i=\sqrt{-1}$ is the complex number. Function could be continuos or piecewise continuous in any interval. For example if

$F(t)=1 \quad F(s)=\int_{t=0}^{\infty} e^{-st}dt=-\frac{1}{s}e^{-st}\Big|_0^{\infty}=-\frac{1}{s}e^{-\infty}+\frac{1}{s}e^{0}=\frac{1}{s}$

$F(t)=t \quad F(s)=\int_{t=0}^{\infty} te^{-st}dt=-\frac{1}{s}e^{-st}\Big|_0^{\infty}$

$u=t \quad dv=e^{-st}dt \quad du=dt \quad v=-\frac{1}{s}e^{-st}dt$

$$\int_a^b udv=uv|_a^b-\int_a^b vdu$$

$$F(s)=\int_{t=0}^{\infty} te^{-st}dt=-\frac{t}{s}e^{-st}\Big|_0^{\infty}-\int_{t=0}^{\infty}-\frac{1}{s}e^{-st}dt=0-\frac{1}{s^2}e^{-st}\Big|_0^{\infty}=\frac{1}{s^2}$$

Laplace transforms have the following relational properties:

$$L[C_1F(t)+C_2G(t)]=C_1F(s)+C_2G(s)$$

Derivatives:

$$L[F'(t)]=\int_{t=0}^{\infty} F'(t)e^{-st}dt=[F(t)e^{-st}]_0^{\infty}+s\int_{t=0}^{\infty} e^{-st}F(t)dt=sF(s)-F(0)$$

Second deriatives:

$$L[F"(t)]=\int_{t=0}^{\infty} F"(t)e^{-st}dt=sL\left[sF^{'(s)}-F^{'}(0)\right]=s^2F(s)-sF^{'}(0)-F"(0)$$

In general nth derivative can be written as:

$$L\left[F^{(n)}(t)\right]=s^nF(s)-s^{n-1}F(0)-s^{n-2}F^{(1)}(0)-s^{n-3}F^{(2)}(0)-\cdots-F^{(n-1)}(0)$$

Laplace transform of the integral:

$$G(t)=\int_0^t F(t)dt$$

$$G(s)=\frac{1}{s}F(s)$$

Change of scale:

$$L[F(at)] = \int_{t=0}^{\infty} e^{-st}F(at)dt = \frac{1}{a}\int_{t=0}^{\infty} e^{-s/au}F(u)du = \frac{1}{a}F\left(\frac{s}{a}\right)$$

Similarly

$$L\left[F\left(\frac{t}{a}\right)\right] = \int_{t=0}^{\infty} e^{-st}F\left(\frac{t}{a}\right)dt = a\int_{t=0}^{\infty} e^{-asu}F(u)du = a\,F(as)$$

Shift property:

$$L[e^{\pm at}F(t)] = \int_{t=0}^{\infty} e^{-st}e^{\pm at}F(t)dt = \int_{t=0}^{\infty} e^{-(s\mp a)t}F(t)dt = F(\,(s \mp a)$$

Some of the Laplace Transform functions

No	F(s)	F(t)
1	$\frac{1}{s}$	1
2	$\frac{1}{s^2}$	t
3	$\frac{1}{s^n}\quad (n = 1,2,3,..)$	$\frac{t^{n-1}}{(n-1)!}$
4	$\frac{1}{\sqrt{s}}$	$\frac{1}{\sqrt{\pi t}}$
5	$\frac{1}{s^{-3/2}}$	$2\sqrt{\frac{t}{\pi}}$
6	$s^{(-n+172)}\quad (n\ = 1,2,3,..)$	$\frac{2^n}{[1.3.5..(2n-1)]\sqrt{\pi}}t^{n-1/2}$
7	$\frac{1}{s^n}\quad (n > 0\ \ real\ number)$	$\frac{t^{n-1}}{\Gamma(n)}$
8	$\frac{1}{s+a}$	e^{-at}
9	$\frac{1}{(s+a)^n}\quad (n = 1,2,3,..)$	$\frac{t^{n-1}e^{-at}}{(n-1)!}$
10	$\frac{\Gamma(k)}{(s+a)^k}\quad (k > 0)$	$t^{k-1}e^{-at}$
11	$\frac{1}{(s+a)(s+b)}\quad (a \neq b)$	$\frac{e^{-at}-e^{-bt}}{b-a}$
12	$\frac{s}{(s+a)(s+b)}\quad (a \neq b)$	$\frac{ae^{-at}-be^{-bt}}{a-b}$
13	$\frac{1}{s^2+a^2}$	$\frac{\sin(at)}{a}$
14	$\frac{s}{s^2+a^2}$	$\cos(at)$
15	$\frac{1}{s^2-a^2}$	$\frac{\sinh(at)}{a}$
16	$\frac{s}{s^2-a^2}$	$\cosh(at)$
17	$\frac{1}{s(s^2+a^2)}$	$\frac{1}{a^2}(1-\cos(at))$
18	$\frac{1}{s^2(s^2+a^2)}$	$\frac{1}{a^3}(at-\sin(at))$
19	$\frac{1}{(s^2+a^2)^2}$	$\frac{1}{2a^3}(\sin(at)-(at)\cos(at))$
20	$\frac{s}{(s^2+a^2)^2}$	$\frac{t}{2a}\sin(at)$
21	$\frac{s^2}{(s^2+a^2)^2}$	$\frac{1}{2a}(\sin(at)-(at)\cos(at))$
22	$\frac{s^2-a^2}{(s^2+a^2)^2}$	$tcos(at)$
23	$\frac{1}{\sqrt{s}+a}$	$\frac{1}{\sqrt{\pi t}}-ae^{a^2t}erfc(a\sqrt{t})$
24	$\frac{\sqrt{s}}{s-a^2}$	$\frac{1}{\sqrt{\pi t}}+ae^{a^2t}erfc(a\sqrt{t})$
25	$\frac{\sqrt{s}}{s+a^2}$	$\frac{1}{\sqrt{\pi t}}+\frac{2a}{\sqrt{\pi}}e^{-a^2t}\int_0^{a\sqrt{t}} e^{\lambda^2}d\lambda$

26	$\frac{1}{\sqrt{s}(s-a^2)}$	$\frac{1}{a}e^{a^2t}erfc(a\sqrt{t})$
27	$\frac{1}{\sqrt{s}(s+a^2)}$	$\frac{2}{a\sqrt{\pi}}e^{-a^2t}\int_0^{a\sqrt{t}} e^{\lambda^2}d\lambda$
28	$\frac{b^2-a^2}{(s-a^2)(b+\sqrt{s})}$	$e^{a^2t}\left[b-aerf\left(a\sqrt{t}\right)\right]-be^{b^2t}erf\left(b\sqrt{t}\right)$
29	$\frac{1}{\sqrt{s}(s+a)}$	$e^{a^2t}\ erfc\left(a\sqrt{t}\right)$
30	$\frac{1}{(s+a)\sqrt{s+b}}$	$\frac{1}{\sqrt{b-a}}e^{at}erf\left(\sqrt{b-a\sqrt{t}}\right)$
31	$\frac{\sqrt{s+2a}}{\sqrt{s}}-1$	$e^{-at}[I_1(at)+I_0(at)]$
32	$\frac{1}{\sqrt{s+a}\sqrt{s+b}}$	$e^{-(a+b)t}I_0\left(\frac{a-b}{2}t\right)$
33	$\frac{1}{\sqrt{s^2+a^2}}$	$J_0(at)$
34	$\frac{\left(\sqrt{s^2-a^2}-s\right)^{\nu}}{\sqrt{s^2+a^2}}$ $where\ (\nu>-1)$	$a^{\nu}J_{\nu}(at)$
35	$\frac{\left(s-\sqrt{s^2-a^2}\right)^{\nu}}{\sqrt{s^2-a^2}}$ $where\ (\nu>-1)$	$a^{\nu}I_{\nu}(at)$
36	$\frac{1}{s}e^{-ks}$	u(t-k)
37	$\frac{1}{s^2}e^{-ks}$	(t-k) u(t-k)
38	$\frac{1}{s}e^{-k/s}$	$J_0\left(2\sqrt{kt}\right)$
39	$\frac{1}{s^{\mu}}e^{-k/s}$ $\mu>0$	$\left(\frac{t}{k}\right)^{(\mu-1)/2}J_0\left(2\sqrt{kt}\right)$
40	$\frac{1}{s^{\mu}}e^{-k/s}$ $\mu<0$	$\left(\frac{t}{k}\right)^{(\mu-1)/2}I_{(\mu-1)}\left(2\sqrt{kt}\right)$
41	$e^{-k\sqrt{s}}$ $k\geq 0$	$\frac{k}{2\sqrt{\pi t^3}}\exp\left(-\frac{k^2}{4t}\right)$
42	$\frac{e^{-k\sqrt{s}}}{s}$ $k\geq 0$	$erfc\left(\frac{k}{2\sqrt{t}}\right)$
43	$\frac{e^{-k\sqrt{s}}}{\sqrt{s}}$ $k\geq 0$	$\frac{1}{\sqrt{\pi t}}\exp\left(-\frac{k^2}{4t}\right)$
44	$\frac{e^{-k\sqrt{s}}}{s^{3/2}}$ $k\geq 0$	$2\sqrt{\frac{t}{\pi}}\exp\left(-\frac{k^2}{4t}\right)-kerfc\left(\frac{k}{2\sqrt{t}}\right)$
45	$\frac{e^{-k\sqrt{s}}}{s^{(1+n/2)}}$ $(n=0,1,2,..,k\geq 0)$	$(4t)^{n/2}i^nerfc\left(\frac{k}{2\sqrt{t}}\right)$
46	$\frac{e^{-k\sqrt{s}}}{a+\sqrt{s}}$ $k\geq 0$	$\frac{1}{\sqrt{\pi t}}\exp\left(-\frac{k^2}{4t}\right)-ae^{ak}e^{a^2t}erfc\left(a\sqrt{t}+\frac{k}{2\sqrt{t}}\right)$
47	$\frac{e^{-k\sqrt{s}}}{\sqrt{s}(a+\sqrt{s})}$ $k\geq$	$e^{ak}e^{a^2t}erfc\left(a\sqrt{t}+\frac{k}{2\sqrt{t}}\right)$
48	$\frac{e^{-k\sqrt{s(s+a)}}}{\sqrt{s(s+a)}}$ $k\geq 0$	$e^{-at/2}I_0\left(\frac{1}{2}a\sqrt{t^2-k^2}\right)\mathrm{u(t-k)}$
49	$\frac{e^{-k\sqrt{s^2+a^2}}}{\sqrt{s^2+a^2}}$ $k\geq 0$	$J_0\left(a\sqrt{t^2-k^2}\right)\mathrm{u(t-k)}$
50	$\frac{e^{-k\sqrt{s^2+a^2}}}{\sqrt{s^2-a^2}}$ $k\geq 0$	$I_0\left(a\sqrt{t^2-k^2}\right)\mathrm{u(t-k)}$
51	$\frac{ae^{-k\sqrt{s}}}{s(a+\sqrt{s})}$ $k\geq 0$	$-e^{ak}e^{a^2t}erfc\left(a\sqrt{t}+\frac{k}{2\sqrt{t}}\right)+erfc\left(\frac{k}{2\sqrt{t}}\right)$
52	$\frac{e^{-k\sqrt{s}}}{s^2}$ $k\geq 0$	$\left(t+\frac{k^2}{2}\right)erfc\left(\frac{k}{2\sqrt{t}}\right)-k\left(\frac{t}{\pi}\right)^{1/2}exp\left(-\frac{k^2}{4t}\right)$
53	$\frac{1}{s}\ln(s)$	$-\gamma-\ln(t)$ γ =0.5772156649.. Euler's constant
54	$ln\left(\frac{s+a}{s+b}\right)$	$\frac{1}{t}(e^{-bt}-e^{-at})$
55	$ln\left(\frac{s^2+a^2}{s^2}\right)$	$\frac{2}{t}(1-\cos(at))$
56	$ln\left(\frac{s^2-a^2}{s^2}\right)$	$\frac{2}{t}(1-\cosh(at))$

57	$K_0(ks) \quad (k>0)$	$\frac{1}{\sqrt{t^2-k^2}}u(t-k)$
58	$K_0(k\sqrt{s}) \quad (k>0)$	$\frac{1}{2t}exp\left(-\frac{k^2}{4t}\right)$
59	$\frac{1}{\sqrt{s}}K_1(k\sqrt{s}) \quad (k>0)$	$\frac{1}{k}exp\left(-\frac{k^2}{4t}\right)$

Note that u(t) is unit step function where

$$u(t)=\begin{cases}0 & t<0\\ 1 & t\geq 0\end{cases}$$

After these initial definitions, now A heat transfer problem can be investigated

PROBLEM 6.3.1 SEMI-INFINITE WALL PROBLEM LAPLACE TRANSFORM SOLUTION TEMPERATURE PROFILE BOUNDARY

$\frac{\partial^2 T^*(x,t)}{\partial T^{*2}}=\frac{1}{\alpha}\frac{\partial T^*(x,t)}{\partial t}$
$T^*(x,t)=F(x)+Ti$ x=0 t > 0
$T^*(x,t)=T_i \quad x\to\infty$ t > 0
$T^*(x,t)=T_i$ at t=0 $0\leq x\leq\infty$

$$T(x,t)=T^*(x,t)-T_i$$

$\frac{\partial^2 T(x,t)}{\partial T^2}=\frac{1}{\alpha}\frac{\partial T}{\partial t}$
$T(x,t)=F(x)+T_i-T_i=F(x)$ x=0 t > 0
$T(x,t)=0 \quad x\to\infty$ t > 0
$T(x,t)=0$ at t=0 $0\leq x\leq\infty$

Laplace transform of the differential equation:

$$\frac{\partial^2 T(x,s)}{\partial x^2}-\frac{s}{\alpha}T(x,s)=0$$

Boundary conditions:
$T(x,s)=F(s)$ x=0
$T(x,s)=0 \quad x\to\infty$
Solution of this ordinary differential equation is:

$$T(x,s)=C_1e^{-x\sqrt{\frac{s}{\alpha}}}+C_2e^{x\sqrt{\frac{s}{\alpha}}}$$

Boundary condition $T(x,s)=0 \quad x\to\infty \quad \to \quad C_2=0$
Boundary condition $T(x,s)=F(s)$ x=0 $\to \quad C_1=F(s)$
So the solution is:

$$T(x,s)=F(s)e^{-x\sqrt{\frac{s}{\alpha}}}$$

Inverse transform for a multiplication of two laplace transform function can be written as:

$$T(x,t)=\int_0^t F(\tau)\,G(x,t-\tau)d\tau$$

Inverse laplace transfrom of $G(x,s)=e^{-x\sqrt{\frac{s}{\alpha}}}$ is $G(x,t)=\frac{x}{2\sqrt{\pi\alpha t^3}}e^{-\frac{x^2}{4\alpha t}}$

In the table it is given as case 41:
For $G(x,s)=e^{-k\sqrt{s}} \quad G(x,t)=\frac{k}{2\sqrt{\pi t^3}}e^{-\frac{k^2}{4t}}$ and our case $k=\frac{x}{\sqrt{\alpha}}$

$$T(x,t)=T^*(x,t)-T_i=\frac{x}{\sqrt{4\pi\alpha}}\int_0^t F(\tau)\frac{x}{(t-\tau)^{3/2}}\exp\left[-\frac{x^2}{4\alpha(t-\tau)}\right]d\tau$$

Special case of this problem: constant surface temperature T_0 for t>0

$\frac{\partial^2 T^*(x,t)}{\partial T^{*2}} = \frac{1}{\alpha}\frac{\partial T^*(x,t)}{\partial t}$
$T^*(x,t) = T_0 = constant$ x=0 t > 0
$T^*(x,t) = T_i \quad x \to \infty \quad$ t > 0
$T^*(x,t) = T_i$ at t=0 $\quad 0 \le x \le \infty$

$$T(x,t) = T^*(x,t) - T_i$$

$\frac{\partial^2 T(x,t)}{\partial T^2} = \frac{1}{\alpha}\frac{\partial T}{\partial t}$
$T(x,t) = T_0 - T_i = F(x)$ x=0 t > 0
$T(x,t) = 0 \quad x \to \infty \quad$ t > 0
$T(x,t) = 0$ at t=0 $\quad 0 \le x \le \infty$

Laplace transform of the differential equation:

$$\frac{\partial^2 T(x,s)}{\partial x^2} - \frac{s}{\alpha}T(x,s) = 0$$

Boundary conditions:
$T(x,s) = T_0 - T_i$ x=0
$T(x,s) = 0 \quad x \to \infty$
Solution of this ordinary differential equation is:

$$T(x,s) = C_1 e^{-x\sqrt{\frac{s}{\alpha}}} + C_2 e^{x\sqrt{\frac{s}{\alpha}}}$$

Boundary condition $T(x,s) = 0 \quad x \to \infty \quad \rightarrow \quad C_2 = 0$
Boundary condition $T(x,s) = \frac{T_0 - T_i}{s}$ x=0 $\quad \rightarrow \quad C_1 = \frac{T_0 - T_i}{s}$
So the solution is:

$$T(x,s) = \frac{T_0 - T_i}{s} e^{-x\sqrt{\frac{s}{\alpha}}}$$

Inverse laplace transfrom of $G(x,s) = \frac{e^{-x\sqrt{\frac{s}{\alpha}}}}{s}$ is $G(x,t) = erfc\left(\frac{x}{2\sqrt{\alpha t}}\right)$

In the table it is given as case 42:
For $G(x,s) = \frac{e^{-k\sqrt{s}}}{s} \quad G(x,t) = erfc\left(\frac{k}{2\sqrt{t}}\right)$ and our case $k = \frac{x}{\sqrt{\alpha}}$

$$T(x,t) = T^*(x,t) - T_i = (T_0 - T_i)erfc\left(\frac{x}{2\sqrt{\alpha t}}\right) = (T_0 - T_i)\left[1 - erf\left(\frac{x}{2\sqrt{\alpha t}}\right)\right]$$

$$\frac{T^*(x,t) - T_0}{T_i - T_0} = erf\left(\frac{x}{2\sqrt{\alpha t}}\right)$$

erf(x) is error function

$$\text{erf(x)} = \frac{2}{\sqrt{\pi}}\int_0^x e^{-v^2} dv$$

Serial formulation of error function:

$$\text{erf(x)} = \frac{2}{\sqrt{\pi}}\sum_{n=0}^{\infty}(-1)^n \frac{x^{2n+1}}{n!\,(2n+1)}$$

Error function can be calculated from this series

```
public class erf1
{  public static double erf(double x)
   {double A=2.0/Math.sqrt(Math.PI);
    double total=0;
    double plusminus=1;
    double power=x;
    double factorial=1;
    for(int n=1;n<200;n++)
    {total+=power*plusminus/factorial/(2.0*(n-1)+1.);
```

```
    power*=x*x;
    factorial*=n;
    plusminus*=-1;
    }
   return A*total;
  }
  public static void main(String arg[])
  {   double x=0;
  double dx=0.02;
                for(int i=0;i<151;i++)
     {x=dx*i;
                System.out.println(x+" "+erf(x));
     }}}
```

Error function

x	erf(x)	x	erf(x)	x	erf(x)
0	0	1	0.842701	2	0.995322
0.02	0.022565	1.02	0.850838	2.02	0.995719
0.04	0.045111	1.04	0.85865	2.04	0.996086
0.06	0.067622	1.06	0.866144	2.06	0.996423
0.08	0.090078	1.08	0.873326	2.08	0.996734
0.1	0.112463	1.1	0.880205	2.1	0.997021
0.12	0.134758	1.12	0.886788	2.12	0.997284
0.14	0.156947	1.14	0.893082	2.14	0.997525
0.16	0.179012	1.16	0.899096	2.16	0.997747
0.18	0.200936	1.18	0.904837	2.18	0.997951
0.2	0.222703	1.2	0.910314	2.2	0.998137
0.22	0.244296	1.22	0.915534	2.22	0.998308
0.24	0.2657	1.24	0.920505	2.24	0.998464
0.26	0.2869	1.26	0.925236	2.26	0.998607
0.28	0.30788	1.28	0.929734	2.28	0.998738
0.3	0.328627	1.3	0.934008	2.3	0.998857
0.32	0.349126	1.32	0.938065	2.32	0.998966
0.34	0.369365	1.34	0.941914	2.34	0.999065
0.36	0.38933	1.36	0.945561	2.36	0.999155
0.38	0.409009	1.38	0.949016	2.38	0.999237
0.4	0.428392	1.4	0.952285	2.4	0.999311
0.42	0.447468	1.42	0.955376	2.42	0.999379
0.44	0.466225	1.44	0.958297	2.44	0.999441
0.46	0.484655	1.46	0.961054	2.46	0.999497
0.48	0.50275	1.48	0.963654	2.48	0.999547
0.5	0.5205	1.5	0.966105	2.5	0.999593
0.52	0.537899	1.52	0.968413	2.52	0.999635
0.54	0.554939	1.54	0.970586	2.54	0.999672
0.56	0.571616	1.56	0.972628	2.56	0.999706
0.58	0.587923	1.58	0.974547	2.58	0.999736
0.6	0.603856	1.6	0.976348	2.6	0.999764
0.62	0.619411	1.62	0.978038	2.62	0.999789
0.64	0.634586	1.64	0.979622	2.64	0.999811
0.66	0.649377	1.66	0.981105	2.66	0.999831
0.68	0.663782	1.68	0.982493	2.68	0.999849
0.7	0.677801	1.7	0.98379	2.7	0.999866
0.72	0.691433	1.72	0.985003	2.72	0.99988
0.74	0.704678	1.74	0.986135	2.74	0.999893
0.76	0.717537	1.76	0.98719	2.76	0.999905
0.78	0.73001	1.78	0.988174	2.78	0.999916
0.8	0.742101	1.8	0.989091	2.8	0.999925
0.82	0.753811	1.82	0.989943	2.82	0.999933
0.84	0.765143	1.84	0.990736	2.84	0.999941
0.86	0.7761	1.86	0.991472	2.86	0.999948
0.88	0.786687	1.88	0.992156	2.88	0.999954
0.9	0.796908	1.9	0.99279	2.9	0.999959
0.92	0.806768	1.92	0.993378	2.92	0.999964
0.94	0.816271	1.94	0.993923	2.94	0.999968
0.96	0.825424	1.96	0.994426	2.96	0.999972

	0.98	0.834232	1.98	0.994892	2.98	0.999975	

Problem 6.3.1 Java Solution

```
 import java.util.*;
public class HT_infinite_wall1
{
public static double f(double t)
{  double ff=0;
            if(t<100)ff=30+0.5*t;
            else     ff=70.0;
   return ff;
}
//integration function
public static double intfr(double to,double t,double x,double alpha)
{ double T=x/Math.sqrt(4.0*Math.PI*alpha)*f(to)/Math.pow((t-to),1.5)*Math.exp(-x*x/(4.0*alpha*(t-to)));
            return T;
}

//integral
//Gauss Kronrod integration formula
public static double integral(double a,double b,double x,double alpha)
   {
            //eps hata miktari
            int n=50;
            double eps=1e-10;
      int Aused=0;
            boolean sonuc;
      double integral=0;
      double A[][];
      int Aw = 0;
      double toplamhata = 0;
      double[] c;
      double[] xg;
      double[] xk;
      int nn = 0;
      int ng = 0;
      int i = 0;
      int j = 0;
      int h = 0;
      double v = 0;
      double k1 = 0;
      double k2 = 0;
      double intg = 0;
      double intk = 0;
      double ta = 0;
      double tb = 0;
      Aw = 4;
      A = new double[n-1+1][Aw-1+1];
      nn = 61;
      ng = 15;
      c = new double[nn];
      xk = new double[nn];
      xg = new double[nn];
      //Gauss katsayilari
      xg[0] = 0.00796819249616660561546588347467 4;
      xg[1] = 0.01846646831109095914230213191204 7;
      xg[2] = 0.02878470788332336934971917961129 2;
      xg[3] = 0.03879919256962704959680193644634 8;
      xg[4] = 0.04840267283059405290293814042280 8;
      xg[5] = 0.05749315621761906648172168940205 6;
      xg[6] = 0.06597422988218049512812851511596 2;
      xg[7] = 0.07375597473770520626824385002219 1;
      xg[8] = 0.08075589522942021535469493846053 0;
      xg[9] = 0.08689978720108297980238753071512 6;
      xg[10] = 0.09212252223778612871763270708761 9;
      xg[11] = 0.09636873717464425963946862635181 0;
      xg[12] = 0.09959342058679526706278028210356 9;
      xg[13] = 0.10176238974840550459642895216855 4;
      xg[14] = 0.10285265289355884034128563670541 5;
      c[0] = 0.99948441005049063757132589570581 1;
      c[1] = 0.99689348407464954027163005091869 5;
      c[2] = 0.99163099687040459485862836610948 6;
      c[3] = 0.98366812327974720997003258160566 3;
      c[4] = 0.97311632250112626837469386842370 7;
      c[5] = 0.96002186496830751221687102558179 8;
      c[6] = 0.94437444474855997941583132403743 9;
```

```
c[7] = 0.92620004742927432587932427708047 4;
c[8] = 0.90557330769990779854652255892595 8;
c[9] = 0.88256053579205268154311646253022 6;
c[10] = 0.85720523354606109895865851065894 4;
c[11] = 0.82956576238276839744289811973250 2;
c[12] = 0.79972783582183908301366894232268 3;
c[13] = 0.76777743210482619491797734097450 3;
c[14] = 0.73379006245322680472617113136952 8;
c[15] = 0.69785049479331579693229238802664 0;
c[16] = 0.66006106412662696137005366814927 1;
c[17] = 0.62052618298924286114047755643118 9;
c[18] = 0.57934523582636169175602493217254 0;
c[19] = 0.53662414814201989926416979331107 3;
c[20] = 0.49248046786177857499369306120770 9;
c[21] = 0.44703376953808917678060990032285 4;
c[22] = 0.40040125483039439253547621154266 1;
c[23] = 0.35270472553087811347103720708937 4;
c[24] = 0.30407320227362507737267710719925 7;
c[25] = 0.25463692616788984643980512981780 5;
c[26] = 0.20452511668230989143895767100202 5;
c[27] = 0.15386991360858354696379467274325 6;
c[28] = 0.10280693796673703014709675131800 1;
c[29] = 0.05147184255531769583302521316672 3;
c[30] = 0.00000000000000000000000000000000 0;
//Kronrod katsayilari
xk[0] = 0.00138901369867700762455159122676 0;
xk[1] = 0.00389046112709988405126720184451 6;
xk[2] = 0.00663070391593129217331982636975 0;
xk[3] = 0.00927327965951776342844114689202 4;
xk[4] = 0.01182301525349634174223289885325 1;
xk[5] = 0.01436972950704580481245143244358 0;
xk[6] = 0.01692088918905327262757228942032 2;
xk[7] = 0.01941414119394238117340895105012 8;
xk[8] = 0.02182803582160919229716748573833 9;
xk[9] = 0.02419116207808060136568637072523 2;
xk[10] = 0.02650995488233310161060170933507 5;
xk[11] = 0.02875404876504129284397878535433 4;
xk[12] = 0.03090725756238776247288425294309 2;
xk[13] = 0.03298144705748372603181419101685 4;
xk[14] = 0.03497933802806002413749967073146 8;
xk[15] = 0.03688236465182122922391106561713 6;
xk[16] = 0.03867894562472759295034865153228 1;
xk[17] = 0.04037453895153595911199527975246 8;
xk[18] = 0.04196981021516424614714754128597 0;
xk[19] = 0.04345253970135606931683172811707 3;
xk[20] = 0.04481480013316266319235555161672 3;
xk[21] = 0.04605923827100698811627173555937 4;
xk[22] = 0.04718554656929915394526147818109 9;
xk[23] = 0.04818586175708712914077949229830 5;
xk[24] = 0.04905543455502977888752816536723 8;
xk[25] = 0.04979568342707420635781156937994 2;
xk[26] = 0.05040592140278234684089308565358 5;
xk[27] = 0.05088179589874960649229747304980 5;
xk[28] = 0.05122154784925877217065628260494 4;
xk[29] = 0.05142612853745902593386287921578 1;
xk[30] = 0.05149472942945156755834043364709 9;
for(i=nn-1; i>=nn/2; i--)
{c[i] = -c[nn-1-i];}
for(i=nn-1; i>=nn/2; i--)
{ xk[i] = xk[nn-1-i]; }
for(i=ng-1; i>=0; i--)
{ xg[nn-2-2*i] = xg[i];
  xg[1+2*i] = xg[i];
}
for(i=0; i<=nn/2; i++)
{ xg[2*i] = 0; }
k1 = 0.5*(b-a);
k2 = 0.5*(b+a);
intg = 0;
intk = 0;
for(i=0; i<=nn-1; i++)
{ v = intfr((k1*c[i]+k2),b,x,alpha);
  intk = intk+v*xk[i];
  if( i%2==1 )
  {intg = intg+v*xg[i];}
}
intk = intk*(b-a)*0.5;
```

```
    intg = intg*(b-a)*0.5;
    A[0][0] = Math.abs(intg-intk);
    A[0][1] = intk;
    A[0][2] = a;
    A[0][3] = b;
    toplamhata = A[0][0];
    if( toplamhata<eps )
    {
       sonuc = true;
       integral = intk;
       Aused = 1;
       return integral;
    }
    Aused = 1;
    for(h=1; h<=n-1; h++)
    {
       Aused = h+1;
       gir(A, h, Aw);
       toplamhata = toplamhata-A[h-1][0];
       ta = A[h-1][2];
       tb = A[h-1][3];
       A[h-1][2] = ta;
       A[h-1][3] = 0.5*(ta+tb);
       A[h][2] = 0.5*(ta+tb);
       A[h][3] = tb;
       for(j=h-1; j<=h; j++)
       {
          k1 = 0.5*(A[j][3]-A[j][2]);
          k2 = 0.5*(A[j][3]+A[j][2]);
          intg = 0;
          intk = 0;
          for(i=0; i<=nn-1; i++)
          {   v = intfr((k1*c[i]+k2),b,x,alpha);
             intk = intk+v*xk[i];
             if( i%2==1 )
             { intg = intg+v*xg[i];}
          }
          intk = intk*(A[j][3]-A[j][2])*0.5;
          intg = intg*(A[j][3]-A[j][2])*0.5;
          A[j][0] = Math.abs(intg-intk);
          A[j][1] = intk;
          toplamhata = toplamhata+A[j][0];
       }
       cik(A, h-1, Aw);
       cik(A, h, Aw);
       if( toplamhata<eps )
       {break;}
    }
    sonuc = toplamhata<eps;
    integral = 0;
    for(j=0; j<=Aused-1; j++)
    { integral = integral+A[j][1];}
    return integral;
 }
 public static void gir(double A[][],int n,int Awidth)
 {
    int i = 0;
    int p = 0;
    double t = 0;
    int maxcp = 0;
    if( n==1 )
    { return; }
    for(i=0; i<=Awidth-1; i++)
    {   t = A[n-1][i];
       A[n-1][i] = A[0][i];
       A[0][i] = t;
    }//end of for
    p = 0;
    while( 2*p+1<n-1 )
    {   maxcp = 2*p+1;
       if( 2*p+2<n-1 )
       {   if( A[2*p+2][0]>A[2*p+1][0] )
          { maxcp = 2*p+2; }
       }
       if( A[p][0]<A[maxcp][0] )
       {   for(i=0; i<=Awidth-1; i++)
          {   t = A[p][i];
```

```
                A[p][i] = A[maxcp][i];
                A[maxcp][i] = t;
              }
              p = maxcp;
            }
            else
            { break; }
        } //end of while
    }

    public static void cik(double A[][],int n,int Awidth)
    {
        int i = 0;
        int p = 0;
        double t = 0;
        int kk = 0;
        if( n==0 )
        {return;}
        p = n;
        while( p!=0 )
        {   kk = (p-1)/2;
            if( A[p][0]>A[kk][0] )
            { for(i=0; i<=Awidth-1; i++)
                { t = A[p][i];
                  A[p][i] = A[kk][i];
                  A[kk][i] = t;
                }
                p = kk;
            }
            else
            {break;}
        }
    }


     public static void main (String args[])
     {
       double alpha=4.0e-5;
       double x=0.1; //m
       double t=300.0;//s
       double T1=20.0;// degree C;
       double T=integral(0.0,t,x,alpha)+T1;
       System.out.println("T="+T);
     }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" HT_infinite_wall1
T=55.14859399275895

> Terminated with exit code 0.
```

Numerical example(variable surface temperature $F(x)+T_i$ for x=0 t>0): A water pipe will be burried underground in a cold climate. When meterological data is examined it is detected that after a long stable period of 20 ℃ temperatures, which it can be assumed that upper layers of the earth will be reached to this temperatures, air cools according to the table given below:

day	temperature	day	temperature	day	temperature	day	temperature
1	20	21	-12	41	-12	61	-8
2	16	22	-15	42	-12	62	-4
3	12	23	-17	43	-11	63	0
4	10	24	-14	44	-11	64	4
5	7	25	-13	45	-11	65	6
6	3	26	-15	46	-12	66	8
7	2	27	-14	47	-11	67	10
8	0	28	-13	48	-14	68	12
9	-2	29	-12	49	-11	69	12
10	-6	30	-11	50	-15	70	16
11	-10	31	-14	51	-12	71	16
12	-11	32	-15	52	-11	72	17
13	-13	33	-14	53	-14	73	18
14	-14	34	-13	54	-15	74	19
15	-15	35	-14	55	-15	75	20
16	-16	36	-14	56	-14	76	20
17	-15	37	-12	57	-15	77	20
18	-17	38	-12	58	-15	78	20
19	-18	39	-15	59	-14	79	20

20	-14	40	-12	60	-15	80	20

When observed the temperature profile, it is seen that temperature starts to warm up again after 60 days. Determine the required depth for the pipe to avoid freezing. Take thermal diffusivity coefficient $\alpha = 0.138 \times 10^{-6}$ m^2/s

```
 import java.util.*;


public class HT_infinite_walla
{
double a[][];
public HT_infinite_walla(String s)
{double d[][]=Text.readDoubleT(s);
 a=cubic_spline(d[0],d[1],0.0,0.0);
}
public  double f(double t,double Ti)
{   double t1=t/86400.0;
            double ff=funcSpline(a,t1)-Ti;
   return ff;
}

//integration function
public double  intfr(double to,double t,double x,double alpha,double Ti)
{ double T=x/Math.sqrt(4.0*Math.PI*alpha)*f(to,Ti)/Math.pow((t-to),1.5)*Math.exp(-x*x/(4.0*alpha*(t-to)));
  //System.out.println("TT="+T);
            return T;
}

//integral
//Gauss Kronrod integration formula
public  double integral(double a,double b,double x,double alpha,double Ti)
   {
            //eps hata miktari
            int n=50;
            double eps=1e-10;
      int Aused=0;
            boolean sonuc;
      double integral=0;
      double A[][];
      int Aw = 0;
      double toplamhata = 0;
      double[] c;
      double[] xg;
      double[] xk;
      int nn = 0;
      int ng = 0;
      int i = 0;
      int j = 0;
      int h = 0;
      double v = 0;
      double k1 = 0;
      double k2 = 0;
      double intg = 0;
      double intk = 0;
      double ta = 0;
      double tb = 0;
      Aw = 4;
      A = new double[n-1+1][Aw-1+1];
      nn = 61;
      ng = 15;
      c = new double[nn];
      xk = new double[nn];
      xg = new double[nn];
      //Gauss katsayilari
      xg[0] = 0.00796819249616660561546588347467 4;
      xg[1] = 0.01846646831109095914230213191204 7;
      xg[2] = 0.02878470788332336934971917961129 2;
      xg[3] = 0.03879919256962704959680193644634 8;
      xg[4] = 0.04840267283059405290293814042280 8;
      xg[5] = 0.05749315621761906648172168940205 6;
      xg[6] = 0.06597422988218049512812851511596 2;
      xg[7] = 0.07375597473770520626824385002219 1;
      xg[8] = 0.08075589522942021535469493846053 0;
      xg[9] = 0.08689978720108297980238753071512 6;
      xg[10] = 0.09212252223778612871763270708761 9;
      xg[11] = 0.09636873717464425963946862635181 0;
```

```
xg[12] = 0.09959342058679526706278028210356 9;
xg[13] = 0.10176238974840550459642895216855 4;
xg[14] = 0.10285265289355884034128563670541 5;
c[0] = 0.99948441005049063757132589570581 1;
c[1] = 0.99689348407464954027163005091869 5;
c[2] = 0.99163099687040459485862836610948 6;
c[3] = 0.98366812327974720997003258160566 3;
c[4] = 0.97311632250112626837469386842370 7;
c[5] = 0.96002186496830751221687102558179 8;
c[6] = 0.94437444474855997941583132403743 9;
c[7] = 0.92620004742927432587932427708047 4;
c[8] = 0.90557330769990779854652255892595 8;
c[9] = 0.88256053579205268154311646253022 6;
c[10] = 0.85720523354606109895865851065894 4;
c[11] = 0.82956576238276839744289811973250 2;
c[12] = 0.79972783582183908301366894232268 3;
c[13] = 0.76777743210482619491797734097450 3;
c[14] = 0.73379006245322680472617113136952 8;
c[15] = 0.69785049479331579693229238802664 0;
c[16] = 0.66006106412662696137005366814927 1;
c[17] = 0.62052618298924286114047755643118 9;
c[18] = 0.57934523582636169175602493217254 0;
c[19] = 0.53662414814201989926416979331107 3;
c[20] = 0.49248046786177857499369306120770 9;
c[21] = 0.44703376953808917678060990032285 4;
c[22] = 0.40040125483039439253547621154266 1;
c[23] = 0.35270472553087811347103720708937 4;
c[24] = 0.30407320227362507737267710719925 7;
c[25] = 0.25463692616788984643980512981780 5;
c[26] = 0.20452511668230989143895767100202 5;
c[27] = 0.15386991360858354696379467274325 6;
c[28] = 0.10280693796673703014709675131800 1;
c[29] = 0.05147184255531769583302521316672 3;
c[30] = 0.00000000000000000000000000000000 0;
//Kronrod katsayilari
xk[0] = 0.00138901369867700762455159122676 0;
xk[1] = 0.00389046112709988405126720184451 6;
xk[2] = 0.00663070391593129217331982636975 0;
xk[3] = 0.00927327965951776342844114689202 4;
xk[4] = 0.01182301525349634174223289885325 1;
xk[5] = 0.01436972950704580481245143244358 0;
xk[6] = 0.01692088918905327262757228942032 2;
xk[7] = 0.01941414119394238117340895105012 8;
xk[8] = 0.02182803582160919229716748573833 9;
xk[9] = 0.02419116207808060136568637072523 2;
xk[10] = 0.02650995488233310161060170933507 5;
xk[11] = 0.02875404876504129284397878535433 4;
xk[12] = 0.03090725756238776247288425294309 2;
xk[13] = 0.03298144705748372603181419101685 4;
xk[14] = 0.03497933802806002413749967073146 8;
xk[15] = 0.03688236465182122922391106561713 6;
xk[16] = 0.03867894562472759295034865153228 1;
xk[17] = 0.04037453895153595911199527975246 8;
xk[18] = 0.04196981021516424614714754128597 0;
xk[19] = 0.04345253970135606931683172811707 3;
xk[20] = 0.04481480013316266319235555161672 3;
xk[21] = 0.04605923827100698811627173555937 4;
xk[22] = 0.04718554656929915394526147818109 9;
xk[23] = 0.04818586175708712914077949229830 5;
xk[24] = 0.04905543455502977888752816536723 8;
xk[25] = 0.04979568342707420635781156937994 2;
xk[26] = 0.05040592140278234684089308565358 5;
xk[27] = 0.05088179589874960649229747304980 5;
xk[28] = 0.05122154784925877217065628260494 4;
xk[29] = 0.05142612853745902593386287921578 1;
xk[30] = 0.05149472942945156755834043364709 9;
for(i=nn-1; i>=nn/2; i--)
{c[i] = -c[nn-1-i];}
for(i=nn-1; i>=nn/2; i--)
{ xk[i] = xk[nn-1-i]; }
for(i=ng-1; i>=0; i--)
{ xg[nn-2-2*i] = xg[i];
  xg[1+2*i] = xg[i];
}
for(i=0; i<=nn/2; i++)
{ xg[2*i] = 0; }
k1 = 0.5*(b-a);
```

```
    k2 = 0.5*(b+a);
    intg = 0;
    intk = 0;
    for(i=0; i<=nn-1; i++)
    { v = intfr((k1*c[i]+k2),b,x,alpha,Ti);
      intk = intk+v*xk[i];
      if( i%2==1 )
      {intg = intg+v*xg[i];}
    }
    intk = intk*(b-a)*0.5;
    intg = intg*(b-a)*0.5;
    A[0][0] = Math.abs(intg-intk);
    A[0][1] = intk;
    A[0][2] = a;
    A[0][3] = b;
    toplamhata = A[0][0];
    if( toplamhata<eps )
    {
       sonuc = true;
       integral = intk;
       Aused = 1;
       return integral;
    }
    Aused = 1;
    for(h=1; h<=n-1; h++)
    {
       Aused = h+1;
       gir(A, h, Aw);
       toplamhata = toplamhata-A[h-1][0];
       ta = A[h-1][2];
       tb = A[h-1][3];
       A[h-1][2] = ta;
       A[h-1][3] = 0.5*(ta+tb);
       A[h][2] = 0.5*(ta+tb);
       A[h][3] = tb;
       for(j=h-1; j<=h; j++)
       {
          k1 = 0.5*(A[j][3]-A[j][2]);
          k2 = 0.5*(A[j][3]+A[j][2]);
          intg = 0;
          intk = 0;
          for(i=0; i<=nn-1; i++)
          {   v = intfr((k1*c[i]+k2),b,x,alpha,Ti);
             intk = intk+v*xk[i];
             if( i%2==1 )
             { intg = intg+v*xg[i];}
          }
          intk = intk*(A[j][3]-A[j][2])*0.5;
          intg = intg*(A[j][3]-A[j][2])*0.5;
          A[j][0] = Math.abs(intg-intk);
          A[j][1] = intk;
          toplamhata = toplamhata+A[j][0];
       }
       cik(A, h-1, Aw);
       cik(A, h, Aw);
       if( toplamhata<eps )
       {break;}
    }
    sonuc = toplamhata<eps;
    integral = 0;
    for(j=0; j<=Aused-1; j++)
    { integral = integral+A[j][1];}
    return integral;
}
public  void gir(double A[][],int n,int Awidth)
{
    int i = 0;
    int p = 0;
    double t = 0;
    int maxcp = 0;
    if( n==1 )
    { return; }
    for(i=0; i<=Awidth-1; i++)
    {   t = A[n-1][i];
       A[n-1][i] = A[0][i];
       A[0][i] = t;
    }//end of for
```

```
        p = 0;
        while( 2*p+1<n-1 )
        {   maxcp = 2*p+1;
            if( 2*p+2<n-1 )
            {   if( A[2*p+2][0]>A[2*p+1][0] )
                { maxcp = 2*p+2; }
            }
            if( A[p][0]<A[maxcp][0] )
            {   for(i=0; i<=Awidth-1; i++)
                {   t = A[p][i];
                    A[p][i] = A[maxcp][i];
                    A[maxcp][i] = t;
                }
                p = maxcp;
            }
            else
            { break; }
        } //end of while
    }

    public  void cik(double A[][],int n,int Awidth)
    {
        int i = 0;
        int p = 0;
        double t = 0;
        int kk = 0;
        if( n==0 )
        {return;}
        p = n;
        while( p!=0 )
        {   kk = (p-1)/2;
            if( A[p][0]>A[kk][0] )
            { for(i=0; i<=Awidth-1; i++)
                { t = A[p][i];
                  A[p][i] = A[kk][i];
                  A[kk][i] = t;
                }
                p = kk;
            }
            else
            {break;}
        }
    }
//Cubic spline interpolation
  public static double [] thomas(double f[],double e[],double g[],double r[])
  {
  // 3 band matrix system of equation solving algorithm
  int n=f.length;
  double x[]=new double[n];
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
     }
  for(int k=1;k<n;k++)
     {r[k]=r[k]-e[k]*r[k-1];
     }
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
     {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
  return x;
  }


  public static double [][] cubic_spline(double xi[],double yi[],double c0,double cn)
  {
  int n=xi.length;
  double h[]=new double[n];
  double w[]=new double[n];
  double f[]=new double[n];
  double e[]=new double[n];
  double g[]=new double[n];
  double d[]=new double[n];
  double x[]=new double[n];
  double S[][]=new double[4][n];
  int k;
  for(k=0;k<(n-1);k++)
     {h[k]=xi[k+1]-xi[k];
```

```
    w[k]=(yi[k+1]-yi[k])/h[k];
              }
 d[0]=c0;
 d[n-1]=cn;
 for(k=1;k<(n-1);k++)
   {d[k]=6.0*(w[k]-w[k-1]);}
 f[0]=1.0;
 f[n-1]=1.0;
 g[0]=0.0;
 g[n-1]=0.0;
 e[0]=0.0;
 e[n-1]=0.0;
 for(k=1;k<(n-1);k++)
 {f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];}
 S[2]=thomas(f,e,g,d);
 S[3]=xi;
 for(k=0;k<(n-1);k++)
  {S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
   S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
  }
 return S;
 }

 public static double funcSpline(double S[][],double x)
 {
 int n=S[0].length;
 double xx1=0;
 double xx2=0;
 double y=0;
 double hk=0;
 for(int k=0;k<(n-1);k++)
   {if(S[3][k]<=x && x<=S[3][k+1])
    {hk=(S[3][k+1]-S[3][k]);
     xx1=(x-S[3][k]);
     xx2=(S[3][k+1]-x);
     y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
     break;
     }
   }
     if(y==0 && S[3][n-2]<=x )
     {
               int k=n-2;
               hk=(S[3][k+1]-S[3][k]);
     xx1=(x-S[3][k]);
     xx2=(S[3][k+1]-x);
     y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
     }
 return y;
 }
public static double[][]  funcSpline(double xi[],double yi[],int numberofmidpoints,double x0,double xn)
 {
              //numberofmidpoints :  in x--o--o--x--o--o--x chain if x's are esxperimental points
              // numberofmidpoints is 2
              int n=xi.length;
              int nn=(n-1)*(numberofmidpoints+1)+1;
              double z[][]=new double[2][nn];
  double S[][]=cubic_spline(xi,yi,x0,xn);
  double dx=0;
  int k=0;
  int i;
  for(i=0;i<(n-1);i++)
    {  z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);k++;
       for(int j=0;j<numberofmidpoints;j++)
          {dx=(xi[i+1]-xi[i])/((double)numberofmidpoints+1.0);
       z[0][k]=z[0][k-1]+dx;z[1][k]=funcSpline(S,z[0][k]);k++;}
    }
    z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);
  return z;
}
public static double[][]  funcSpline(double xi[],double yi[],int numberofmidpoints)
{
              //numberofmidpoints :  in x--o--o--x--o--o--x chain if x's are esxperimental points
              // numberofmidpoints is 2
              int n=xi.length;
              int nn=(n-1)*(numberofmidpoints+1)+1;
              double z[][]=new double[2][nn];
  double S[][]=cubic_spline(xi,yi,0,0);
```

```
    double dx=0;
    int k=0;
    int i;
    for(i=0;i<(n-1);i++)
      {  z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);k++;
        for(int j=0;j<numberofmidpoints;j++)
          {dx=(xi[i+1]-xi[i])/((double)numberofmidpoints+1.0);
        z[0][k]=z[0][k-1]+dx;z[1][k]=funcSpline(S,z[0][k]);k++;}
      }
      z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);
    return z;
  }
   // Türev formülleri
// =================
  public static double dfSpline(double xi[],double yi[],double c0,double cn,double x)
  { //kübik şerit türev formülü
              double S[][]=cubic_spline(xi,yi,c0,cn);
    return dfSpline(S,x);
  }

  public static double dfSpline(double S[][],double x)
  {
    //kübik şerit türev formülü
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y=0;
  double hk=0;
  for(int k=0;k<(n-1);k++)
     {if(S[3][k]<=x && x<=S[3][k+1])
      {hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]-S[1][k]+(xx1*xx1*S[2][k+1]-xx2*xx2*S[2][k])/(2.0*hk);
       break;
       }
     }
       if(y==0 && S[3][n-2]<=x )
       {
                int k=n-2;
                hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
       }
  return y;
  }

  public static double  dfSpline(double xi[],double yi[],double x)
  {
    //doğal kübik şerit türev formülü
              double S[][]=cubic_spline(xi,yi,0,0);
    return dfSpline(S,x);
  }

  public static double intSpline(double S[][],double a,double b)
  {
    //kübik şerit integral[S(x)]dx  formülü
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y1,y2;
  double hk=0;
  double toplam=0;
  y1=0;y2=0;
  for(int k=0;k<(n-1);k++)
     { hk=(S[3][k+1]-S[3][k]);
      if(a>S[3][k+1])
      {toplam=0;
      }
              else if(S[3][k]<=a && a<=S[3][k+1] && b>S[3][k+1])//şart 2
      {
       xx1=(a-S[3][k]);
       xx2=(S[3][k+1]-a);
       y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       xx1=hk;
       xx2=0;
```

```
      y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      toplam+=(y2-y1);
      }
      else if(S[3][k]<=a && a<=S[3][k+1] && S[3][k]<=b && b<=S[3][k+1])//şart 3
      {
      xx1=(a-S[3][k]);
      xx2=(S[3][k+1]-a);
      y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      xx1=(b-S[3][k]);
      xx2=(S[3][k+1]-b);
      y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      toplam+=(y2-y1);
      }
      else if(a<S[3][k] && b>=S[3][k+1]) //şart 4
      {
      xx1=0;
      xx2=hk;
      y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      xx1=hk;
      xx2=0;
      y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      toplam+=(y2-y1);
      }
      else if(S[3][k]<=b && b<=S[3][k+1])//şart 5
      {xx1=0;
       xx2=hk;
       y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       xx1=(b-S[3][k]);
       xx2=(S[3][k+1]-b);
       y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       toplam+=(y2-y1);
       }
       else break;
    }
  return toplam;
  }


public static double golden(double S[][],double a,double b)
{
//  find the minimum of the function
//  note  maximum f(x) = minimum (-f(x))
    double epsilon=1.0e-10;
    double delta=1.0e-8;
    int print=0;
    double r1 = (Math.sqrt(5.0)-1.0)/2.0; // golden ratio
    double r2 = r1*r1;
    double h = b - a;
    double ya = -funcSpline(S,a);
    double yb = -funcSpline(S,b);
    double c = a + r2*h;
    double d = a + r1*h;
    double yc = -funcSpline(S,c);
    double yd = -funcSpline(S,d);
    int k = 1;
    double dp,dy,p,yp;
    while ((Math.abs(yb-ya)>epsilon) || (h>delta))
    {
      k++;
      if (yc<yd)
        {
        b = d;
        yb = yd;
        d = c;
        yd = yc;
        h = b - a;
        c = a + r2 * h;
        yc = -funcSpline(S,c);
        }
      else
       {
         a = c;
         ya = yc;
         c = d;
         yc = yd;
         h = b - a;
         d = a + r1 * h;
```

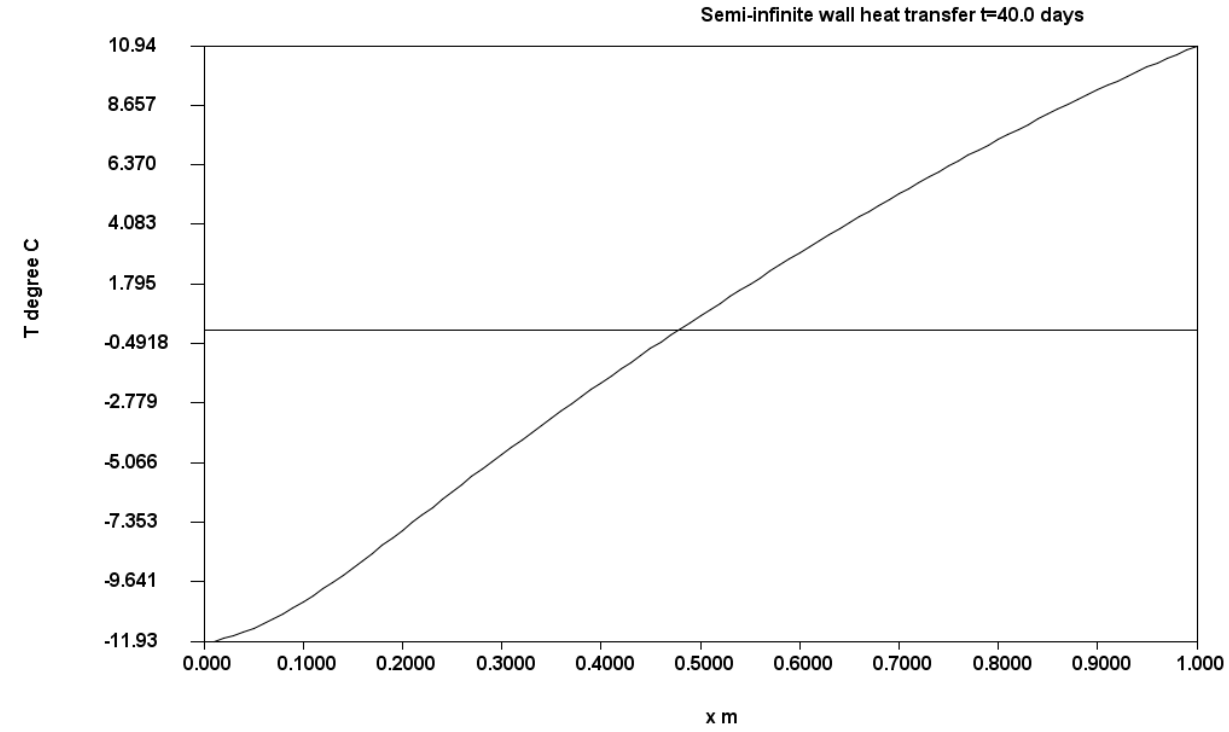
```
        yd = -funcSpline(S,d);
       }//end of if
    }//end of while
    dp = Math.abs(b-a);
    dy = Math.abs(yb-ya);
    p = a;
    yp = ya;
    if (yb<ya)
    {
      p = b;
     yp = yb;
    }
    if(print==1)
    {System.out.println("x min = "+p+"ymin = "+yp+"error of x ="+dp+"error of y"+dy); }
    return p;
}
public static void  printData(double c[][])
  {       String s3[]={"x[i]","y[i]"};
       Text.print(Text.T(c),s3,"Input data");
  }
  public static void  printOutputData(double c[][],int xarray[],int yarray[],int ngroup)
  {
                int n=c[0].length;
                int m=xarray.length;
                double z[][];
    double E[][];
    double xi[]=new double[n];
    double yi[]=new double[n];
    double dx=0;
    int k=0;
    int i;
    String s="";
    String s1[]={"x","y"};
    for(i=0;i<m;i++)
    {
                s="data set no : "+i+" x ref : "+xarray[i]+" y ref : "+yarray[i];
                xi=c[xarray[i]];
                yi=c[yarray[i]];
    z=funcSpline(xi,yi,ngroup);
    Text.printT(z,s1,s);
    }

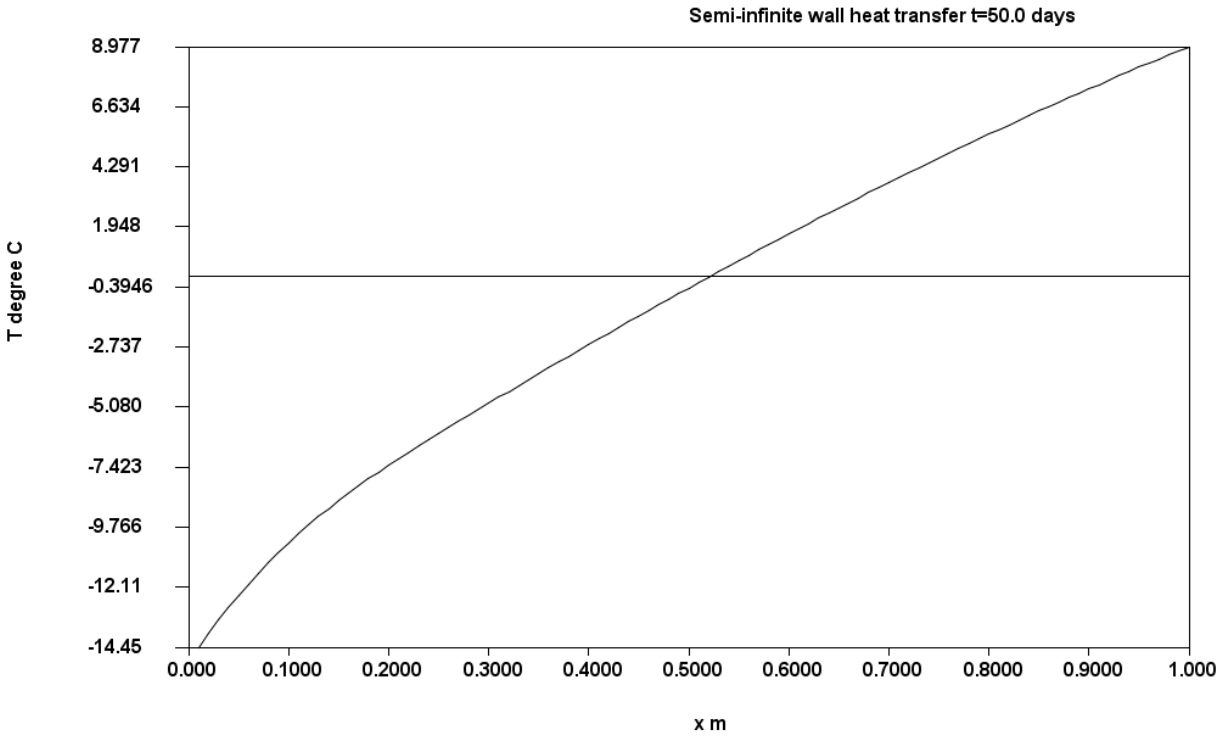
  }

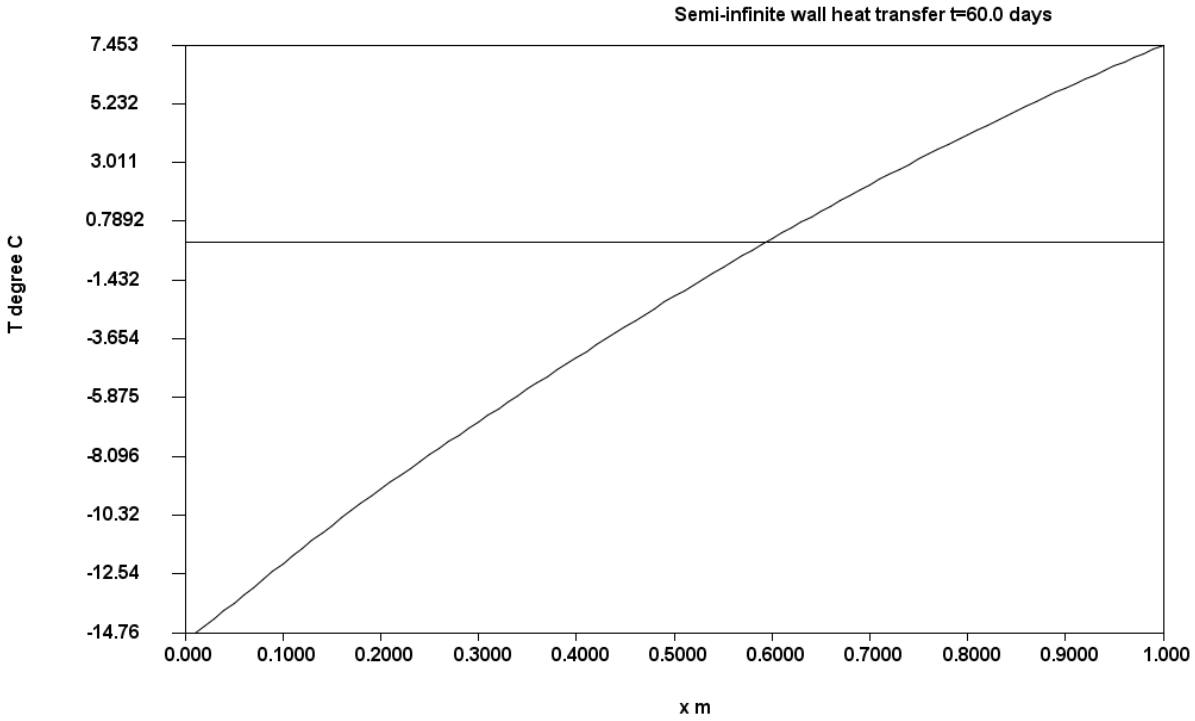
  public static void printCoefficients(int datano,double c[][],int x1array[],int y1array[],String s)
  {    int n=c[0].length;
       String s2[]={"dataset number : k","a[k]","b[k]","c[k]","x[k]"};
       double d[][]=Text.T(cubic_spline(c[x1array[datano]],c[y1array[datano]],0,0));
       double e[][]=new double[d.length][d[0].length+1];
       for(int i=0;i<e.length;i++)
       {for(int j=0;j<e[0].length;j++)
        {if(j==0) e[i][j]=i;
         else e[i][j]=d[i][j-1];
        }
       }
       int i=1;
       Text.print(e,s2,s);
  }
  public static void  funcEKKPlot(double c[][],int xarray[],int yarray[],int ngroup)
  { int sp[]={20,22,24,25,26,27,32,33,34,35,21,23};
    int sp1[]={0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9};
                int n=c[0].length;
                int m=xarray.length;
                double z[][];
    double E[][];
    double xi[]=new double[n];
    double yi[]=new double[n];
    double dx=0;
    int k=0;
    int i;
    String s="";
    String s1[]={"x","y"};
    xi=c[xarray[0]];
                yi=c[yarray[0]];
    Plot pp=new Plot(xi,yi);
    pp.setPlotType(0,sp[0]);
    for(i=0;i<m;i++)
```

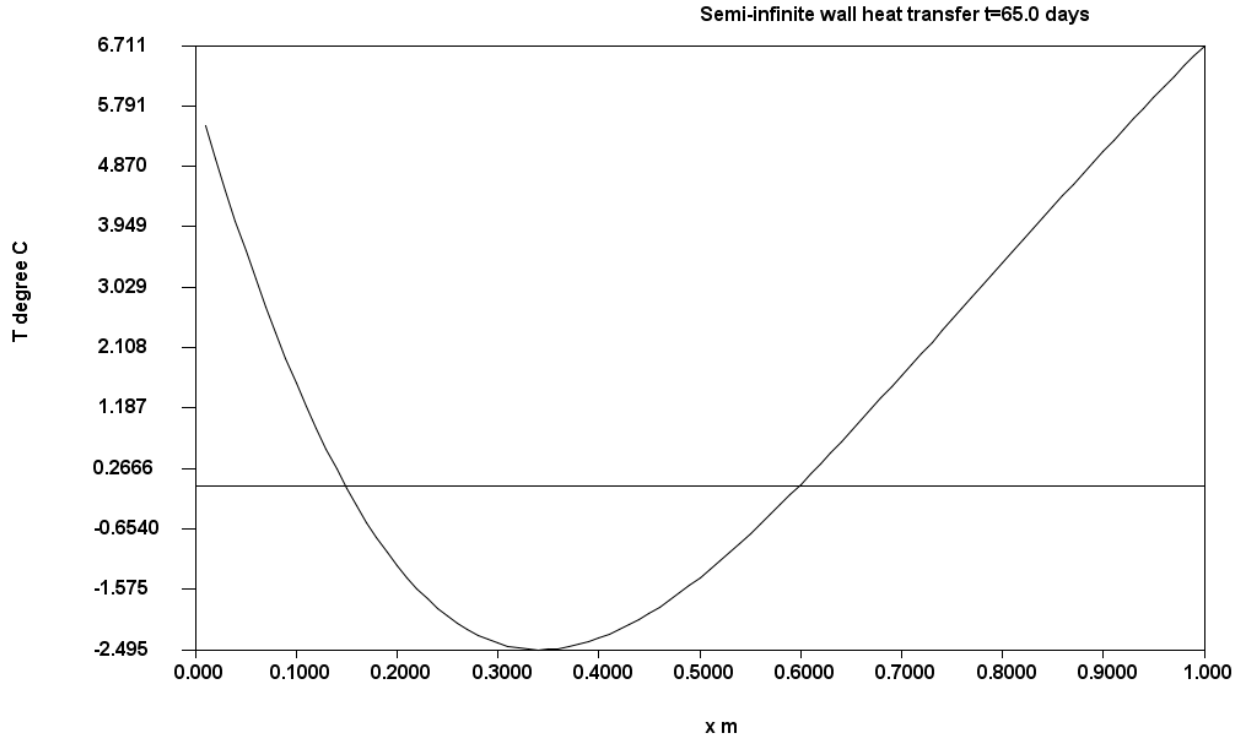
```
    {
              s="data set no : "+i+" x ref : "+xarray[i]+" y ref : "+yarray[i];
              xi=c[xarray[i]];
              yi=c[yarray[i]];
   z=funcSpline(xi,yi,ngroup);
   if(i!=0) {pp.addData(xi,yi,sp[i]);}
   pp.addData(z[0],z[1],sp1[i]);
   }
   pp.setPlabel("y=f(x)");
   pp.setXlabel("x");
   pp.setYlabel("y");
   pp.plot();
  }
  public static void  errorEKKPlot(double c[][],int xarray[],int yarray[],int ngroup)
  { int sp[]={1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9};
              int n=c[0].length;
              int m=xarray.length;
              double z[][];
   double E[][];
   double xi[]=new double[n];
   double yi[]=new double[n];
   double dx=0;
   int k=0;
   int i;
   String s="";
   String s1[]={"x","y"};
   xi=c[xarray[0]];
              yi=c[yarray[0]];
   Plot pp=new Plot(xi,yi);
   i=0;
   do
   {
              s="data set no : "+i+" x ref : "+xarray[i]+" y ref : "+yarray[i];
              xi=c[xarray[i]];
              yi=c[yarray[i]];
   z=funcSpline(xi,yi,ngroup);
   if(i==0) {pp=new Plot(z[0],z[1]);pp.setPlotType(i,sp[i]);}
   else     {pp.addData(z[0],z[1],sp[i]);}
   i++;
   } while(i<m);
   pp.setPlabel("Error=y-yi");
   pp.setXlabel("x");
   pp.setYlabel("dy");
   pp.plot();
  }
 // end of cubbi spline interpolation programs

    public static void main (String args[])
    { HT_infinite_walla iw=new HT_infinite_walla("a.txt");
      double alpha=0.138e-6;
      double x0=1.0; //m
      double x[]=new double[100];
      double T[]=new double[100];
      double t=60*86400;//s
      double Ti=20.0;// degree C;
      for(int i=0;i<100;i++)
      {x[i]=x0/100*(i+1);
       T[i]=iw.integral(0.0,t,x[i],alpha,Ti)+Ti;
       System.out.println(i+" "+x[i]+" "+T[i]);
      }
      Plot pp=new Plot(x,T);
      double xx[]={0,x0};
      double yy[]={0,0};
      pp.setPlabel("Semi-infinite wall heat transfer t="+t/86400+" days");
      pp.setXlabel(" x m");
      pp.setYlabel(" T degree C");
      pp.addData(xx,yy);
      pp.plot();
    }
}
```


Semi-infinite wall heat transfer t=40.0 days
T degree C
10.94
8.657
6.370
4.083
1.795
-0.4918
-2.779
-5.066
-7.353
-9.641
-11.93
0.000
0.1000
0.2000
0.3000
0.4000
0.5000
0.6000
0.7000
0.8000
0.9000
1.000
x m


Semi-infinite wall heat transfer t=50.0 days
T degree C
8.977
6.634
4.291
1.948
-0.3946
-2.737
-5.080
-7.423
-9.766
-12.11
-14.45
0.000
0.1000
0.2000
0.3000
0.4000
0.5000
0.6000
0.7000
0.8000
0.9000
1.000
x m





65 day plot clearly shows reheating effect of the surface. So the distance should be determined from 60 day case as 0.6 m.

Numerical example(constant surface temperature T_0 for x=0 t>0):

PROBLEM: A thick concrete wall (α=6e-7 m^2/s) is initially at a uniform temperature T_i=10 ℃. Suddenly its surface temperature is raised to T_0=105 ℃. Calculate temperatures at x =0.05,0.1 and 0.15 m from the surface at t=1000 s.

```
public class erf1
{  public static double erf(double x)
   {double A=2.0/Math.sqrt(Math.PI);
    double total=0;
    double plusminus=1;
    double power=x;
```

```
    double factorial=1;
    for(int n=1;n<200;n++)
    {total+=power*plusminus/factorial/(2.0*(n-1)+1.);
     power*=x*x;
     factorial*=n;
     plusminus*=-1;
     }
    return A*total;
   }
   public static void main(String arg[])
   {   double x[]={0.05,0.1,0.15};
       double t=1000.0;
       double T0=105.0;
       double Ti=10.0;
       double alpha=6e-7;
                  if_xj T=(double z[])->
                  {double x1=z[0];double t1=z[1];
                  double y1=x1/(2.0*Math.sqrt(alpha*t1));
                  double T1=T0+(Ti-T0)*erf(y1);
                  return T1;};
                  String s="";
                  for(int i=0;i<x.length;i++)
                  {   double z[]={x[i],t};
                              s+="x = "+x[i]+" t = "+t+" T = "+T.func(z)+"\n";
                  }
                  System.out.println(s);
   }
}
```

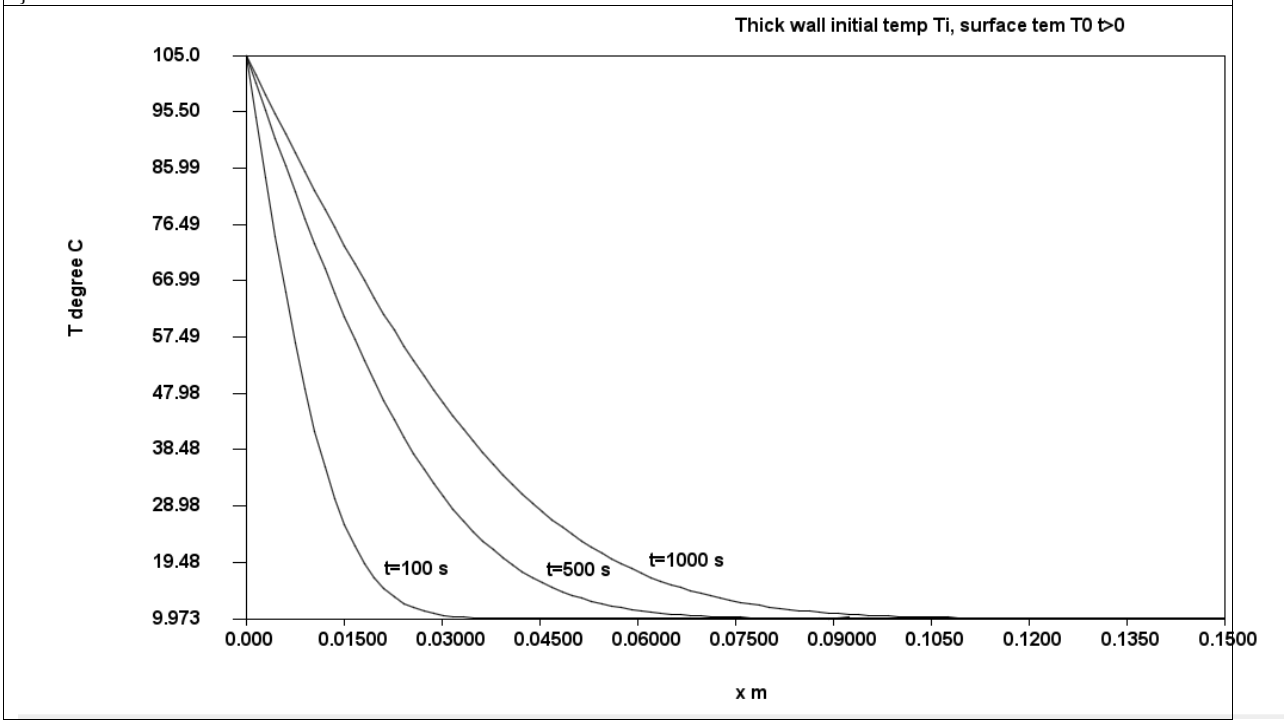
```
---------- Capture Output ----------
> "F:\co\java\bin\javaw.exe" erf1
x = 0.05 t = 1000.0 T = 24.146893951982747
x = 0.1 t = 1000.0 T = 10.369779626663998
x = 0.15 t = 1000.0 T = 10.00141572190877

> Terminated with exit code 0.
```

PROBLEM: A thick concrete wall (α=6e-7 m^2/s) is initially at a uniform temperature T_i=10 ℃. Suddenly its surface temperature is raised to T_0=105 ℃. Plot temperature profile for t=100 s,500 s and 1000 s

```
public class erf1
{  public static double erf(double x)
   {double A=2.0/Math.sqrt(Math.PI);
    double total=0;
    double plusminus=1;
    double power=x;
    double factorial=1;
    for(int n=1;n<200;n++)
    {total+=power*plusminus/factorial/(2.0*(n-1)+1.);
     power*=x*x;
     factorial*=n;
     plusminus*=-1;
     }
    return A*total;
   }
   public static void main(String arg[])
   {   double x[]=new double[100];
       double dx=0.15/(x.length-1);
       double t[]={100,500,1000};
       double T1[][]=new double[t.length][x.length];
       double T0=105.0;
       double Ti=10.0;
       double alpha=6e-7;
                  if_xj T=(double z[])->
                  {double x1=z[0];double t1=z[1];
                  double y1=x1/(2.0*Math.sqrt(alpha*t1));
                  double T2=T0+(Ti-T0)*erf(y1);
                  return T2;};
                  String s="";
                  for(int i=0;i<x.length;i++)
                  {   x[i]=dx*i;
                      for(int j=0;j<t.length;j++)
                      {
                              double z[]={x[i],t[j]};
                              T1[j][i]=T.func(z);
                      }
```

```
            }
            Plot p=new Plot(x,T1[0]);
            for(int i=1;i<t.length;i++)
            {p.addData(x,T1[i]);}
            p.setPlabel("Thick wall initial temp Ti, surface tem T0 t>0");
            p.setXlabel("x m");
            p.setYlabel("T degree C");
            p.plot();
  }
}
```



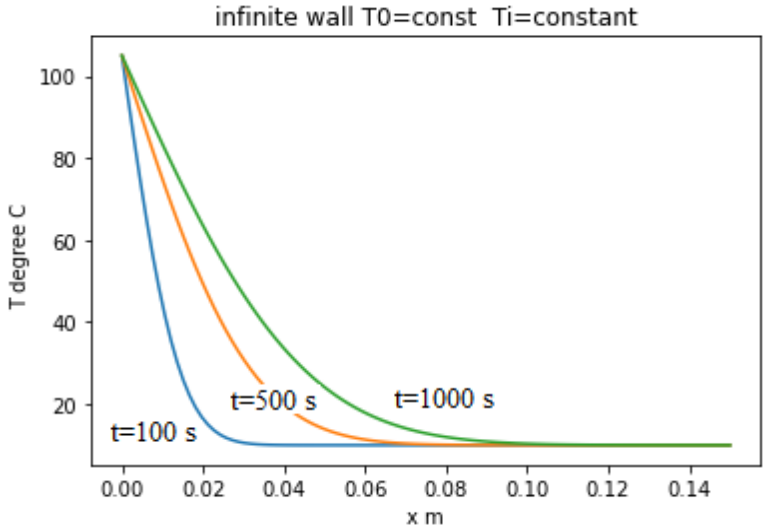
Python version:

```
from math import *
from f_xj import *
import numpy as np;
import matplotlib.pyplot as plt;

class f1(f_xj):
   def func(self,z):
      x1=z[0]
      t1=z[1]
      y1=x1/(2.0*sqrt(alpha*t1))
      T2=T0+(Ti-T0)*erf(y1)
      return T2

np1=100
x=[0 for x in range(np1)]
dx=0.15/(len(x)-1)
t=[100,500,1000]
np2=len(t)
T1= [[0 for x in range(np1)] for y in range(np2)];
T0=105.0
Ti=10.0
alpha=6e-7
s=""
f=f1()
for i in range(np2):
```

```
    for j in range(np1):
        x[j]=dx*j
        z=[x[j],t[i]]
        T1[i][j]=f.func(z)
plt.title('infinite wall T0=const  Ti=constant')
plt.xlabel('x m')
plt.ylabel('T degree C ')
for i in range(np2):
   plt.plot(x,T1[i])
```



PROBLEM: A thick bronze [α=0.86x10^{-5} m^2/s k=26 W/(mK) is initially at a uniform temperature T_i=250 ℃ . Suddenly surface temperature is reduced to T_a=25 ℃. Detemine the temperature 0.05 m from the surface 600 s after exposure.

```
ublic class HT_T_infinite
{  public static double erf(double x)
   {double A=2.0/Math.sqrt(Math.PI);
    double total=0;
    double plusminus=1;
    double power=x;
    double factorial=1;
    for(int n=1;n<200;n++)
    {total+=power*plusminus/factorial/(2.0*(n-1)+1.);
     power*=x*x;
     factorial*=n;
     plusminus*=-1;
     }
    return A*total;
   }
   public static void main(String arg[])
   {   double x[]={0.05};
       double t=600.0;
       double T0=25.0;
       double Ti=250.0;
       double alpha=0.86e-5;
                  if_xj T=(double z[])->
                  {double x1=z[0];double t1=z[1];
                  double y1=x1/(2.0*Math.sqrt(alpha*t1));
                  System.out.println("ksi="+y1);
                  double T1=T0+(Ti-T0)*erf(y1);
                  return T1;};
                  String s="";
                  for(int i=0;i<x.length;i++)
                  {   double z[]={x[i],t};
                              s+="x = "+x[i]+" t = "+t+" T = "+T.func(z)+"\n";
                  }
                  System.out.println(s);
   }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_T_infinite
ksi=0.3480287789935653
x = 0.05 t = 600.0 T = 109.91789350004223
> Terminated with exit code 0.
```

PROBLEM 6.3.2 SEMI-INFINITE WALL PROBLEM LAPLACE TRANSFORM SOLUTION, CONVECTIVE BOUNDARY

A second example: Semi-infinite medium at initially zero temperature. At t>0 boundary surface at x=0 is subjected to **convective heat transfer**

$$\frac{\partial^2 T^*(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T^*(x,t)}{\partial t}$$

with boundary condition:

$-k\frac{\partial T^*(x,t)}{\partial x} + h[T^*(x,t) - T_\infty] \quad x = 0 \quad t > 0$

$T^*(x,t) = T_i \quad x \to \infty \quad t > 0$

$T^*(x,t) = T_i$ at t=0

Shift of variables:

$T^*(x,t) = T^*(x,t) - T_i$

So the equation becomes:

$$\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T(x,t)}{\partial t}$$

$-k\frac{\partial T(x,t)}{\partial x} + h[T(x,t) + T_i - T_\infty] \quad x = 0 \quad t > 0$

$T(x,t) = 0 \quad x \to \infty \quad t > 0$

$T(x,t) = 0$ at t=0

Laplace transform of the differential equation:

$\frac{\partial^2 T(x,s)}{\partial x^2} - \frac{s}{\alpha}T(x,s) = 0 \quad 0 \le x \le \infty$

Boundary conditions:

$-k\frac{\partial T(x,s)}{\partial x} + hT(x,s) + h\frac{[T_i - T_\infty]}{s} \quad at\ x = 0$ (define H=h/k)

$-\frac{\partial T(x,s)}{\partial x} + HT(x,s) + H\frac{[T_i - T_\infty]}{s} \quad at\ x = 0$

$T(x,s) = 0 \quad x \to \infty$

Solution of this ordinary differential equation is:

$$T(x,s) = C_1 e^{-x\sqrt{\frac{s}{\alpha}}} + C_2 e^{x\sqrt{\frac{s}{\alpha}}}$$

$T(x,s) = 0 \quad x \to \infty$ therefore $C_2 = 0$

$$T(x,s) = C_1 e^{-x\sqrt{\frac{s}{\alpha}}}$$

$$\frac{\partial T(x,s)}{\partial x} = -C_1\sqrt{\frac{s}{\alpha}}e^{-x\sqrt{\frac{s}{\alpha}}}$$

$-\frac{\partial T(x,s)}{\partial x} + HT(x,s) + H\frac{[T_i - T_\infty]}{s} \quad at\ x = 0$

$C_1\sqrt{\frac{s}{\alpha}}e^{-x\sqrt{\frac{s}{\alpha}}} + HC_1 e^{-x\sqrt{\frac{s}{\alpha}}} + H\frac{[T_i - T_\infty]}{s} = 0$ at x=0

$$C_1\sqrt{\frac{s}{\alpha}}e^{-x\sqrt{\frac{s}{\alpha}}} + HC_1 e^{-x\sqrt{\frac{s}{\alpha}}} + H\frac{[T_i - T_\infty]}{s} = 0$$

$$C_1\left[\sqrt{\frac{s}{\alpha}} + H\right] + H\frac{[T_i - T_\infty]}{s} = 0$$

$$C_1 = -H\frac{[T_i - T_\infty]}{s\left[\sqrt{\frac{s}{\alpha}} + H\right]}$$

$$T(x,s) = H\frac{[T_\infty - T_i]}{s\left[\sqrt{\frac{s}{\alpha}} + H\right]}e^{-x\sqrt{\frac{s}{\alpha}}}$$

$$T(x,s) = H\sqrt{\alpha}\frac{[T_\infty - T_i]}{s\left[\sqrt{s} + H\sqrt{\alpha}\right]}e^{-x\sqrt{\frac{s}{\alpha}}}$$

51	$\frac{ae^{-k\sqrt{s}}}{s(\sqrt{s}+a)}$	$-e^{ak}e^{a^2t}erfc\left(a\sqrt{t}+\frac{k}{2\sqrt{t}}\right)+erfc\left(\frac{k}{2\sqrt{t}}\right)$

$$T(x,s) = [T^*(x,s) - T_i] = [T_\infty - T_i]\left[erfc\left(\frac{x}{2\sqrt{\alpha t}}\right) - e^{H\sqrt{\alpha}\frac{x}{\sqrt{\alpha}}}e^{H^2\alpha t}erfc\left(H\sqrt{\alpha}\sqrt{t}+\frac{x}{2\sqrt{\alpha t}}\right)\right]$$

$$T(x,s) = [T^*(x,s) - T_i] = [T_\infty - T_i]\left[erfc\left(\frac{x}{2\sqrt{\alpha t}}\right) - e^{Hx}a^{H^2\alpha t}erfc\left(H\sqrt{\alpha}\sqrt{t}+\frac{x}{2\sqrt{\alpha t}}\right)\right]$$

$$\frac{[T^*(x,s) - T_i]}{[T_\infty - T_i]} = \left[erfc\left(\frac{x}{2\sqrt{\alpha t}}\right) - e^{\frac{h}{k}x}e^{\frac{h^2}{k^2}\alpha t}erfc\left(\frac{h}{k}\sqrt{\alpha}\sqrt{t}+\frac{x}{2\sqrt{\alpha t}}\right)\right]$$

Problem 6.3.2 Java solution:

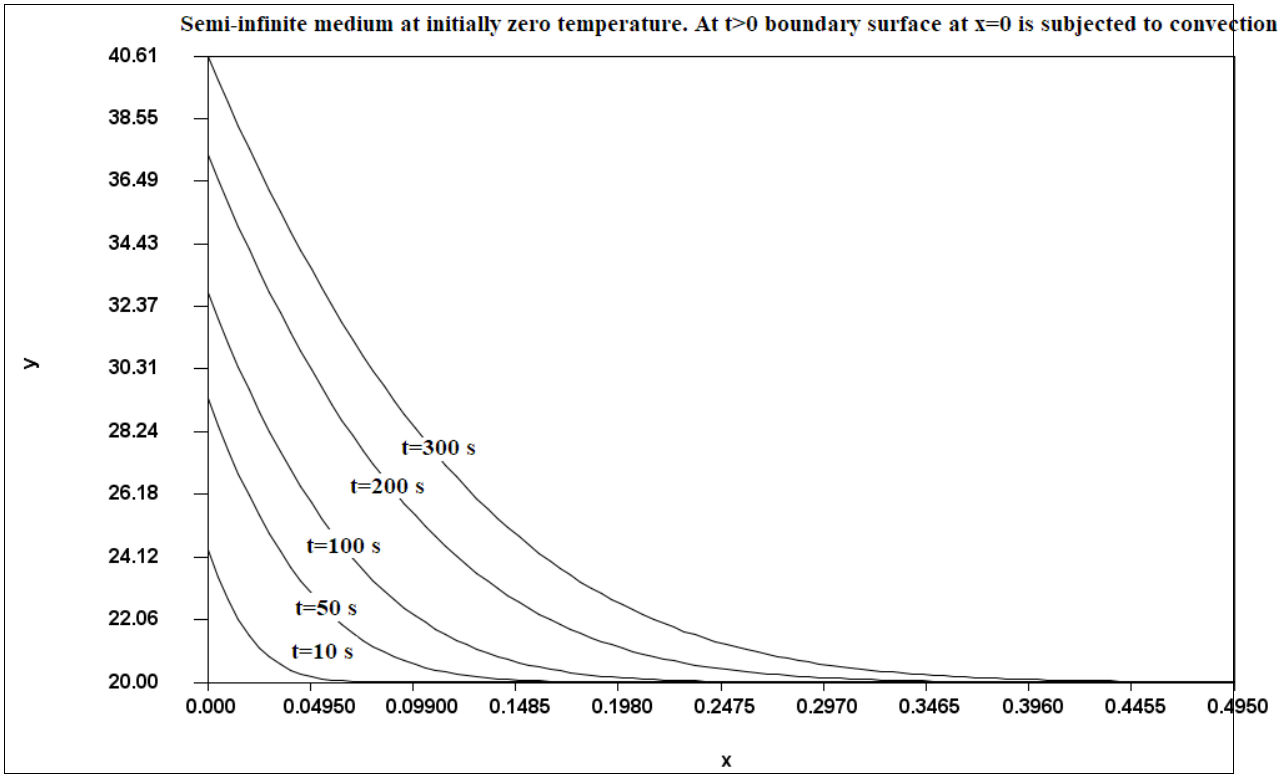
```
 import java.util.*;
public class HT_infinite_wall2
{
        //incomplete gamma function P(a,x)
public static double gammp(double a,double x)
{
        double gamser,gammcf;
        if (x < 0.0 || a <= 0.0) System.out.println("Invalid arguments in routine GAMMP");
        if (x < (a+1.0)) {
                gamser=gser(a,x);
                return gamser;
        } else {
                gammcf=gcf(a,x);
                return 1.0-gammcf;
        }
}

//incomplete gamma function Q(a,x)=1-P(a,x);
public static double gammq(double a,double x)
{
        double gamser,gammcf;
        if (x < 0.0 || a <= 0.0) System.out.println("Invalid arguments in routine GAMMQ");
        if (x < (a+1.0)) {
                gamser=gser(a,x);
                return 1.0-gamser;
        } else {
                gammcf=gcf(a,x);
                return gammcf;
        }
}
public static double gser(double a,double x)
{       int ITMAX=200;
        double EPS=3.0e-7;
        int n;
        double sum,del,ap;
        double gln=lngamma(a)[0];
        double gamser=0;
        if (x <= 0.0) {
                if (x < 0.0) System.out.println("x less than 0 in routine GSER");
                gamser=0.0;
                return gamser;
        } else {
                ap=a;
                del=sum=1.0/a;
                for (n=1;n<=ITMAX;n++) {
                        ap += 1.0;
                        del *= x/ap;
                        sum += del;
                        if (Math.abs(del) < Math.abs(sum)*EPS) {
                                gamser=sum*Math.exp(-x+a*Math.log(x)-(gln));
                                return gamser;
```

```
                                        }
                                }
                                //System.out.println("a too large, ITMAX too small in routine GSER");
                                return gamser;
                        }
        }
        public static double gcf(double a,double x)
        {
                        int ITMAX=200;
                        double EPS=3.0e-7;
                        int n;
                        double gln;
                        double gammcf=0;
                        double gold=0.0,g,fac=1.0,b1=1.0;
                        double b0=0.0,anf,ana,an,a1,a0=1.0;
                        gln=lngamma(a)[0];
                        a1=x;
                        for (n=1;n<=ITMAX;n++) {
                                an=(double) n;
                                ana=an-a;
                                a0=(a1+a0*ana)*fac;
                                b0=(b1+b0*ana)*fac;
                                anf=an*fac;
                                a1=x*a0+anf*a1;
                                b1=x*b0+anf*b1;
                                if (a1!=0) {
                                        fac=1.0/a1;
                                        g=b1*fac;
                                        if (Math.abs((g-gold)/g) < EPS) {
                                                gammcf=Math.exp(-x+a*Math.log(x)-gln)*g;
                                                return gammcf;
                                        }
                                        gold=g;
                                }
                        }
                        //System.out.println("a too large, ITMAX too small in routine GCF");
           return gammcf;
        }
        //error function
        public static double  erf(double x)
        {
          //error function
          return x < 0.0 ? -gammp(0.5,x*x) : gammp(0.5,x*x);
        }
        //natural logarithm of gamma function
                        public static double [] lngamma(double x)
           {//logaithm of gamma function
           //xx[0] value of logarithm of gamma function
           //xx[1] sign of gamma functon
                        double xx[]=new double[2];
           double result;
           double a;
           double b;
           double c;
           double p;
           double q;
           double u;
           double w;
           double z;
           int i;
           double logpi;
           double ls2pi;
           double tmp;
           double sgngam = 1;
           logpi = 1.14472988584940017414;
           ls2pi = 0.91893853320467274178;
           if( x<-34.0 )
           {
              q = -x;
              double yy[]=lngamma(q);
              w = yy[0];
              tmp=yy[1];
              p = Math.floor(q);
              i = (int)Math.round(p);
              if( i%2==0 )
              {
                 sgngam = -1;
```

```
    }
    else
    {
       sgngam = 1;
    }
    z = q-p;
    if( z>0.5 )
    {
       p = p+1;
       z = p-q;
    }
    z = q*Math.sin(Math.PI*z);
    result = logpi-Math.log(z)-w;
    xx[0]=result;
    xx[1]=sgngam;
    return xx;
 }
 if( x<13 )
 {
    z = 1;
    p = 0;
    u = x;
    while(u>=3)
    {
       p = p-1;
       u = x+p;
       z = z*u;
    }
    while(u<2)
    {
       z = z/u;
       p = p+1;
       u = x+p;
    }
    if( z<0 )
    {
       sgngam = -1;
       z = -z;
    }
    else
    {
       sgngam = 1;
    }
    if( u==2 )
    {
       result = Math.log(z);
       //return result;
       xx[0]=result;
       xx[1]=sgngam;
       return xx;
    }
    p = p-2;
    x = x+p;
    b = -1378.25152569120859100;
    b = -38801.6315134637840924+x*b;
    b = -331612.992738871184744+x*b;
    b = -1162370.97492762307383+x*b;
    b = -1721737.00820839662146+x*b;
    b = -853555.664245765465627+x*b;
    c = 1;
    c = -351.815701436523470549+x*c;
    c = -17064.2106651881159223+x*c;
    c = -220528.590553854454839+x*c;
    c = -1139334.44367982507207+x*c;
    c = -2532523.07177582951285+x*c;
    c = -2018891.41433532773231+x*c;
    p = x*b/c;
    result = Math.log(z)+p;
    //return result;
    xx[0]=result;
    xx[1]=sgngam;
    return xx;
 }
 q = (x-0.5)*Math.log(x)-x+ls2pi;
 if( x>100000000 )
 {
    result = q;
```

```
        //return result;
        xx[0]=result;
        xx[1]=sgngam;
        return xx;
    }
    p = 1/(x*x);
    if( x>=1000.0 )
    {
        q = q+((7.936507936507936507936507936507936507936*0.0001*p-2.777777777777777777778*0.001)*p+0.08333333333333333333333)/x;
    }
    else
    {
        a = 8.11614167470508450300*0.0001;
        a = -5.95061904284301438324*0.0001+p*a;
        a = 7.9365034045771694394 5*0.0001+p*a;
        a = -2.77777777730099687205*0.001+p*a;
        a = 8.33333333333331927722*0.01+p*a;
        q = q+a/x;
    }
    xx[0]= q;
    xx[1]=sgngam;
    return xx;
    }
//complimentary error function
public static double  erfc(double x)
{
                return x < 0.0 ? 1.0+gammp(0.5,x*x) : gammq(0.5,x*x);
}
public static double T(double x,double t,double Tinf,double alpha,double H)
{
//H = h/k
double x1=x/Math.sqrt(4.0*alpha*t);
double TT=Tinf*(erfc(x1)-Math.exp(H*x+H*H*alpha*t)*erfc(H*Math.sqrt(alpha*t)+x1));
return TT;
}
    public static void main (String args[])
    {
      double alpha=4.0e-5;
      double H=2.0;
      double x[]=new double[100]; //m
      double t[]={10.0,50.0,100.0,200.0,300.0};//s
      double T1=20.0;// degree C;
      double Tinf=100.0;
      double T[][]=new double[t.length][100];
      for(int i=0;i<t.length;i++)
      {for(int j=0;j<x.length;j++)
       {
                    x[j]=0.5*j/100.0;
                    T[i][j]=T(x[j],t[i],Tinf,alpha,H)+T1;}
      }
      Plot pp=new Plot(x,T[0]);
      for(int i=1;i<t.length;i++)
      {pp.addData(x,T[i]); }
       pp.plot();
    }
}
```

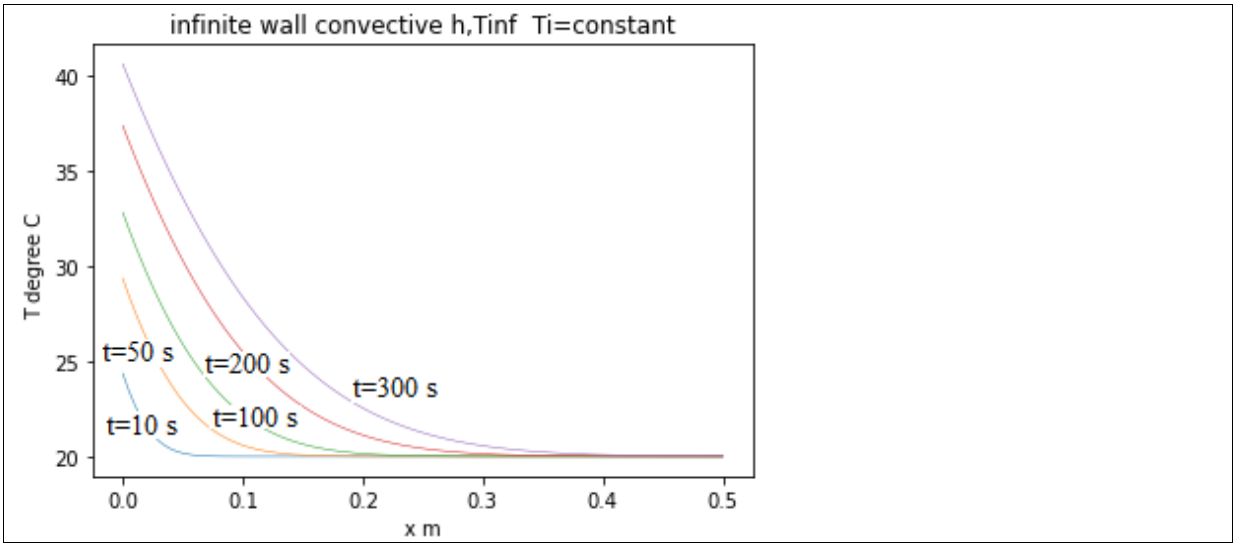


Problem 6.3.2 Python solution:

```
 from math import *
from f_xj import *
import numpy as np;
import matplotlib.pyplot as plt;
# infinite wall convective heat flow at the wall surface at t>0 seconds
class f1(f_xj):
    def __init__(self,Tinf,alpha,H,Ti):
        self.name = 'HT_infinite_wall2'
        self.Tinf=Tinf
        self.alpha=alpha
        self.H=H
        self.Ti=Ti

    def func(self,z):
         x1=z[0]
         t1=z[1]
         x2=x1/sqrt(4.0*alpha*t1)
         T2=Tinf*(erfc(x2)-exp(H*x1+H*H*alpha*t1)*erfc(H*sqrt(alpha*t1)+x2))+Ti
         return T2

np1=100
x=[0 for x in range(np1)]
dx=0.5/(len(x)-1)
alpha=4.0e-5
H=2.0
Tinf=100.0
t=[10.0,50.0,100.0,200.0,300.0]
np2=len(t)
T1= [[0 for x in range(np1)] for y in range(np2)];
T0=105.0
Ti=20.0
s=""
T=f1(Tinf,alpha,H)
for i in range(np2):
    for j in range(np1):
        x[j]=dx*j
        z=[x[j],t[i]]
        T1[i][j]=T.func(z)
plt.title('infinite wall convective h,Tinf  Ti=constant')
plt.xlabel('x m')
plt.ylabel('T degree C ')
for i in range(np2):
    plt.plot(x,T1[i],linewidth=0.5)
```



PROBLEM: A thick bronze [α=0.88x10^{-5} m^2/s k=26 W/(mK) is initially at a uniform temperature T_i=200 ℃ . Suddenly surface is exposed to a coolantT_a=30 ℃. Assuming that the convective heat transfer coefficient h=150 W/(m^{2}K). Detemine the temperature 0.05 m from the surface 900 s after exposure.

```
public class HT_Tinf_h_infinite
{  public static double erf(double x)
   {double A=2.0/Math.sqrt(Math.PI);
    double total=0;
    double plusminus=1;
    double power=x;
    double factorial=1;
    for(int n=1;n<200;n++)
    {total+=power*plusminus/factorial/(2.0*(n-1)+1.);
     power*=x*x;
     factorial*=n;
     plusminus*=-1;
     }
    return A*total;
   }
   public static double erfc(double x)
   {return 1.0-erf(x);}
   public static void main(String arg[])
   {   double x[]={0.05};
       double t=900.0;
       double Tinf=30;
       double h=100.0;
       double k=25.0;
       double Ti=200.0;
       double alpha=0.88e-5;
                  if_xj T=(double z[])->
                  {double x1=z[0];double t1=z[1];
                  double y1=x1/(2.0*Math.sqrt(alpha*t1));
                  double y2=h/k*Math.sqrt(alpha);
                  double y3=y2*Math.sqrt(t1);
                  double T1=Ti+(Tinf-Ti)*(erfc(y1)-Math.exp(h/k*x1+y2*y2*t1)*erfc(y1+y3));
                  return T1;};
                  String s="";
                  for(int i=0;i<x.length;i++)
                  {   double z[]={x[i],t};
                              s+="x = "+x[i]+" t = "+t+" T = "+T.func(z)+"\n";
                  }
                  System.out.println(s);
   }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_Tinf_h_infinite
x = 0.05 t = 900.0 T = 169.1761791261158


> Terminated with exit code 0.
```

PROBLEM:

A thick wood Wall[α=0.82x10^{-7}m^2/s, k=0.15 W/(mK)] is initially at a uniform temperature of T_i=20 C. The wood may ignite at T=400C. If the surface is exposed to hot gases at T_{inf}=500 C, and the heat transfer coefficient between the gas and the surface is h=45 W/(m^{2}K). How long will it take for the surface (x=0) of the wood to ignition temperature[T=400C]

$$\frac{[T^*(x,s)-T_i]}{[T_\infty - T_i]} = \left[erfc\left(\frac{x}{2\sqrt{\alpha t}}\right) - e^{\frac{h}{k}x} e^{\frac{h^2}{k^2}\alpha t} erfc\left(\frac{h}{k}\sqrt{\alpha}\sqrt{t} + \frac{x}{2\sqrt{\alpha t}}\right)\right]$$

x=0 $erfc(0) = 1$

$$\frac{[400-20]}{[500-20]} = 1 - e^{z^2} erfc(z)$$

$$e^{z^2} erfc(z) - 0.20833 = 0$$

$$z = \frac{h}{k}\sqrt{\alpha}\sqrt{t} = 2.528226516$$

0.791667		
0.208333		
z	2.528226516	
	0.000394155	
h=	45	W/(m^2K)
k	0.15	
α	8.20E-08	
h/k*α^0.5	8.59E-02	
t^0.5=	2.94E+01	
t	8.66E+02	s

PROBLEM 6.3.3 SEMI-INFINITE WALL PROBLEM LAPLACE TRANSFORM SOLUTION, CONSTANT SURFACE HEAT FLUX BOUNDARY

Semi-infinite medium: **Constant surface heat flux**

$$\frac{\partial^2 T^*(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T^*(x,t)}{\partial t}$$

with boundary condition:

$-k\frac{\partial T^*(x,t)}{\partial x} = q_0 \quad x = 0 \quad t > 0$

$T^*(x,t)$= T_i $\quad x \to \infty \quad$ t > 0

T^* (x,t)= T_i $\quad$ at t=0

Shift of variables:

$T^*(x,t)=T^*(x,t) - T_i$

So the equation becomes:

$$\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T(x,t)}{\partial t}$$

$-k\frac{\partial T(x,t)}{\partial x} = q_0 \quad x = 0 \quad t > 0$

$x = 0 \quad t > 0$

$T(x,t)$= 0 $\quad x \to \infty \quad$ t > 0

$T(x,t)$= 0 $\quad$ at t=0

Laplace transform of the differential equation:

$\frac{\partial^2 T(x,s)}{\partial x^2} - \frac{s}{\alpha}T(x,s) = 0 \quad 0 \le x \le \infty$

Boundary conditions:

$-k\frac{\partial T(x,t)}{\partial x}=\frac{q_0}{s}\quad x=0$

(define H=h/k)

$T(x,s)=0\quad x\to\infty\quad t>0$

Solution of this ordinary differential equation is:

$$T(x,s)=C_1e^{-x\sqrt{\frac{s}{\alpha}}}+C_2e^{x\sqrt{\frac{s}{\alpha}}}$$

$T(x,s)=0\ \ x\to\infty$ therefore $C_2=0$

$$T(x,s)=C_1e^{-x\sqrt{\frac{s}{\alpha}}}$$

$$\frac{\partial T(x,s)}{\partial x}=-C_1\sqrt{\frac{s}{\alpha}}e^{-x\sqrt{\frac{s}{\alpha}}}$$

$kC_1\sqrt{\frac{s}{\alpha}}e^{-x\sqrt{\frac{s}{\alpha}}}=\frac{q_0}{s}\quad x=0\quad t>0$

$kC_1\sqrt{\frac{s}{\alpha}}=\frac{q_0}{s}$

$$C_1=\frac{q_0\sqrt{\alpha}}{ks^{3/2}}$$

So the solution is:

$$T(x,s)=\frac{q_0\sqrt{\alpha}}{ks^{3/2}}e^{-x\sqrt{\frac{s}{\alpha}}}$$

Case 44 in the table:

44	$\frac{e^{-k\sqrt{s}}}{s^{3/2}}\quad k\geq 0$	$2\sqrt{\frac{t}{\pi}}\exp\left(-\frac{k^2}{4t}\right)-k*erfc\left(\frac{k}{2\sqrt{t}}\right)$

So $k=\frac{x}{\sqrt{\alpha}}$

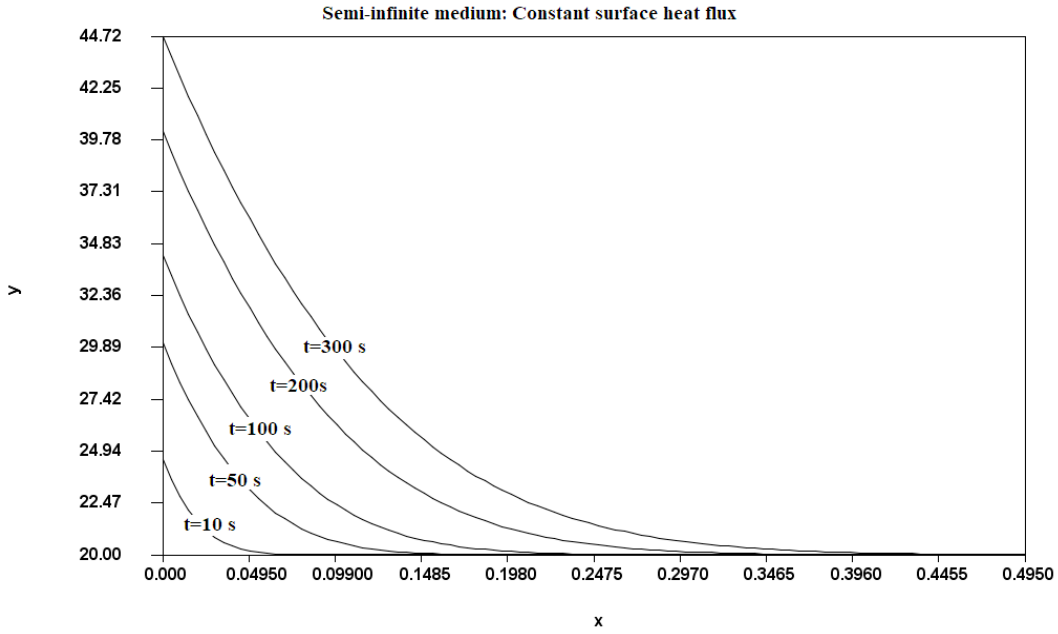
The inversion of Laplace transform results:

$$T(x,t)=2\frac{q_0}{k}\sqrt{\frac{\alpha t}{\pi}}\exp\left(-\frac{x^2}{4\alpha t}\right)-\frac{q_0}{k}x*erfc\left(\frac{x}{2\sqrt{\alpha t}}\right)$$

$$T(x,t)=T^*(x,t)-T_i=2\frac{q_0}{k}\sqrt{\frac{\alpha t}{\pi}}\exp\left(-\frac{x^2}{4\alpha t}\right)-\frac{q_0}{k}x*erfc\left(\frac{x}{2\sqrt{\alpha t}}\right)$$

```
class HT_infinite_wall3
{   //Function  is written by using lambda variables
    public static void main (String args[])
    {
      double alpha=4.0e-5;
      double q0=10000.0;  //W/m^2
      double k=50.0; //W/mK;
      double t0=20;
     if_xj f1=(double x[])->2.0*q0*Math.sqrt(alpha*x[0]/Math.PI)/k*Math.exp(-x[1]*x[1]/(4.0*alpha*x[0]))-
q0*x[1]/k*Mathd.erfc(x[1]/(2.0*Math.sqrt(alpha*x[0])))+t0;
      double xx[]=new double[100]; //m
      double t[]={10.0,50.0,100.0,200.0,300.0};//s
      double T1=20.0;// degree C;
      double T[][]=new double[t.length][100];
      for(int i=0;i<t.length;i++)
      {for(int j=0;j<xx.length;j++)
       {   xx[j]=0.5*j/100.0;
                  double y[]={t[i],xx[j]};
```

```
                    T[i][j]=f1.func(y);}
      }
      Plot pp=new Plot(xx,T[0]);
      for(int i=1;i<t.length;i++)
      {pp.addData(xx,T[i]); }
       pp.plot();
    }
}
```



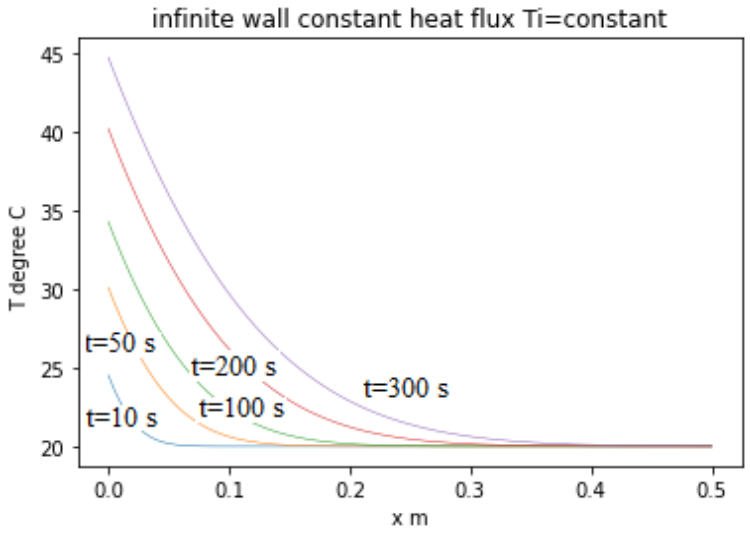
Python version:

```
from math import *
from f_xj import *
import numpy as np;
import matplotlib.pyplot as plt;
# infinite wall constant heat flux q0 at the wall surface at t>0 seconds
class f1(f_xj):
   def __init__(self,Ti,alpha,k,q0):
      self.name = 'HT_infinite_wall3'
      self.q0=q0
      self.alpha=alpha
      self.k=k
      self.Ti=Ti

   def func(self,z):
      x=z[0]
      t=z[1]
      T=2.0*q0*sqrt(alpha*t/pi)/k*exp(-x*x/(4.0*alpha*t))-q0*x/k*erfc(x/(2.0*sqrt(alpha*t)))+Ti
      return T

np1=100
x=[0 for x in range(np1)]
dx=0.5/(len(x)-1)
alpha=4.0e-5
q0=10000.0
k=50.0
t=[10.0,50.0,100.0,200.0,300.0]
np2=len(t)
T1= [[0 for x in range(np1)] for y in range(np2)];
Ti=20.0
s=""
T=f1(Ti,alpha,k,q0)
for i in range(np2):
   for j in range(np1):
      x[j]=dx*j
      z=[x[j],t[i]]
      T1[i][j]=T.func(z)
plt.title('infinite wall constant heat flux Ti=constant')
plt.xlabel('x m')
plt.ylabel('T degree C ')
```

```
for i in range(np2):
    plt.plot(x,T1[i],linewidth=0.5)
```



7.0 INTEGRAL TRANSFORM METHODS

When nonhomogenities in the boundary layers existed, methods like seperation of variables are not always good enough to give us homogeneous solutions, eventhough several methods are existed to convert the problem to the homogeneous one. Integral transform method is derived from seperation of variable methods. Integral transform and inverse integral transform is derived from arbitrary function in terms of eigenfunctions of the corresponding eigenvalue problem. Consider the following eigenvalue problem:

$\nabla^2\varphi(r) + \lambda^2\varphi(r) = 0$ in region R

With boundary condition:

$k_i \frac{\partial\varphi(r_i)}{\partial n_i} + h_i\varphi(r_i) = 0$ on the boundary S_i

Assuming that eigenfunction $\varphi(\lambda_m, r)$ of this eigenvalue problem satisfy the orthogonality conditions. The general can be applied.

$$\int_R \varphi(\lambda_m, r)\varphi(\lambda_n, r)dv = \begin{cases} 0 & m \neq n \\ N(\lambda_m) & m = n \end{cases}$$

$$N(\lambda_m) = \int_R [\varphi(\lambda_m, r)]^2 dv$$

Now the representation of the function can be given as:

$$T(r,t) = \sum_{m=1}^{\infty} C_m(t)\varphi(\lambda_m, r)$$

By applying the operator $\int_R \varphi(\lambda_m, r)dv$ into the both side of the equation and considering orthogonality condition

$$\int_R \varphi(\lambda_m, r)T(r,t)dv = C_m(t)\int_R [\varphi(\lambda_m, r)]^2 dv = C_m(t)N(\lambda_m)$$

$$C_m(t) = \frac{1}{N(\lambda_m)}\int_R \varphi(\lambda_m, r)T(r,t)dv$$

This expression is introduced into the $T(r,t)$ equation by spliting it into two parts

Inversion formula: $T(r,t) = \sum_{m=1}^{\infty} \frac{\varphi(\lambda_m, r)}{N(\lambda_m)}\bar{T}(\lambda_m, t)$

Integral transform: $\bar{T}(\lambda_m, t) = \int_R \varphi(\lambda_m, r)T(r,t)dv$ where $\bar{T}(\lambda_m, t)$ is called integral transform of the function $T(r,t)$

Now let us look at the heat transfer problem:

$$\nabla^2 T(r,t) + \frac{1}{k}g(r,t) = \frac{1}{\alpha}\frac{\partial T(r,t)}{\partial t}$$ in region R

With the boundary conditions:

$$k_i \frac{\partial T(r_i,t)}{\partial n_i} + h_i T(r_i,t) = f_i(r_i,t)$$ on boundary S_i t>0

$$T(r,t) = F(r)$$ for t=0 in region R

Integrated form of the equation:

$$\int_R \varphi(\lambda_m, r)\nabla^2 T(r,t)dv + \frac{1}{k}\int_R \varphi(\lambda_m, r)g(r,t)dv = \frac{1}{\alpha}\int_R \varphi(\lambda_m, r)\frac{\partial T(r,t)}{\partial t}dv$$

Integral transform of this expression:

$$\int_R \varphi(\lambda_m, r)\nabla^2 T(r,t)dv + \frac{1}{k}\bar{g}(\lambda_m, t) = \frac{1}{\alpha}\frac{\partial \bar{T}(\lambda_m, t)}{\partial t}$$

Where $\bar{g}(\lambda_m, t) = \int_R \varphi(\lambda_m, r)g(r,t)dv$ and $\bar{T}(\lambda_m, t) = \int_R \varphi(\lambda_m, r)T(r,t)dv$

The integral $\int_R \varphi(\lambda_m,r)\nabla^2 T(r,t)dv$ can be evaluated making use of Green's theorem

$$\int_R \varphi(\lambda_m,r)\nabla^2 T(r,t)dv = \int_R T\nabla^2\varphi(\lambda_m,r)\,\mathrm{dv} + \sum_{i=1}^{N}\int_{S_i}\left(\varphi(\lambda_m,r)\frac{\partial T(r,t)}{\partial n_i} - T(r,t)\frac{\partial\varphi(\lambda_m,r)}{\partial n_i}\right)ds_i$$

Where s_i continious boundaries of region R. The integral $\int_R T\nabla^2\varphi(\lambda_m,r)\,\mathrm{dv}$ is evaluated by writing equation $\nabla^2\varphi(r)+\lambda^2\varphi(r)=0$ in region R for the eigenfunction $\varphi(\lambda_m,r)$, multiplying both sides with T(r,t) and integrating in region R and utilizing integral transform.

$\int_R T(r,t)\nabla^2\varphi(\lambda_m,r)dv + \lambda_m^2\int_R T(r,t)\varphi(\lambda_m,r)dv = 0$

The surface integral $\int_{S_i}\left(\varphi(\lambda_m,r)\frac{\partial T(r,t)}{\partial n_i} - T(r,t)\frac{\partial\varphi(\lambda_m,r)}{\partial n_i}\right)ds_i$ is evaluated by using boundary condition $k_i\frac{\partial T(r_i,t)}{\partial n_i} + h_i T(r_i,t) = f_i(r_i,t)$ on boundary S_i t>0

$\left(\varphi(\lambda_m,r)\frac{\partial T(r,t)}{\partial n_i} - T(r,t)\frac{\partial\varphi(\lambda_m,r)}{\partial n_i}\right) = \frac{\varphi(\lambda_m,r_i)}{k_i}f_i(r_i,t)$ so equation becomes

$\int_R \varphi(\lambda_m,r)\nabla^2 T(r,t)dv = -\lambda_m^2\int_R T(r,t)\varphi(\lambda_m,r)dv + \sum_{i=1}^{N}\int_{S_i}\left(\frac{\varphi(\lambda_m,r_i)}{k_i}f_i(r_i,t)\right)ds_i$ Substituting this equation into $\int_R \varphi(\lambda_m,r)\nabla^2 T(r,t)dv + \frac{1}{k}\bar{g}(\lambda_m,t) = \frac{1}{\alpha}\frac{\partial\bar{T}(\lambda_m,t)}{\partial t}$ equation becomes

$\frac{\partial\bar{T}(\lambda_m,t)}{\partial t} + \alpha\lambda_m^2\bar{T}(\lambda_m,t) = A(\lambda_m,t)$ where $A(\lambda_m,t) = \frac{\alpha}{k}g(\lambda_m,t) + \alpha\sum_{i=1}^{N}\int_{S_i}\left(\frac{\varphi(\lambda_m,r_i)}{k_i}f_i(r_i,t)\right)ds_i$

Integral transform of the boundary condition $T(r,t)=F(r)$ for t=0 in region R becomes

$\bar{T}(\lambda_m,t) = \int_R \varphi(\lambda_m,r)F(r)dv = \bar{F}(\lambda_m)$ for t=0

Our integral transform equation then becomes:

$$\bar{T}(\lambda_m,t) = \exp(-\alpha\lambda_m^2 t)\left[\bar{F}(\lambda_m) + \int_0^t \exp(-\alpha\lambda_m^2 t)A(\lambda_m,t)dt\right]$$

Introducing integral transform into the inversion formula gives:

$$T(r,t) = \sum_{m=1}^{\infty}\frac{1}{N(\lambda_m)}\exp(-\alpha\lambda_m^2 t)\varphi(\lambda_m,r)\left[\bar{F}(\lambda_m) + \int_0^t \exp(-\alpha\lambda_m^2 t)A(\lambda_m,t)dt\right]$$

Where

$$A(\lambda_m,t) = \frac{\alpha}{k}g(\lambda_m,t) + \alpha\sum_{i=1}^{N}\int_{S_i}\left(\frac{\varphi(\lambda_m,r_i)}{k_i}f_i(r_i,t)\right)ds_i$$

$$\bar{F}(\lambda_m) = \int_R \varphi(\lambda_m,r)F(r)dv$$

$$\bar{g}(\lambda_m,t) = \int_R \varphi(\lambda_m,r)g(r,t)dv$$

$$N(\lambda_m) = \int_R [\varphi(\lambda_m,r)]^2 dv$$

Note: in the boundary equation $k_i\frac{\partial T(r_i,t)}{\partial n_i} + h_i T(r_i,t) = f_i(r_i,t)$ on boundary S_i t>0 if $k_i = 0$ (which can be happened when a temperature type of boundary is given) the the term $\frac{\varphi(\lambda_m,r_i)}{k_i}$ will be replaced by $-\frac{1}{h_i}\frac{\partial\varphi(\lambda_m,r_i)}{\partial n_i}$

Example problem: Consider the following two dimensional (x,t) heat conduction problem:

$\frac{\partial^2 T(x,t)}{\partial x^2} + \frac{1}{k}g(x,t) = \frac{1}{\alpha}\frac{\partial T(x,t)}{\partial t}$ in $0\le x\le L$ t>0

$\frac{\partial T(x,t)}{\partial x} = 0$ at x=0 t>0

$T(x,t)$=0 at x=l t>0
$T(x,t) = F(x)$ at t=0 $\quad 0 \le x \le L$
Solution of the problem then has a form:

$$T(r,t) = \sum_{m=1}^{\infty} \frac{1}{N(\beta_m)} \exp(-\alpha\lambda_m^2 t) X(\beta_m, x) \left[\bar{F}(\beta_m) + \frac{\alpha}{k} \int_0^t \bar{g}(\beta_m, t) dt \right]$$

Where $X(\beta_m, x) = \cos(\beta_m x)$ $\quad \frac{1}{N(\beta_m)} = \frac{2}{L}$ and $\cos(\beta_m L) = 0 \rightarrow \beta_m = \frac{(2m-1)\pi}{L}$ m=1..N

$$T(x,t) = \frac{2}{L} \sum_{m=1}^{\infty} \exp(-\alpha\lambda_m^2 t) \cos(\beta_m x) \left[\int_{x=0}^{L} F(x) \cos(\beta_m x) dx + \frac{\alpha}{k} \int_{t=0}^{t} \int_{x=0}^{L} \exp(\alpha\lambda_m^2 t) \cos(\beta_m x) g(x,t) \cos(\beta_m x) dx dt \right]$$

Note: equation involve double integration in order to solve double integration numerical analysis info about numerical information solution is given in below info box

MULTIDIMENSIONAL INTEGRATION BY PARAMETRIC METHODS

A two dimensional integration

$$I(D) = \int\int_D f(x,y) dy dx$$

Where D is a region in xy plane. If this region is defined as $a \le x \le b$ and $c(x) \le y \le d(x)$, one way two calculate two dimensional integration is to repeat one dimensional integration twice. In the first step intehration function g(x) is calculated as an integration in y dimension and then function g(x) is solved to get integration result.

$$I = \int_a^b \int_{c(x)}^{d(x)} f(x,y) dy dx$$

$$g(x) = \int_{c(x)}^{d(x)} f(x,y) dy dx \quad a \le x \le b$$

$$= \int_a^b g(x) dx$$

Any integration formulation can be used for this purpose. All it has to be done is to evaluate integration Formula twice once for each independent variable. As an example Gauss-Legendre formulation will be used to evaluate double integration. Two code is defined. In the first one Gauss-Legendre coefficients are defined as constants, in the second one they are calculated through Legendre polynomials. In the two dimensional integration x dimesion limits are still given as constant values, but y dimension limits c(x) and d(x) should be given as functions for proper evaluation of the integral.

$$I = \int_a^b \int_{c(x)}^{d(x)} f(x,y) dy dx = \int_a^b \frac{d(x) - c(x)}{2} \sum_{j=1}^{n} C_{n,j} f(x, \frac{(d(x) - c(x)) r_{n,j} + (d(x) + c(x))}{2} dx$$

In this equation $r_{n,j}$ root values and $C_{n,j}$ are equation coefficients. These values can be calculated from Legendre polynomials.

Now we can create numerical solution for the given final formulation. (HT_int_tr1.java)

```
//=====================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
```

```
// and differentiation (derivative) functions
// Gauss-Legendre integration with two variables
// constant number of coefficients (60 coefficients)
// and roots
// Dr. Turhan Coban
// =====================================================
import java.io.*;
abstract class fdi extends Mathd
{
public double a,b;
public fdi(double ai,double bi)
{a=ai;b=bi;}
public fdi()
{a=0;b=1;}
public void setab(double ai,double bi)
{a=ai;b=bi;}
abstract public  double c(double x);
abstract public  double d(double x);
abstract public  double f(double x[]);
}
class f1 extends fdi
{ int m;
  double L,t;
  double beta;
  double alpha;
  if_xj g;
  public f1( if_xj gi,int mi,double alphai,double Li,double ti)
  {super(0.0,Li);g=gi;alpha=alphai;m=mi;L=Li;t=ti;beta=(2.0*m-1.0)*Math.PI/L;}
  //lower integration limit function
  public double c(double x)
  {return 0.0;}
  //upper integration limit function
  public double d(double x)
  {return L;}
  public double f(double x[])
  {
  //function required to integrate

  double ff=exp(alpha*beta*beta*x[1])*g.func(x)*cos(beta*x[0]);
  return ff;
  }
}
class HT_int_tr1
{
public static double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{   //calculates legendre gauss-coefficients as coefficients of the integral
            //for n terms
            double EPS=3.0e-15;
            int m,j,i;
            double z1,z,xm,xl,pp,p3,p2,p1;
   //double x[]=new double[n];
   //double w[]=new double[n];
   double a[][]=new double[2][n];//a[0][i]=x[i]   a[1][i]=w[i]
            m=(n+1)/2;
            xm=0.5*(x2+x1);
            xl=0.5*(x2-x1);
            for (i=1;i<=m;i++)  {
                        z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                        do {
                                    p1=1.0;
                                    p2=0.0;
                                    for (j=1;j<=n;j++) {
                                                p3=p2;
                                                p2=p1;
                                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                                    }
                                    pp=n*(z*p1-p2)/(z*z-1.0);
                                    z1=z;
                                    z=z1-p1/pp;
                        } while (Math.abs(z-z1) > EPS);
                        a[0][i-1]=xm-xl*z;
                        a[0][n-i]=xm+xl*z;
                        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                        a[1][n-i]=a[1][i-1];
                                    }
return a;
}
```

```
public static double I(fdi f)
{
//integral f(x)dx
//integral of a function by using gauss-legendre quadrature
//coefficients are pre-calculated for 60 terms for [-1,1]
//band then utilises variable transform
 int i,j;
 int n=60;
 double r[],c[];
 r=new double[n];
 c=new double[n];
 r[ 0] = .15532579626752470000E-02;
 r[ 1] = .81659383601264120000E-02;
 r[ 2] = .19989067515846230000E-01;
 r[ 3] = .36899976285362850000E-01;
 r[ 4] = .58719732103973630000E-01;
 r[ 5] = .85217118808615820000E-01;
 r[ 6] = .11611128394758690000E+00;
 r[ 7] = .15107475260334210000E+00;
 r[ 8] = .18973690850537860000E+00;
 r[ 9] = .23168792592899010000E+00;
 r[10] = .27648311523095540000E+00;
 r[11] = .32364763723456090000E+00;
 r[12] = .37268153691605510000E+00;
 r[13] = .42306504319570830000E+00;
 r[14] = .47426407872234120000E+00;
 r[15] = .52573592127765890000E+00;
 r[16] = .57693495680429170000E+00;
 r[17] = .62731846308394490000E+00;
 r[18] = .67635236276543910000E+00;
 r[19] = .72351688476904450000E+00;
 r[20] = .76831207407100990000E+00;
 r[21] = .81026309149462140000E+00;
 r[22] = .84892524739665800000E+00;
 r[23] = .88388871605241310000E+00;
 r[24] = .91478288119138420000E+00;
 r[25] = .94128026789602640000E+00;
 r[26] = .96310002371463720000E+00;
 r[27] = .98001093248415370000E+00;
 r[28] = .99183406163987350000E+00;
 r[29] = .99844674203732480000E+00;
c[ 0] = .39840962480827790000E-02;
c[ 1] = .92332341555455000000E-02;
c[ 2] = .14392353941661670000E-01;
c[ 3] = .19399596284813530000E-01;
c[ 4] = .24201336415292590000E-01;
c[ 5] = .28746578108808720000E-01;
c[ 6] = .32987114941090080000E-01;
c[ 7] = .36877987368852570000E-01;
c[ 8] = .40377947614710090000E-01;
c[ 9] = .43449893600541500000E-01;
c[10] = .46061261118893050000E-01;
c[11] = .48184368587322120000E-01;
c[12] = .49796710293397640000E-01;
c[13] = .50881194874202750000E-01;
c[14] = .51426326446779420000E-01;
c[15] = .51426326446779420000E-01;
c[16] = .50881194874202750000E-01;
c[17] = .49796710293397640000E-01;
c[18] = .48184368587322120000E-01;
c[19] = .46061261118893050000E-01;
c[20] = .43449893600541500000E-01;
c[21] = .40377947614710090000E-01;
c[22] = .36877987368852570000E-01;
c[23] = .32987114941090080000E-01;
c[24] = .28746578108808720000E-01;
c[25] = .24201336415292590000E-01;
c[26] = .19399596284813530000E-01;
c[27] = .14392353941661670000E-01;
c[28] = .92332341555455000000E-02;
c[29] = .39840962480827790000E-02;
for(i=0;i<30;i++) {r[i+30]=-r[i];c[i+30]=c[i];}
int m=n;
double x[],h1,h2,J,a1,b1,c1,d1,k1,k2,JX,Q;
x=new double[2];
//1=========================
```

```
a1=f.a;
b1=f.b;
h1=(b1-a1)/2.0;
h2=(b1+a1)/2.0;
J=0;
//2=========================
for(i=0;i<m;i++)
{
//3=========================
JX=0;
x[0]=h1*r[i]+h2;
d1=f.d(x[0]);
c1=f.c(x[0]);
k1=(d1-c1)/2.0;
k2=(d1+c1)/2.0;
//4=========================
    for(j=0;j<n;j++)
    {
                x[1]=k1*r[j]+k2;
                Q=f.f(x);
                JX+=c[j]*Q;
                }
J=J+c[i]*k1*JX;
}
J=h1*J;
return J;
}
public static void main(String args[]) throws IOException
  { double L=1.0;  //m
    double t=3600;
    double alpha=1.0e-6;
    double T=10;
    double integ1=0;
    double integ2=0;
    double x=0.1;
    double k=50.0; //  W/(mK)
    //x1[0]=x  x1[1]=t
    if_xj g=(double x1[])->100*x1[0];
    if_x F=(x1)->20.0*x1;
    for(int m=1;m<50;m++)
    {f1 ff=new f1(g,m,alpha,L,t);
     double beta=(2.0*m-1.0)*Math.PI/L;
     integ2=I(ff);
     if_x y1=(x1)->F.func(x1)*Math.cos(beta*x1);
     integ1=y1.integral(0.0,t);
     System.out.println("m="+m+"beta="+beta+"integ1="+integ1+"integ2="+integ2);
     T+=Math.exp(-alpha*beta*beta*t)*Math.cos(beta*x)*(y1.integral(0.0,L)+alpha/k*integ2);
    }
    T*=2.0/L;
    System.out.println("x = "+x+"t="+t+"T="+T);
}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" HT_int_tr1
m=1beta=3.141592653589793integ1=1.6360812510762941E7integ2=-20.26433672879602
m=2beta=9.42477796076938integ1=1.3378589112125183E7integ2=-2.2516818616792733
m=3beta=15.707963267948966integ1=9705177.59864646integ2=-0.8106694773636043
m=4beta=21.991148575128552integ1=8720971.602538412integ2=-0.4136558677234771
m=5beta=28.274333882308138integ1=1.3050998575900134E7integ2=-0.2502757887329145
m=6beta=34.55751918948772integ1=2.2945813251362227E7integ2=-0.16757307063293492
m=7beta=40.840704496667314integ1=3.567774863499815E7integ2=-0.12000678182580861
m=8beta=47.12388980384689integ1=4.630412345322993E7integ2=-0.09016334841184619
m=9beta=53.40707511102649integ1=4.954023133207446E7integ2=-0.07021856133348224
m=10beta=59.69026041820606 6integ1=4.1972597972257584E7integ2=-0.05623373861661546
m=11beta=65.97344572538566integ1=2.365693112556217E7integ2=-0.04605079541921173
m=12beta=72.25663103256524integ1=-1623589.61221613integ2=-0.0384068599486 4523
m=13beta=78.53981633974483integ1=-2.7606452680663705E7integ2=-0.03252298469988887
m=14beta=84.82300164692441integ1=-4.782444433133388E7integ2=-0.02789754716134818
m=15beta=91.106186954104integ1=-5.797368951710145E7integ2=-0.024195683589059554
m=16beta=97.38937226128358integ1=-5.738689837913661E7integ2=-0.0211869316094581
m=17beta=103.67255756846318integ1=-4.899992406146612E7integ2=-0.018708473766491245
m=18beta=109.95574287564276integ1=-3.785017090878593E7integ2=-0.016642638294184813
m=19beta=116.23892818282235integ1=-2.878650013768683E7integ2=-0.01490267011736603
m=20beta=122.52211349000193integ1=-2.441563417084229E7integ2=-0.013423471827733192
m=21beta=128.8052987971815integ1=-2.4198906073778387E7integ2=-0.01215542554799932
```

```
m=22beta=135.0884841043611integ1=-2.5086898664658315E7integ2=-0.011060179753506549
m=23beta=141.3716694115407integ1=-2.336915985368206E7integ2=-0.010107695437541886
m=24beta=147.6548547187203integ1=-1.6843445054274373E7integ2=-0.00927399499869856
m=25beta=153.93804002589985integ1=-6241877.52025964integ2=-0.00854085875742046
m=26beta=160.22122533307945integ1=4820651.528156524integ2=-0.007901241038355473
m=27beta=166.50441064025904integ1=1.142472281391428E7integ2=-0.0073135524924237295
m=28beta=172.78759594743863integ1=9765027.911342267integ2=-0.006555576851762927
m=29beta=179.0707812546182integ1=-718015.944600611integ2=-0.006291931602300288
m=30beta=185.3539665617978integ1=-1.6548858425989337E7integ2=-0.009976501890803399
m=31beta=191.63715186897738integ1=-3.125090952243381E7integ2=-0.007262575021289047
m=32beta=197.92033717615698integ1=-3.785698714610449E7integ2=0.039051484110085306
m=33beta=204.20352248333654integ1=-3.1842626838082083E7integ2=0.021249021486473132
m=34beta=210.48670779051614integ1=-1.3179771327419568E7integ2=-0.32668676317057493
m=35beta=216.76989309769573integ1=1.3448131605640331E7integ2=-0.23482953724525646
m=36beta=223.05307840487532integ1=4.042899943900705E7integ2=1.5532892857783265
m=37beta=229.3362637120549integ1=6.0261448620811075E7integ2=1.2623235265568737
m=38beta=235.61944901923448integ1=6.852209758517803E7integ2=-4.871138309542406
m=39beta=241.90263432641407integ1=6.5442147005385354E7integ2=-4.359354083492212
m=40beta=248.18581963359367integ1=5.539400103293332E7integ2=9.06071206626194
m=41beta=254.46900494077323integ1=4.454507727295937E7integ2=8.83498426317714
m=42beta=260.75219024795285integ1=3.77897563094229E7integ2=-7.571687802222459
m=43beta=267.0353755551324integ1=3.641735753539598E7integ2=-8.343554685516132
m=44beta=273.318560862312integ1=3.7637523155200385E7integ2=-2.5427046498639174
m=45beta=279.6017461694916integ1=3.620672003842143E7integ2=-1.5033151341933508
m=46beta=285.88493147667117integ1=2.7384559259971775E7integ2=6.605926242152686
m=47beta=292.16811678385073integ1=9777865.891168052integ2=6.980448899306837
m=48beta=298.45130209103036integ1=-1.336392960787008E7integ2=3.42632459693947
m=49beta=304.7344873982099integ1=-3.5137308079091646E7integ2=2.6442578297236365
x = 0.1t=3600.0T=12.194514058635923

> Terminated with exit code 0.
```

Python version of the program:

```
#integral HT_int_tr1.py
# Heat transfer integral equations example
from math import *
from f_xj import *
from f_x import *

#2 dimensional function
class f1:
  def __init__(self, gi,mi,alphai,Li,ti):
    self.a = 0.0
    self.b = Li
    self.L=Li
    self.g=gi
    self.m=mi
    self.alpha=alphai
    self.t=t
    self.beta=(2.0*self.m-1)*(pi*self.L)
  def c(self,x):
      return 0.0
  def d(self,x):
      return L
  def f(self,x):
      ff=exp(alpha*beta*beta*x[1])*g.func(x)*cos(beta*x[0])
      return ff


def I(f):
    #integral f(x)dx
    #integral of a function by using gauss-legendre quadrature
    #coefficients are pre-calculated for 60 terms for [-1,1]
    #band then utilises variable transform
    n=60;
    r=[0 for z in range(n)]
    c=[0 for z in range(n)]
    r[ 0] = .1553257962675247000E-02;
    r[ 1] = .8165938360126412000E-02;
    r[ 2] = .1998906751584623000E-01;
    r[ 3] = .3689997628536285000E-01;
    r[ 4] = .5871973210397363000E-01;
    r[ 5] = .8521711880861582000E-01;
    r[ 6] = .1161112839475869000E+00;
    r[ 7] = .1510747526033421000E+00;
    r[ 8] = .1897369085053786000E+00;
```

```
    r[ 9] = .23168792592899010000E+00;
    r[10] = .27648311523095540000E+00;
    r[11] = .32364763723456090000E+00;
    r[12] = .37268153691605510000E+00;
    r[13] = .42306504319570830000E+00;
    r[14] = .47426407872234120000E+00;
    r[15] = .52573592127765890000E+00;
    r[16] = .57693495680429170000E+00;
    r[17] = .62731846308394490000E+00;
    r[18] = .67635236276543910000E+00;
    r[19] = .72351688476904450000E+00;
    r[20] = .76831207407100990000E+00;
    r[21] = .81026309149462140000E+00;
    r[22] = .84892524739665800000E+00;
    r[23] = .88388871605241310000E+00;
    r[24] = .91478288119138420000E+00;
    r[25] = .94128026789602640000E+00;
    r[26] = .96310002371463720000E+00;
    r[27] = .98001093248415370000E+00;
    r[28] = .99183406163987350000E+00;
    r[29] = .99844674203732480000E+00;
    c[ 0] = .39840962480827790000E-02;
    c[ 1] = .92332341555455000000E-02;
    c[ 2] = .14392353941661670000E-01;
    c[ 3] = .19399596284813530000E-01;
    c[ 4] = .24201336415292590000E-01;
    c[ 5] = .28746578108808720000E-01;
    c[ 6] = .32987114941090080000E-01;
    c[ 7] = .36877987368852570000E-01;
    c[ 8] = .40377947614710090000E-01;
    c[ 9] = .43449893600541500000E-01;
    c[10] = .46061261118893050000E-01;
    c[11] = .48184368587322120000E-01;
    c[12] = .49796710293397640000E-01;
    c[13] = .50881194874202750000E-01;
    c[14] = .51426326446779420000E-01;
    c[15] = .51426326446779420000E-01;
    c[16] = .50881194874202750000E-01;
    c[17] = .49796710293397640000E-01;
    c[18] = .48184368587322120000E-01;
    c[19] = .46061261118893050000E-01;
    c[20] = .43449893600541500000E-01;
    c[21] = .40377947614710090000E-01;
    c[22] = .36877987368852570000E-01;
    c[23] = .32987114941090080000E-01;
    c[24] = .28746578108808720000E-01;
    c[25] = .24201336415292590000E-01;
    c[26] = .19399596284813530000E-01;
    c[27] = .14392353941661670000E-01;
    c[28] = .92332341555455000000E-02;
    c[29] = .39840962480827790000E-02;
    for i in range(30):
       r[i+30]=-r[i]
       c[i+30]=c[i]
    m=n;
    x=[0 for z in range(2)]
    #1=========================
    a1=f.a
    b1=f.b;
    h1=(b1-a1)/2.0;
    h2=(b1+a1)/2.0;
    J=0;
    #2=========================
    for i in range(m):
       JX=0;
       x[0]=h1*r[i]+h2;
       d1=f.d(x[0]);
       c1=f.c(x[0]);
       k1=(d1-c1)/2.0;
       k2=(d1+c1)/2.0;
       #3=========================
       for j in range(n):
          x[1]=k1*r[j]+k2
          Q=f.f(x)
          JX+=c[j]*Q
       J=J+c[i]*k1*JX
    J=h1*J;
```

```
    return J



L=1.0;  #m
t=3600.0
alpha=1.0e-6
T=10;
integ1=0.0
integ2=0.0
x=0.1
k=50.0  #  W/(mK)
# x1[0]=x  x1[1]=t
x1=[x,t]
class f2(f_xj):func=lambda self,x: 100.0*x[0]
g=f2()
class f3(f_x):func=lambda self,x: 20.0*x
F=f3()
for m in range(1,50):
    ff=ff=f1(g,m,alpha,L,t)
    beta=(2.0*m-1.0)*pi/L;
    integ2=I(ff)
    class f4(f_x):func=lambda self,x: F.func(x)*cos(beta*x)
    y1=f4()
    integ1=y1.integral(0.0,t,20)
    print("m=",m,"beta=",beta,"integ1=",integ1,"integ2=",integ2)
    T=T+exp(-alpha*beta*beta*t)*cos(beta*x)*(y1.integral(0.0,L,20)+alpha/k*integ2)
T*=2.0/L;
print("x = ",x,"t=",t,"T=",T)
```

```
runfile('E:/okul/SCO1/HT_int_tr1.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj, f_x
m= 1 beta= 3.141592653589793 integ1= 15638386.162158405 integ2= -20.26433672879602
m= 2 beta= 9.42477796076938 integ1= 13181558.626988266 integ2= -2.2516818616792733
m= 3 beta= 15.707963267948966 integ1= 22727930.6175761 integ2= -0.8106694773636043
m= 4 beta= 21.991148575128552 integ1= 27782615.67461563 integ2= -0.4136558677234771
m= 5 beta= 28.274333882308138 integ1= 91790485.44692326 integ2= -0.2502757887329145
m= 6 beta= 34.55751918948772 integ1= -5598955.686731523 integ2= -0.16757307063293492
m= 7 beta= 40.840704496667314 integ1= 13535877.892333847 integ2= -0.12000678182580861
m= 8 beta= 47.12388980384689 integ1= 38090075.1837646 integ2= -0.09016334841184619
m= 9 beta= 53.40707511102649 integ1= 12075843.004274044 integ2= -0.07021856133348224
m= 10 beta= 59.690260418206066 integ1= -12567983.030709447 integ2= -0.05623373861661546
m= 11 beta= 65.97344572538566 integ1= -11886261.362219937 integ2= -0.04605079541921173
m= 12 beta= 72.25663103256524 integ1= 43249935.38679126 integ2= -0.0384068599486 4512
m= 13 beta= 78.53981633974483 integ1= -19898430.82201991 integ2= -0.03252298469988887
m= 14 beta= 84.82300164692441 integ1= 27898912.087679036 integ2= -0.02789754716134818
m= 15 beta= 91.106186954104 integ1= -3013665.654328853 integ2= -0.024195683589059554
m= 16 beta= 97.38937226128358 integ1= 6533876.179262815 integ2= -0.0211869316094581
m= 17 beta= 103.67255756846318 integ1= 48274785.636048876 integ2= -0.0187084737664913
m= 18 beta= 109.95574287564276 integ1= -43066823.046316676 integ2= -0.016642638294184813
m= 19 beta= 116.23892818282235 integ1= -36472148.6358899 integ2= -0.014902670117366032
m= 20 beta= 122.52211349000193 integ1= -1722841.1542558589 integ2= -0.013423471827734191
m= 21 beta= 128.8052987971815 integ1= -21972297.57788502 integ2= -0.01215542554799932
m= 22 beta= 135.0884841043611 integ1= -49624775.558276534 integ2= -0.011060179753506604
m= 23 beta= 141.37166941154 07 integ1= 414968.90834910586 integ2= -0.010107695437541997
m= 24 beta= 147.6548547187203 integ1= 11874604.29970065 integ2= -0.00927399499869856
m= 25 beta= 153.93804002589985 integ1= 4265749.050800093 integ2= -0.008540858757420904
m= 26 beta= 160.22122533307945 integ1= -32961032.774677373 integ2= -0.007901241038355473
m= 27 beta= 166.50441064025904 integ1= -19554461.994246118 integ2= -0.007313552492423785
m= 28 beta= 172.78759594743863 integ1= -32153256.90881077 integ2= -0.006555576851762934
m= 29 beta= 179.0707812546182 integ1= -32164971.408064824 integ2= -0.006291931602300288
m= 30 beta= 185.3539665617978 integ1= -17403023.25943925 integ2= -0.009976501890803399
m= 31 beta= 191.63715186897738 integ1= -84657834.54259369 integ2= -0.007262575021289047
m= 32 beta= 197.92033717615698 integ1= 41667579.68818247 integ2= 0.039051484110085306
m= 33 beta= 204.20352248333654 integ1= -20114848.192499895 integ2= 0.02124902148647314
m= 34 beta= 210.48670779051614 integ1= -46524073.001039654 integ2= -0.32668676317057493
m= 35 beta= 216.76989309769573 integ1= 14846229.143789887 integ2= -0.23482953724525646
m= 36 beta= 223.05307840487532 integ1= -13523432.722009338 integ2= 1.5532892857783265
m= 37 beta= 229.3362637120549 integ1= 11648712.291166909 integ2= 1.2623235265568737
m= 38 beta= 235.61944901923448 integ1= -66453569.22388059 integ2= -4.871138309542406
m= 39 beta= 241.90263432641407 integ1= -795013.573989505 integ2= -4.359354083492212
m= 40 beta= 248.18581963359367 integ1= -34771749.35818569 integ2= 9.06071206626194
m= 41 beta= 254.46900494077323 integ1= -20016563.27065926 integ2= 8.83498426317714
m= 42 beta= 260.75219024795285 integ1= 9366978.104340265 integ2= -7.571687802222459
m= 43 beta= 267.0353755551324 integ1= -43048706.20740456 integ2= -8.343554685516132
m= 44 beta= 273.318560862312 integ1= 67091114.78322326 integ2= -2.5427046498639174
m= 45 beta= 279.6017461694916 integ1= 12255709.455442268 integ2= -1.5033151341933508
m= 46 beta= 285.88493147667117 integ1= -35824183.6724518 integ2= 6.605926242152686
```

m= 47 beta= 292.16811678385073 integ1= 424545.3320821072 integ2= 6.980448899306837
m= 48 beta= 298.45130209103036 integ1= 5052474.932048841 integ2= 3.426324596939469
m= 49 beta= 304.7344873982099 integ1= -22637550.60972654 integ2= 2.6442578297236365
x = 0.1 t= 3600.0 T= 12.198963350154287

8.0 APPROXIMATION METHODS IN HEAT CONDUCTION

Approximate methods used an approximation equation such as a polynomial to approximate heat transfer temperature profile. As a first example case let us consider semi-infinite wall with constant surface temperature. Let us solve the problem first as an exact solution by using Laplace transform method.

PROBLEM 8.1 SEMI-INFINITE WALL WITH CONSTANT TEMPERATURE APPROXIMATION METHODS

$$\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T(x,t)}{\partial t}$$

with boundary condition:
$T(x,t)= T_0$ x=0 t > 0
$T(x,t)=T_i$ $x \to \infty$ t > 0
$T(x,t)= T_i$ at t=0
Defining $\theta(x,t) = T(x,t) - T_i$

$$\frac{\partial^2 \theta(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial \theta(x,t)}{\partial t}$$

with boundary condition:
$\theta(x,t) = \theta_0 = T_0 - T_i$ x=0 t > 0
$\theta(x,t) = 0$ $x \to \infty$ t > 0
$\theta(x,t) = 0$ at t=0
Laplace transform of the equation is:
Laplace transform of the differential equation:

$$\frac{\partial^2 \theta(x,s)}{\partial x^2} - \frac{s}{\alpha}\theta(x,s) = 0$$

Boundary conditions:
$\theta(x,s) = T_0 - T_i$ x=0
$\theta(x,s) = 0$ $x \to \infty$
Solution of this ordinary differential equation is:

$$\theta(x,s) = C_1 e^{-x\sqrt{\frac{s}{\alpha}}} + C_2 e^{x\sqrt{\frac{s}{\alpha}}}$$

Boundary condition $\theta(x,s) = 0$ $x \to \infty$ $\rightarrow$ $C_2 = 0$
Boundary condition $\theta(x,s) = \frac{T_0 - T_i}{s}$ x=0 $\rightarrow$ $C_1 = \frac{T_0 - T_i}{s}$
So the solution is:

$$\theta(x,s) = \frac{T_0 - T_i}{s} e^{-x\sqrt{\frac{s}{\alpha}}}$$

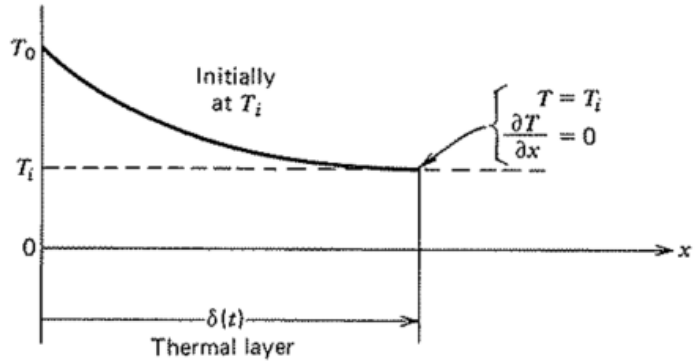
Inverse laplace transfrom of $G(x,s) = \frac{e^{-x\sqrt{\frac{s}{\alpha}}}}{s}$ is $G(x,t) = erfc\left(\frac{x}{2\sqrt{\alpha t}}\right)$

In the table it is given as case 42:
For $G(x,s) = \frac{e^{-k\sqrt{s}}}{s}$ $G(x,t) = erfc\left(\frac{k}{2\sqrt{t}}\right)$ and our case $k = \frac{x}{\sqrt{\alpha}}$

$$\theta(x,t) = T(x,t) - T_i = (T_0 - T_i) erfc\left(\frac{x}{2\sqrt{\alpha t}}\right) = (T_0 - T_i)\left[1 - erf\left(\frac{x}{2\sqrt{\alpha t}}\right)\right]$$

$$\frac{T(x,t) - T_0}{T_i - T_0} = erf\left(\frac{x}{2\sqrt{\alpha t}}\right)$$

Now that we have the exact solution let us look at the approximate solution method. It will be assumed that at time t, effect of surface temperature will be penetrate some thermal layer with penetration depth $\delta(t)$. For the remaining profile, temperature will remain constant at initial temperature, T_i.



Considering this condition differential equation can be integrate for the thermal layer only for integration depth $\delta(t)$. Therefore differential equation can be integrate for this integration depth.

$$\left.\frac{\partial T(x,t)}{\partial x}\right|_{x=\delta(t)} - \left.\frac{\partial T(x,t)}{\partial x}\right|_{x=0} = \frac{1}{\alpha}\int_{x=0}^{x=\delta(t)} \frac{\partial T(x,t)}{\partial t}\,dx$$

and considering

$$\frac{\partial}{\partial t}\left[\int_{x=0}^{x=\delta(t)} T(x,t)\,dx\right] = \int_{x=0}^{x=\delta(t)} \frac{\partial T(x,t)}{\partial t}\,dx + \left[T\frac{\partial x}{\partial t}\right]_{x=\delta(t)} - \left[T\frac{\partial x}{\partial t}\right]_{x=0}$$

$$\int_{x=0}^{x=\delta(t)} \frac{\partial T(x,t)}{\partial t}\,dx = \frac{\partial}{\partial t}\left[\int_{x=0}^{x=\delta(t)} T(x,t)\,dx\right] - T_i\frac{\partial \delta(t)}{\partial t}$$

Defining

$$\theta(x,t) = \int_{x=0}^{x=\delta(t)} T(x,t)\,dx$$

differential equation becomes:

$$-\alpha\left.\frac{\partial T(x,t)}{\partial t}\right|_{x=0} = \frac{\partial}{\partial t}[\theta(x,t) - T_i\delta(t)]$$

In order to solve this differential equation, an approximate temperature profile T(x,t) will be assumed. As a first case, let us take a cubic polynomial

$$T(x,t) = a + bx + cx^2 + dx^3$$

Coefficients can be found by applying boundary conditions:

$T(x,t)|_{x=0} = T_0$

$T(x,t)|_{x=\delta(t)} = T_i$

$\left.\frac{\partial T(x,t)}{\partial x}\right|_{x=\delta(t)} = 0$

$\left.\frac{\partial^2 T(x,t)}{\partial x^2}\right|_{x=\delta(t)} = 0$

$\left.\frac{\partial^2 T(x,t)}{\partial x^2}\right|_{x=0} = 0$

$$\frac{\partial T(x,t)}{\partial x} = b + 2cx + 3dx^2$$

$$\frac{\partial^2 T(x,t)}{\partial x^2} = 2c + 6dx$$

$\left.\frac{\partial^2 T(x,t)}{\partial x^2}\right|_{x=0} = 0 \quad \rightarrow \quad 2c = 0 \quad c = 0$

$T(x,t)|_{x=0} = T_0 \quad \rightarrow \quad a = T_0$

$\left.\frac{\partial T(x,t)}{\partial x}\right|_{x=\delta(t)} = 0 \quad \rightarrow \quad b + 3d\delta^2 = 0 \qquad b = -3d\delta^2$

$T(x,t)|_{x=\delta(t)} = T_i \quad \rightarrow \quad T_i = T_0 + b\delta + d\delta^2 = T_0 + (-3d\delta^2)\delta + d\delta^2$

$d = \frac{T_0 - T_i}{2\delta^3} \qquad b = -3\frac{T_0 - T_i}{2\delta^3}\delta^2 = -\frac{3}{2}\frac{T_0 - T_i}{\delta}$

$$T(x,t) - T_i = (T_0 - T_i) - \frac{3}{2}(T_0 - T_i)\left(\frac{x}{\delta}\right) + \frac{1}{2}(T_0 - T_i)\left(\frac{x}{\delta}\right)^3$$

$$\frac{T(x,t) - T_i}{T_0 - T_i} = 1 - \frac{3}{2}\left(\frac{x}{\delta}\right) + \frac{1}{2}\left(\frac{x}{\delta}\right)^3$$

This equation can now be substituted into the differential equation

$$\frac{\partial T(x,t)}{\partial x} = \frac{\partial}{\partial x}\left[(T_0 - T_i) - \frac{3}{2}(T_0 - T_i)\left(\frac{x}{\delta}\right) + \frac{1}{2}(T_0 - T_i)\left(\frac{x}{\delta}\right)^3\right]$$

$$\frac{\partial T(x,t)}{\partial x} = -\frac{3}{2\delta}(T_0 - T_i) + \frac{3}{2}(T_0 - T_i)\frac{x^2}{\delta^3}$$

$$\left.\frac{\partial T(x,t)}{\partial x}\right|_{x=0} = -\frac{3}{2\delta}(T_0 - T_i)$$

$$\theta(x,t) = \int_{x=0}^{x=\delta(t)} T(x,t)\,dx = \int_{x=0}^{x=\delta(t)} \left[(T_0 - T_i) - \frac{3}{2}(T_0 - T_i)\left(\frac{x}{\delta}\right) + \frac{1}{2}(T_0 - T_i)\left(\frac{x}{\delta}\right)^3 + T_i\right] dx$$

$$\theta(x,t) = (T_0 - T_i)\left[x - \frac{3}{4}\frac{x^2}{\delta} + \frac{1}{8}\frac{x^4}{\delta^3}\right]_{x=0}^{x=\delta(t)} + T_i\delta$$

$$\theta(x,t) = (T_0 - T_i)\left[\delta - \frac{3}{4}\delta + \frac{1}{8}\delta\right] + T_i\delta = (T_0 - T_i)\left[\frac{3}{8}\delta\right] + T_i\delta$$

Substituting the results into the differential equation gives

$$-\alpha\left.\frac{\partial T(x,t)}{\partial t}\right|_{x=0} = \frac{\partial}{\partial t}[\theta(x,t) - T_i\delta(t)]$$

$$-\alpha\left[-\frac{3}{2\delta}(T_0 - T_i)\right] = \frac{\partial}{\partial t}\left[(T_0 - T_i)\left[\frac{3}{8}\delta\right] + T_i\delta - T_i\delta\right]$$

$$\left[\frac{3\alpha}{2\delta}\right] = \frac{\partial}{\partial t}\left[\frac{3}{8}\delta\right]$$

$$4\alpha\partial t = \delta\partial(\delta)$$

$$\int_{t=0}^{t} 4\alpha\partial t' = \int_{\delta'=0}^{\delta} \delta'\partial(\delta')$$

$$4\alpha t = \frac{\delta^2}{2}$$

$$\delta(t) = \sqrt{8\alpha t}$$

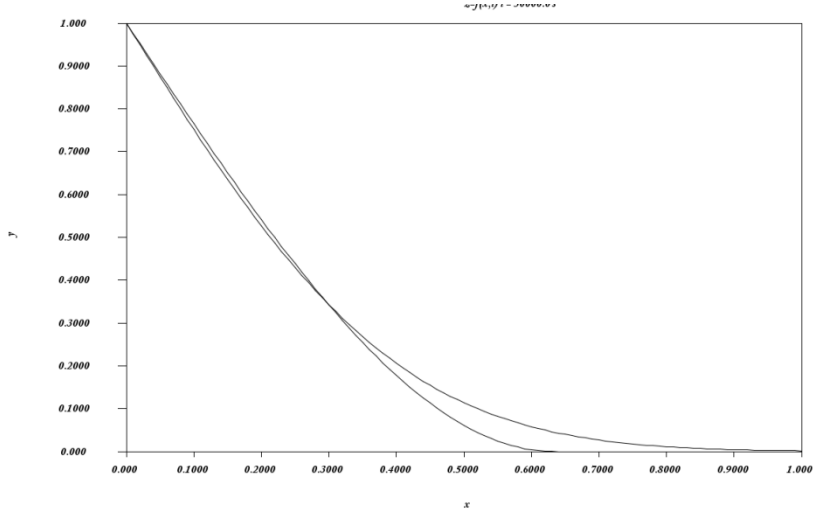
By using this result, together with approximation temperature profile will give us approximation solution. The results can be compare by using a graphic program

Problem 9.1 java solution

```
class i_wall extends fi_xt
{
public double[] func(double x,double t)
{ double a[]=new double[2];
        double alpha=1.0e-6;
```

```
  double delta=Math.sqrt(8.0*alpha*t);
  double eta=x/2/Math.sqrt(alpha*t);
  double xx=x/delta;

  a[0]=erfc(eta);
  if(xx<1) a[1]=1.0-1.5*xx+0.5*xx*xx*xx;
  else a[1]=0.0;
  return a;
}               }
public class timeplot_ntest1
{
public static void main(String args[])
 {
 i_wall ff=new i_wall();
  timeplot_n p=new timeplot_n(ff,0.0,1.0,0.01,0.0,50000.0,0.1,100.0);
  p.start();
 }}
```



Python solution

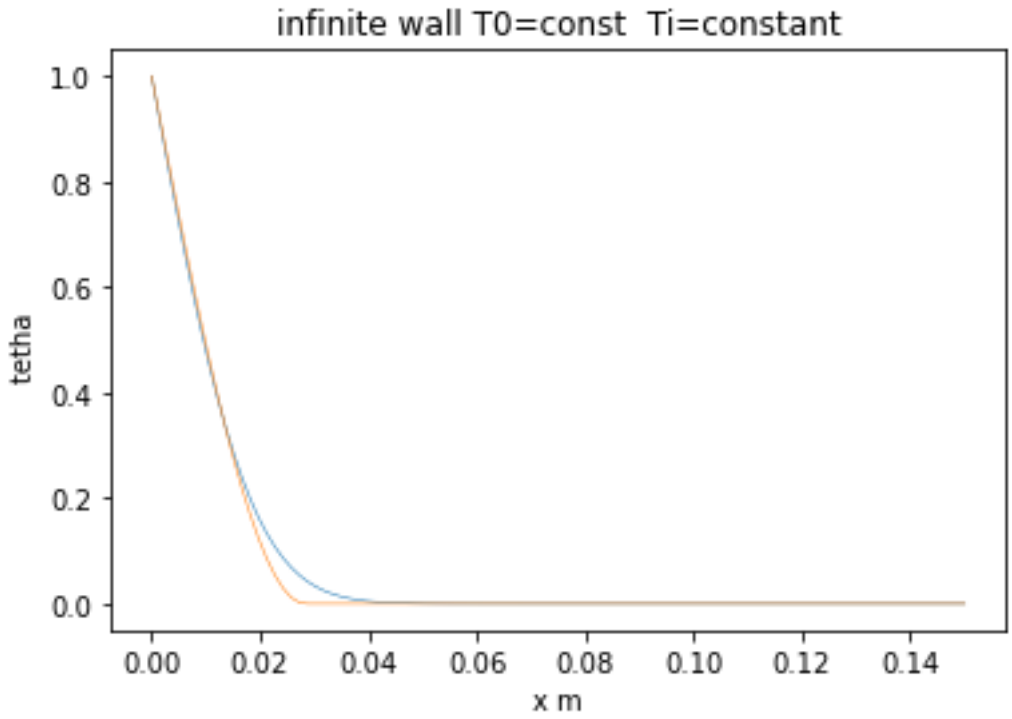
```
from math import *
from f_xj import *
import numpy as np;
import matplotlib.pyplot as plt;
#exact solution
class f1(f_xj):
    def func(self,z):
        x=z[0]
        t=z[1]
        eta=x/(2.0*sqrt(alpha*t))
        return erfc(eta)

#approximate solution
class f2(f_xj):
    def func(self,z):
        x=z[0]
        t=z[1]
        delta=sqrt(8.0*alpha*t)
        xx=x/delta
        if xx<1:
            T2=1.0-3.0/2.0*xx+0.5*xx*xx*xx
        else:
            T2=0
        return T2

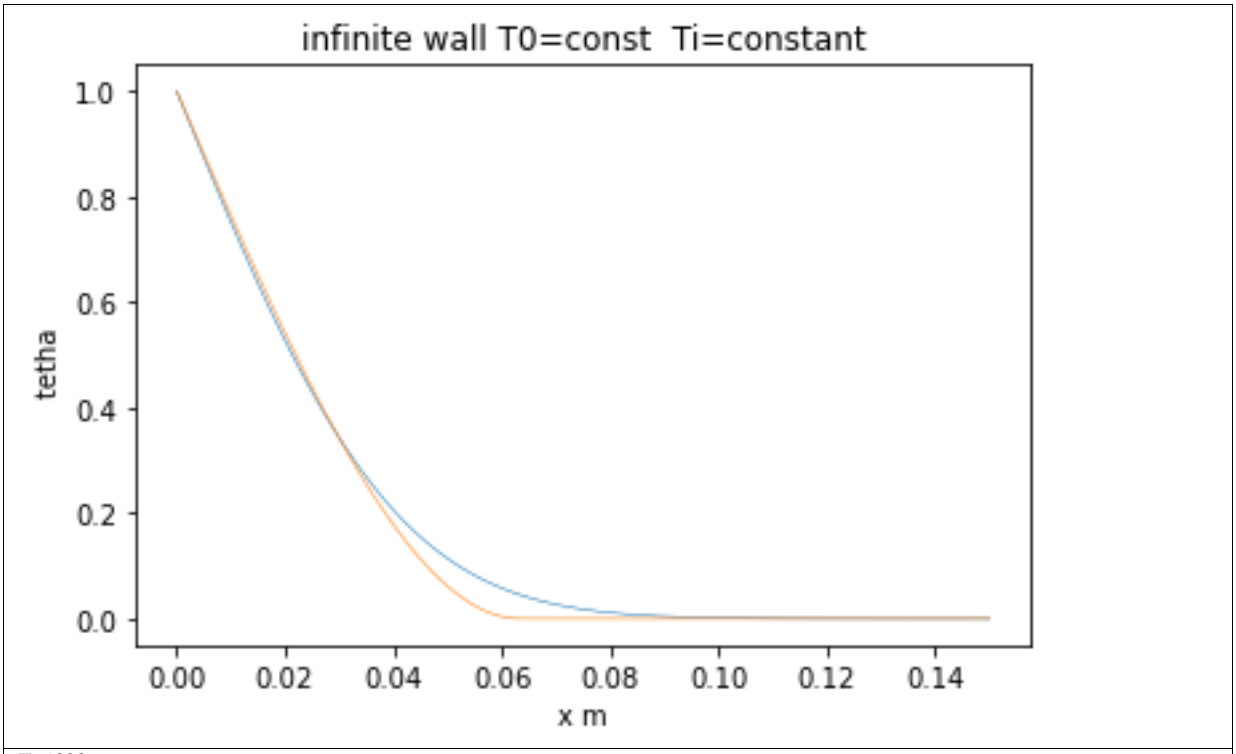
np1=100
x=[0 for x in range(np1)]
dx=0.15/(len(x)-1)
t=[100]
np2=len(t)
T1= [[0 for x in range(np1)] for y in range(np2)]
T2= [[0 for x in range(np1)] for y in range(np2)]
T0=105.0
Ti=10.0
alpha=1e-6
s=""
```

```
f=f1()
ff=f2()
for i in range(np2):
    for j in range(np1):
        x[j]=dx*j
        z=[x[j],t[i]]
        T1[i][j]=f.func(z)
        T2[i][j]=ff.func(z)
plt.title('infinite wall T0=const  Ti=constant')
plt.xlabel('x m')
plt.ylabel('tetha ')
for i in range(np2):
    plt.plot(x,T1[i],linewidth=0.5)
    plt.plot(x,T2[i],linewidth=0.5)
```

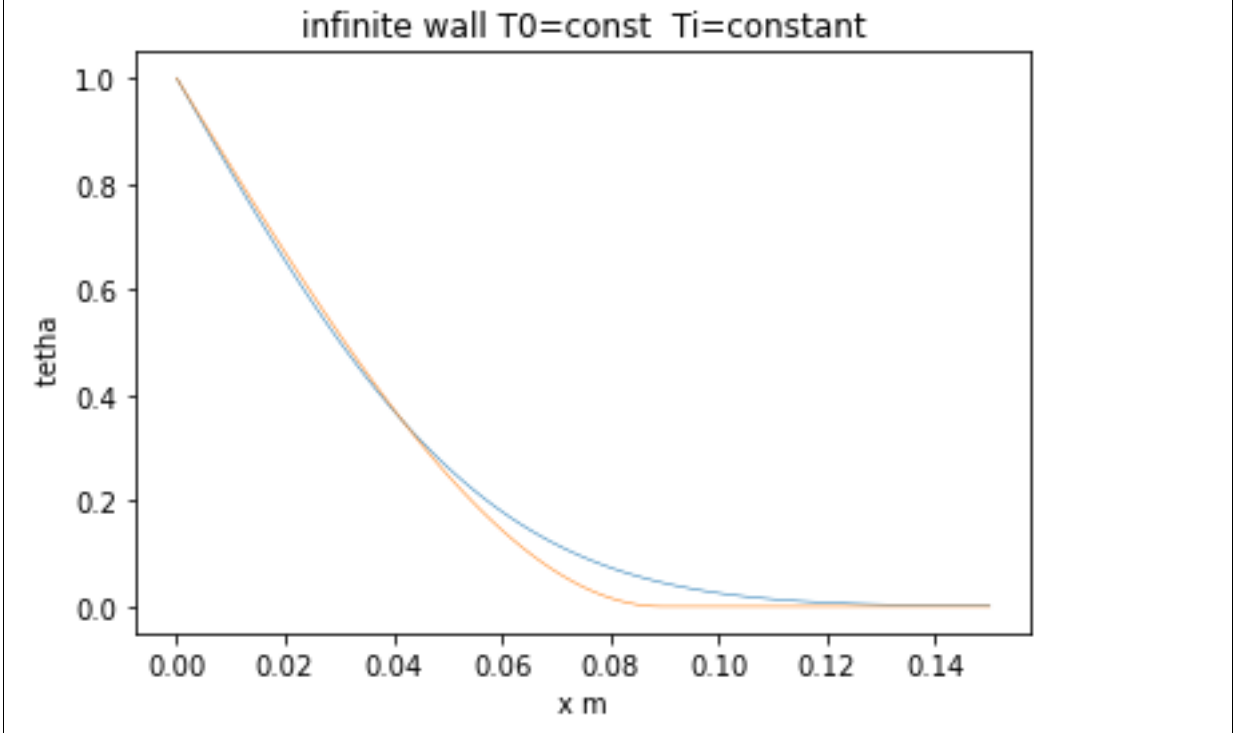
t=100 s



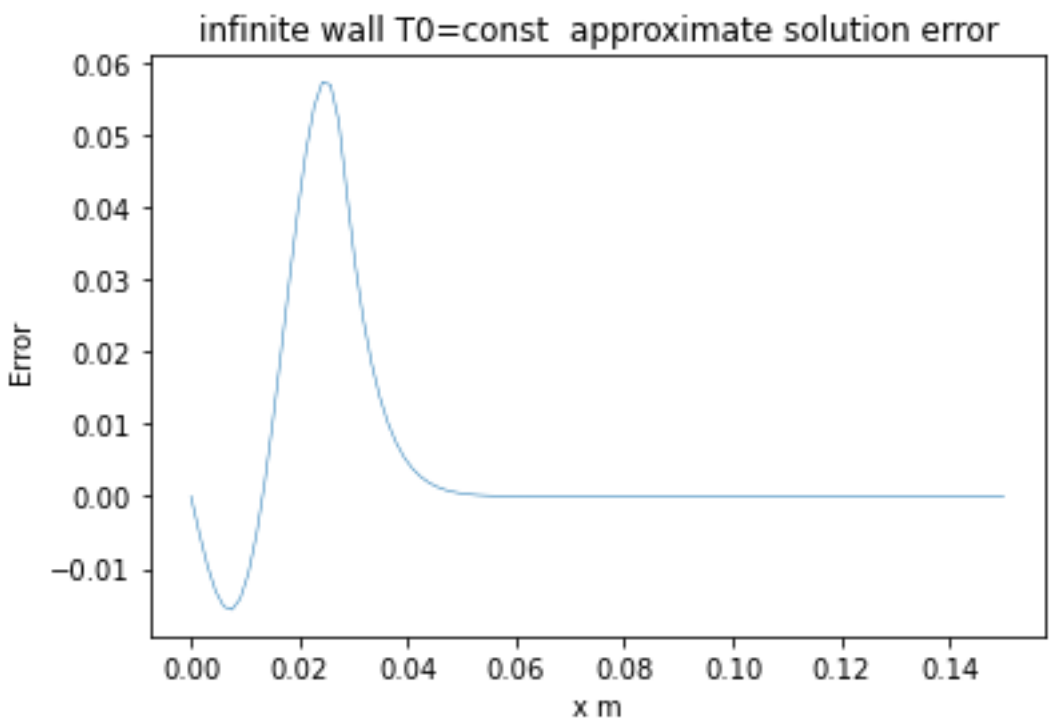
t=500 s



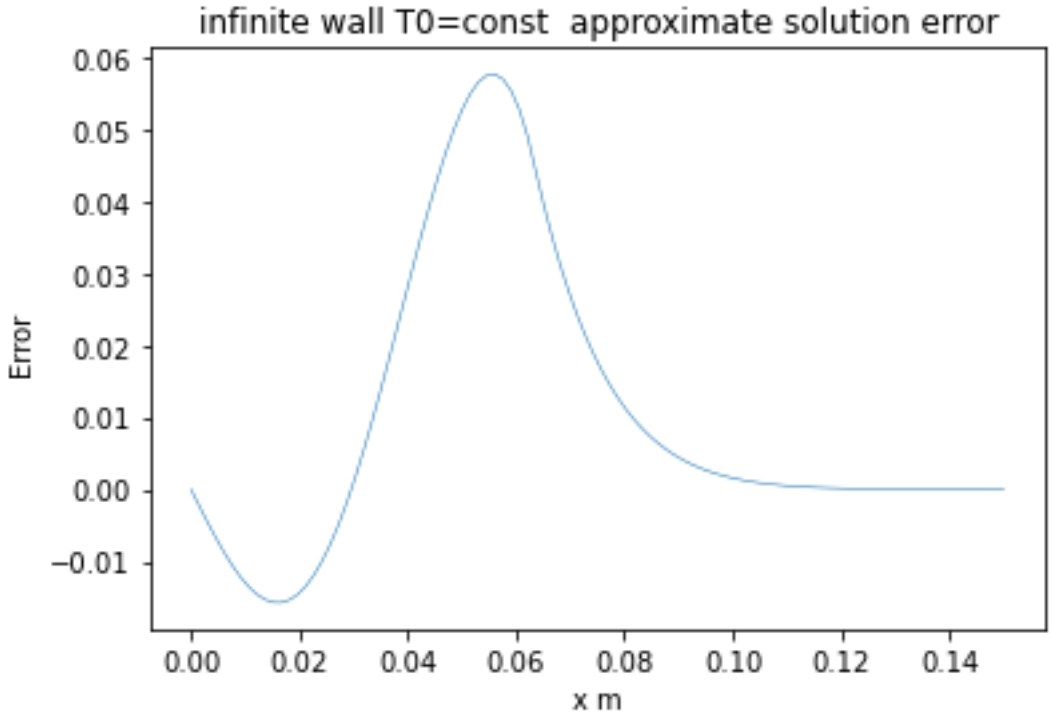
T=1000 s



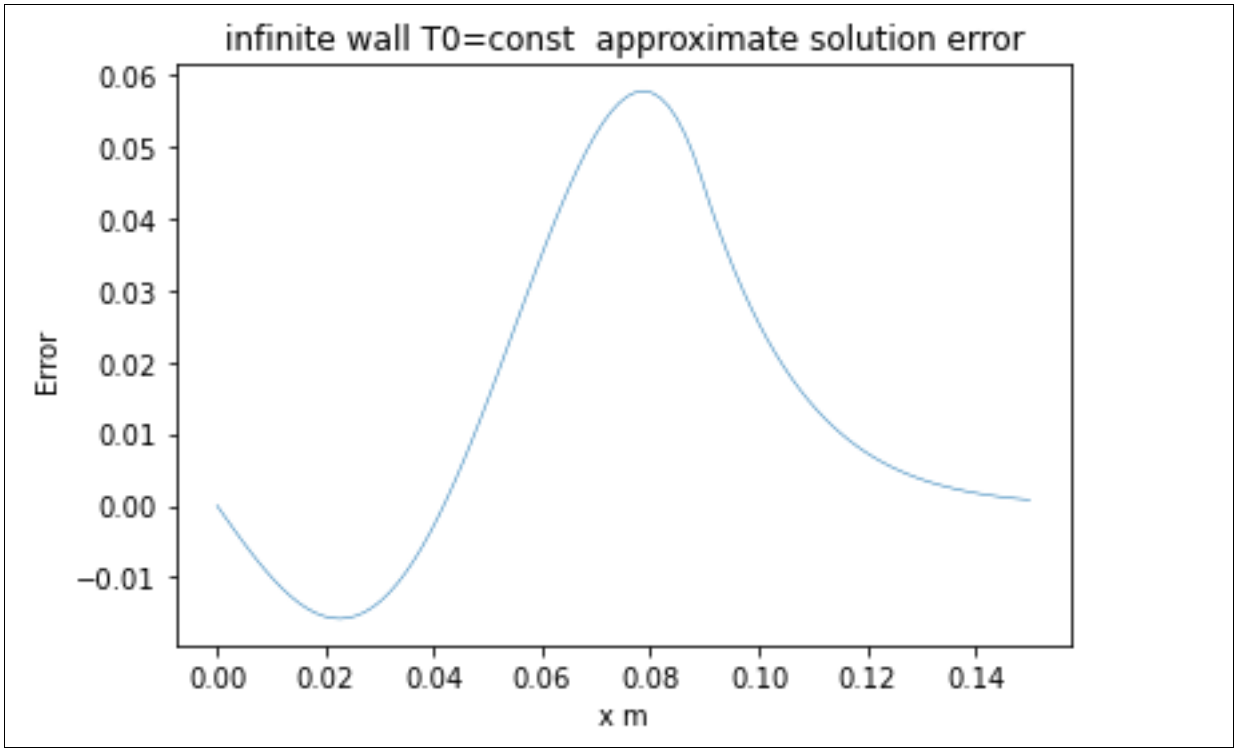
Error t=100 s



Error t=500 s



Error t=1000 s



For better accuracy, degree of polynomial can be increased. Of course additional boundary conditions to solve thic new equations is required. If a 4th degree polynomial is assumed:

$$T(x,t) = a + bx + cx^2 + dx^3 + ex^4$$

Coefficients can be found by applying boundary conditions:

$T(x,t)|_{x=0} = T_0$

$T(x,t)|_{x=\delta(t)} = T_i$

$\frac{\partial T(x,t)}{\partial x}\Big|_{x=\delta(t)} = 0$

$\frac{\partial^2 T(x,t)}{\partial x^2}\Big|_{x=\delta(t)} = 0$

$\frac{\partial^2 T(x,t)}{\partial x^2}\Big|_{x=0} = 0$

$$\frac{\partial T(x,t)}{\partial x} = b + 2cx + 3dx^2 + 4ex^3$$

$$\frac{\partial^2 T(x,t)}{\partial x^2} = 2c + 6dx + 12ex^2$$

$\frac{\partial^2 T(x,t)}{\partial x^2}\Big|_{x=0} = 0 \quad \rightarrow \quad 2c = 0 \quad c = 0$

$T(x,t)|_{x=0} = T_0 \quad \rightarrow \quad a = T_0$

$\frac{\partial^2 T(x,t)}{\partial x^2}\Big|_{x=\delta(t)} = 0 \quad \rightarrow \; 6d\delta + 12e\delta^2 = 0 \quad d = -2e\delta$

$\frac{\partial T(x,t)}{\partial x}\Big|_{x=\delta(t)} = 0 \quad \rightarrow \quad b + 3d\delta^2 + 4e\delta^3 = b + 3(-2e\delta)\delta^2 + 4e\delta^3 = 0 \quad b = 2e\delta^3$

$T(x,t)|_{x=\delta(t)} = T_i \quad \rightarrow \quad T_i = T_0 + b\delta + d\delta^3 + e\delta^3 = T_0 + (2e\delta^3)\delta + (-2e\delta)\delta^3 + e\,\delta^4$

$d = 2\frac{T_0 - T_i}{\delta^3} \quad e = -\frac{T_0 - T_i}{\delta^4} \quad b = -2\frac{T_0 - T_i}{\delta}$

$$T(x,t) - T_i = (T_0 - T_i) - 2(T_0 - T_i)\left(\frac{x}{\delta}\right) + 2(T_0 - T_i)\left(\frac{x}{\delta}\right)^3 - (T_0 - T_i)\left(\frac{x}{\delta}\right)^4$$

$$\frac{T(x,t)-T_i}{T_0-T_i}=1-2\left(\frac{x}{\delta}\right)+2\left(\frac{x}{\delta}\right)^3-\left(\frac{x}{\delta}\right)^4$$

We can apply this new temperature profile into the differential equation

$$T(x,t)=(T_0-T_i)-2(T_0-T_i)\left(\frac{x}{\delta}\right)+2(T_0-T_i)\left(\frac{x}{\delta}\right)^3-(T_0-T_i)\left(\frac{x}{\delta}\right)^4+T_i$$

$$\frac{\partial T(x,t)}{\partial x}=-2(T_0-T_i)\left(\frac{1}{\delta}\right)+6(T_0-T_i)\left(\frac{x^2}{\delta^3}\right)-4(T_0-T_i)\left(\frac{x}{\delta}\right)^4$$

$$\left.\frac{\partial T(x,t)}{\partial x}\right|_{x=0}=-2(T_0-T_i)\left(\frac{1}{\delta}\right)$$

$$\theta(x,t)=\int_{x=0}^{x=\delta(t)}T(x,t)\,dx=\int_{x=0}^{x=\delta(t)}\left[(T_0-T_i)-2(T_0-T_i)\left(\frac{x}{\delta}\right)+2(T_0-T_i)\left(\frac{x}{\delta}\right)^3-(T_0-T_i)\left(\frac{x}{\delta}\right)^4+T_i\right]dx$$

$$\theta(x,t)=(T_0-T_i)\left[x-\left(\frac{x^2}{\delta}\right)+\frac{2}{4}\left(\frac{x^4}{\delta^3}\right)-\frac{1}{5}\left(\frac{x^5}{\delta^4}\right)\right]_0^{\delta}+T_i x|_0^{\delta}=(T_0-T_i)\frac{3}{10}\delta+T_i\delta$$

differential equation becomes:

$$-\alpha\left.\frac{\partial T(x,t)}{\partial t}\right|_{x=0}=\frac{\partial}{\partial t}[\theta(x,t)-T_i\delta(t)]$$

$$-\alpha\left[-2(T_0-T_i)\left(\frac{1}{\delta}\right)\right]=\frac{\partial}{\partial t}\left[(T_0-T_i)\frac{3}{10}\delta+T_i\delta-T_i\delta\right]$$

$$\left(\frac{20\alpha}{3\delta}\right)=\frac{\partial}{\partial t}(\delta)$$

$$\left(\frac{20\alpha}{3}\right)\partial t=\delta\partial\delta$$

$$\left(\frac{20\alpha}{3}\right)t=\frac{\delta^2}{2}$$

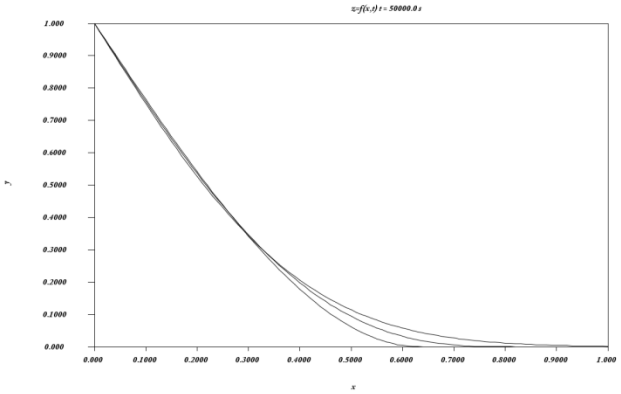
$$\delta(t)=\sqrt{\frac{40\alpha t}{3}}$$

By using this result, together with approximation temperature profile will give us approximation solution. The results can be compare by using a graphic program

Problem 9.1 java solution

```
 class i_wall extends fi_xt
{
public double[] func(double x,double t)
{ double a[]=new double[3];
            double alpha=1.0e-6;
  double delta=Math.sqrt(8.0*alpha*t);
  double delta1=Math.sqrt(40.0/3.0*alpha*t);
  double eta=x/2/Math.sqrt(alpha*t);
  double xx=x/delta;
  double xx1=x/delta1;

  a[0]=erfc(eta);
  if(xx<1) a[1]=1.0-1.5*xx+0.5*xx*xx*xx;
  else a[1]=0.0;
  if(xx1<1) a[2]=1.0-2*xx1+2.0*xx1*xx1*xx1-xx1*xx1*xx1*xx1;
  else a[2]=0.0;
  return a;
}          }
public class timeplot_ntest
{
public static void main(String args[])
  {
 i_wall ff=new i_wall();
  timeplot_n p=new timeplot_n(ff,0.0,1.0,0.01,0.0,50000.0,0.1,100.0);
  p.start();
  }
}
```



Now let us consider 1 dimensional, time dependent (Lumped) solution

Python solution of the problem

```
from math import *
from f_xj import *
import numpy as np;
import matplotlib.pyplot as plt;
#exact solution
class f1(f_xj):
    def func(self,z):
        x=z[0]
        t=z[1]
        eta=x/(2.0*sqrt(alpha*t))
        return erfc(eta)

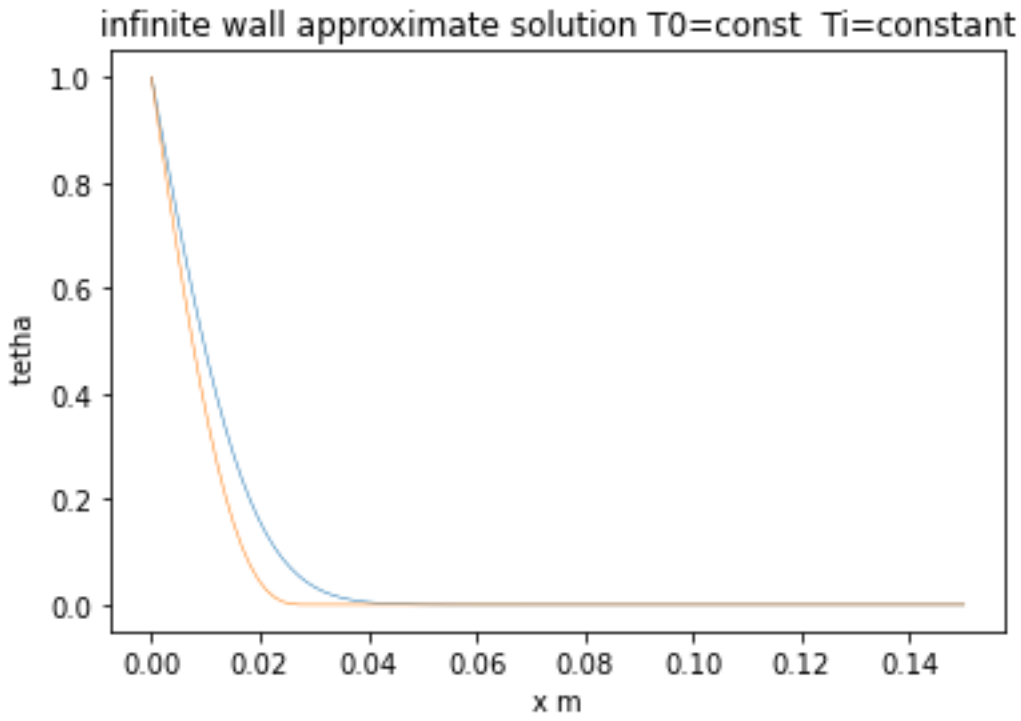
#approximate solution
class f2(f_xj):
    def func(self,z):
        x=z[0]
        t=z[1]
        delta=sqrt(8.0*alpha*t)
        xx=x/delta
        if xx<1:
            T2=1.0-2.0*xx+2.0*xx*xx*xx-xx*xx*xx*xx
        else:
            T2=0
        return T2
# difference between approximate and exact solutionapproximate solution
class f3(f_xj):
    def func(self,z):
        T1=f1()
        T2=f2()
        dT=T1.func(z)-T2.func(z)
        return dT

np1=100
x=[0 for x in range(np1)]
dx=0.15/(len(x)-1)
t=[100]
np2=len(t)
T1= [[0 for x in range(np1)] for y in range(np2)]
T2= [[0 for x in range(np1)] for y in range(np2)]
dT= [[0 for x in range(np1)] for y in range(np2)]
T0=105.0
Ti=10.0
alpha=1e-6
s=""
f=f1()
ff=f2()
df=f3()
for i in range(np2):
    for j in range(np1):
        x[j]=dx*j
        z=[x[j],t[i]]
        T1[i][j]=f.func(z)
        T2[i][j]=ff.func(z)
        dT[i][j]=df.func(z)

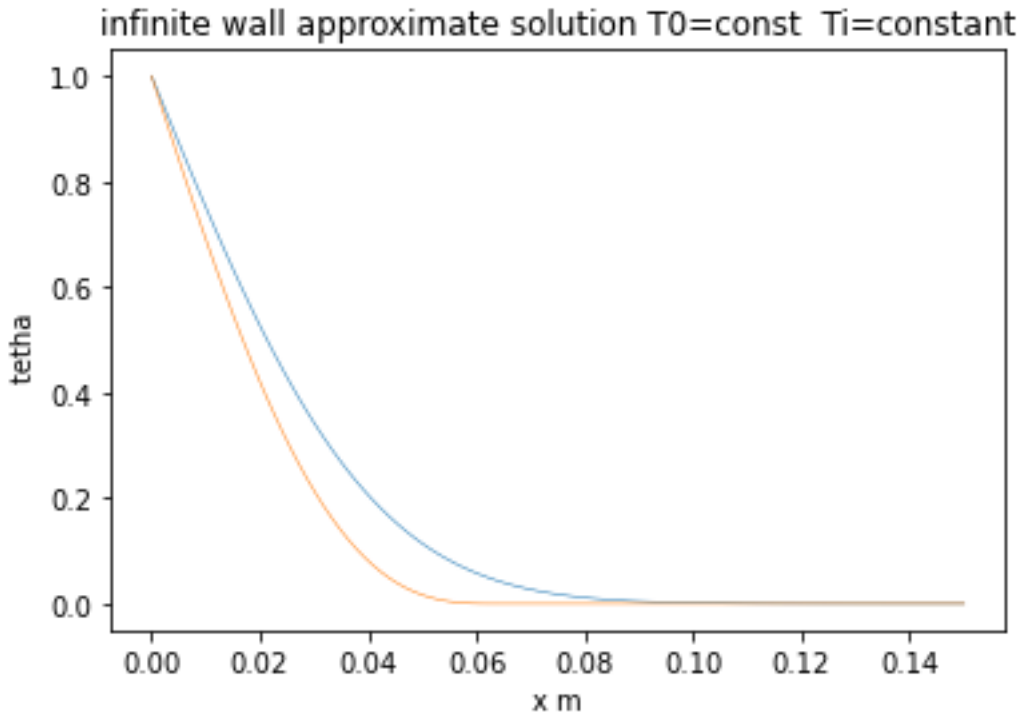
plt.title('infinite wall approximate solution T0=const  Ti=constant')
plt.xlabel('x m')
```

```
plt.ylabel('tetha ')
for i in range(np2):
    plt.plot(x,T1[i],linewidth=0.5)
    plt.plot(x,T2[i],linewidth=0.5)
"""
plt.title('infinite wall T0=const  approximate solution error')
plt.xlabel('x m')
plt.ylabel('Error ')
for i in range(np2):
    plt.plot(x,dT[i],linewidth=0.5)
"""
```

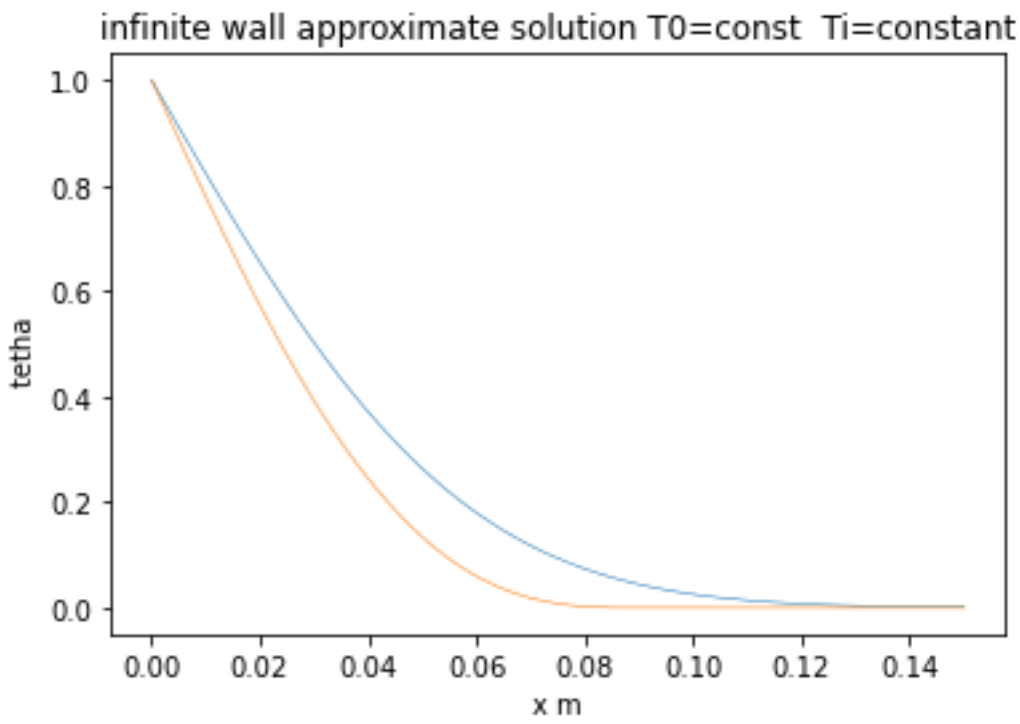
t=100 s



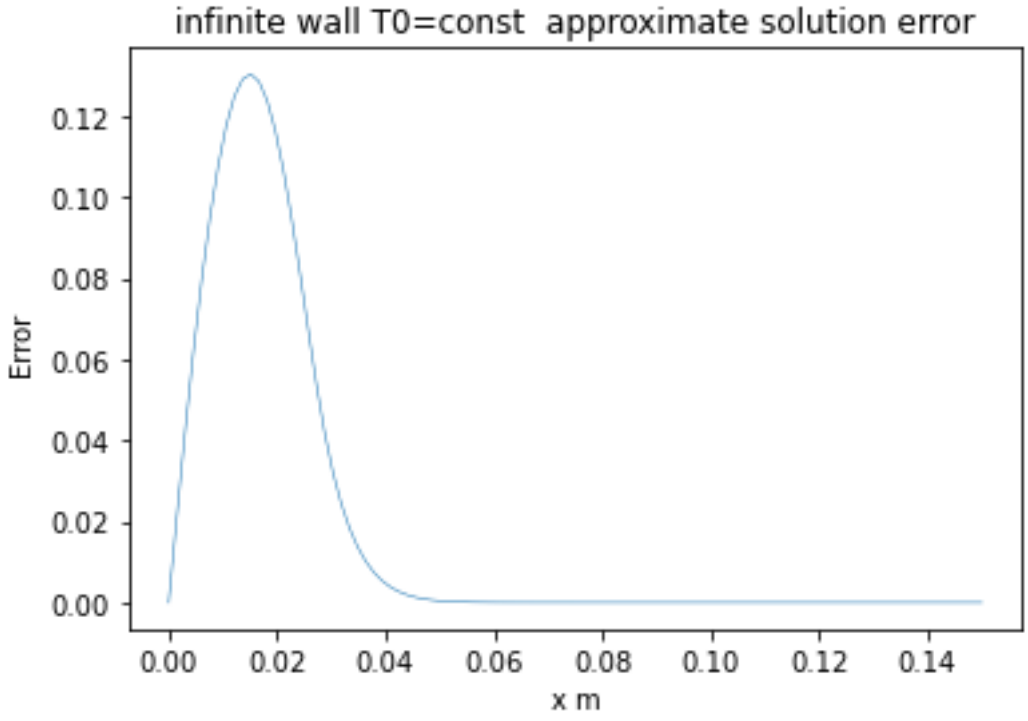
t=500 s



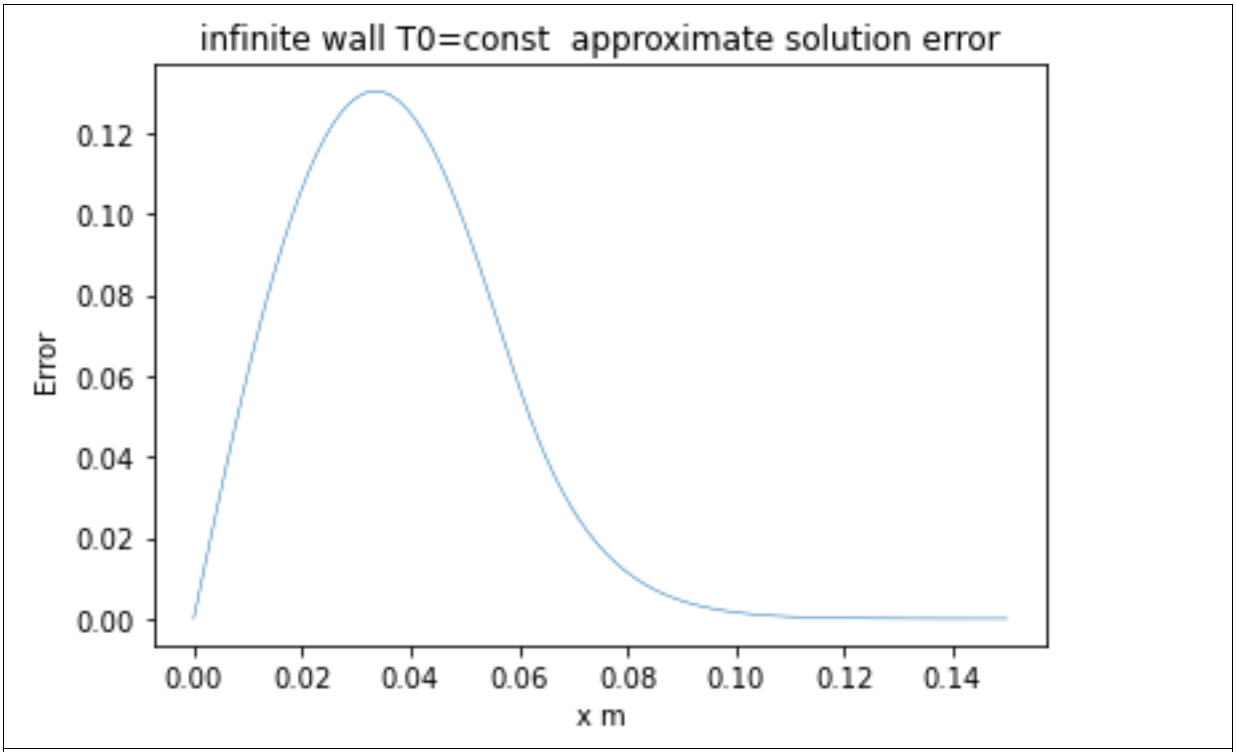
t=1000 s



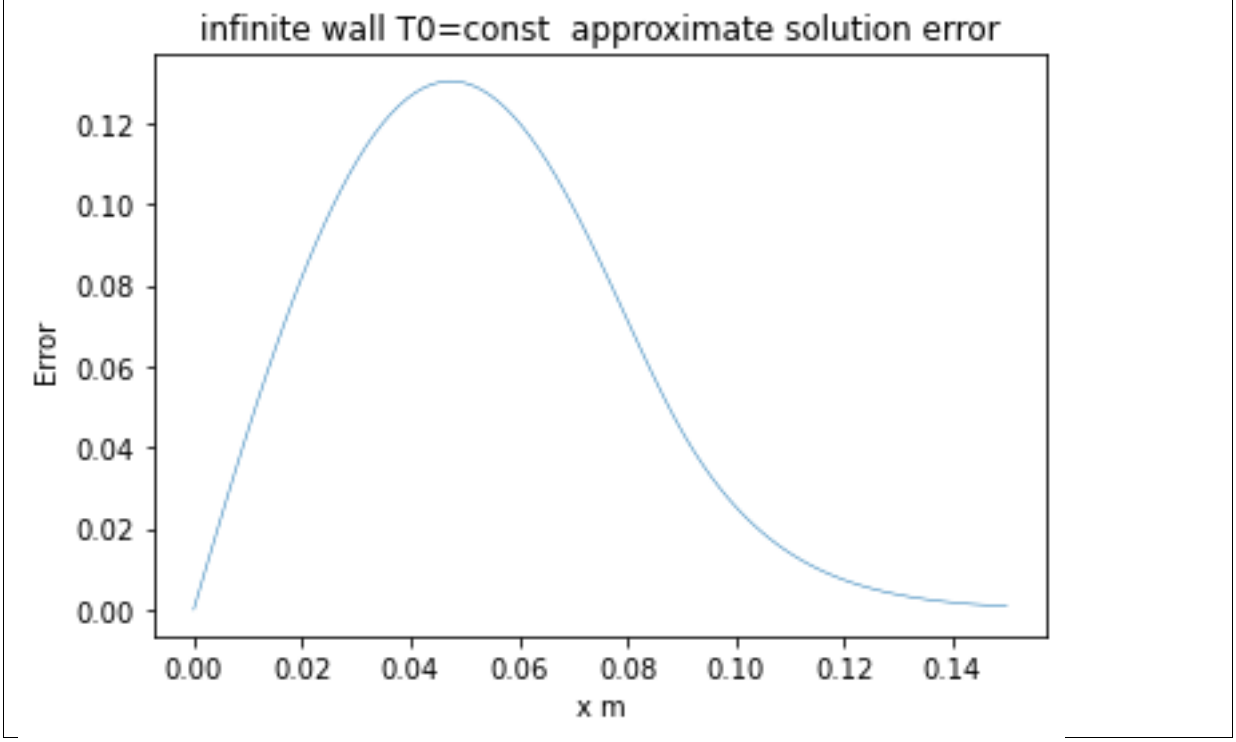
Error t=100 s



Error t=500 s



Error t=1000 s



In higher degree of polynomial version error is bigger. That indicates that increasing degree of polynomial not necessarily improve solution.

PROBLEM 8.2 TIME DEPENDENT LUMPED SOLUTION APPROXIMATION METHODS (COLLOCATION METHOD)

$$-hA_s(T(t)-T_\infty)=\rho V C_p\frac{dT(t)}{dt}$$

where A_s is surface area, V is volume, C_p is specific heat. By changing variables as:

$\theta(t)=\frac{T(t)-T_\infty}{T_i-T_\infty}\quad \tau=\frac{t}{\tau_{max}}\quad X=\frac{\rho V C_p}{\tau_{max}hA_s}$

equation becomes:

$$X\frac{d\theta(t)}{dt}+\theta(t)=0$$

Boundary condition t=0 θ =1so exact solution is:

$$\theta(t)=exp\left(-\frac{\tau}{X}\right)$$

Now let us create approximate solution, for this the following polynomial is assumed:

$$\theta(t)=1+c_1\tau+c_2\tau^2+c_3\tau^3+c_4\tau^4$$

This equation automatically satisfy the boundary condition but if it is applied to differential equation a residue is remained as:

$$\frac{d\theta(t)}{dt}=c_1\tau+2c_2\tau\ +3c_3\tau^2+4c_4\tau^3$$

$$X\frac{d\theta(t)}{dt}+\theta(t)=0$$

$$R=X(c_1\tau+2c_2\tau\ +3c_3\tau^2+4c_4\tau^3)+(1+c_1\tau+c_2\tau^2+c_3\tau^3+c_4\tau^4)$$

$$R=1+(X+\tau)c_1+(2\tau X+\tau^2)c_2+(3\tau^2X+\tau^3)c_3+(4\tau^3+\tau^4)c_4$$

Assuming that equation is valid in the range of $0\le\tau\le1$ the equation will be evaluate at collocation locations such as:

$\tau_1=0.25\quad\tau_2=0.50\quad\tau_3=0.75\quad\tau_4=1.0\quad\tau_{max}=60\ s$

This method is called **Collocation method**

Coefficients can be obtained by solving the system of equations

$$\begin{bmatrix}(X+\tau_1) & (2\tau_1X+\tau_1^2) & (3\tau_1^2X+\tau_1^3) & (4\tau_1^3+\tau_1^4)\\ (X+\tau_2) & (2\tau_2X+\tau_2^2) & (3\tau_2^2X+\tau_2^3) & (4\tau_2^3+\tau_2^4)\\ (X+\tau_3) & (2\tau_3X+\tau_3^2) & (3\tau_3^2X+\tau_3^3) & (4\tau_3^3+\tau_3^4)\\ (X+\tau_4) & (2\tau_4X+\tau_4^2) & (3\tau_4^2X+\tau_4^3) & (4\tau_4^3+\tau_4^4)\end{bmatrix}\begin{Bmatrix}c_1\\c_2\\c_3\\c_3\end{Bmatrix}=\begin{Bmatrix}-1\\-1\\-1\\-1\end{Bmatrix}$$

Let us write a small program to do this task:

```
 import java.io.*;

class collocation
{

  public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss eleme
  int i,j,k,p,ii,jj;
  System.out.println("Original matrix :\n"+Matrix.toString(a));
  for(k=0;k<(n-1);k++)
  { //pivoting
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
                System.out.println("pivoted matrix : k="+k+"\n"+Matrix.toString(a));
              //Solving gauss elimination
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
```

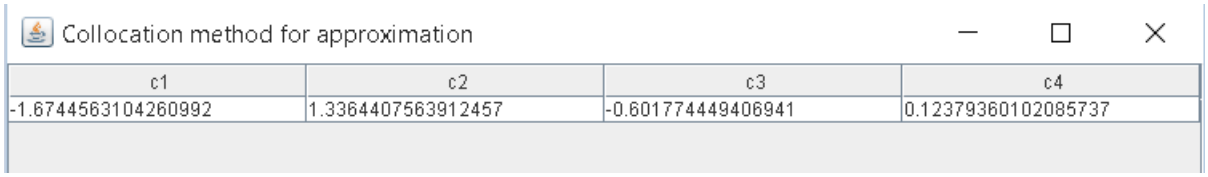
```
      {   a[i][j]-=carpan*a[k][j]; }
          b[i]   =b[i]   -carpan*b[k];
               }
  }
  System.out.println("final upper triangular matrix : \n"+Matrix.toString(a));
  //back substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }
  public static double[] c(double X, double to[])
  {  double a[][]=new double[4][4];
     a[0][0]=(X+to[0]);a[1][0]=(X+to[1]);a[2][0]=(X+to[2]);a[3][0]=(X+to[3]);
     a[0][1]=(2.0*X*to[0]+to[0]*to[0]);a[1][1]=(2.0*X*to[1]+to[1]*to[1]);
     a[2][1]=(2.0*X*to[2]+to[2]*to[2]);a[3][1]=(2.0*X*to[3]+to[3]*to[3]);
     a[0][2]=(3.0*X*to[0]*to[0]+to[0]*to[0]*to[0]);a[1][2]=(3.0*X*to[1]*to[1]+to[1]*to[1]*to[1]);
              a[2][2]=(3.0*X*to[2]*to[2]+to[2]*to[2]*to[2]);a[3][2]=(3.0*X*to[3]*to[3]+to[3]*to[3]*to[3]);
     a[0][3]=(4.0*X*to[0]*to[0]*to[0]+to[0]*to[0]*to[0]*to[0]);
     a[1][3]=(4.0*X*to[1]*to[1]*to[1]+to[1]*to[1]*to[1]*to[1]);
     a[2][3]=(4.0*X*to[2]*to[2]*to[2]+to[2]*to[2]*to[2]*to[2]);
     a[3][3]=(4.0*X*to[3]*to[3]*to[3]+to[3]*to[3]*to[3]*to[3]);
              double b[]={-1,-1,-1,-1};
              Text.print(a,"a");
              Text.print(b,"b");
              double c[]=pivotlugauss(a,b);
              return c;
  }

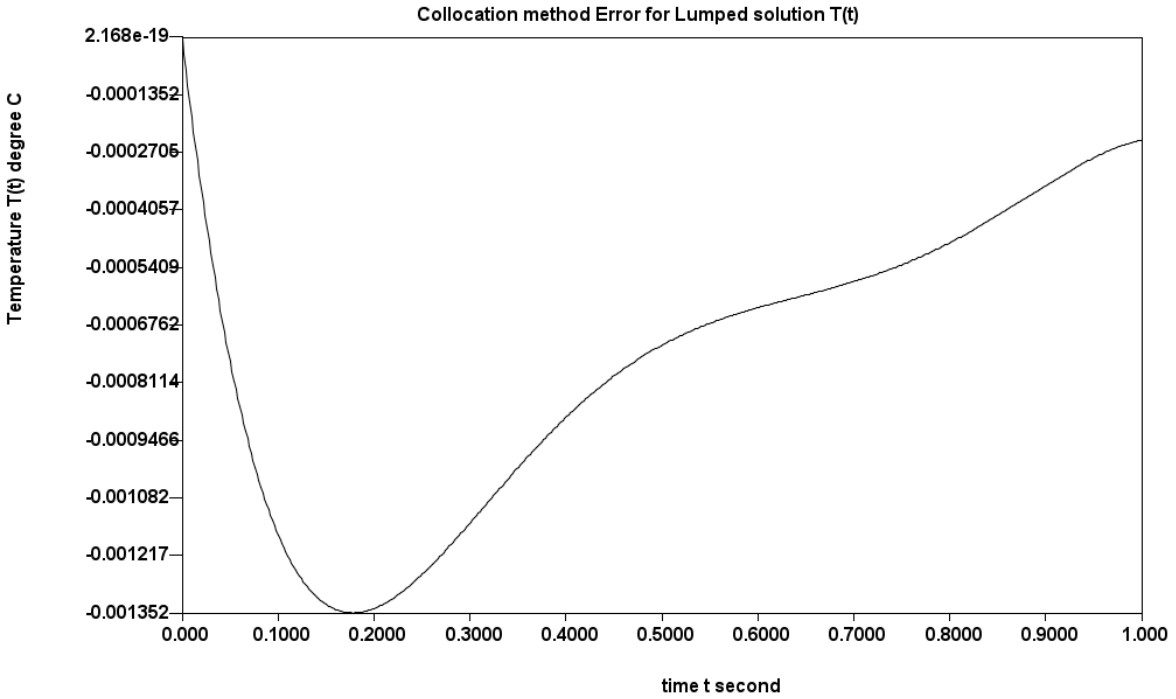
  public static void main(String args[]) throws IOException
  {
  double toi[]={0.25,0.5,0.75,1.0};
  double ro=8500; //kg/m^3
  double Cp=400; //J/kgK
  double h=80; //W/m2K
  double D=0.02;
  double k=20; //W/mK
  double V=Math.PI*D*D*D/6;
  double A=Math.PI*D*D;
  double Bi=h*D/6/k;
  System.out.println("Bi="+Bi);
  double tomax=240;
  double X=ro*V*Cp/(tomax*h*A);
  double c[]=c(X,toi);
  String c1[]={"c1","c2","c3","c4"};
  Text.print(c,c1,"Collocation method for approximation");
  if_x f=to->Math.exp(-to/X);
  if_x f1=to->1+c[0]*to+c[1]*to*to+c[2]*to*to*to+c[3]*to*to*to*to;
   if_x df=to->f.func(to)-f1.func(to);
  Plot pp=new Plot(df,0,1);
  pp.setPlabel("Collocation method Error for Lumped solution T(t)");
  pp.setXlabel("time t second");
  pp.setYlabel("Temperature T(t) degree C");
  pp.plot();
   }
}
```

0.840278	0.357639	0.126302	0.040799	C1
1.090278	0.840278	0.567708	0.357639	c2
1.340278	1.447917	1.417969	1.3125	c3
1.590278	2.180556	2.770833	3.361111	c4

-1
-1
-1
-1

c1	c2	c3	c4
-1.67446	1.336441	-0.60177	0.123794





Can ve increase the number of data points while keeping degree of polynomial constant. It is possible if we use singular value decomposition method to solve the matrix system.

SINGULAR VALUE DECOMPOSITION FOR SOLVING SYSTEM OF EQUATIONS

Let A be an m x n matrix such that the number of rows m is greater than or equal to the number of columns n. Then there exists:
1) an m x n column orthogonal matrix **U** (a matrix Q is orthogonal if its transpose is equal to its inverse)
2) an n x n diagonal matrix **S,** with positive or zero elements, and

3) an n x n orthogonal matrix V such that:

$$\boldsymbol{A=USV^T}$$

This is the **Singular Value Decomposition (SVD) of matrix A.** If a matrix can be decomposed this way, its inverse value will be existed as:

$$\boldsymbol{A^{-1}=V\ [diag\ (s_1^{-1},s_2^{-1},\ldots,s_n^{-1})]\ (U^T)}$$

It should ve note that because of the orhogonal properties of U and V inverses are equal to transpose values. Then a system of equation can be simply solve as $x=A^{-1}b$
Now let us look at these processes in more details with an example

$$A=\begin{bmatrix}4 & 0\\ 3 & -5\end{bmatrix}$$

The singular values of A are the square roots of the eigenvalues of A^TA

$$A^T=\begin{bmatrix}4 & 3\\ 0 & -5\end{bmatrix}$$

$$A^TA=\begin{bmatrix}4 & 3\\ 0 & -5\end{bmatrix}\begin{bmatrix}4 & 0\\ 3 & -5\end{bmatrix}=\begin{bmatrix}25 & -15\\ -15 & 25\end{bmatrix}$$

Eigenvalues of A^TA is

$\lambda = \begin{bmatrix} 40 \\ 10 \end{bmatrix}$ Root of the eigenvalues constructs the matrix S, by using eigenvalues as the diagonal of the matrix. And then the inverse can be easily computed by inverting the eigenvalues.

$$S = \begin{bmatrix} \sqrt{40} & 0 \\ 0 & \sqrt{10} \end{bmatrix} = \begin{bmatrix} 6.3245 & 0 \\ 0 & 3.1622 \end{bmatrix}$$

$$S^{-1} = \begin{bmatrix} 1/6.3245 & 0 \\ 0 & 1/3.1622 \end{bmatrix} = \begin{bmatrix} 0.1581 & 0 \\ 0 & 0.3162 \end{bmatrix}$$

The eigenvectors corresponding to these eigenvalues are:

$$\left[A^T A - \lambda I\right][X^1] = \begin{bmatrix} 25-40 & -15 \\ -15 & 25-40 \end{bmatrix}\begin{bmatrix} x_1^1 \\ x_2^1 \end{bmatrix} = 0$$

$$\left[A^T A - \lambda I\right][X^2] = \begin{bmatrix} 25-10 & -15 \\ -15 & 25-10 \end{bmatrix}\begin{bmatrix} x_1^2 \\ x_2^2 \end{bmatrix} = 0$$

$$[X] = \begin{bmatrix} 0.707106781186548 & 0.707106781186548 \\ -0.707106781186548 & 0.707106781186548 \end{bmatrix}$$

The eigenvector values will be normalized by using Length factor

$L = \sqrt{(x_1)^2 + (x_2)^2}$ Then the normalized Value will be V vector

$$V = \begin{bmatrix} 0.707106781186548 & 0.707106781186548 \\ -0.707106781186548 & 0.707106781186548 \end{bmatrix}$$

The U vector will be calculated as

$$U = AVS^{-1} = \begin{bmatrix} 0.4472 & 0.8944 \\ 0.8944 & -0.4472 \end{bmatrix}$$

Assuming that equation is valid in the range of $0 \le \tau \le 1$ the equation will be evaluate at collocation locations such as:

$$\theta(t) = 1 + c_1\tau + c_2\tau^2 + c_3\tau^3 + c_4\tau^4$$

$\tau_1 = 0.1 \quad \tau_2 = 0.2 \quad \tau_3 = 0.3 \quad \tau_4 = 0.4 \quad \tau_5 = 0.5 \quad \tau_6 = 0.6 \quad \tau_7 = 0.7 \quad \tau_8 = 0.8 \quad \tau_8 = 0.9$
$\tau_{10} = 1.0 \quad \tau_{max} = 60\ s$

$$\begin{bmatrix} (X+\tau_1) & (2\tau_1 X + \tau_1^2) & (3\tau_1^2 X + \tau_1^3) & (4\tau_1^3 + \tau_1^4) \\ (X+\tau_2) & (2\tau_2 X + \tau_2^2) & (3\tau_2^2 X + \tau_2^3) & (4\tau_2^3 + \tau_2^4) \\ (X+\tau_3) & (2\tau_3 X + \tau_3^2) & (3\tau_3^2 X + \tau_3^3) & (4\tau_3^3 + \tau_3^4) \\ (X+\tau_4) & (2\tau_4 X + \tau_4^2) & (3\tau_4^2 X + \tau_4^3) & (4\tau_3^3 + \tau_3^4) \\ (X+\tau_5) & (2\tau_5 X + \tau_5^2) & (3\tau_5^2 X + \tau_5^3) & (4\tau_5^3 + \tau_5^4) \\ (X+\tau_6) & (2\tau_6 X + \tau_6^2) & (3\tau_6^2 X + \tau_6^3) & (4\tau_6^3 + \tau_6^4) \\ (X+\tau_7) & (2\tau_7 X + \tau_7^2) & (3\tau_7^2 X + \tau_7^3) & (4\tau_7^3 + \tau_7^4) \\ (X+\tau_8) & (2\tau_8 X + \tau_8^2) & (3\tau_8^2 X + \tau_8^3) & (4\tau_8^3 + \tau_8^4) \\ (X+\tau_9) & (2\tau_9 X + \tau_9^2) & (3\tau_9^2 X + \tau_9^3) & (4\tau_9^3 + \tau_9^4) \\ (X+\tau_{10}) & (2\tau_{10} X + \tau_{10}^2) & (3\tau_{10}^2 X + \tau_{10}^3) & (4\tau_{10}^3 + \tau_{10}^4) \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_3 \end{Bmatrix} = \begin{Bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{Bmatrix}$$

Collocation with singular value decomposion **collocation_SVD.java**

```
import java.io.*;

class collocation_SVD
{

  public static double[] pivotlugauss(double a[][],double b[])
```

```
{ //gauss elimination with partial pivoting
int n=b.length;
double x[]=new double[n];
double carpan=0;
double toplam=0;
double buyuk;
double dummy=0;
//gauss eleme
int i,j,k,p,ii,jj;
System.out.println("Original matrix :\n"+Matrix.toString(a));
for(k=0;k<(n-1);k++)
{ //pivoting
            p=k;
            buyuk=Math.abs(a[k][k]);
            for(ii=k+1;ii<n;ii++)
            {  dummy=Math.abs(a[ii][k]);
              if(dummy > buyuk) {buyuk=dummy;p=ii;}
            }
            if(p!=k)
            { for(jj=k;jj<n;jj++)
              { dummy=a[p][jj];
                a[p][jj]=a[k][jj];
                a[k][jj]=dummy;
              }
              dummy=b[p];
              b[p]=b[k];
              b[k]=dummy;
            }
              System.out.println("pivoted matrix : k="+k+"\n"+Matrix.toString(a));
            //Solving gauss elimination
  for(i=k+1;i<n;i++)
  {  carpan=a[i][k]/a[k][k];
     a[i][k]=0;
     for(j=k+1;j<n;j++)
     {   a[i][j]-=carpan*a[k][j]; }
        b[i]   =b[i]   -carpan*b[k];
            }
}
System.out.println("final upper triangular matrix : \n"+Matrix.toString(a));
//back substitution
x[n-1]=b[n-1]/a[n-1][n-1];
for(i=n-2;i>=0;i--)
{
  toplam=0;
  for(j=i+1;j<n;j++)
  {  toplam+=a[i][j]*x[j];}
  x[i]=(b[i]-toplam)/a[i][i];
}
return x;
}
public static double[] c(double X, double to[])
{  double a[][]=new double[10][4];

a[0][0]=(X+to[0]);a[1][0]=(X+to[1]);a[2][0]=(X+to[2]);a[3][0]=(X+to[3]);a[4][0]=(X+to[4]);a[5][0]=(X+to[5]);a[6][0]=(X+to[6]);a[7][0]=
(X+to[7]);a[8][0]=(X+to[8]);a[9][0]=(X+to[9]);

a[0][1]=(2.0*X*to[0]+to[0]*to[0]);a[1][1]=(2.0*X*to[1]+to[1]*to[1]);a[2][1]=(2.0*X*to[2]+to[2]*to[2]);a[3][1]=(2.0*X*to[3]+to[3]*to[3]
);a[4][1]=(2.0*X*to[4]+to[4]*to[4]);a[5][1]=(2.0*X*to[5]+to[5]*to[5]);a[6][1]=(2.0*X*to[6]+to[6]*to[6]);a[7][1]=(2.0*X*to[7]+to[7]*to[
7]);a[8][1]=(2.0*X*to[8]+to[8]*to[8]);a[9][1]=(2.0*X*to[9]+to[9]*to[9]);

a[0][2]=(3.0*X*to[0]*to[0]+to[0]*to[0]*to[0]);a[1][2]=(3.0*X*to[1]*to[1]+to[1]*to[1]*to[1]);a[2][2]=(3.0*X*to[2]*to[2]+to[2]*to[2]*to[
2]);a[3][2]=(3.0*X*to[3]*to[3]+to[3]*to[3]*to[3]);a[4][2]=(3.0*X*to[4]*to[4]+to[4]*to[4]*to[4]);a[5][2]=(3.0*X*to[5]*to[5]+to[5]*to[5]*
to[5]);a[6][2]=(3.0*X*to[6]*to[6]+to[6]*to[6]*to[6]);a[7][2]=(3.0*X*to[7]*to[7]+to[7]*to[7]*to[7]);a[8][2]=(3.0*X*to[8]*to[8]+to[8]*to[
8]*to[8]);a[9][2]=(3.0*X*to[9]*to[9]+to[9]*to[9]*to[9]);
    a[0][3]=(4.0*X*to[0]*to[0]*to[0]+to[0]*to[0]*to[0]*to[0]); a[1][3]=(4.0*X*to[1]*to[1]*to[1]+to[1]*to[1]*to[1]*to[1]);
a[2][3]=(4.0*X*to[2]*to[2]*to[2]+to[2]*to[2]*to[2]*to[2]); a[3][3]=(4.0*X*to[3]*to[3]*to[3]+to[3]*to[3]*to[3]*to[3]);
a[4][3]=(4.0*X*to[4]*to[4]*to[4]+to[4]*to[4]*to[4]*to[4]);a[5][3]=(4.0*X*to[5]*to[5]*to[5]+to[5]*to[5]*to[5]*to[5]);
a[6][3]=(4.0*X*to[6]*to[6]*to[6]+to[6]*to[6]*to[6]*to[6]);a[7][3]=(4.0*X*to[7]*to[7]*to[7]+to[7]*to[7]*to[7]*to[7]);
a[8][3]=(4.0*X*to[8]*to[8]*to[8]+to[8]*to[8]*to[8]*to[8]);a[9][3]=(4.0*X*to[9]*to[9]*to[9]+to[9]*to[9]*to[9]*to[9]);
            double b[]={-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};
            Text.print(a);
            Text.print(b);
            SVD q=new SVD(a);
   double c[]=q.solve(b);
            return c;
}
```

```
  public static void main(String args[]) throws IOException
  {
  double toi[]={0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0};
  double ro=8500; //kg/m^3
  double Cp=400; //J/kgK
  double h=80; //W/m2K
  double D=0.02;
  double k=20; //W/mK
  double V=Math.PI*D*D*D/6;
  double A=Math.PI*D*D;
  double Bi=h*D/6/k;
  System.out.println("Bi="+Bi);
  double tomax=240;
  double X=ro*V*Cp/(tomax*h*A);
  double c[]=c(X,toi);
  String c1[]={"c1","c2","c3","c4"};
  Text.print(c,c1,"Collocation method for approximation");
  if_x f=to->Math.exp(-to/X);
  if_x f1=to->1+c[0]*to+c[1]*to*to+c[2]*to*to*to+c[3]*to*to*to*to;
  Plot pp=new Plot(f,0,1);
  pp.addFunction(f1,0,1);
  pp.plot();
  }
}
```

Singular value decomposition linear system of equation solving method by using Jacobi eigenvalue method SVD.java

```
import java.io.*;
import javax.swing.*;

class SVD
{
double U[][];
double V[][];
double S[][];
double invS[][];

public static double[][] Transpose(double [][] left)
{
//transpose matrix (if A=a(i,j) Transpose(A)=a(j,i)
int i,j;
int n=left.length;
int m=left[0].length;
double b[][];
b=new double[m][n];
for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    {
    b[j][i]=left[i][j];
    }
  }
return b;
}

public static double[][] T(double [][] left)
{return Transpose(left);}

public static double[][] I(int n)
{
//unit matrix
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
    b[i][i]=1.0;
return b;
}
//Converts vector to diagonal values of matrix
public static double[][] S(double[] left)
{
int n=left.length;
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
```

```
    b[i][i]=left[i];
return b;
}
//Converts double to diagonal values of matrix
public static double[][] S(double left,int n)
{
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
    b[i][i]=left;
return b;
}
//takes invert value of a diagonal matrix
public static double[][] invS(double[][] S)
{
int n=S.length;
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
    {  if(S[i][i]==0) b[i][i]=0;
                else b[i][i]=1.0/S[i][i];
              }
return b;
}
//square root of a diagonal matrix
public static double[][] sqrtS(double[] left)
{
int n=left.length;
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
    b[i][i]=Math.sqrt(left[i]);
return b;
}
//multiply two matrices
public static double[][] multiply(double[][] left,double[][] right)
{
//multiplication of two matrices
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right[0].length;
int n2=right.length;
double[][] b;
b=new double[n1][m2];
 if(m1 !=n2 )
 {
 System.out.println("inner matrix dimensions must agree"+"m1="+m1+"n2="+n2);
 return b;
 }
 for(i=0;i<n1;i++)
   {
   for(j=0;j<m2;j++)
     {
     for(k=0;k<m1;k++)
                 {b[i][j]+=left[i][k]*right[k][j];
                 }
     }
   }
return b;
//end of multiply of two matrices
}

//multiply a matrix with a double value
public static double[][] multiply(double[][] left,double right)
{
//multiplying a matrix with a constant
int i,j;
int n=left.length;
int m=left[0].length;
double b[][];
b=new double[n][m];
for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    b[i][j]=left[i][j]*right;
  }
```

```
return b;
//end of multiplying a matrix with a constant double
}

public static double[] multiply(double[][] left,double[] right)
{
//multiplication of one matrix with one vector
int ii,jj,i,j,k;
int n1=left.length;
int m1=left[0].length;

int m2=right.length;
double[] b;
b=new double[m2];
 for(i=0;i<n1;i++)
   {
     b[i]=0;
     for(k=0;k<m1;k++)
                {b[i]+=left[i][k]*right[k];
                }
   }
return b;
//end of multiply of a matrix and a vector
}

//substract two matrices
public static double[][] substract(double[][] left,double[][] right)
{
//addition of two matrices
int n1=left.length;
int m1=left[0].length;
int n2=right.length;
int m2=right[0].length;
int nMax,mMax;
int i,j;
if(m1>=m2) mMax=m1;
else       mMax=m2;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[][];
b=new double[nMax][mMax];
for(i=0;i<n1;i++)
  {
  for(j=0;j<m1;j++)
    {
    b[i][j]=b[i][j]+left[i][j];
    }
  }
for(i=0;i<n2;i++)
  {
  for(j=0;j<m2;j++)
    {
    b[i][j]=b[i][j]-right[i][j];
    }
  }
return b;
//end of matrix substraction method
}

public SVD(double A[][])
{initilize(A);}

public static double[] solve(double a[][],double b[])
{ SVD q=new SVD(a);
 double x[]=multiply(multiply(q.V,q.invS),multiply(SVD.T(q.U),b));
 return x;
}

public  double[] solve(double b[])
{
 double x[]=multiply(multiply(V,invS),multiply(T(U),b));
 return x;
}

public void initilize(double A[][])
{
  double ATA[][]=multiply(Transpose(A),A);
```

```
  double c[][];
  c=eigen_jacobi(ATA);
  int n=c[0].length;
  int n1=c.length;
  System.out.println("c ilk ="+Matrix.toString(c));
  double dummy[]=new double[n1];
  for(int i=0;i<n;i++)
  {for(int k=i+1;k<n;k++)
    {if(c[0][i]<c[0][k])
      { for(int j=0;j<n1;j++)
        { dummy[j]=c[j][k];
          c[j][k]=c[j][i];
          c[j][i]=dummy[j];
        }
      }
    }
  }
  double x[]=new double[n];
  for(int i=0;i<n;i++)
  {x[i]=c[0][i];}
  System.out.println("c"+Matrix.toString(c));
  S=sqrtS(x);
  invS=invS(S);
  double xx[][]=new double[n][n];
  V=new double[n][n];
  U=new double[n][n];
  double L[]=new double[n];
  for(int i=0;i<n;i++)
  { for(int j=0;j<n;j++)
    {xx[i][j]=c[i+1][j];}
  }
  System.out.println("xx"+Matrix.toString(xx));
  double bb[]=new double[n];
  for(int j=0;j<n;j++)
  { for(int i=0;i<n;i++)
    {L[j]+=xx[i][j]*xx[i][j];}
    L[j]=Math.sqrt(L[j]);
    System.out.println("j)="+j+"LJ="+L[j]);
  }
  for(int i=0;i<n;i++)
  { for(int j=0;j<n;j++)
    {V[i][j]=xx[i][j]/L[i];}
  }
  U=multiply(A,multiply(V,invS));
}

  public static void print(String s)
  {JOptionPane.showMessageDialog(null,s,
     "Singular value decomposition : ",JOptionPane.PLAIN_MESSAGE);}

  public static double[][] eigen_jacobi(double A[][])
  {
   double eps=1e-15;
   double err=1;
   int m=100;
   int n=A.length;
   double P[][]=new double[n][n];
   double R[][]=new double[n][n];
   int p=n,q=n;
   double max=-9.99e100;
   double fi;
   P=I(n);
   for(int i=0;i<n;i++)
   {for(int j=i+1;j<n;j++)
      {if(Math.abs(A[i][j]) > max) {max=Math.abs(A[i][j]);p=i;q=j;}}
   }
   while(Math.abs(A[p][q]) > eps )
   {
    fi=0.5*Math.atan(2.0*A[p][q]/(A[p][p]-A[q][q]));
    //System.out.println("fi="+fi+"Apq="+A[p][q]+"App="+A[p][p]+"Aqq="+A[q][q]+"p="+p+"q="+q);
    R=I(n);
    R[p][p]=Math.cos(fi);
    R[q][q]=Math.cos(fi);
    R[p][q]=-Math.sin(fi);
    R[q][p]=Math.sin(fi);
    P=multiply(P,R);
    A=multiply(T(R),multiply(A,R));
```

```
 max=-9.99e100;
 for(int i=0;i<n;i++)
{for(int j=i+1;j<n;j++){if(Math.abs(A[i][j])>max) {max=Math.abs(A[i][j]);p=i;q=j;}}}
 }
double lambda[]=new double[n];
for(int i=0;i<n;i++) {lambda[i]=A[i][i];}
double PP[][]=new double[n+1][n];
PP[0]=lambda;
for(int i=0;i<n;i++)
{for(int j=0;j<n;j++)
  {PP[i+1][j]=P[i][j];}
}
return PP;
}

public double[][] A()
{return multiply(U,multiply(S,Transpose(V)));}

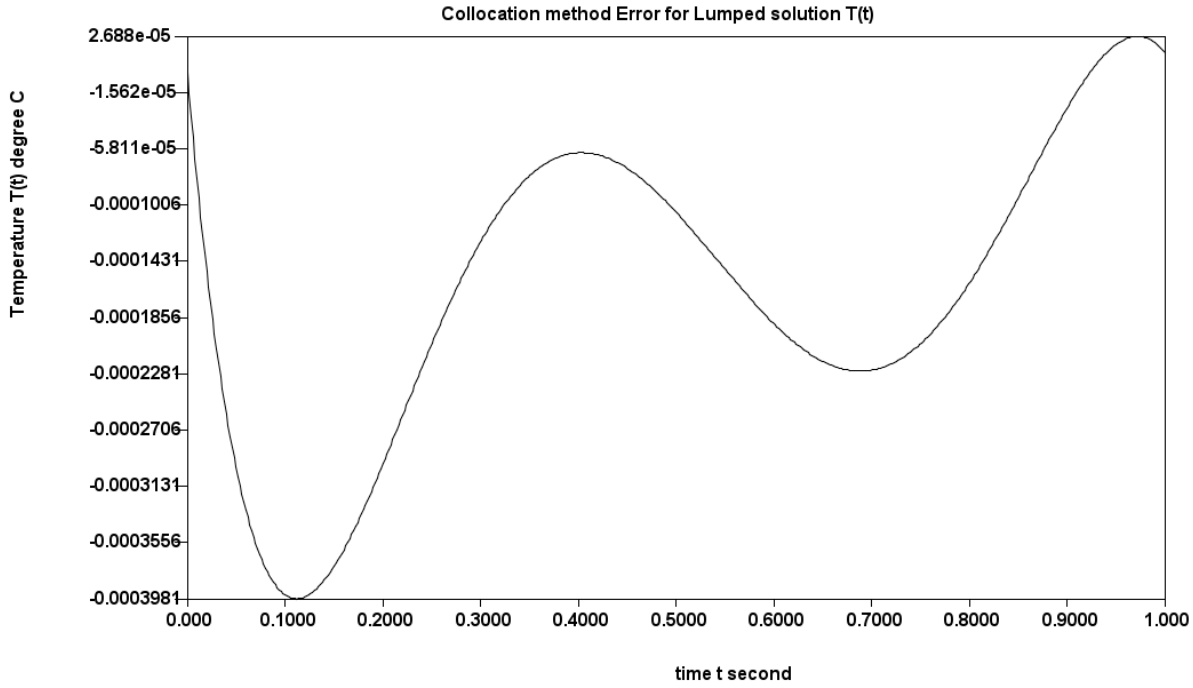
public static void main(String args[]) throws IOException
{
double a[][]={{3,2,-1},{2,-2,4},{-1,0.5,-1}};
double b[]={1,-2,0};
//Dynamic call to SVD
SVD q=new SVD(a);
double x[]=q.solve(b);
String s="a=\n"+Matrix.toString(a)+"\n";
s+="b=\n"+Matrix.toString(b)+"\n";
s+="S:\n"+Matrix.toString(q.S)+"\n";
s+="inv(S):\n"+Matrix.toString(q.invS)+"\n";
s+="V:\n"+Matrix.toString(q.V)+"\n";
s+="U:\n"+Matrix.toString(q.U)+"\n";
s+="Reconstituted a matrix : A=U*S*VT\n"+Matrix.toString(q.A())+"\n";
//Static call to SVD
//double x2[]=solve(a,b);
s+="x2="+Matrix.toString(x);
s+="SINGULAR VALUE DECOMPOSITION : \n"+Matrix.toString(x);
print(s);
}
}
```

0.690278	0.128056	0.018708	0.002461	c1
0.790278	0.276111	0.078833	0.020489	c2
0.890278	0.444167	0.186375	0.07185	c3
0.990278	0.632222	0.347333	0.176711	c4
1.090278	0.840278	0.567708	0.357639	
1.190278	1.068333	0.8535	0.6396	
1.290278	1.316389	1.210708	1.049961	
1.390278	1.584444	1.645333	1.618489	
1.490278	1.8725	2.163375	2.37735	
1.590278	2.180556	2.770833	3.361111	

-1
-1
-1
-1
-1
-1
-1
-1
-1
-1

Solution:

c1	c2	c3	c4
-1.68545	1.373155	-0.64461	0.140645

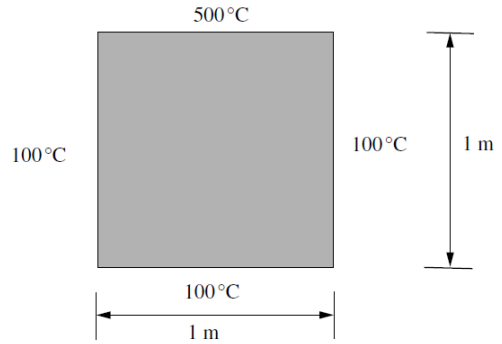


It is clear that Collocation method with SVD reduces the error. Maximum error in collocation (4 equation) is 0.001352. Maximum error in Collocation with SVD(10 equation) is 0.0003981

PROBLEM 8.3 TWO DIMENSIONAL STEADY-STATE PROBLEM APPROXIMATION WITH COLLOCATION METHOD

Now let us consider the following problem:

A square plate with temperatures of 100 C in three sides and 500 C in the fourth side is given. Plate sides are given as 1 m. Calculate the temperature profile



Differential equation:

$$\frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial x^2} = 0$$

Polynomial equation

$$T(x,y) = \sum_{i=0}^{n} \sum_{j=0}^{m} a_{i(n+1)+j} x^i y^j$$

$$\frac{\partial T(x,y)}{\partial x} = \sum_{i=1}^{n} \sum_{j=0}^{m} i * a_{i(n+1)+j} x^{i-1} y^j$$

$$\frac{\partial T(x,y)}{\partial y} = \sum_{i=0}^{n} \sum_{j=1}^{m} j * a_{i(n+1)+j} x^i y^{j-1}$$

$$\frac{\partial^2 T(x,y)}{\partial x^2} = \sum_{i=2}^{n}\sum_{j=0}^{m} i(i-1)a_{i(n+1)+j}x^{i-2}y^j$$

$$\frac{\partial^2 T(x,y)}{\partial y^2} = \sum_{i=0}^{n}\sum_{j=2}^{m} j(j-1)a_{i(n+1)+j}x^{i}y^{j-2}$$

$$\frac{\partial^2 T(x,y)}{\partial x \partial y} = \sum_{i=1}^{n}\sum_{j=1}^{m} (i-1)(j-1)a_{i(n+1)+j}x^{i-1}y^{j-1}$$

Differential equation becomes:

$$R = \frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial y^2}$$
$$= \sum_{i=2}^{n}\sum_{j=0}^{m} i(i-1)a_{i(n+1)+j}x^{i-2}y^j + \sum_{i=0}^{n}\sum_{j=2}^{m} j(j-1)a_{i(n+1)+j}x^{i}y^{j-2}$$

Equation have total K=(n+1)(m+1) coefficient

Equation divided to P=Px*Py pieces (greater than coefficient of polynomial surface)

x=dx*iPx $0 \le iPx \le (Px-1)$

y=dy*iPy $0 \le iPy \le (Py-1)$

Boundary conditions:

x=0 $0 \le y \le (Py-1)$ $T=T_1$

x=L $0 \le y \le (Py-1)$ $T=T_2$

y=0 $0 \le x \le (Px-1)$ $T=T_3$

y=w $0 \le x \le (Px-1)$ $T=T_4$

$$T_1 = \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j}y^j$$

$$T_2 = \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j}L^i y^j$$

$$T_3 = \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j}x^i$$

$$T_4 = \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j}x^i W^j$$

Not that number of equations are not equal to number of unkowns in the polynomial surface. Therefore Singular value decomposition method is used to solve the equations for finding the polynomial coefficients.

A code is given this program **poly2.java** , defines the two dimensional polynomial function and its coefficients and derivatives.

```
//Polynomial with two independent variable
//polinomial of order k=n*m
public class poly2
```

```
{ public int k;
  public double func(double a[],double x,double y,int n,int m)
  {   k=n*m;
      int k1=0;
      double xx=0;
      double total=0.0;
      for(int i=0;i<=n;i++)
      {  for(int j=0;j<=m;j++)
        { k1=i*(m+1)+j;
          xx=pow(x,y,i,j);
          total+=a[k1]*xx;
        }
      }
                return total;
  }

  public double dfuncx(double a[],double x,double y,int n,int m)
  {   k=n*m;
      int k1=0;
      double total=0.0;
      for(int i=0;i<=n;i++)
      {for(int j=0;j<=m;j++)
       {k1=i*(m+1)+j;
        double xx=dpowx(x,y,i,j);
        total+=a[k1]*xx;
        System.out.println("i="+i+"j="+j+"k1="+k1+"xx="+xx+"a="+a[k1]+"total="+total);
       }
      }
                return total;
  }

  public double dfuncy(double a[],double x,double y,int n,int m)
  {   k=n*m;
      int k1=0;
      double total=0.0;
      for(int i=0;i<=n;i++)
      {for(int j=0;j<=m;j++)
       {k1=i*(m+1)+j;
       double xx=dpowy(x,y,i,j);
        total+=a[k1]*xx;
        System.out.println("i="+i+"j="+j+"k1="+k1+"xx="+xx+"a="+a[k1]+"total="+total);
       }
      }
                return total;
  }

  public double d2funcx2(double a[],double x,double y,int n,int m)
  {   k=n*m;
      int k1=0;
      double total=0.0;
      if(a.length<k) System.out.println("error");
      for(int i=0;i<=n;i++)
      {for(int j=0;j<=m;j++)
       {k1=i*(m+1)+j;
        total+=a[k1]*d2powx2(x,y,i,j);
       }
      }
                return total;
  }
  public double d2funcy2(double a[],double x,double y,int n,int m)
  {   k=n*m;
      int k1=0;
      double total=0.0;
      for(int i=0;i<=n;i++)
      {for(int j=0;j<=m;j++)
       {k=i*(m+1)+j;
        total+=a[k]*d2powy2(x,y,i,j);
       }
      }
                return total;
  }
  public double d2funcxy(double a[],double x,double y,int n,int m)
  {   k=n*m;
      int k1=0;
      double total=0.0;
      if(a.length<k) System.out.println("error");
      for(int i=0;i<=n;i++)
```

```
    {for(int j=0;j<=m;m++)
     {k=i*(m-1)+j;
      total+=a[k]*d2powxy(x,y,i,j);
     }
    }
                return total;
}
//=====================================
public static double pow(double x,double y,int n,int m)
{ double pow=1.0;
                if(n==0 && m==0) return pow;
 else if(m==0 && n!=0)
 {for(int i=0;i<n;i++)
  {pow*=x;}
 }
 else if(n==0 && m!=0)
 {for(int j=0;j<m;j++)
  {pow*=y;}
 }
 else
 {for(int i=0;i<n;i++)
  {pow*=x;}
  for(int j=0;j<m;j++)
  {pow*=y;}
 }
 return pow;
}
public static double dpowx(double x,double y,int n,int m)
{ double powx=n;
  double powy=1.0;
  for(int i=1;i<n;i++)
  {powx*=x;}
  for(int j=0;j<m;j++)
  {powy*=y;}
  return powx*powy;
}
public static double dpowy(double x,double y,int n,int m)
{ double powx=1.0;
  double powy=m;
  for(int i=0;i<n;i++)
  {powx*=x;}
  for(int j=1;j<m;j++)
  {powy*=y;}
  return powx*powy;
}
public static double d2powx2(double x,double y,int n,int m)
{ double powx=n*(n-1);
  double powy=1.0;
  for(int i=2;i<n;i++)
  {powx*=x;}
  for(int j=0;j<m;j++)
  {powy*=y;}
  return powx*powy;
}
public static double d2powy2(double x,double y,int n,int m)
{ double powx=1.0;
  double powy=m*(m-1);
  for(int i=0;i<n;i++)
  {powx*=x;}
  for(int j=2;j<m;j++)
  {powy*=y;}
  return powx*powy;
 }

public static double d2powxy(double x,double y,int n,int m)
{ double powx=n;
  double powy=m;
  for(int i=1;i<n;i++)
  {powx*=x;}
  for(int j=1;j<m;j++)
  {powy*=y;}
  return powx*powy;
}
public static void main(String arg[])
{ double x=0.1;double y=0.1;
                double a[]={1,2,3,4,5,6,7,8,9};
  poly2 p=new poly2();
```

```
    System.out.println(p.func(a,x,y,2,2));
    System.out.println(p.dfuncx(a,x,y,2,2));
    System.out.println(p.dfuncy(a,x,y,2,2));
  }
}
```

Now we can construct solution of the differential equation by using **collocation** method:

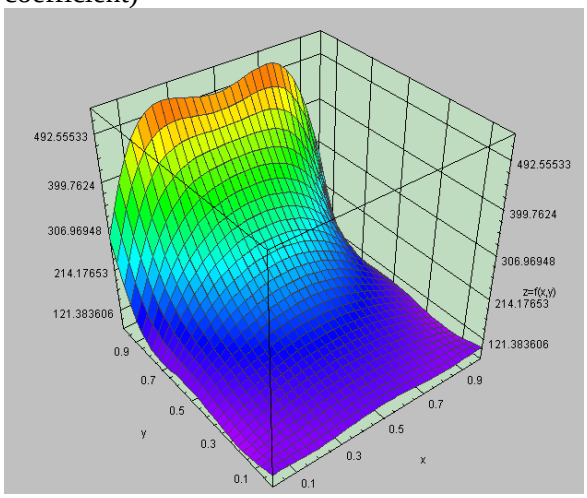
HT_poly2_T.java

```
 public class HT_poly2_T
{ int P,Px,Py,m,n,k,ip;
double T1,T2,T3,T4;
double dx,dy;
double L,W;
double A[][];
double B[];
double TT;
double x,y;
 public HT_poly2_T(double T1i,double T2i,double T3i,double T4i,int ni,int mi, int Pxi,int Pyi,double Li,double Wi)
 {Px=Pxi;Py=Pyi;P=Px*Py;n=ni;m=mi;k=(n+1)*(m+1);T1=T1i;T2=T2i;T3=T3i;T4=T4i;W=Wi;L=Li;dx=L/Px;dy=W/Py;TT=0;
 A=new double[P][k];B=new double[P];
 int k1=0;
 for(int ipx=0;ipx<Px;ipx++)
 {for(int ipy=0;ipy<Py;ipy++)
  {   x=dx*ipx;y=dy*ipy;
     ip=ipx*Py+ipy;
                if(ipx==0)
                {TT=T3;
                 for(int i=0;i<=n;i++)
                 {for(int j=0;j<=m;j++)
                  {k1=i*(m+1)+j;
                   A[ip][k1]=poly2.pow(x,y,i,j);
                   B[ip]=TT;
                  }
                 }
                }
                else if(ipx==(Px-1))
                {TT=T4;
                 for(int i=0;i<=n;i++)
                 {for(int j=0;j<=m;j++)
                  {k1=i*(m+1)+j;
                   A[ip][k1]=poly2.pow(x,y,i,j);
                   B[ip]=TT;
                  }
                 }
                }
                else if(ipy==0)
                {TT=T1;
                 for(int i=0;i<=n;i++)
                 {for(int j=0;j<=m;j++)
                  {k1=i*(m+1)+j;
                   A[ip][k1]=poly2.pow(x,y,i,j);
                   B[ip]=TT;
                  }
                 }
                }
                else if(ipy==(Py-1))
                {TT=T2;
                 for(int i=0;i<=n;i++)
                 {for(int j=0;j<=m;j++)
                  {k1=i*(m+1)+j;
                   A[ip][k1]=poly2.pow(x,y,i,j);
                   B[ip]=TT;
                  }
                 }
                }
                else
                {for(int i=0;i<=n;i++)
                 {for(int j=0;j<=m;j++)
                  {k1=i*(m+1)+j;
                   A[ip][k1]=poly2.d2powx2(x,y,i,j)+poly2.d2powy2(x,y,i,j);
                            B[ip]=0.0;
                  }
                 }
                }
```

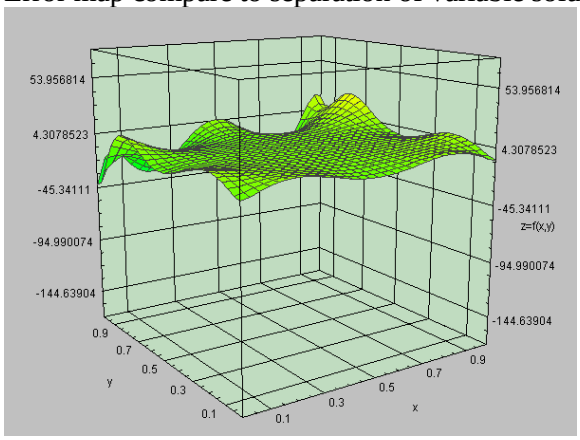
```
  }
 }
 Text.print(A,"A");
 Text.printT(B,"B");
 SVD q=new SVD(A);
 double a[]=q.solve(B);
 Text.printT(a,"a");
 double T[][]=new double[Px][Py];
 poly2 p=new poly2();
 for(int ipx=0;ipx<Px;ipx++)
 {for(int ipy=0;ipy<Py;ipy++)
  {   x=dx*ipx;y=dy*ipy;
      ip=ipx*Py+ipy;
      T[ipx][ipy]=p.func(a,x,y,n,m);
   } // end int ipy=0;ipy<Py;ipy++)
 }//end for(int ipx=0;ipx<Px;ipx++)
  Text.print(T,"T");
     double b[][]=surfacePlot.dataformat(0.0,L,0.0,W,T);
     surfacePlot p1;
     p1=new surfacePlot(b);
     p1.plot();
}
public static void main(String arg[])
{HT_poly2_T h=new HT_poly2_T(100,500.0,100.0,100.0,5,5,80,80,1.0,1.0);

}
}
```

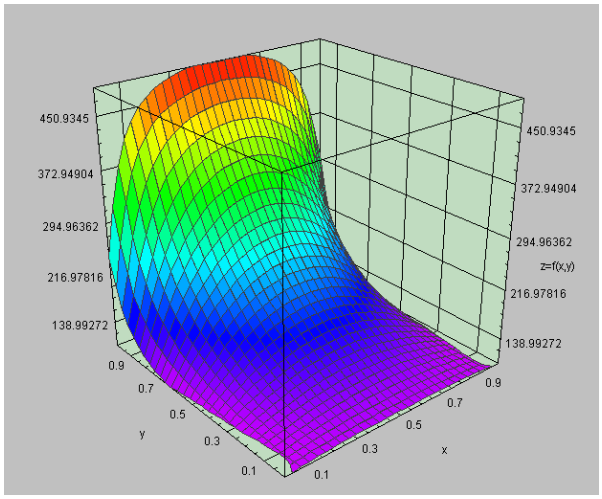
Output of the polynomial function (Temperature profile) n=7,m=7 Px=80,Py=80 (64 polynomial coefficient)



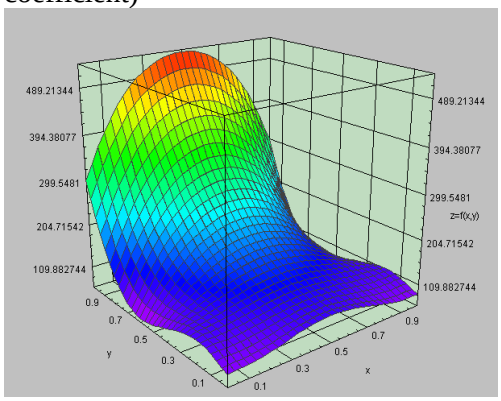
Error map compare to separation of variable solution



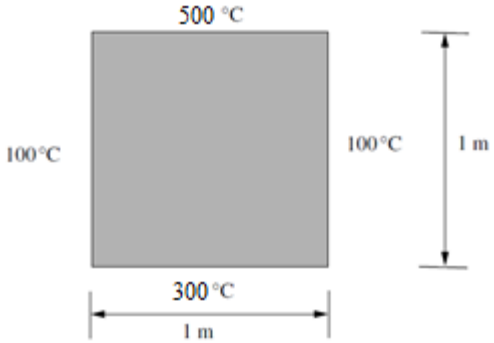
Separation of variable Solution of this problem



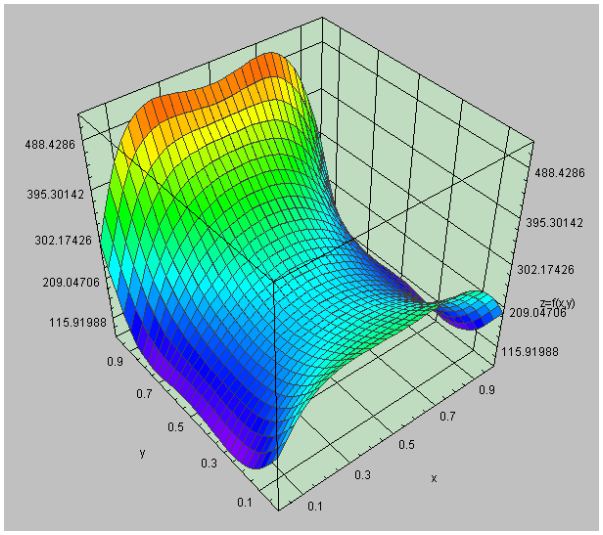
Output of the polynomial function(Temperature profile) n=5,m=5,Px=80,Py=80 (36 polynomial coefficient)



A square plate with temperatures as seen in the figures are given. Plate sides are given as 1 m. Calculate the temperature profile



Output of the polynomial function(Temperature profile) n=7,m=7,Px=80,Py=80



PROBLEM 8.4 TWO DIMENSIONAL STEADY-STATE PROBLEM APPROXIMATION WITH COLLOCATION METHOD

Now let us consider the following problem:

A square plate exposed to convective flux in each side.

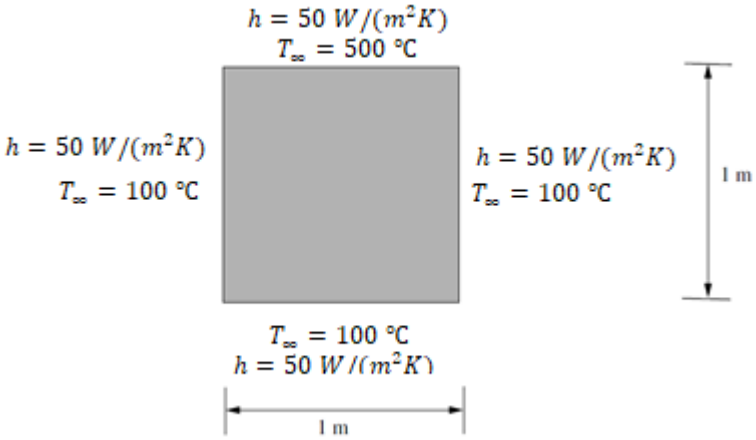
x=0 $h = 50\ W/(m^2K)$ $T_\infty = 100$ ℃

x=L=1 m $h = 50\ W/(m^2K)$ $T_\infty = 100$ ℃

y=0 $h = 50\ W/(m^2K)$ $T_\infty = 100$ ℃

y=W=1 m $h = 50\ W/(m^2K)$ $T_\infty = 500$ ℃

$k = 50\ W/(mK)$ Calculate the temperature profile



Differential equation:

$$\frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial x^2} = 0$$

Polynomial equation

$$T(x,y) = \sum_{i=0}^{n} \sum_{j=0}^{m} a_{i(n+1)+j} x^i y^j$$

$$\frac{\partial T(x,y)}{\partial x} = \sum_{i=1}^{n} \sum_{j=0}^{m} i * a_{i(n+1)+j} x^{i-1} y^j$$

$$\frac{\partial T(x,y)}{\partial y} = \sum_{i=0}^{n} \sum_{j=1}^{m} j * a_{i(n+1)+j} x^i y^{j-1}$$

$$\frac{\partial^2 T(x,y)}{\partial x^2} = \sum_{i=2}^{n}\sum_{j=0}^{m} i(i-1)a_{i(n+1)+j}x^{i-2}y^j$$

$$\frac{\partial^2 T(x,y)}{\partial y^2} = \sum_{i=0}^{n}\sum_{j=2}^{m} j(j-1)a_{i(n+1)+j}x^{i}y^{j-2}$$

$$\frac{\partial^2 T(x,y)}{\partial x \partial y} = \sum_{i=1}^{n}\sum_{j=1}^{m} (i-1)(j-1)a_{i(n+1)+j}x^{i-1}y^{j-1}$$

Differential equation becomes:

$$R = \frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial y^2}$$
$$= \sum_{i=2}^{n}\sum_{j=0}^{m} i(i-1)a_{i(n+1)+j}x^{i-2}y^j + \sum_{i=0}^{n}\sum_{j=2}^{m} j(j-1)a_{i(n+1)+j}x^{i}y^{j-2}$$

Equation have total K=(n+1)(m+1) coefficient

Equation divided to P=Px*Py pieces (greater than coefficient of polynomial surface)

x=dx*iPx $0 \le iPx \le (Px-1)$

y=dy*iPy $0 \le iPy \le (Py-1)$

Boundary conditions:

x=0 $h_1 = 50\ W/(m^2K)$ $T_{1\infty} = 100$ ℃

$$-k\left.\frac{\partial T(x,y)}{\partial x}\right|_{x=0} = h_1(T(x,y) - T_{1\infty})|_{x=0}$$

x=L=1 m $h_2 = 50\ W/(m^2K)$ $T_{2\infty} = 100$ ℃

$$-k\left.\frac{\partial T(x,y)}{\partial x}\right|_{x=L} = h_2(T(x,y) - T_{2\infty})|_{x=L}$$

y=0 $h_3 = 50\ W/(m^2K)$ $T_{3\infty} = 100$ ℃

$$-k\left.\frac{\partial T(x,y)}{\partial x}\right|_{y=0} = h_3(T(x,y) - T_{3\infty})|_{y=0}$$

y=W=1 m $h_4 = 50\ W/(m^2K)$ $T_{4\infty} = 500$ ℃

$$-k\left.\frac{\partial T(x,y)}{\partial x}\right|_{y=W} = h_4 T\big((x,y) - T_{4\infty}\big)\big|_{y=W}$$

x=0 $h_1 = 50\ W/(m^2K)$ $T_{1\infty} = 100$ ℃

$$T_1 = \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j}y^j$$

$$\frac{\partial T_1(x,y)}{\partial x} = \sum_{i=1}^{n}\sum_{j=0}^{m} i * a_{i(n+1)+j}y^j$$

$$-k\left[\sum_{i=1}^{n}\sum_{j=0}^{m} i * a_{i(n+1)+j}y^j\right] = h_1\left(\sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j}y^j - T_{1\infty}\right)$$

x=L=1 m $h_2 = 50\ W/(m^2K)$ $T_{2\infty} = 100$ ℃

$$T_2 = \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j} L^i y^j$$

$$\frac{\partial T_2(x,y)}{\partial x} = \sum_{i=1}^{n}\sum_{j=0}^{m} i * a_{i(n+1)+j} L^{i-1} y^j$$

$$-k\left[\sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j} L^i y^j\right] = h_2\left(\sum_{i=1}^{n}\sum_{j=0}^{m} i * a_{i(n+1)+j} L^{i-1} y^j - T_{2\infty}\right)$$

y=0 $h_3 = 50\ W/(m^2K)$ $T_{3\infty} = 100$ ℃

$$T_3 = \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j} x^i$$

$$\frac{\partial T_3(x,y)}{\partial y} = \sum_{i=0}^{n}\sum_{j=1}^{m} j * a_{i(n+1)+j} x^i$$

$$-k\left[\sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j} x^i\right] = h_3\left(\sum_{i=0}^{n}\sum_{j=1}^{m} j * a_{i(n+1)+j} x^i - T_{2\infty}\right)$$

y=W=1 m $h_4 = 50\ W/(m^2K)$ $T_{4\infty} = 500$ ℃

$$T_4 = \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j} x^i W^j$$

$$\frac{\partial T_4(x,y)}{\partial y} = \sum_{i=0}^{n}\sum_{j=1}^{m} j * a_{i(n+1)+j} x^i W^{j-1}$$

$$-k\left[\sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j} x^i W^j\right] = h_4\left(\sum_{i=0}^{n}\sum_{j=1}^{m} j * a_{i(n+1)+j} x^i W^{j-1} - T_{2\infty}\right)$$

Not that number of equations are not equal to number of unkowns in the polynomial surface. Therefore Singular value decomposition method is used to solve the equations for finding the polynomial coefficients. **poly2.java** , defined in the previous example problem will also be used in here.

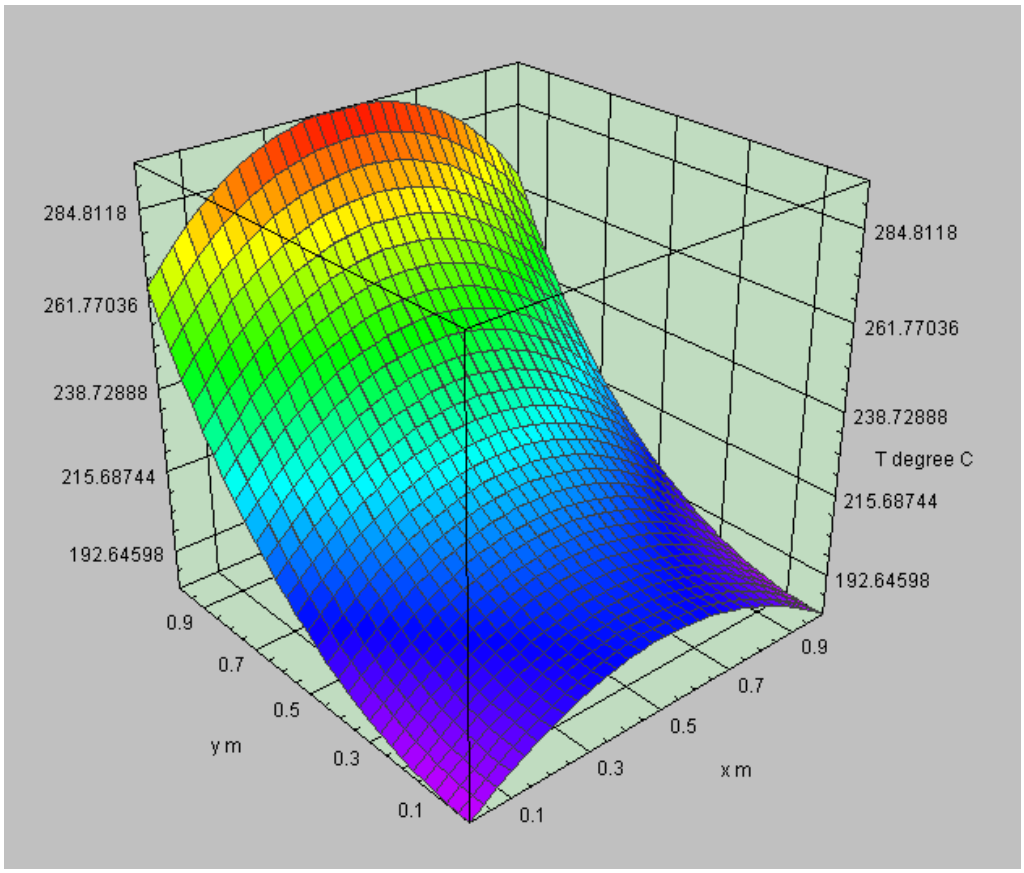
HT_poly2_hTinf.java

```
public class HT_poly2_hTinf
{ int P,Px,Py,m,n,k,ip;
double T1,T2,T3,T4;
double dx,dy;
double L,W;
double A[][];
double B[];
double TT;
double x,y;
double kcond;
double H1,H2,H3,H4;
 public HT_poly2_hTinf(double kcond1,double h1,double h2,double h3,double h4,double T1i,double T2i,double T3i,double T4i,int ni,int
mi, int Pxi,int Pyi,double Li,double Wi)
```

```
{kcond=kcond1;H1=h1/kcond;H2=h2/kcond;H3=H3/kcond;H4=h4/kcond;Px=Pxi;Py=Pyi;P=Px*Py;n=ni;m=mi;k=(n+1)*(m+1);T1=T1i;T
2=T2i;T3=T3i;T4=T4i;W=Wi;L=Li;dx=L/Px;dy=W/Py;TT=0;
 A=new double[P][k];B=new double[P];
 int k1=0;
 for(int ipx=0;ipx<Px;ipx++)
 {for(int ipy=0;ipy<Py;ipy++)
  {   x=dx*ipx;y=dy*ipy;
      ip=ipx*Py+ipy;
                 if(ipx==0)
                 {TT=T3;
                  for(int i=0;i<=n;i++)
                  {for(int j=0;j<=m;j++)
                   {k1=i*(m+1)+j;
                    A[ip][k1]=-poly2.dpowx(x,y,i,j)+H3*poly2.pow(x,y,i,j);
                    B[ip]=H3*TT;
                   }
                  }
                 }
                 else if(ipx==(Px-1))
                 {TT=T4;
                  for(int i=0;i<=n;i++)
                  {for(int j=0;j<=m;j++)
                   {k1=i*(m+1)+j;
                    A[ip][k1]=poly2.dpowx(x,y,i,j)+H4*poly2.pow(x,y,i,j);
                    B[ip]=H4*TT;
                   }
                  }
                 }
                 else if(ipy==0)
                 {TT=T1;
                  for(int i=0;i<=n;i++)
                  {for(int j=0;j<=m;j++)
                   {k1=i*(m+1)+j;
                    A[ip][k1]=-poly2.dpowy(x,y,i,j)+H1*poly2.pow(x,y,i,j);
                    B[ip]=H1*TT;
                   }
                  }
                 }
                 else if(ipy==(Py-1))
                 {TT=T2;
                  for(int i=0;i<=n;i++)
                  {for(int j=0;j<=m;j++)
                   {k1=i*(m+1)+j;
                    A[ip][k1]=poly2.dpowy(x,y,i,j)+H2*poly2.pow(x,y,i,j);
                    B[ip]=H2*TT;
                   }
                  }
                 }
                 else
                 {for(int i=0;i<=n;i++)
                  {for(int j=0;j<=m;j++)
                   {k1=i*(m+1)+j;
                    A[ip][k1]=poly2.d2powx2(x,y,i,j)+poly2.d2powy2(x,y,i,j);
                              B[ip]=0.0;
                   }
                  }
                 }
  }
 }
 Text.print(A,"A");
 Text.printT(B,"B");
 SVD q=new SVD(A);
 double a[]=q.solve(B);
 Text.printT(a,"a");
 double T[][]=new double[Px][Py];
 poly2 p=new poly2();
 for(int ipx=0;ipx<Px;ipx++)
 {for(int ipy=0;ipy<Py;ipy++)
  {   x=dx*ipx;y=dy*ipy;
      ip=ipx*Py+ipy;
      T[ipy][ipx]=p.func(a,x,y,n,m);
   } // end int ipy=0;ipy<Py;ipy++)
 }//end for(int ipx=0;ipx<Px;ipx++)
  Text.print(T,"T");
    double b[][]=surfacePlot.dataformat(0.0,L,0.0,W,T);
    surfacePlot p1;
```

```
    p1=new surfacePlot(b,"x m","y m","T degree C");
    p1.plot();
}
public static void main(String arg[])
{HT_poly2_hTinf h=new HT_poly2_hTinf(50,50,50,50,50,100,100.0,100.0,500.0,7,7,80,80,1.0,1.0);

}
}
```



Now let us consider the following problem:

A square plate exposed to convective flux in each side.

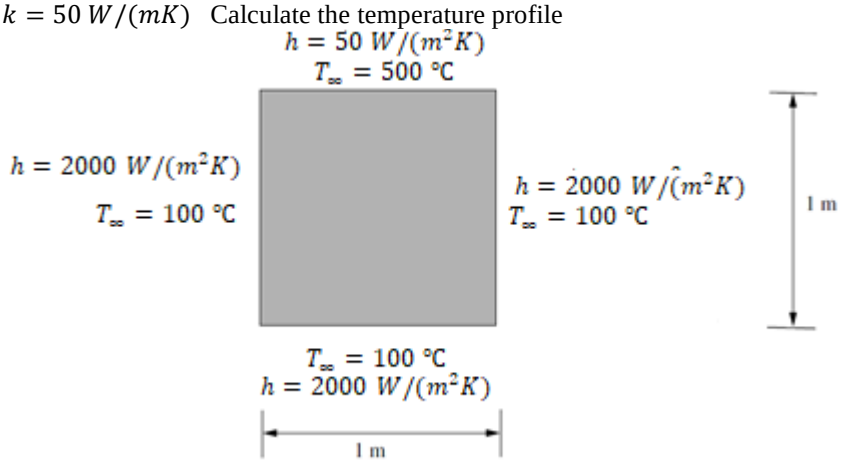
x=0 $h = 2000\ W/(m^2K)$ $T_\infty = 100$ ℃ (approaching to a constant temperature case)

x=L=1 m $h = 2000\ W/(m^2K)$ $T_\infty = 100$ ℃ (approaching to a constant temperature case)

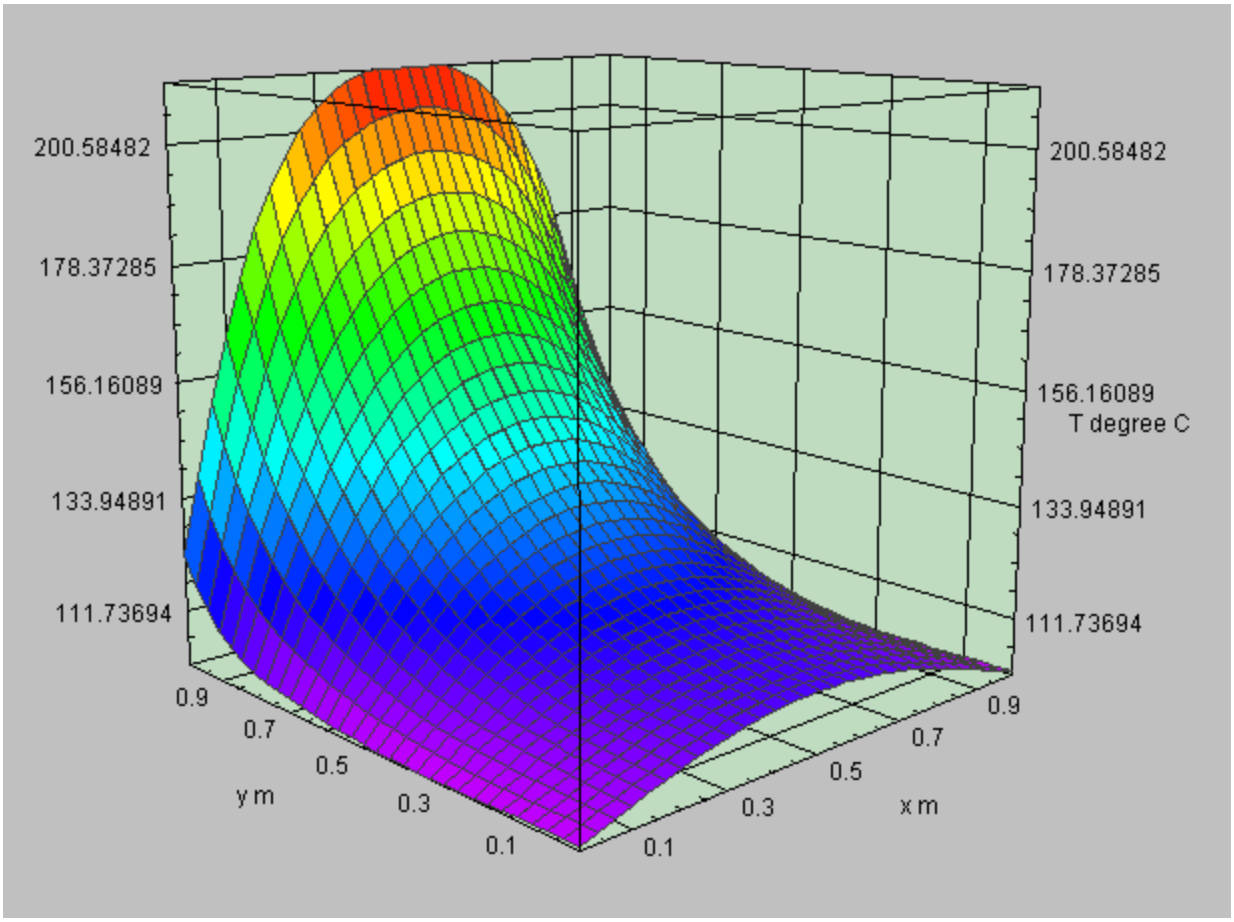
y=0 $h = 2000\ W/(m^2K)$ $T_\infty = 100$ ℃ (approachingto a constant temperature case)

y=W=1 m $h = 50\ W/(m^2K)$ $T_\infty = 500$ ℃

$k = 50\ W/(mK)$ Calculate the temperature profile




```
public static void main(String arg[])
{HT_poly2_hTinf h=new HT_poly2_hTinf(50,2000,2000,2000,50,100,100.0,100.0,500.0,7,7,80,80,1.0,1.0);
```



PROBLEM 8.5 TIME DEPENDENT LUMPED SOLUTION APPROXIMATION METHODS (LEAST SQUARE METHOD)

As a next approximation method we will consider **least square method**.

$$-hA_s(T(t)-T_\infty)=\rho V C_p\frac{dT(t)}{dt}$$

where A_s is surface area, V is volume, C_p is specific heat. By changing variables as:

$\theta(t)=\frac{T(t)-T_\infty}{T_i-T_\infty}\quad \tau=\frac{t}{\tau_{max}}\quad X=\frac{\rho V C_p}{\tau_{max}hA_s}$

equation becomes:

$$X\frac{d\theta(t)}{dt}+\theta(t)=0$$

Boundary condition t=0 θ =1so exact solution is:

$$\theta(t)=exp\left(-\frac{\tau}{X}\right)$$

Now let us create approximate solution, for this the following polynomial is assumed:

$$\theta(t)=1+c_1\tau+c_2\tau^2+c_3\tau^3+c_4\tau^4$$

This equation automatically satisfy the boundary condition but if it is applied to differential equation a residue is remained as:

$$\frac{d\theta(t)}{dt}=c_1\tau+2c_2\tau\ +3c_3\tau^2+4c_4\tau^3$$

$$X\frac{d\theta(t)}{dt}+\theta(t)=0$$

$$R=X(c_1\tau+2c_2\tau\ +3c_3\tau^2+4c_4\tau^3)+(1+c_1\tau+c_2\tau^2+c_3\tau^3+c_4\tau^4)$$

$$R=1+(X+\tau)c_1+(2\tau X+\tau^2)c_2+(3\tau^2X+\tau^3)c_3+(4\tau^3+\tau^4)c_4$$

It is possible to write least square equations as:

$$\frac{1}{2}\frac{\partial}{\partial c_i}\int_{\tau=0}^{\tau=1} R^2(\tau, c_1, c_2, c_3, c_4)d\tau = 0 \qquad i = 1,2,3,4$$

Considering R as

$$R = 1 + \phi_1 c_1 + \phi_2 c_2 + + \phi_3 c_3 + + \phi_4 c_4 = 1 + \sum_{i=1}^{4} \phi_i c_i$$

$$\frac{\partial R}{\partial c_i} = \phi_i \;\; i = 1,2,3,4$$

$$\int_{\tau=0}^{\tau=1} \phi_i \left(1 + \sum_{i=1}^{4} \phi_i c_i\right) d\tau = 0 \quad i = 1,2,3,4$$

So

$$\begin{bmatrix} \int \phi_1\phi_1 d\tau & \int \phi_1\phi_2 d\tau & \int \phi_1\phi_3 d\tau & \int \phi_1\phi_4 d\tau \\ \int \phi_2\phi_1 d\tau & \int \phi_2\phi_2 d\tau & \int \phi_2\phi_3 d\tau & \int \phi_2\phi_4 d\tau \\ \int \phi_3\phi_1 d\tau & \int \phi_3\phi_2 d\tau & \int \phi_3\phi_3 d\tau & \int \phi_3\phi_4 d\tau \\ \int \phi_4\phi_1 d\tau & \int \phi_4\phi_2 d\tau & \int \phi_4\phi_3 d\tau & \int \phi_4\phi_4 d\tau \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{Bmatrix} = \begin{Bmatrix} -\int \phi_1 d\tau \\ -\int \phi_2 d\tau \\ -\int \phi_3 d\tau \\ -\int \phi_4 d\tau \end{Bmatrix}$$

Each integration should be calculated and carried out into the matrix to determine the coefficients.

$$\int_{\tau=0}^{\tau=0}\phi_1\phi_1 d\tau = \int_{\tau=0}^{\tau=1}(X^2 + 2X\tau + \tau^2)d\tau = \left(X^2\tau + X\tau^2 + \frac{\tau^3}{3}\right)\Big|_{\tau=0}^{\tau=1} = X^2 + X + \frac{1}{3}$$

$$\int_{\tau=0}^{\tau=0}\phi_1\phi_2 d\tau = \int_{\tau=0}^{\tau=1}(X+\tau)*(2X\tau+\tau^2)d\tau = \int_{\tau=0}^{\tau=1}(2X^2\tau + 3X\tau^2 + \tau^3)d\tau = \left(X^2\tau^2 + X\tau^3 + \frac{\tau^4}{4}\right)\Big|_{\tau=0}^{\tau=1} = X^2 + X + \frac{1}{4}$$

$$\int_{\tau=0}^{\tau=0}\phi_1\phi_3 d\tau = \int_{\tau=0}^{\tau=1}(X+\tau)*(3X\tau^2+\tau^3)d\tau = \int_{\tau=0}^{\tau=1}(3X^2\tau^2 + 4X\tau^3 + \tau^4)d\tau = \left(X^2\tau^3 + X\tau^4 + \frac{\tau^5}{5}\right)\Big|_{\tau=0}^{\tau=1} = X^2 + X + \frac{1}{5}$$

$$\int_{\tau=0}^{\tau=0}\phi_1\phi_4 d\tau = \int_{\tau=0}^{\tau=1}(X+\tau)*(3X\tau^3+\tau^4)d\tau = \int_{\tau=0}^{\tau=1}(4X^2\tau^2 + 5X\tau^4 + \tau^5)d\tau = \left(X^2\tau 4^3 + X\tau^5 + \frac{\tau^6}{6}\right)\Big|_{\tau=0}^{\tau=1} = X^2 + X + \frac{1}{6}$$

$$\int_{\tau=0}^{\tau=0}\phi_2\phi_2 d\tau = \int_{\tau=0}^{\tau=1}(2X\tau+\tau^2)*(2X\tau+\tau^2)d\tau = \int_{\tau=0}^{\tau=1}(4X^2\tau^2 + 4X\tau^3 + \tau^4)d\tau = \left(\frac{4}{3}X^2\tau^2 + X\tau^4 + \frac{\tau^5}{5}\right)\Big|_{\tau=0}^{\tau=1} = \frac{4}{3}X^2 + X + \frac{1}{5}$$

$$\int_{\tau=0}^{\tau=0}\phi_2\phi_3 d\tau = \int_{\tau=0}^{\tau=1}(2X\tau+\tau^2)*(3X\tau^2+\tau^3)d\tau = \int_{\tau=0}^{\tau=1}(6X^2\tau^3 + 5X\tau^4 + \tau^5)d\tau = \left(\frac{6}{4}X^2\tau^4 + X\tau^5 + \frac{\tau^6}{6}\right)\Big|_{\tau=0}^{\tau=1} = \frac{6}{4}X^2 + X + \frac{1}{6}$$

$$\int_{\tau=0}^{\tau=0}\phi_2\phi_4 d\tau = \int_{\tau=0}^{\tau=1}(2X\tau+\tau^2)*(4X\tau^3+\tau^4)d\tau = \int_{\tau=0}^{\tau=1}(8X^2\tau^4 + 5X\tau^5 + \tau^6)d\tau = \left(\frac{8}{5}X^2\tau^5 + X\tau^6 + \frac{\tau^7}{7}\right)\Big|_{\tau=0}^{\tau=1} = \frac{8}{5}X^2 + X + \frac{1}{7}$$

$$\int_{\tau=0}^{\tau=0}\phi_3\phi_3 d\tau = \int_{\tau=0}^{\tau=1}(3X\tau^2+\tau^3)*(3X\tau^2+\tau^3)d\tau = \int_{\tau=0}^{\tau=1}(9X^2\tau^4 + 6X\tau^5 + \tau^6)d\tau = \left(\frac{9}{5}X^2\tau^5 + X\tau^6 + \frac{\tau^7}{7}\right)\Big|_{\tau=0}^{\tau=1} = \frac{9}{5}X^2 + X + \frac{1}{7}$$

$$\int_{\tau=0}^{\tau=0}\phi_3\phi_4 d\tau = \int_{\tau=0}^{\tau=1}(3X\tau^2+\tau^3)*(4X\tau^3+\tau^4)d\tau = \int_{\tau=0}^{\tau=1}(12X^2\tau^5 + 7X\tau^6 + \tau^7)d\tau = \left(\frac{12}{6}X^2\tau^6 + X\tau^7 + \frac{\tau^8}{8}\right)\Big|_{\tau=0}^{\tau=1} = 2X^2 + X + \frac{1}{8}$$

$$\int_{\tau=0}^{\tau=0}\phi_4\phi_4 d\tau = \int_{\tau=0}^{\tau=1}(4X\tau 3+\tau^4)*(4X\tau^3+\tau^4)d\tau = \int_{\tau=0}^{\tau=1}(16X^2\tau^6 + 8X\tau 7 + \tau^8)d\tau = \left(\frac{16}{7}X^2\tau^7 + X\tau^8 + \frac{\tau 9}{8}\right)\Big|_{\tau=0}^{\tau=1} = \frac{16}{7}X^2 + X + \frac{1}{9}$$

$$-\int_{\tau=0}^{\tau=0}\phi_1 d\tau = \int_{\tau=0}^{\tau=1}(X+\tau)d\tau = \left(X\tau + \frac{\tau^2}{2}\right)\Big|_{\tau=0}^{\tau=1} = -\left(X + \frac{1}{2}\right)$$

$$-\int_{\tau=0}^{\tau=0}\phi_2 d\tau=\int_{\tau=0}^{\tau=1}(2X\tau+\tau^2)d\tau=\left(X\tau^2+\frac{\tau^3}{3}\right)\Bigg|_{\tau=0}^{\tau=1}=-\left(X+\frac{1}{3}\right)$$

$$-\int_{\tau=0}^{\tau=0}\phi_3 d\tau=\int_{\tau=0}^{\tau=1}(3X\tau^2+\tau^3)d\tau=\left(X\tau^3+\frac{\tau^4}{4}\right)\Bigg|_{\tau=0}^{\tau=1}=-\left(X+\frac{1}{4}\right)$$

$$-\int_{\tau=0}^{\tau=0}\phi_4 d\tau=\int_{\tau=0}^{\tau=1}(3X\tau^4+\tau^4)d\tau=\left(X\tau^4+\frac{\tau^5}{5}\right)\Bigg|_{\tau=0}^{\tau=1}=-\left(X+\frac{1}{5}\right)$$

Now that all coefficients of the matrix is established, solution can be done by using a small application program **residue_least_square.java**

```
import java.io.*;

class residue_least_square
{

  public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss eleme
  int i,j,k,p,ii,jj;
  System.out.println("Original matrix :\n"+Matrix.toString(a));
  for(k=0;k<(n-1);k++)
  { //pivoting
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                   if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
                  b[k]=dummy;
                }
                  System.out.println("pivoted matrix : k="+k+"\n"+Matrix.toString(a));
                //Solving gauss elimination
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
                }
  }
  System.out.println("final upper triangular matrix : \n"+Matrix.toString(a));
  //back substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }
  public static double[] c(double X)
  {  double a[][]=new double[4][4];
     a[0][0]=(X*X+X+1.0/3.0);
     a[1][0]=(X*X+X+1.0/4.0);
     a[2][0]=(X*X+X+1.0/5.0);
     a[3][0]=(X*X+X+1.0/6.0);
```

```
    a[0][1]=a[1][0];
    a[1][1]=(4.0/3.0*X*X+X+1.0/5.0);
    a[2][1]=(6.0/4.0*X*X+X+1.0/6.0);
    a[3][1]=(8.0/5.0*X*X+X+1.0/7.0);
    a[0][2]=a[2][0];
    a[1][2]=a[2][1];
            a[2][2]=(9.0/5.0*X*X+X+1.0/7.0);
            a[3][2]=(2.0*X*X+X+1.0/8.0);
    a[0][3]=a[3][0];
    a[1][3]=a[3][1];
    a[2][3]=a[3][2];
    a[3][3]=(16.0/7.0*X*X+X+1.0/9.0);;
            double b[]={-(X+1.0/2.0),-(X+1.0/3.0),-(X+1.0/4.0),-(X+1.0/5.0)};
            double c[]=pivotlugauss(a,b);
            return c;
  }

  public static void main(String args[]) throws IOException
  {
  double toi[]={0.25,0.5,0.75,1.0};
  double ro=8500; //kg/m^3
  double Cp=400; //J/kgK
  double h=80; //W/m2K
  double D=0.02;
  double k=20; //W/mK
  double V=Math.PI*D*D*D/6;
  double A=Math.PI*D*D;
  double Bi=h*D/6/k;
  System.out.println("Bi="+Bi);
  double tomax=240;
  double X=ro*V*Cp/(tomax*h*A);
  double c[]=c(X);
  String c1[]={"c1","c2","c3","c4"};
  Text.print(c,c1,"Least square method for approximation");
  if_x f=to->Math.exp(-to/X);
  if_x f1=to->1+c[0]*to+c[1]*to*to+c[2]*to*to*to+c[3]*to*to*to*to;
  Plot pp=new Plot(f,0,1);
  pp.addFunction(f1,0,1);
  pp.plot();
  }
}
```

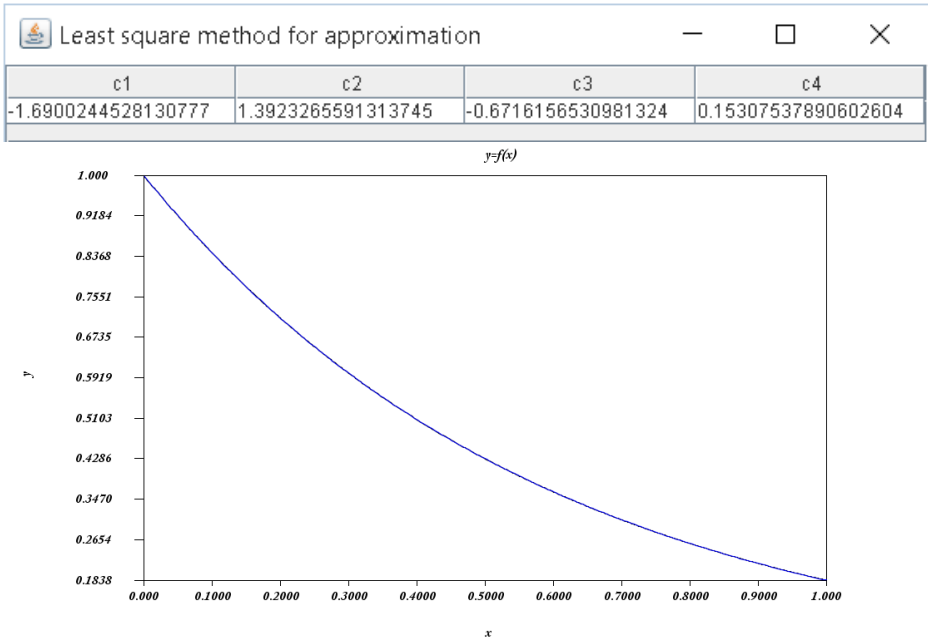
```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" residue_least_square
Bi=0.013333333333333332
Original matrix :
      1.272038966049383      1.188705632716049      1.138705632716049      1.105372299382716
      1.188705632716049      1.254848251028807      1.279586226851852      1.290619488536155
      1.138705632716049      1.279586226851852      1.360305059523810      1.412133487654321
      1.105372299382716      1.290619488536155      1.412133487654321      1.497795414462081

pivoted matrix : k=0
      1.272038966049383      1.188705632716049      1.138705632716049      1.105372299382716
      1.188705632716049      1.254848251028807      1.279586226851852      1.290619488536155
      1.138705632716049      1.279586226851852      1.360305059523810      1.412133487654321
      1.105372299382716      1.290619488536155      1.412133487654321      1.497795414462081

pivoted matrix : k=1
      1.272038966049383      1.188705632716049      1.138705632716049      1.105372299382716
      0.000000000000000      0.257661919158115      0.422624756279086      0.537252575088716
      0.000000000000000      0.215479044806210      0.340956947880412      0.422624756279086
      0.000000000000000      0.144016649981762      0.215479044806210      0.257661919158115

pivoted matrix : k=2
      1.272038966049383      1.188705632716049      1.138705632716049      1.105372299382716
      0.000000000000000      0.257661919158115      0.422624756279086      0.537252575088716
      0.000000000000000      0.000000000000000     -0.020741355156435     -0.042628152076166
      0.000000000000000      0.000000000000000     -0.012478201139771     -0.026672027964582

final upper triangular matrix :
      1.272038966049383      1.188705632716049      1.138705632716049      1.105372299382716
      0.000000000000000      0.257661919158115      0.422624756279086      0.537252575088716
      0.000000000000000      0.000000000000000     -0.020741355156435     -0.042628152076166
      0.000000000000000      0.000000000000000      0.000000000000000     -0.001026516771543
```



The last method we will look for the approximation is **Galarkin Method**. This Method multiplies residue with a weight function and integrates similar to least square method

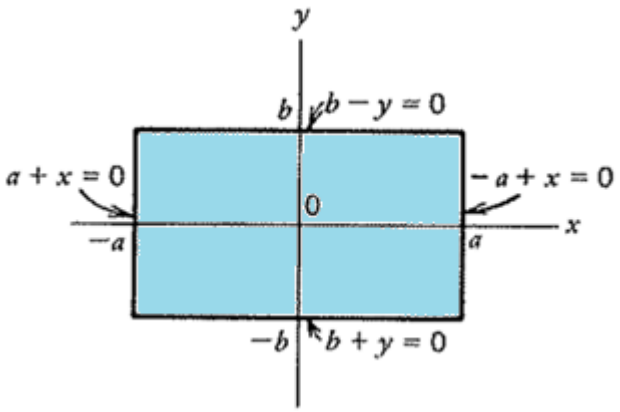
$$\int_{\tau=0}^{\tau=1} w_i R(\tau, c_1, c_2, c_3, c_4) d\tau = 0 \quad i = 1..4$$

$w_i$is called weight function. One possible way is defining the weight function in here is to utilised derivatives of residue with respect to c_i

$$w_i = \frac{\partial R(\tau, c_1, c_2, c_3, c_4)}{\partial c_i} \quad i = 1..4$$

in this case the results will be same as the previous least square case. Other considerations of weight functions are also possible. In fact selection of the weight factor is one of the most important criteria in solution. In general, weight functions should be selected so that it should satisfy the boundary conditions. As an example let us consider the following problem with a rectangular region:

PROBLEM 8.6 TWO DIMENSIONAL STEADY-STATE PROBLEM APPROXIMATION WITH GALARKIN METHOD



For differential equation: $\frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial y} + \frac{1}{k} g(x,y) = 0$

Boundary conditions:

$T(x,y) = 0 \;\; at \; x = -a \qquad T(x,y) = 0 \;\; at \; y = -b$

$T(x,y) = 0 \;\; at \; x = \;\; a \qquad T(x,y) = 0 \;\; at \; y = \;\; b$

Where

$$\bar{T}(x,y) = \phi_0(x,y) + \sum_{j=1}^{n} C_j\,\phi_j(x,y)$$

when this function is substituted into the differential equation

$$\frac{\partial^2 \bar{T}(x,y)}{\partial x^2} + \frac{\partial^2 \bar{T}(x,y)}{\partial y^2} + \frac{1}{k} g(x,y) = R(C_1, C_2, C_3, C_4)$$

Then the Galarkin method to evalute the problem will be

$$\int_{x=-a}^{a}\int_{y=-b}^{b}\left[\frac{\partial^2 \bar{T}(x,y)}{\partial x^2} + \frac{\partial^2 \bar{T}(x,y)}{\partial y^2} + \frac{1}{k} g(x,y)\right] dxdy = \int_{x=-a}^{a}\int_{y=-b}^{b} [R(C_1, C_2, C_3, C_4)]\, dxdy$$

Before going into the details of approximate solution, let us solve the differential equation with exact solution so that the results can be compared. This equation can be solved by using separation of variable methods. In order to homogenize the equation, system is divided up into two equation as:

$$\bar{T}(x,y) = \theta(x,y) + \psi(x)$$

And

$$\frac{\partial^2 \psi(x)}{\partial x^2} + \frac{g}{k} = 0$$

Boundary conditions:

$$\psi(x) = 0 \;\; at \;\; x = -a$$
$$\psi(x) = 0 \;\; at \;\; x = \;\; a$$

$$\frac{\partial^2 \theta(x,y)}{\partial x^2} + \frac{\partial^2 \theta(x,y)}{\partial y^2} = 0$$

Boundary conditions:

$$\theta(x,y) = 0 \;\; at \;\; x = -a$$
$$\theta(x,y) = 0 \;\; at \;\; x = \;\; a$$
$$\theta(x,y) = -\psi(x) \;\; at \;\; y = -b$$
$$\theta(x,y) = -\psi(x) \;\; at \;\; x = \;\; b$$

For g(x,y)=g=const Solution of $\psi(x)$

$$\psi(x) = -\frac{g}{k}\frac{x^2}{2} + C_1 x + C_2$$

applying boundary conditions $C_2 = \frac{g}{k}\frac{a^2}{2} \quad C_1 = 0$

$$\psi(x) = \frac{g}{2k}(a^2 - x^2)$$

Solution of $\theta(x,y)$

$$\theta(x,y) = X(x)Y(y)$$
$$\frac{1}{X(x)}\frac{\partial^2 X(x)}{\partial x^2} = -\frac{1}{Y(y)}\frac{\partial^2 Y(y)}{\partial y^2} = -\beta^2$$
$$\frac{\partial^2 X(x)}{\partial x^2} + \beta^2 X(x) = 0$$
$$\frac{\partial^2 Y(y)}{\partial y^2} - \beta^2 Y(y) = 0$$
$$X(x) = C_1 \cos(\beta x) + C_2 \sin(\beta x)$$
$$Y(x) = C_3 \cos h(\beta y) + C_4 \sinh(\beta y)$$

Boundary conditions:

$$X(x) = 0 \;\; at\, x = -a$$
$$X(x) = 0 \;\; at\, x = \;\; a$$
$$X(x) = C_1 \cos(-\beta a) + C_2 \sin(-\beta a) = 0$$
$$X(x) = C_1 \cos(\beta a) + C_2 \sin(\beta a) = 0$$

Considering that $\cos(-\beta a) = \cos(\beta a)$ and $\sin(-\beta a) = -\sin(\beta a)$

$$X(x) = C_1 \cos(\beta a) - C_2 \sin(\beta a) = 0$$

$$X(x) = C_1 \cos(\beta a) + C_2 \sin(\beta a) = 0$$

$$C_1 \cos(\beta a) = 0$$
$$\beta a = (2n+1)\frac{\pi}{2}$$
$$\beta_n = (2n+1)\frac{\pi}{2a}$$
$$C_2 = 0$$

$$\theta(x,y) = \bar{T}(x,y) - \psi(x)$$
$$\theta(x,y) = \sum_{n=0}^{\infty} [C_3 \cos h(\beta_n y) + C_4 \sinh(\beta_n y)] \cos(\beta_n x)$$

Boundary conditions:

$$-\psi(x) = \sum_{n=0}^{\infty} [C_3 \cos h(-\beta_n b) + C_4 \sinh(-\beta_n b)] \cos(\beta_n x)$$
$$-\psi(x) = \sum_{n=0}^{\infty} [C_3 \cos h(\beta_n b) + C_4 \sinh(\beta_n b)] \cos(\beta_n x)$$

Considering that $\cosh(-\beta a) = \cos h(\beta a)$ and $\sinh(-\beta a) = -\sin h(\beta a)$

$$-\psi(x) = \sum_{n=0}^{\infty} [C_3 \cos h(\beta_n b) - C_4 \sinh(\beta_n b)] \cos(\beta_n x)$$
$$-\psi(x) = \sum_{n=0}^{\infty} [C_3 \cos h(\beta_n b) + C_4 \sinh(\beta_n b)] \cos(\beta_n x)$$

So $C_4 = 0$

$$-\psi(x) = \sum_{n=0}^{\infty} C_3 \cos h(\beta_n b) \cos(\beta_n x)$$
$$-\frac{\psi(x)}{\cos h(\beta_n b)} = \sum_{n=0}^{\infty} C_3 \cos(\beta_n x)$$

It is known that $\cos(\beta_n x)$is an orthogonal function so orthogonality can be applied.

$$-\int_{x=-a}^{a} \frac{\psi(x) cos(\beta_n x)}{\cos h(\beta_n b)} dx = C_n \int_{x=-a}^{a} \cos^2(\beta_n x) dx$$

To solve the integral variable change is aplied as

$\bar{x} = \frac{x}{a} \quad \psi(\bar{x}) = \frac{ga^2}{2k}(1-\bar{x}^2) \quad \bar{\beta}_n = (2n+1)\frac{\pi}{2}$

$$-\int_{\bar{x}=-1}^{1} \frac{\psi(\bar{x}) cos(\bar{\beta}_n \bar{x})}{\cos h\left(\bar{\beta}_n \frac{b}{a}\right)} d\bar{x} = C_n \int_{\bar{x}=-1}^{1} \cos^2(\bar{\beta}_n \bar{x}) d\bar{x}$$

$$\int_{\bar{x}=-1}^{1} \cos^2(\bar{\beta}_n \bar{x}) d\bar{x} = \int_{\bar{x}=-1}^{1} \frac{1+\cos(2\bar{\beta}_n \bar{x})}{2} d\bar{x} = \left[\frac{x}{2} + \frac{1}{2\bar{\beta}_n} \sin(2\bar{\beta}_n)\right]_{-1}^{1} = 1$$

$$\frac{ga^2}{2k \cos h\left(\bar{\beta}_n \frac{b}{a}\right)} \int_{\bar{x}=-1}^{1} (1-\bar{x}^2) cos(\bar{\beta}_n \bar{x}) d\bar{x}$$

$$\int_{\bar{x}=-1}^{1} \left[cos(\bar{\beta}_n \bar{x}) - \bar{x}^2 cos(\bar{\beta}_n \bar{x})\right] d\bar{x} = \left[\frac{sin(\bar{\beta}_n \bar{x})}{\bar{\beta}_n} - x^2 \frac{sin(\bar{\beta}_n \bar{x})}{\bar{\beta}_n} - 2x \frac{cos(\bar{\beta}_n \bar{x})}{\bar{\beta}_n{}^2} + 2 \frac{cos(\bar{\beta}_n \bar{x})}{\bar{\beta}_n{}^3}\right]_{-1}^{1}$$

$$\int_{\bar{x}=-1}^{1}\left[cos\left(\bar{\beta}_n\bar{x}\right)-\bar{x}^2cos\left(\bar{\beta}_n\bar{x}\right)\right]d\bar{x}=\frac{4(-1)^n}{\bar{\beta}_n{}^3}$$

$$C_n=\frac{ga^2}{2k\cos h\left(\bar{\beta}_n\frac{b}{a}\right)}\frac{4(-1)^n}{\bar{\beta}_n{}^3}$$

so the exact solution:

$$T(x,y)=\frac{g}{k}\left[\frac{a^2-x^2}{2}-2a^2\sum_{n=0}^{\infty}\frac{(-1)^n}{\bar{\beta}_n^3}\frac{\cosh\left(\bar{\beta}_n\frac{y}{a}\right)\cos\left(\bar{\beta}_n\frac{x}{a}\right)}{\cosh\left(\bar{\beta}_n\frac{b}{a}\right)}\right]$$

Now that exact solution formula is established, Galarkin approximation

$$\int_{x=-a}^{a}\int_{y=-b}^{b}\left[\frac{\partial^2\overline{T}(x,y)}{\partial x^2}+\frac{\partial^2\overline{T}(x,y)}{\partial y^2}+\frac{1}{k}g\right]\phi_1 dxdy=0$$

Take

$$\phi_1(x,y)=(a^2-x^2)(b^2-y^2)$$

$$\overline{T}_1(x,y)=C_1\phi_1(x,y)=C_1(a^2-x^2)(b^2-y^2)$$

$$\frac{\partial^2\overline{T}_1(x,y)}{\partial x^2}=-2C_1(b^2-y^2)$$

$$\frac{\partial^2\overline{T}_1(x,y)}{\partial y^2}=-2C_1(a^2-x^2)$$

$$\int_{x=-a}^{a}\int_{y=-b}^{b}\left[-2C_1\left((a^2-x^2)+(b^2-y^2)\right)+\frac{g}{k}\right](a^2-x^2)(b^2-y^2)dxdy=0$$

$$\int_{x=-a}^{a}\int_{y=-b}^{b}\left[-2C_1\left((a^2-x^2)^2(b^2-y^2)+(a^2-x^2)(b^2-y^2)^2\right)+\frac{g}{k}(a^2-x^2)(b^2-y^2)\right]dxdy=0$$

$$\int_{x=-a}^{a}\int_{y=-b}^{b}\left[-2C_1\left((a^4-2a^2x^2+x^4)(b^2-y^2)+(a^2-x^2)(b^4-2b^2y^2+y^4)\right)+\frac{g}{k}(a^2-x^2)(b^2-y^2)\right]dxdy=0$$

Variable conversion: $\bar{x}=\frac{x}{a}$ and $\bar{y}=\frac{y}{b}$

$$\int_{\bar{x}=-1}^{1}\int_{\bar{y}=-1}^{1}\left[-2C_1\left(a^2(1-2\bar{x}^2+\bar{x}^4)(1-\bar{y}^2)+(1-2\bar{y}^2+\bar{y}^4)(1-\bar{x}^2)\right)+\frac{g}{k}(1-\bar{x}^2)(1-\bar{y}^2)\right]d\bar{x}d\bar{y}=0$$

$$\int_{\bar{x}=-1}^{1}\int_{\bar{y}=-1}^{1}\left[-2C_1\left(a^2(1-2\bar{x}^2+\bar{x}^4)(1-\bar{y}^2)+(1-2\bar{y}^2+\bar{y}^4)(1-\bar{x}^2)\right)+\frac{g}{k}(1-\bar{x}^2)(1-\bar{y}^2)\right]d\bar{x}d\bar{y}=0$$

$$-2C_1\left(a^2(\bar{x}-\frac{2}{3}\bar{x}^3+\frac{1}{5}\bar{x}^5)\Big|_{-1}^{1}(\bar{y}-\frac{1}{3}\bar{y}^3)\Big|_{-1}^{1}\right\}+b^2(\bar{y}-\frac{2}{3}\bar{y}^3+\frac{1}{5}\bar{y}^5)\Big|_{-1}^{1}(\bar{x}-\frac{1}{3}\bar{x}^3)\Big|_{-1}^{1}\Big)+\frac{g}{k}(\bar{x}-\frac{1}{3}\bar{x}^3)\Big|_{-1}^{1}(\bar{y}-\frac{1}{3}\bar{y}^3)\Big|_{-1}^{1}=0$$

$$-2C_1\left(a^2\left[\left(1-\frac{2}{3}+\frac{1}{5}\right)-\left(-1+\frac{2}{3}-\frac{1}{5}\right)\right]\left[\left(1-\frac{1}{3}\right)-\left(-1+\frac{1}{3}\right)\right]+b^2\left[\left(1-\frac{2}{3}+\frac{1}{5}\right)-\left(-1+\frac{2}{3}-\frac{1}{5}\right)\right]\left[\left(1-\frac{1}{3}\right)-\left(-1+\frac{1}{3}\right)\right]\right)+$$

$$\frac{g}{k}\left[\left(1-\frac{1}{3}\right)-\left(-1+\frac{1}{3}\right)\right]\left[\left(1-\frac{1}{3}\right)-\left(-1+\frac{1}{3}\right)\right]=0$$

$$-2C_1\left(a^2\frac{16}{15}\frac{4}{3}+b^2\frac{16}{15}\frac{4}{3}\right)+\frac{g}{k}\frac{4}{3}\frac{4}{3}=0$$

$$C_1=\frac{5}{8}\frac{g/k}{a^2+b^2}$$

$$\overline{T}_1(x,y)=\frac{5}{8}\frac{g/k}{a^2+b^2}(a^2-x^2)(b^2-y^2)$$

Now let us apply a two coefficient approximation

$$\phi_1 = (a^2-x^2)(b^2-y^2)$$

$$\phi_2 = x^2(a^2-x^2)(b^2-y^2)$$

$$T_2(x,y) = C_1\phi_1 + C_2\phi_2$$

$$T_2(x,y) = C_1(a^2-x^2)(b^2-y^2) + C_2(a^2x^2 - x^4)(b^2-y^2)$$

$$\frac{\partial T_2(x,y)}{\partial x} = -2C_1x(b^2-y^2) + C_2(2a^2x - 4x^3)(b^2-y^2)$$

$$\frac{\partial^2 T_2(x,y)}{\partial x^2} = -2C_1(b^2-y^2) + C_2(2a^2 - 12x^2)(b^2-y^2)$$

$$\frac{\partial T_2(x,y)}{\partial y} = -2C_1(a^2 - x^2)y - 2C_2(a^2x^2 - x^4)y$$

$$\frac{\partial^2 T_2(x,y)}{\partial y^2} = -2C_1(a^2 - x^2) - 2C_2{}^2(a^2x^2 - x^4)$$

$$\int_{x=-a}^{a}\int_{y=-b}^{b}\left[\frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial T(x,y)}{\partial y^2} + \frac{1}{k}g\right]\phi_1(x,y)dxdx = 0$$

$$\int_{x=-a}^{a}\int_{y=-b}^{b}\left[\frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial T(x,y)}{\partial y^2} + \frac{1}{k}g\right]\phi_2(x,y)dxdx = 0$$

$$\int_{\bar{x}=-a}^{a}\int_{\bar{y}=-a}^{a}\left[\left(-2C_1(a^2-x^2) + C_2(2a^4 - 14a^2x^2 + 12x^4)\right)(b^4 - 2b^2y^2 + y^4) + \left((a^4 - 2a^2x^2 + x^4)C_1 + (a^4x^2 - 2a^2x^4 + x^6)C_2\right)\left((-2)(b^2-y^2)\right) + \frac{g}{k}(a^2-x^2)(b^2-y^2)\right]dxdy = 0$$

$$\int_{\bar{x}=-a}^{a}\int_{\bar{y}=-a}^{a}\left[\left(-2C_1(a^2x^4-x^2) + C_2(2a^4x^2 - 14a^2x^4 + 12x6)\right)(b^4 - 2b^2y^2 + y^4) + \left((a^4x^2 - 2a^2x^4 + x^6)C_1 + (a^4x^4 - 2a^2x^6 + x^8)C_2\right)\left((-2)(b^2-y^2)\right) + \frac{g}{k}(a^2x^2 - x^4)(b^2-y^2)\right]dxdy =$$

$$\left(-2C_1(a^2x - \frac{x^3}{3}) + C_2(2a^4x - \frac{14}{3}a^2x^3 + \frac{12}{5}x^5)\right)\Bigg|_{-a}^{a}(b^4x - 2b^2\frac{y^3}{3} + \frac{y^5}{5})\Bigg|_{-b}^{b} +$$

$$\left((a^4 - \frac{2}{3}a^2x^3 + \frac{1}{5}x^5)C_1 + (a^4\frac{x^3}{3} - \frac{2}{5}a^2x^5 + \frac{x^7}{7})C_2\right)\Bigg|_{-a}^{a}\left((-2)(b^2y - \frac{y^3}{3})\right)\Bigg|_{-b}^{b} + \frac{g}{k}(a^2x - \frac{x^3}{3})\Bigg|_{-a}^{a}(b^2y - \frac{y^3}{3})\Bigg|_{-b}^{b} = 0$$

$$\left(-2C_1(a^2\frac{x^3}{3} - \frac{x^5}{5}) + C_2(2a^4\frac{x^3}{3} - \frac{14}{5}a^2x^5 + \frac{12}{7}x^7)\right)\Bigg|_{-a}^{a}(b^4x - 2b^2\frac{y^3}{3} + \frac{y^5}{5})\Bigg|_{-b}^{b} +$$

$$\left((a^4\frac{x^3}{3} - \frac{2}{5}a^2x^5 + \frac{1}{7}x^7)C_1 + (a^4\frac{x^5}{5} - \frac{2}{7}a^2x^7 + \frac{x^9}{9})C_2\right)\Bigg|_{-a}^{a}\left((-2)(b^2y - \frac{y^3}{3})\right)\Bigg|_{-b}^{b} + \frac{g}{k}(a^2\frac{x^3}{3} - \frac{x^5}{5})\Bigg|_{-a}^{a}(b^2y - \frac{y^3}{3})\Bigg|_{-b}^{b} = 0$$

$$\left(-\frac{8}{3}a^3C_1 - \frac{4}{15}a^5C_2\right)\frac{16}{15}b^5 + \left(\frac{16}{15}a^5C_1 + \frac{16}{105}a^7C_2\right)\frac{(-8)}{3}b^3 + \frac{16}{9}\frac{g}{k}a^3b^3 = 0$$

$$\left(-\frac{8}{15}a^5C_1 - \frac{88}{105}a^7C_2\right)\frac{16}{15}b^5 + \left(\frac{16}{105}a^7C_1 + \frac{158}{315}a^9C_2\right)\frac{(-8)}{3}b^3 + \frac{16}{45}\frac{g}{k}a^5b^3 = 0$$

$$\begin{bmatrix} \frac{8b^2(a^2+b^2)}{5} & \left(\frac{4}{25}a^2b^2 + \frac{8}{35}a^4\right) \\ 8(\frac{a^2}{7} + \frac{b^2}{5}) & \left(\frac{88}{35}a^2b^2 + \frac{79}{21}a^4\right) \end{bmatrix}\begin{Bmatrix} C_1 \\ C_2 \end{Bmatrix} = \begin{Bmatrix} \frac{g}{k} \\ \frac{g}{k} \end{Bmatrix}$$

```
 class fa extends f_xj
{
double a,b,k,g;
public fa(double ai,double bi,double ki,double gi)
{a=ai;b=bi;k=ki;g=gi;}

public double func(double x[])
{
//çözümü istenen fonksiyon
double ff=0;
double sign=1.0;
```

```
double beta=1.0;
double x1,y1,y2,ffx;
for(int n=0;n<200;n++)
{   beta=(2.0*n+1.0)/2.0*PI;
    y1=Math.cosh(beta*x[1]/a);
    x1=Math.cos(beta*x[0]/a);
    y2=Math.cosh(beta*b/a);
                ff+=-2.0*a*a*sign/(beta*beta*beta)*y1*x1/y2;
                sign*=-1.0;
}
ff+=(a*a-x[0]*x[0])/2.0;
ff*=g/k;
double ff1=5.0/8.0*g/k/(a*a+b*b)*(a*a-x[0]*x[0])*(b*b-x[1]*x[1]);

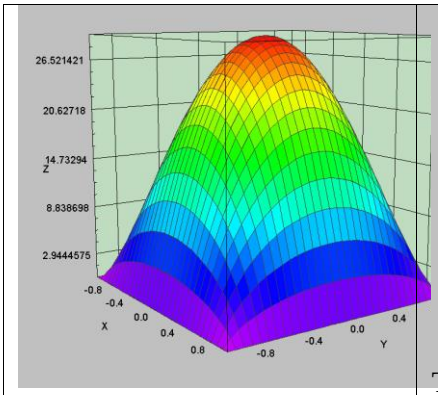
double aa[][]={{8.0/5.0*(b*b+a*a),(4.0/25.0*a*a*b*+8/35*a*a*a*a)},
            {(8.0/7.0*a*a+8.0/5.0*b*b),(88.0/35.0*a*a*b*b+79.0/21.0*a*a*a*a)}};
double bb[]={g/k,g/k};
double C[]=pivotlugauss(aa,bb);
double ff2=C[0]*(a*a-x[0]*x[0])*(b*b-x[1]*x[1])+C[1]*x[0]*x[0]*(a*a-x[0]*x[0])*(b*b-x[1]*x[1]);
return (ff-ff2); //maksimum testi
}

public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss eleme
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivoting
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                   if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
                  b[k]=dummy;
                }
                //Solving gauss elimination
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
                }
  }
  //back substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }

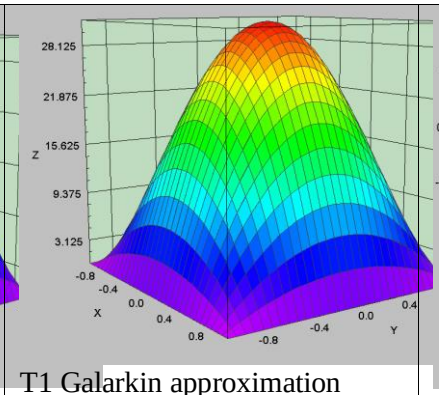
}

public class surfacePlotE4
{       public static void main(String arg[])
        {  double a=1.0;
           double b=1.0;
```

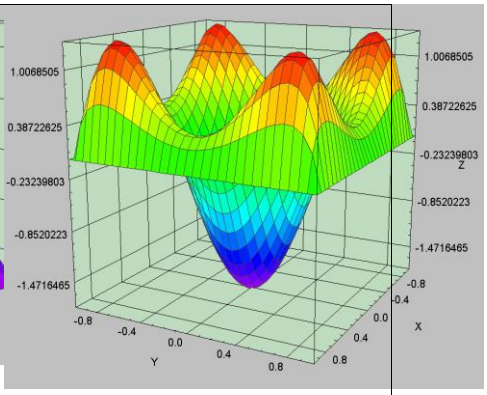
```
        double k=100.0;
        double g=10000.0;
                fa f=new fa(a,b,k,g);
                double x[]={0,0};
                //System.out.println("T(0,0)="+f.func(x));
        surfacePlot p=new surfacePlot(f,-a,a,-b,b);
        p.plot();
      }
}
```



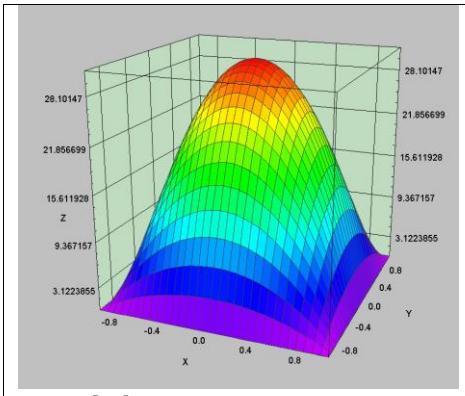
Exact solution T0



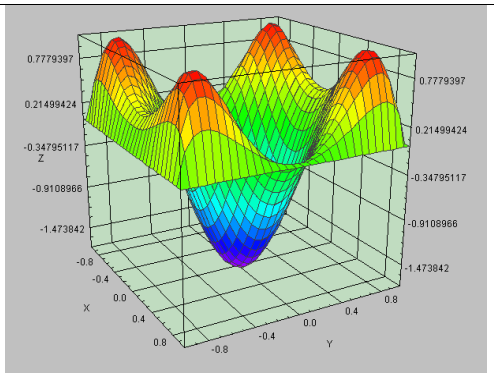
T1 Galarkin approximation



T0-T1 difference between solutions



T2 Galarkin approximation



T0-T2 difference between solutions

9. PHASE-CHANGE PROBLEM IN HEAT CONDUCTION

PROBLEM 9.1 PHASE CHANGE HEAT CONDUCTION PROBLEM

A thick liquid layer of initially at an initial temperature of Ti can be cooled from one side (x=0) to a temperature of T_0 . Assuming this temperature is below the frezing temperature, T_m, liquid will start to change phase from liquid to solid as a function of time. The solid surface will be tickened as a function of time. This process is expresses as

$$\frac{\partial^2 T_s(x,t)}{\partial x^2} = \frac{1}{\alpha_s}\frac{\partial T_s(x,t)}{\partial t} \quad \text{in } 0 \le x \le s(t) \quad t>0$$

$T_s(x,t) = T_0$ at x=0 t>0

$T_s(x,t) = T_m$ at x=s(t) t>0

$T_s(x,t) = T_i$ at $0 \le x \le s(t)$ t=0

$$\frac{\partial^2 T_l(x,t)}{\partial x^2} = \frac{1}{\alpha_l}\frac{\partial T_l(x,t)}{\partial t} \quad \text{in } s(t) \le x \le \infty \quad t>0$$

$T_l(x,t) = T_i$ at $x = \infty$ t>0

$T_l(x,t) = T_m$ at x=s(t) t>0

$T_l(x,t) = T_i$ at $0 \le x \le s(t)$ t=0

Furthermore at the interphase

$T_s(x,t) = T_l(x,t) = T_m$ at x=s(t) t>0

And from the energy balance at the interface

$$k_s \frac{\partial T_s(x,t)}{\partial x^2} - k_l \frac{\partial T_l(x,t)}{\partial x^2} = (\rho_l H_l - \rho_s H_s) v_x - \rho_l H_l v_1$$

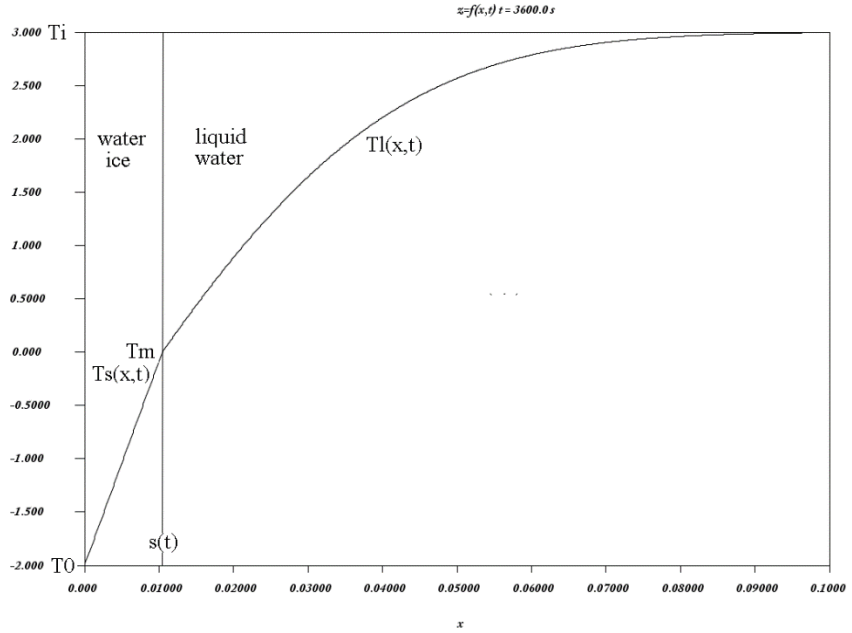
$$(\rho_l - \rho_s) v_x = \rho_l v_1$$

$$\frac{(\rho_l - \rho_s)}{\rho_l} v_x = v_1$$

So

$$k_s \frac{\partial T_s(x,t)}{\partial x^2} - k_l \frac{\partial T_l(x,t)}{\partial x^2} = \rho_s (H_l - H_s) v_x = \rho_s (H_l - H_s) \frac{ds(t)}{dt}$$

The temperatures $T_s(x,t)$ and $T_l(x,t)$ indicates temperatures of solid and liquid phases. Where v_x is the velocity of the interphase, v_l velocity of the liquid at the interphase, H_l and H_s antalpy of solid and liquid at melting temperature of T_m.The difference of liquid and solid enthalpies are called latent heat.



If a solution in the form of

$$T_s(x,t) = T_0 + C_1 erf\left(\frac{x}{2\sqrt{\alpha_s t}}\right)$$

$$T_l(x,t) = T_i + C_2 erfc\left(\frac{x}{2\sqrt{\alpha_l t}}\right)$$

Then interface BC can be applied

$T_s(x,t) = T_l(x,t) = T_m$ at x=s(t) t>0

$$T_0 + C_1 erf\left(\frac{s(t)}{2\sqrt{\alpha_s t}}\right) = T_i + C_2 erfc\left(\frac{s(t)}{2\sqrt{\alpha_l t}}\right) = T_m \text{ define } \lambda = \frac{s(t)}{2\sqrt{\alpha_s t}} \quad s(t) = 2\lambda\sqrt{\alpha_s t}$$

$$T_0 + C_1 erf(\lambda) = T_i + C_2 erfc\left(\lambda\left(\frac{\alpha_s}{\alpha_l}\right)^{1/2}\right) = T_m \text{ so}$$

$$C_1 = \frac{(T_m - T_0)}{erf(\lambda)}$$

$$C_2 = \frac{(T_m - T_i)}{erfc\left(\lambda\left(\frac{\alpha_s}{\alpha_l}\right)^{1/2}\right)}$$ and

$$k_s \frac{\partial T_s(x,t)}{\partial x} - k_l \frac{\partial T_l(x,t)}{\partial x} = \rho_s (H_l - H_s) v_x = \rho_s (H_l - H_s) \frac{ds(t)}{dt}$$

$$\frac{d(erf(u))}{dx} = \frac{du}{dx}\frac{2}{\sqrt{\pi}} e^{-u^2} \qquad u = \frac{x}{2(\alpha_s t)^{1/2}} \qquad \frac{du}{dx} = \frac{1}{2(\alpha_s t)^{1/2}}$$

$$T_s(x,t) = T_0 + \frac{(T_m - T_0)}{erf(\lambda)} erf\left(\frac{x}{2(\alpha_s t)^{1/2}}\right)$$

$$T_l(x,t) = T_i + \frac{(T_m - T_i)}{erfc\left[\lambda\left(\frac{\alpha_s}{\alpha_l}\right)^{1/2}\right]} erf\left(\frac{x}{2(\alpha_s t)^{1/2}}\right)$$

$$\frac{dT_s(x,t)}{dx} = \frac{(T_m - T_0)}{2(\alpha_s t)^{1/2} erf(\lambda)} \frac{2}{\sqrt{\pi}} \exp\left(-\left(\frac{x}{2(\alpha_s t)^{1/2}}\right)^2\right) = \frac{(T_m - T_0)}{(\alpha_s t)^{1/2} erf(\lambda)} \frac{1}{\sqrt{\pi}} \exp\left(-\lambda^2\right)$$

$$erfc(u) = 1 - erf(u) \qquad \frac{d(erfc(u))}{dx} = -\frac{du}{dx}\frac{2}{\sqrt{\pi}} e^{-u^2}$$

$$\frac{dT_l(x,t)}{dx} = -\frac{(T_m - T_i)}{2(\alpha_l t)^{1/2} erfc\left[\lambda\left(\frac{\alpha_s}{\alpha_l}\right)^{1/2}\right]} \frac{2}{\sqrt{\pi}} \exp\left(-\left(\frac{x}{2(\alpha_l t)^{1/2}}\right)^2\right) = -\frac{(T_m - T_i)}{(\alpha_l t)^{1/2} erfc\left[\lambda\left(\frac{\alpha_s}{\alpha_l}\right)^{1/2}\right]} \frac{1}{\sqrt{\pi}} \exp\left(-\lambda^2\left(\frac{\alpha_s}{\alpha_l}\right)\right)$$

$$s(t) = 2\lambda\left(\alpha_s t\right)^{1/2} \qquad \frac{ds(t)}{dt} = \frac{\lambda \alpha_s^{1/2}}{t^{1/2}}$$

$$\frac{k_s (T_m - T_0)}{(\alpha_s t)^{1/2} erf(\lambda)} \frac{1}{\sqrt{\pi}} \exp\left(-\lambda^2\right) + \frac{(T_m - T_i)}{(\alpha_l t)^{1/2} erfc\left[\lambda\left(\frac{\alpha_s}{\alpha_l}\right)^{1/2}\right]} \frac{1}{\sqrt{\pi}} \exp\left(-\lambda^2\left(\frac{\alpha_s}{\alpha_l}\right)\right) = \rho_s (H_l - H_s) \frac{\lambda \alpha_s^{1/2}}{t^{1/2}}$$

$$\alpha_s = \frac{k_s}{\rho_s C_{ps}}$$

$$\frac{k_s (T_m - T_0)}{(\alpha_s t)^{1/2} erf(\lambda)} \frac{1}{\sqrt{\pi}} \exp\left(-\lambda^2\right) + \frac{k_l (T_m - T_i)}{(\alpha_l t)^{1/2} erfc\left[\lambda\left(\frac{\alpha_s}{\alpha_l}\right)^{1/2}\right]} \frac{1}{\sqrt{\pi}} \exp\left(-\lambda^2\left(\frac{\alpha_s}{\alpha_l}\right)\right) = \rho_s (H_l - H_s) \frac{\lambda \alpha_s^{1/2}}{t^{1/2}}$$

Rearranging the equation gives:

$$\frac{\exp\left(-\lambda^2\right)}{erf(\lambda)} + \left(\frac{k_l}{k_s}\right)\left(\frac{\alpha_s}{\alpha_l}\right)^{1/2} \frac{(T_m - T_i)}{(T_m - T_0) erfc\left[\lambda\left(\frac{\alpha_s}{\alpha_l}\right)^{1/2}\right]} \frac{1}{\sqrt{\pi}} \exp\left(-\lambda^2\left(\frac{\alpha_s}{\alpha_l}\right)\right) = \frac{\lambda\sqrt{\pi}(H_l - H_s)}{C_{ps}(T_m - T_0)}$$

Solution of this eqaution will yield the λ value. By using this eqaution a program is developed to do the calculation. For specific properties water-ice is taken.

```
 import javax.swing.*;
//class to calculate lambda root value
class fx extends f_x
{
double ks,kl,alphas,alphal,Tm,T0,Ti,L,Cps,hfg;
steamIAPWS_IF97 st;
double a[];
public fx(double Tmi,double T0i,double Tii)
```

```
{Tm=Tmi;T0=T0i;Ti=Tii;
 hfg=333.55e3;    // J/kgK
 st=new steamIAPWS_IF97();
 //Tm degree C
 Cps=2109.7329400765643+7.637140284403064*Tm+0.00269180790270179*Tm*Tm;   // J/kgK
 ks=2.217352440215741-0.006916860285175764 4*Tm+1.016721166672117 5E-4*Tm*Tm+4.456742855800601E-7*Tm*Tm*Tm;
 double ros=916.8094018597769-0.15255175128486845*Tm-0.0015446216557046906*Tm*Tm-9.285069004045726E-6*Tm*Tm*Tm;
 alphas=ks/(ros*Cps);
 double P=101.325; //kPA;
 kl=st.k_PT(P,(Tm+273.15));
 double a[]=st.property("tx",(Tm+273.15),0.0);
 double rol=1.0/a[2];
 double h1=a[3];
 a=st.property("tx",(Tm+273.25),0.0);
 double h2=a[3];
 double Cpl=(h2-h1)/0.1*1e3;
 alphal=kl/(rol*Cpl);
 }

public double func(double l)
{double al=alphas/alphal;
 double c1=l*hfg/(Cps*(Tm-T0));
 double c3=l*sqrt(al);
 double c4=erfc(c3);
 double c2=exp(-l*l*al)/c3;
 double ff=exp(-l*l)/erf(l)+kl/ks*sqrt(al)*(Tm-Ti)/(Tm-T0)*c2-c1;
 return ff;
}}

//class to plot & evaluate temperature profile

class ice_PC extends fi_xt
{
double alphas,alphal,lambda,Tm,T0,Ti;
fx f;
public ice_PC(double Tmi,double T0i,double Tii)
{Tm=Tmi;T0=T0i;Ti=Tii;
 f=new fx(Tm,T0,Ti);
 double a[][]=rootlimits(f,0.0001,4.0,100,1);
 lambda=bisection(f,a[0][0],a[1][0]);
 System.out.println("lambda="+lambda);
}
public static double[][] rootlimits(f_x f,double x1,double x2,int n,int nbb)
{
        // find root limits
        // n : number of subdivisions
        // x1,x2 : limit values of the root
        // nbb = number of roots in the serched region
        //
        int nb;
        int i;
        double x,fp,fc,dx;
        double xb[][]=new double[2][nbb];
        nb=0;
        dx=(x2-x1)/n;
        x=x1;
        fp=f.func(x1);
        for (i=1;i<=n;i++)
        {
                x+=dx;
                fc=f.func(x);
                // eğer kök olan bölge bulunduysa.....
                if (fc*fp < 0.0 || fp==0) {
                        xb[0][nb]=x-dx;
                        xb[1][nb]=x;
                        nb++;
                }
                fp=fc;
                if (nbb == nb) return xb;
        }
if( nb == 0)
        JOptionPane.showMessageDialog(null,"search is completed no root is found"
   ,"SEARCH RESULT WARNING",JOptionPane.WARNING_MESSAGE);
else if(nb<nbb)
  {     JOptionPane.showMessageDialog(null,"search is completed "+nb+" root is found \n"+
    "You searched for  "+nbb+" root","SEARCH RESULT WARNING",JOptionPane.WARNING_MESSAGE);
   double xc[][]=new double[2][nb];
```

```
   for (i=0;i<nb;i++) {xc[0][i]=xb[0][i];xc[1][i]=xb[1][i];}
   return xc;
  }
return xb;
}


public static double[] enlarge(f_x f,double x1)
{return enlarge(f,x1,(x1+0.00001));}

public static double[] enlarge(f_x f,double x1,double x2)
{
        //kök olan bir bölge bulana kadar verilen siniri genişletir
        int NTRY=50;
        double a[]=new double[2];
        double FACTOR=1.6;
        int j;
        double f1,f2;
        if (x1 == x2) System.out.println("Bad initial range in ZBRAC");
        f1=f.func(x1);
        f2=f.func(x2);
        for (j=1;j<=NTRY;j++) {
                if (f1*f2 < 0.0) {a[0]=x1;a[1]=x2;return a;}
                if (Math.abs(f1) < Math.abs(f2))
                        f1=f.func(x1 += FACTOR*(x1-x2));
                else
                        f2=f.func(x2 += FACTOR*(x2-x1));
        }
        {a[0]=x1;a[1]=x2;return a;}
}



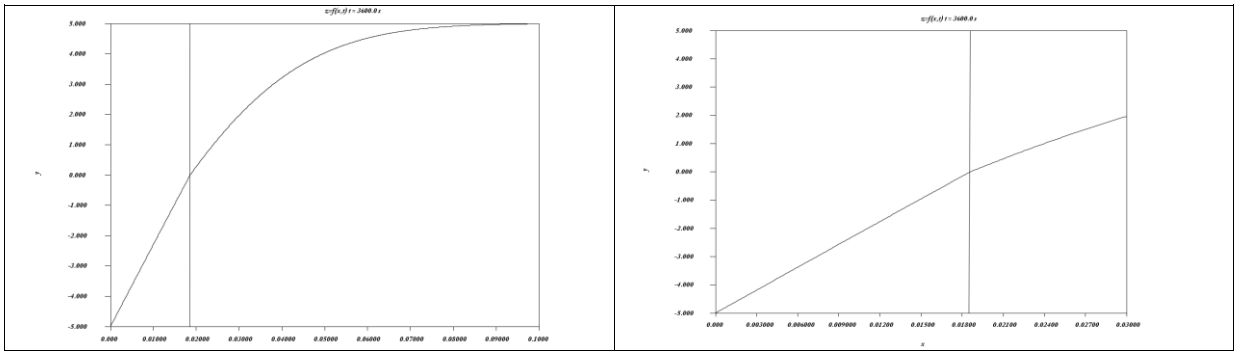
public static double bisection(f_x f,double xl,double xu)
{
//bisection root finding method
double test;
double xr=0;
double es,ea;
double fxl,fxr,fxu;
double xold=0;
int maxit=100,iter=0;
es=0.0000001;
ea=1.1*es;
String s="";
fxl= f.func(xl);
fxu= f.func(xu);
if(fxl*fxu>0) System.out.println("xl="+xu+"xr="+xr+"fxl="+fxl+"fxu="+fxu+"error");
while((ea>es)&&(iter<maxit))
{   xold=xr;
    xr=(xl+xu)/2.0;
    fxr= f.func(xr);
    iter++;
    if(xr!=0)
       { ea=Math.abs((xr-xold)/xr)*100;}
    test= fxl*fxr;
    if(test==0.0)      ea=0;
    else if(test<0.0) {xu=xr;fxu=fxr;}
    else if(test>0)   {xl=xr;fxl=fxr;}
    else              {ea=0;}
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded \n"+
   " result might not be valid","MAKSİMUM NUMBER OF ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
//JOptionPane.showMessageDialog(null,s,
//     "bisection root finding method : ",JOptionPane.PLAIN_MESSAGE);
   return xr;
}
public double s(double t)
{return 2.0*lambda*sqrt(f.alphas*t);}

public double[] func(double x,double t)
{ double a[]=new double[2];
  if(x<s(t)) a[0]=(Tm-T0)*erf(x/2.0/sqrt(f.alphas*t))/erf(lambda)+T0;
  else a[0]=(Tm-Ti)*erfc(x/2.0/sqrt(f.alphal*t))/erfc(lambda*sqrt(f.alphas/f.alphal))+Ti;
  if(x<s(t)) a[1]=T0;
  else       a[1]=Ti;
  return a;
```

```
}
}

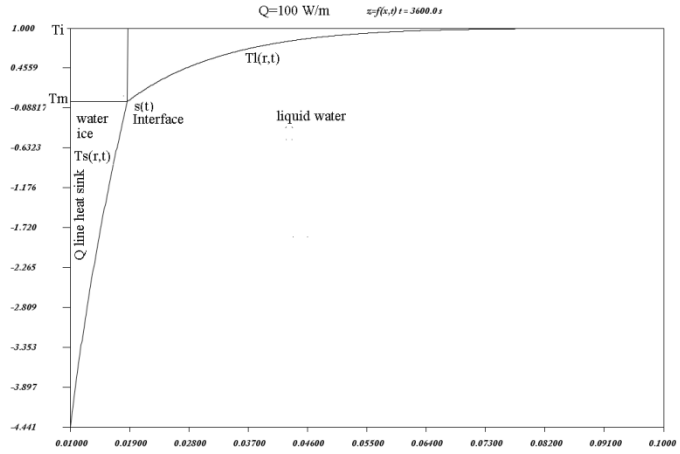
public class HT_phase_change1
{

public static void main(String ... arg)
{   ice_PC ff=new ice_PC(0.0,-2.0,3.0);
    timeplot_n p=new timeplot_n(ff,0.0,0.1,0.0001,0.0,3600.0,0.1,100.0);
    p.start();
}
}
```



PROBLEM 9.2 PHASE CHANGE HEAT TRANSFER PROBLEM FOR A CYLINDER

In the next problem solidification by a heat sink in an infinite cyclinder will be considered. A line heat sink of Q W/m is placed at the center of the cylinder and solidification will ocur in cyclidrical coordinates



$\frac{1}{r}\frac{\partial}{\partial r}\left(\frac{\partial T_s(r,t)}{\partial r}\right)=\frac{1}{\alpha_s}\frac{\partial T_s(r,t)}{\partial t}$ in in $0 \le r \le s(t)$ t>0 for solid phase

$T_s(x,t)=T_m$ at r=s(t) t>0

$\frac{1}{r}\frac{\partial}{\partial r}\left(\frac{\partial T_l(r,t)}{\partial r}\right)=\frac{1}{\alpha_l}\frac{\partial T_l(r,t)}{\partial t}$ in in $s(t) \le r \le \infty$ t>0 for liquid phase

$T_l(r,t)=T_i$ $at\, r=\infty$ t>0

$T_l(x,t)=T_m$ at r=s(t) t>0

$T_l(x,t)=T_i$ at $0 \le r \le s(t)$ t=0

For the liquid-solid interphase

$T_s(r,t)=T_l(r,t)=T_m$ at r=s(t) t>0

$$k_s\frac{\partial T_s(r,t)}{\partial r}-k_l\frac{\partial T_l(r,t)}{\partial r}=\rho_s(H_l-H_s)\frac{ds(t)}{dt}$$

Solution sets of the differential eqautions can be given as:

$$T_s(r,t) = A - BEi\left(\frac{-r^2}{4\alpha_s t}\right)$$

$$T_l(r,t) = T_i \ - CEi\left(\frac{-r^2}{4\alpha_l t}\right)$$

In this equations Ei(x) is exponential integral function
Basic definition of the exponential integral function is:

$$Ei(x) = \int_{-x}^{\infty} \frac{e^{-t}}{t} dt$$

Exponential integral function ca be calculated by using series formulas

$Ei(x) = \gamma + \ln|x| + \sum_{k=1}^{\infty} \frac{x^k}{k\,k!}$ x<2.5 $\gamma = 0.57721566490153286060651209008240243 1$is Euler Mascheroni constant

$Ei(x) = \gamma + \ln + \exp(x/2) \sum_{n=1}^{\infty} \frac{(-1)^{n-1} x^{n\,[(n-1)/2]}}{2^{n-1} n!} \sum_{k=0}^{} \frac{1}{2k+1}$ 2.4<x<10 (Ramanujan formula)

Asymtotic divergent series:

$$Ei(x) = \frac{\exp(-x)}{x} \sum_{n=0}^{N-1} \frac{n!}{(-x)^n}$$ x>10

```
public class Ei
{  //C Euler-Mascheroni constant
   public static double C=0.57721566490153286060651209008240243 1;
   //exponential integral section
   //Argument addition series Ei
 public static double AAS_Ei(double x)
 {
     double ei[] = {
     1.915047433355013959531e2,  4.403798995348382689974e2,
     1.037878290717089587658e3,  2.492228976241877759138e3,
     6.071406374098611507965e3,  1.495953266639752885229e4,
     3.719768849068903560439e4,  9.319251363396537129882e4,
     2.349558524907683035782e5,  5.955609986708370018502e5,
     1.516637894042516884433e6,  3.877904330597443502996e6,
     9.950907251046844760026e6,  2.561565266405658882048e7,
     6.612718635548492136250e7,  1.711446713003636684975e8,
     4.439663698302712208698e8,  1.154115391849182948287e9,
     3.005950906525548689841e9,  7.842940991898186370453e9,
     2.049649711988081236484e10, 5.364511859231469415605e10,
     1.405991957584069047340e11, 3.689732094072741970640e11,
     9.694555759683939661662e11, 2.550043566357786926147e12,
     6.714640184076497558707e12, 1.769803724411626854310e13,
     4.669055014466159544500e13, 1.232852079912097685431e14,
     3.257988998672263996790e14, 8.616388199965786544948e14,
     2.280446200301902595341e15, 6.039718263611241578359e15,
     1.600664914324504111070e16, 4.244796092136850759368e16,
     1.126348290166966760275e17, 2.990444718632336675058e17,
     7.943916035704453771510e17, 2.111342388647824195000e18,
     5.614329680810343111535e18, 1.493630213112993142255e19,
     3.975442747903744836007e19, 1.058563689713169096306e20
   };
   int  k = (int) (x + 0.5);
   int  j = 0;
   double xx = (double) k;
   double dx = x - xx;
   double xxj = xx;
   double edx = Math.exp(dx);
   double Sm = 1.0;
   double Sn = (edx - 1.0) / xxj;
   double max_double=1.7976931348623157e+308;
   double eps=1e-20;
   double term = max_double;
   double fact = 1.0;
   double dxj = 1.0;

   while (Math.abs(term) > eps * Math.abs(Sn) ) {
```

```
      j++;
      fact *= (double) j;
      xxj *= xx;
      dxj *= (-dx);
      Sm += (dxj / fact);
      term = ( fact * (edx * Sm - 1.0) ) / xxj;
      Sn += term;
    }

    return ei[k-7] + Sn * Math.exp(xx);
}
//Power series Ei
public static double PS_Ei( double x )
{  double max_double=1.7976931348623157e+308;
    double eps=1e-20;
    double xn = -x;
    double Sn = -x;
    double Sm1 = 0.0;
    double hsum = 1.0;
    double g = C;//Euler number ;
    double y = 1.0;
    double fact = 1.0;
    if ( x == 0.0 ) return  -max_double;

    while ( Math.abs(Sn - Sm1) > eps * Math.abs(Sm1) ) {
      Sm1 = Sn;
      y += 1.0;
      xn *= (-x);
      fact *= y;
      hsum += (1.0 / y);
      Sn += hsum * xn / fact;
    }
    return (g + Math.log(Math.abs(x)) - Math.exp(x) * Sn);
}
//Continued Faraction Ei
public static double CF_Ei( double x )
{  double max_double=1.7976931348623157e+308;
    double eps=1e-20;
    double Am1 = 1.0;
    double A0 = 0.0;
    double Bm1 = 0.0;
    double B0 = 1.0;
    double a = Math.exp(x);
    double b = -x + 1.0;
    double Ap1 = b * A0 + a * Am1;
    double Bp1 = b * B0 + a * Bm1;
    int j = 1;
    a = 1.0;
    while ( Math.abs(Ap1 * B0 - A0 * Bp1) > eps * Math.abs(A0 * Bp1) ) {
      if ( Math.abs(Bp1) > 1.0) {
        Am1 = A0 / Bp1;
        A0 = Ap1 / Bp1;
        Bm1 = B0 / Bp1;
        B0 = 1.0;
      } else {
        Am1 = A0;
        A0 = Ap1;
        Bm1 = B0;
        B0 = Bp1;
      }
      a = -j * j;
      b += 2.0;
      Ap1 = b * A0 + a * Am1;
      Bp1 = b * B0 + a * Bm1;
      j += 1;
    }
    return (-Ap1 / Bp1);
}
public static double Ei( double x )
{  double max_double=1.7976931348623157e+308;
    if ( x < -5.0 ) return CF_Ei(x);
    if ( x == 0.0 ) return -max_double;
    if ( x < 6.8 )  return PS_Ei(x);
    if ( x < 50.0 ) return AAS_Ei(x);
    return CF_Ei(x);
}
```

```
public static double E1( double x )
{return -Ei(-x);}
//end of exponential integral section
public static void main(String arg[])
{String s="";
 for(double x=0.01;x<=5.0;x+=0.01)
 {s+=x+"   "+E1(x)+"\n";}
System.out.println(s);
}
}
```

x	E1(x)=-Ei(-x)	x	E1(x)=-Ei(-x)	x	E1(x)=-Ei(-x)	x	E1(x)=-Ei(-x)	x	E1(x)=-Ei(-x)	x	E1(x)=-Ei(-x)
0.01	4.03792957654E+00	0.51	5.47822351781E-01	1.01	2.15741623794E-01	1.51	9.85443646984E-02	2.01	4.82288813035E-02	2.51	2.45888647027E-02
0.02	3.35470778331E+00	0.52	5.36219797846E-01	1.02	2.12171083432E-01	1.52	9.70934662966E-02	2.02	4.75672347334E-02	2.52	2.42673393603E-02
0.03	2.95911872402E+00	0.53	5.24951510115E-01	1.03	2.08670559301E-01	1.53	9.56664241155E-02	2.03	4.69154065626E-02	2.53	2.39502739535E-02
0.04	2.68126368903E+00	0.54	5.14003885702E-01	1.04	2.05238351720E-01	1.54	9.42627855447E-02	2.04	4.62732354318E-02	2.54	2.36376017134E-02
0.05	2.46789848851E+00	0.55	5.03364081392E-01	1.05	2.01872813220E-01	1.55	9.28821081642E-02	2.05	4.56405629912E-02	2.55	2.33292569713E-02
0.06	2.29530691814E+00	0.56	4.93019958776E-01	1.06	1.98572346533E-01	1.56	9.15239594682E-02	2.06	4.50172338354E-02	2.56	2.30251751381E-02
0.07	2.15083818026E+00	0.57	4.82960034245E-01	1.07	1.95335402670E-01	1.57	9.01879165982E-02	2.07	4.44030954403E-02	2.57	2.27252926848E-02
0.08	2.02694100259E+00	0.58	4.73173433331E-01	1.08	1.92160479095E-01	1.58	8.88735660844E-02	2.08	4.37979981010E-02	2.58	2.24295471226E-02
0.09	1.91874477003E+00	0.59	4.63649848957E-01	1.09	1.89046117979E-01	1.59	8.75805035957E-02	2.09	4.32017948721E-02	2.59	2.21378769840E-02
0.1	1.82292395842E+00	0.6	4.54379503189E-01	1.1	1.85990904536E-01	1.6	8.63083336975E-02	2.1	4.26143415085E-02	2.6	2.18502218041E-02
0.11	1.73710669407E+00	0.61	4.45353112163E-01	1.11	1.82993465435E-01	1.61	8.50566696173E-02	2.11	4.20354964088E-02	2.61	2.15665221024E-02
0.12	1.65954175209E+00	0.62	4.36561853847E-01	1.12	1.80052467282E-01	1.62	8.38251330173E-02	2.12	4.14651205590E-02	2.62	2.12867193647E-02
0.13	1.58889930486E+00	0.63	4.27997338402E-01	1.13	1.77166615170E-01	1.63	8.26133537748E-02	2.13	4.09030774781E-02	2.63	2.10107560259E-02
0.14	1.52414572217E+00	0.64	4.19651580863E-01	1.14	1.74334651294E-01	1.64	8.14209697688E-02	2.14	4.03492331654E-02	2.64	2.07385754523E-02
0.15	1.46446167052E+00	0.65	4.11516975949E-01	1.15	1.71555353630E-01	1.65	8.02476266733E-02	2.15	3.98034560480E-02	2.65	2.04701219253E-02
0.16	1.40918669869E+00	0.66	4.03586274791E-01	1.16	1.68827534661E-01	1.66	7.90929777578E-02	2.16	3.92656169310E-02	2.66	2.02053406246E-02
0.17	1.35778065254E+00	0.67	3.95852563412E-01	1.17	1.66150040170E-01	1.67	7.79566836923E-02	2.17	3.87355889475E-02	2.67	1.99441776121E-02
0.18	1.30979613542E+00	0.68	3.88309242805E-01	1.18	1.63521748079E-01	1.68	7.68384123599E-02	2.18	3.82132475109E-02	2.68	1.96865798162E-02
0.19	1.26485842441E+00	0.69	3.80950010461E-01	1.19	1.60941567336E-01	1.69	7.57378386740E-02	2.19	3.76984702678E-02	2.69	1.94324950160E-02
0.2	1.22265054418E+00	0.7	3.73768843234E-01	1.2	1.58408436851E-01	1.7	7.46546444013E-02	2.2	3.71911370519E-02	2.7	1.91818718266E-02
0.21	1.18290198625E+00	0.71	3.66759981411E-01	1.21	1.55921324480E-01	1.71	7.35885179903E-02	2.21	3.66911298396E-02	2.71	1.89346596837E-02
0.22	1.14538005500E+00	0.72	3.59917913910E-01	1.22	1.53479226038E-01	1.72	7.25391544047E-02	2.22	3.61983327059E-02	2.72	1.86908088295E-02
0.23	1.10988313890E+00	0.73	3.53237364490E-01	1.23	1.51081164373E-01	1.73	7.15062549618E-02	2.23	3.57126317819E-02	2.73	1.84502702980E-02
0.24	1.07623541487E+00	0.74	3.46713278904E-01	1.24	1.48726188456E-01	1.74	7.04895271757E-02	2.24	3.52339152134E-02	2.74	1.82129959012E-02
0.25	1.04428263444E+00	0.75	3.40340812911E-01	1.25	1.46413372526E-01	1.75	6.94886846046E-02	2.25	3.47620731194E-02	2.75	1.79789382155E-02
0.26	1.01388873678E+00	0.76	3.34115321091E-01	1.26	1.44141815256E-01	1.76	6.85034467038E-02	2.26	3.42969975534E-02	2.76	1.77480505677E-02
0.27	9.84933101328E-01	0.77	3.28032346380E-01	1.27	1.41910638962E-01	1.77	6.75335386814E-02	2.27	3.38385824636E-02	2.77	1.75202870227E-02
0.28	9.57308300357E-01	0.78	3.22087610293E-01	1.28	1.39718988834E-01	1.78	6.65786913593E-02	2.28	3.33867236556E-02	2.78	1.72956023698E-02
0.29	9.30918246013E-01	0.79	3.16277003760E-01	1.29	1.37566032206E-01	1.79	6.56386410378E-02	2.29	3.29413187551E-02	2.79	1.70739521103E-02
0.3	9.05676651676E-01	0.8	3.10596578546E-01	1.3	1.35450957849E-01	1.8	6.47131293639E-02	2.3	3.25022671716E-02	2.8	1.68552924452E-02
0.31	8.81505745679E-01	0.81	3.05042539200E-01	1.31	1.33372975293E-01	1.81	6.38019032036E-02	2.31	3.20694700634E-02	2.81	1.66395802629E-02
0.32	8.58335189327E-01	0.82	2.99611235503E-01	1.32	1.31331314175E-01	1.82	6.29047145178E-02	2.32	3.16428303025E-02	2.82	1.64267731273E-02
0.33	8.36101161455E-01	0.83	2.94299155371E-01	1.33	1.29325223609E-01	1.83	6.20213202416E-02	2.33	3.12222524414E-02	2.83	1.62168292660E-02
0.34	8.14745579688E-01	0.84	2.89102918181E-01	1.34	1.27353971585E-01	1.84	6.11514821669E-02	2.34	3.08076426796E-02	2.84	1.60097075588E-02
0.35	7.94215434621E-01	0.85	2.84019268502E-01	1.35	1.25416844389E-01	1.85	6.02949668284E-02	2.35	3.03989088316E-02	2.85	1.58053675268E-02
0.36	7.74462217809E-01	0.86	2.79045070182E-01	1.36	1.23513146039E-01	1.86	5.94515453928E-02	2.36	2.99959602953E-02	2.86	1.56037693209E-02
0.37	7.55441428159E-01	0.87	2.74177300782E-01	1.37	1.21642197750E-01	1.87	5.86209935508E-02	2.37	2.95987080211E-02	2.87	1.54048737113E-02
0.38	7.37112144155E-01	0.88	2.69413046334E-01	1.38	1.19803337414E-01	1.88	5.78030914126E-02	2.38	2.92070644819E-02	2.88	1.52086420766E-02

0.39	7.19436651662E-01	0.89	2.64749496385E-01	1.39	1.17995919100E-01	1.89	5.69976234054E-02	2.39	2.88209436435E-02	2.89	1.50150363937E-02
0.4	7.02380118866E-01	0.9	2.60183939326E-01	1.4	1.16219312571E-01	1.9	5.62043781745E-02	2.4	2.84402609358E-02	2.9	1.48240192273E-02
0.41	6.85910311352E-01	0.91	2.55713757978E-01	1.41	1.14472902821E-01	1.91	5.54231484870E-02	2.41	2.80649332246E-02	2.91	1.46355537201E-02
0.42	6.69997341530E-01	0.92	2.51336425417E-01	1.42	1.12756089623E-01	1.92	5.46537311370E-02	2.42	2.76948787842E-02	2.92	1.44496035828E-02
0.43	6.54613447544E-01	0.93	2.47049501029E-01	1.43	1.11068287105E-01	1.93	5.38959268553E-02	2.43	2.73300172703E-02	2.93	1.42661330848E-02
0.44	6.39732797606E-01	0.94	2.42850626776E-01	1.44	1.09408923324E-01	1.94	5.31495402196E-02	2.44	2.69702696935E-02	2.94	1.40851070441E-02
0.45	6.25331316323E-01	0.95	2.38737523654E-01	1.45	1.07777439869E-01	1.95	5.24143795680E-02	2.45	2.66155583939E-02	2.95	1.39064908187E-02
0.46	6.11386530112E-01	0.96	2.34707988344E-01	1.46	1.06173291471E-01	1.96	5.16902569151E-02	2.46	2.62658070155E-02	2.96	1.37302502971E-02
0.47	5.97877429243E-01	0.97	2.30759890037E-01	1.47	1.04595945628E-01	1.97	5.09769878694E-02	2.47	2.59209404819E-02	2.97	1.35563518894E-02
0.48	5.84784344415E-01	0.98	2.26891167414E-01	1.48	1.03044882244E-01	1.98	5.02743915536E-02	2.48	2.55808849715E-02	2.98	1.33847625188E-02
0.49	5.72088836049E-01	0.99	2.23099825790E-01	1.49	1.01519593278E-01	1.99	4.95822905267E-02	2.49	2.52455678947E-02	2.99	1.32154496127E-02
0.5	5.59773594776E-01	1	2.19383934396E-01	1.5	1.00019582407E-01	2	4.89005107081E-02	2.5	2.49149178703E-02	3	1.30483810942E-02

The following specific relation is given between E1 and Ei

$E_1(x) = -Ei(-x)$

And derivative of $E_1(x)$ $\frac{dE_1(x)}{dx} = -E_0(x)$ It is also known that $E_0(x) = \frac{e^{-x}}{x}$ therefore

$\frac{dEi(x)}{dx} = -\frac{e^x}{x}$ and therefore $\frac{dEi(u)}{dx} = -\frac{du}{dx}\frac{e^u}{u}$ in our equation $u = -\frac{r^2}{4\alpha t}$ and $\frac{du}{dx} = -\frac{2r}{4\alpha t}$

$$\frac{dEi(u)}{dx} = -\left(-\frac{2r}{4\alpha t}\right)\left(\frac{\exp\left(-\frac{r^2}{4\alpha t}\right)}{-\frac{r^2}{4\alpha t}}\right) = \frac{2}{r}\exp\left(-\frac{r^2}{4\alpha t}\right)$$ Therefore

$$T_s(r,t) = A - BEi\left(\frac{-r^2}{4\alpha_s t}\right) \qquad \frac{dT_s}{dr} = -\frac{2}{r}B\exp\left(-\frac{r^2}{4\alpha_s t}\right)$$

$$T_l(r,t) = T_l - CEi\left(\frac{-r^2}{4\alpha_l t}\right) \qquad \frac{dT_l}{dr} = -\frac{2}{r}C\exp\left(-\frac{r^2}{4\alpha_l t}\right)$$

Since heat sink at the center

$$-Q = \lim_{r\to 0}\left(-k_s A\frac{dT_s}{dr}\right) = \lim_{r\to 0}\left(-k_s(2\pi r)\frac{dT_s}{dr}\right) = \lim_{r\to 0}\left(-k_s(2\pi r)\left(-\frac{2}{r}B\exp\left(-\frac{r^2}{4\alpha_s t}\right)\right)\right) = 4\pi B k_s$$

$B = \frac{-Q}{4\pi k_s}$ Assuming also $\lambda = \frac{s(t)}{2(\alpha_s t)^{1/2}}$ Interface equation:

$T_s(r,t) = T_l(r,t) = T_m$ at r=s(t) t>0

$$A + \frac{Q}{4\pi B}Ei\left(-\lambda^2\right) = T_l - CEi\left(-\lambda^2\left(\frac{\alpha_s}{\alpha_l}\right)\right) = T_m$$

Solving A and B from these equation for constant λ

$$A = T_m - \frac{Q}{4\pi k_s}Ei\left(-\lambda^2\right)$$

$$C = \frac{T_i - T_m}{Ei(-\lambda^2\alpha_s/\alpha_l)}$$

$$s(t) = 2\lambda(\alpha_s t)^{1/2} \qquad s^2(t) = 4\lambda^2\alpha_s t \quad 2s(t)\frac{ds(t)}{dt} = 4\lambda^2\alpha_s \qquad \frac{ds(t)}{dt} = \frac{2\lambda^2\alpha_s}{s(t)}$$

$$T_s(r,t) = T_m - \frac{Q}{4\pi k_s} Ei\left(-\lambda^2\right) + \frac{Q}{4\pi k_s} Ei\left(\frac{-r^2}{4\alpha_s t}\right)$$

$$T_l(r,t) = T_i - \frac{T_i - T_m}{Ei(-\lambda^2 \alpha_s / \alpha_l)} Ei\left(\frac{-r^2}{4\alpha_l t}\right)$$

And by applying the interface boundary :

$$k_s \frac{\partial T_s(r,t)}{\partial r} - k_l \frac{\partial T_l(r,t)}{\partial r} = \rho_s (H_l - H_s) \frac{ds(t)}{dt} \quad r = s(t)$$

$$k_s \left(\frac{2}{s(t)} \frac{Q}{4\pi k_s} \exp\left(-\frac{s(t)^2}{4\alpha_s t} \right) \right) - k_l \left(-\frac{2}{s(t)} \frac{T_i - T_m}{Ei(-\lambda^2 \alpha_s / \alpha_l)} \exp\left(-\frac{s(t)^2}{4\alpha_l t} \right) \right) = \rho_s (H_l - H_s) \frac{2\lambda^2 \alpha_s}{s(t)}$$

$$\left(\frac{Q}{4\pi} \exp\left(-\lambda^2\right) \right) + k_l \left(\frac{T_i - T_m}{Ei(-\lambda^2 \alpha_s / \alpha_l)} \exp\left(-\lambda^2 \frac{\alpha_s}{\alpha_l} \right) \right) = \rho_s (H_l - H_s) \lambda^2 \alpha_s$$

In order to solve this equation a java progaram is written

```
import javax.swing.*;
//class to calculate lambda root value
class fx2 extends f_x
{
double Q,ks,kl,alphas,alphal,Tm,Ti,Cps,hfg,ros,rol;
steamIAPWS_IF97 st;
double a[];
public fx2(double Qi,double Tmi,double Tii)
{Tm=Tmi;Q=Qi;Ti=Tii;
 hfg=333.55e3;                // J/kgK
 st=new steamIAPWS_IF97();
 //Tm degree C
 Cps=2109.7329400765643+7.637140284403064*Tm+0.00269180790270179*Tm*Tm;   // J/kgK
 ks=2.217352440215741-0.006916860285175764 4*Tm+1.0167211666721175E-4*Tm*Tm+4.456742855800601E-7*Tm*Tm*Tm;
 ros=916.8094018597769-0.1525517512848 6845*Tm-0.0015446216557046906*Tm*Tm-9.285069004045726E-6*Tm*Tm*Tm;
 alphas=ks/(ros*Cps);
 double P=101.325; //kPA;
 kl=st.k_PT(P,(Tm+273.15));
 double a[]=st.property("tx",(Tm+273.15),0.0);
 rol=1.0/a[2];
 double h1=a[3];
 a=st.property("tx",(Tm+273.25),0.0);
 double h2=a[3];
 double Cpl=(h2-h1)/0.1*1e3;
 alphal=kl/(rol*Cpl);
 }

public double func(double l)
{double al=alphas/alphal;
 double c1=-l*l*al;
 double ff=Q/(4.0*PI)*exp(-l*l)+kl*(Ti-Tm)/Ei(c1)*exp(c1)-l*l*alphas*ros*hfg;
 return ff;
}}
//class to plot & evaluate temperature profile

class ice_PC2 extends fi_xt
{
double Q,alphas,alphal,lambda,Tm,Ti;
fx2 f;
public ice_PC2(double Qi,double Tmi,double Tii)
{Tm=Tmi;Q=Qi;Ti=Tii;
 f=new fx2(Q,Tm,Ti);
 double a[][]=rootlimits(f,0.0001,4.0,100,1);
 lambda=bisection(f,a[0][0],a[1][0]);
 System.out.println("lambda="+lambda);
}
public static double[][] rootlimits(f_x f,double x1,double x2,int n,int nbb)
{
        // find root limits
        // n : number of subdivisions
        // x1,x2 : limit values of the root
        // nbb = number of roots in the serched region
```

```
                //
                int nb;
                int i;
                double x,fp,fc,dx;
                double xb[][]=new double[2][nbb];
                nb=0;
                dx=(x2-x1)/n;
                x=x1;
                fp=f.func(x1);
                for (i=1;i<=n;i++)
                {
                        x+=dx;
                        fc=f.func(x);
                        // eğer kök olan bölge bulunduysa.....
                        if (fc*fp < 0.0 || fp==0) {
                                xb[0][nb]=x-dx;
                                xb[1][nb]=x;
                                nb++;
                        }
                        fp=fc;
                        if (nbb == nb) return xb;
                }
if( nb == 0)
                JOptionPane.showMessageDialog(null,"search is completed no root is found"
   ,"SEARCH RESULT WARNING",JOptionPane.WARNING_MESSAGE);
else if(nb<nbb)
  {             JOptionPane.showMessageDialog(null,"search is completed "+nb+" root is found \n"+
    "You searched for  "+nbb+" root","SEARCH RESULT WARNING",JOptionPane.WARNING_MESSAGE);
   double xc[][]=new double[2][nb];
   for (i=0;i<nb;i++) {xc[0][i]=xb[0][i];xc[1][i]=xb[1][i];}
   return xc;
  }
return xb;
}
public static double bisection(f_x f,double xl,double xu)
{
//bisection root finding method
double test;
double xr=0;
double es,ea;
double fxl,fxr,fxu;
double xold=0;
int maxit=100,iter=0;
es=0.0000001;
ea=1.1*es;
String s="";
fxl= f.func(xl);
fxu= f.func(xu);
if(fxl*fxu>0) System.out.println("xl="+xu+"xr="+xr+"fxl="+fxl+"fxu="+fxu+"error");
while((ea>es)&&(iter<maxit))
{   xold=xr;
    xr=(xl+xu)/2.0;
    fxr= f.func(xr);
    iter++;
    if(xr!=0)
       { ea=Math.abs((xr-xold)/xr)*100;}
    test= fxl*fxr;
    if(test==0.0)      ea=0;
    else if(test<0.0) {xu=xr;fxu=fxr;}
    else if(test>0)   {xl=xr;fxl=fxr;}
    else              {ea=0;}
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded \n"+
   " result might not be valid","MAKSİMUM NUMBER OF ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
//JOptionPane.showMessageDialog(null,s,
//     "bisection root finding method : ",JOptionPane.PLAIN_MESSAGE);
   return xr;
}
public double s(double t)
{return 2.0*lambda*sqrt(f.alphas*t);}

public double[] func(double r,double t)
{ double a[]=new double[2];
  if(r<s(t))
  {a[0]=Tm+Q/(4.0*PI*f.ks)*(Ei(-r*r/(4.0*f.alphas*t))-Ei(-lambda*lambda));a[1]=Tm;}
  else
  {a[0]=Ti-(Ti-Tm)/Ei(-lambda*lambda*f.alphas/f.alphal)*Ei(-r*r/(4.0*f.alphal*t));
```

```
  a[1]=Ti;
  }
  return a;
}
}

public class HT_phase_change2
{
public static void main(String ... arg)
{   ice_PC2 ff=new ice_PC2(100,0.0,1.0);
    timeplot_n p=new timeplot_n(ff,0.01,0.1,0.0001,0.0,3600.0,0.1,100.0);
    p.start();}}
```

10 HEAT CONDUCTION IN ANISOTROPIC SOLIDS

In all previous chapters heat conduction is taken as isotropic. In isotropic systems thermal conductivity does not depends on direction. On the other hand, in anisotropic materials thermal conductivity changes as a function of direction.



An example of anisotropic solid: wood

For this reason it is a complicated problem and should be investigated for each materials in details. Let us try to get heat transfer equations for the anisotropic material first
Heat transfer of the isotropic materials obeys the Fourier's law which

$$q = -k\nabla T$$

For the anisotropic material heat trasnfer will obey

$$-q_x = k_{11}\frac{\partial T}{\partial x} + k_{12}\frac{\partial T}{\partial y} + k_{13}\frac{\partial T}{\partial z}$$

$$-q_y = k_{21}\frac{\partial T}{\partial x} + k_{22}\frac{\partial T}{\partial y} + k_{23}\frac{\partial T}{\partial z}$$

$$-q_z = k_{31}\frac{\partial T}{\partial x} + k_{32}\frac{\partial T}{\partial y} + k_{33}\frac{\partial T}{\partial z}$$

$$q = \sqrt{q_x^2 + q_y^2 + q_z^2}$$

As it can be seen from the equation heat flux direciton is depends on directional heat transfer coefficients. k_{ij} ,s called the conductivity coefficients and combine as a tensor.

$$k = \begin{vmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{vmatrix}$$

In general coefficients are symmetric i.e $k_{ij} = k_{ji}$ and also $k_{ii} > 0$ and also $k_{ii}k_{jj} > k_{ij}^2$
Similarly in cylindrical coordinates equations can be written as

$$-q_r = k_{11}\frac{\partial T}{\partial r} + k_{12}\frac{1}{r}\frac{\partial T}{\partial \phi} + k_{13}\frac{\partial T}{\partial z}$$
$$-q_\phi = k_{21}\frac{\partial T}{\partial r} + k_{22}\frac{1}{r}\frac{\partial T}{\partial \phi} + k_{23}\frac{\partial T}{\partial z}$$
$$-q_z = k_{31}\frac{\partial T}{\partial r} + k_{32}\frac{1}{r}\frac{\partial T}{\partial \phi} + k_{33}\frac{\partial T}{\partial z}$$

Conservation of energy equation in cartesian system then becomes:

$$k_{11}\frac{\partial^2 T}{\partial x^2} + k_{22}\frac{\partial^2 T}{\partial y^2} + k_{33}\frac{\partial^2 T}{\partial z^2} + (k_{12}+k_{21})\frac{\partial^2 T}{\partial x \partial y} + (k_{13}+k_{31})\frac{\partial^2 T}{\partial x \partial z} + (k_{23}+k_{32})\frac{\partial^2 T}{\partial y \partial z} + g(x,y,z,t) = \rho C_p \frac{\partial T}{\partial t}$$

Where $k_{12} = k_{21}$ $k_{13} = k_{31}$ $k_{23} = k_{32}$

Boundary conditions :

$$\pm \delta_i k_{ref}\frac{\partial T}{\partial n^*} + h_i T = f_i$$

$$\frac{\partial T}{\partial n^*} = \sum_{j=1}^{3} \frac{k_{ij}}{k_{ref}}\frac{\partial T}{\partial x_i}$$

Where δ_i zero or unity if it is zero, it will give temp type boundary layer. k_{ref} is the reference thermal conductivity, can be taken as k_{11} or k_{22} or k_{33} but most of the time it is taken as k_{11} for convinious. Let us write the x axis component boundary layer as open formula:

$-\left(k_{11}\frac{\partial T}{\partial x} + k_{12}\frac{\partial T}{\partial y} + k_{13}\frac{\partial T}{\partial z}\right) + \boldsymbol{h_1 T} = \boldsymbol{f_1}$ **at x=0**

$\left(k_{11}\frac{\partial T}{\partial x} + k_{12}\frac{\partial T}{\partial y} + k_{13}\frac{\partial T}{\partial z}\right) + \boldsymbol{h_2 T} = \boldsymbol{f_2}$ **at x=L**

Assuming $k_{ref} = k_{11}$

$k_{11}\frac{\partial T}{\partial x} + k_{12}\frac{\partial T}{\partial y} + k_{13}\frac{\partial T}{\partial z} = \frac{\partial T}{\partial x} + \frac{k_{12}}{k_{ref}}\frac{\partial T}{\partial y} + \frac{k_{13}}{k_{ref}}\frac{\partial T}{\partial z} = \frac{\partial T}{\partial x} + \epsilon_{12}\frac{\partial T}{\partial y} + \varepsilon_{13}\frac{\partial T}{\partial z}$

Conservation of energy equation in cylindrical system:

$$k_{11}\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial T}{\partial r}\right) + k_{22}\frac{1}{r^2}\frac{\partial^2 T}{\partial \phi^2} + k_{33}\frac{\partial^2 T}{\partial z^2} + (k_{12}+k_{21})\frac{1}{r}\frac{\partial^2 T}{\partial x \partial \phi} + (k_{13}+k_{31})\frac{\partial^2 T}{\partial x \partial z} + (k_{23}+k_{32})\frac{1}{r}\frac{\partial^2 T}{\partial \phi \partial z} + g(x,y,z,t) = \rho C_p \frac{\partial T}{\partial t}$$

r axis boundary conditions

$-\left(k_{11}\frac{\partial T}{\partial r} + k_{12}\frac{1}{r}\frac{\partial T}{\partial \phi} + k_{13}\frac{\partial T}{\partial z}\right) + \boldsymbol{h_1 T} = \boldsymbol{f_1}$ **at x=0**

$\left(k_{11}\frac{\partial T}{\partial r} + k_{12}\frac{1}{r}\frac{\partial T}{\partial \phi} + k_{13}\frac{\partial T}{\partial z}\right) + \boldsymbol{h_2 T} = \boldsymbol{f_2}$ **at x=L**

Now let us consider the cartesian system energy equation

$$k_{11}\frac{\partial^2 T}{\partial x^2} + k_{22}\frac{\partial^2 T}{\partial y^2} + k_{33}\frac{\partial^2 T}{\partial z^2} + (k_{12}+k_{21})\frac{\partial^2 T}{\partial x \partial y} + (k_{13}+k_{31})\frac{\partial^2 T}{\partial x \partial z} + (k_{23}+k_{32})\frac{\partial^2 T}{\partial y \partial z} + g(x,y,z,t) = \rho C_p \frac{\partial T}{\partial t}$$

Where $k_{12} = k_{21}$ $k_{13} = k_{31}$ $k_{23} = k_{32}$

When the thermal conductivity matrix is symmetric, it is possible to find such a cartesian coordinates $[u_1, u_2, u_3]$ that thermal conductivey matrix reduces to

$$k = \begin{vmatrix} k_1 & 0 & 0 \\ 0 & k_2 & 0 \\ 0 & 0 & k_3 \end{vmatrix}$$

Where k_1, k_2, k_3 are called principle thermal conductivities. Then the catesian coordinate heat transfer equation will be resuced to

$$k_1 \frac{\partial^2 T(u_1,u_2,u_3,t)}{\partial u_1^2} + k_2 \frac{\partial^2 T(u_1,u_2,u_3,t)}{\partial u_2^2} + k_3 \frac{\partial^2 T(u_1,u_2,u_3,t)}{\partial u_3^2} + g(u_1,u_2,u_3,t) = \rho C_p \frac{\partial T(u_1,u_2,u_3,t)}{\partial t}$$

Reducing original heat conduction problem to principle thermal conductivities are called eigenvalue calculations. Eigenvalue matrices can be written as:

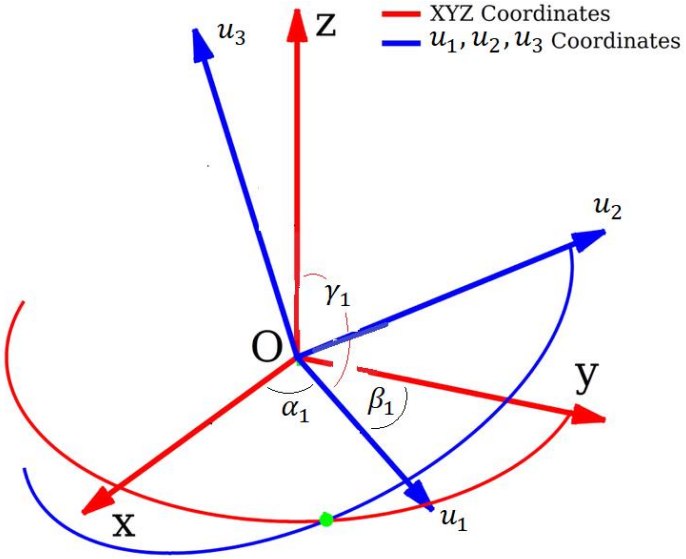
$$\begin{bmatrix} k_{11}-k_1 & k_{12} & k_{13} \\ k_{21} & k_{22}-k_1 & k_{23} \\ k_{31} & k_{32} & k_{33}-k_1 \end{bmatrix} \begin{Bmatrix} l_{11} \\ l_{12} \\ l_{13} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}$$

$$\begin{bmatrix} k_{11}-k_2 & k_{12} & k_{13} \\ k_{21} & k_{22}-k_2 & k_{23} \\ k_{31} & k_{32} & k_{33}-k_2 \end{bmatrix} \begin{Bmatrix} l_{21} \\ l_{22} \\ l_{23} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}$$

$$\begin{bmatrix} k_{11}-k_1 & k_{12} & k_{13} \\ k_{21} & k_{22}-k_1 & k_{23} \\ k_{31} & k_{32} & k_{33}-k_1 \end{bmatrix} \begin{Bmatrix} l_{21} \\ l_{22} \\ l_{23} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}$$

Where $\begin{bmatrix} l_{11} & l_{12} & l_{13} \\ l_{21} & l_{22} & l_{23} \\ l_{31} & l_{32} & l_{33} \end{bmatrix} = \begin{bmatrix} \cos(\alpha_1) & \cos(\beta_1) & \cos(\gamma_1) \\ \cos(\alpha_2) & \cos(\beta_2) & \cos(\gamma_2) \\ \cos(\alpha_3) & \cos(\beta_3) & \cos(\gamma_2) \end{bmatrix}$

l_{ij} are eigen vectors or directional cosines of (xyz) axis and (u_1, u_2, u_3) α_1 is the angle(radian) between x axis and u_1 axis. β_1 is the angle(radian) between y axis and u_1 axis and γ_1 is the angle(radian) between z axis and u_1 axis. Similarly α_2 is the angle(radian) between x axis and u_2 axis. β_2 is the angle(radian) between y axis and u_2 axis and γ_2 is the angle(radian) between z axis and u_2 axis. And α_3 is the angle(radian) between x axis and u_3 axis. β_3 is the angle(radian) between y axis and u_3 axis and γ_3 is the angle(radian) between z axis and u_3 axis.



PROBLEM: Thermal conductivity of single crystalline β-Ga_2O_3 is given by Jiang et al. [113] as

$$k = \begin{vmatrix} 12.7 & -1.2 & 0 \\ -1.2 & 9.7 & 0 \\ 0 & 0 & 22.5 \end{vmatrix}$$

Find the principle thermal conductivities and its direction with respect to given cartesian coordinate system.

When the thermal conductivity matrix is investigated it is immadiately seen that k_3=22.5 and $\{l_{31}\ l_{32}\ l_{33}\} = \{1\ 1\ 1\}$ so that eigenvalue problem is reduced to 2 dimension

$$k = \begin{vmatrix} 12.7 & -1.2 \\ -1.2 & 9.7 \end{vmatrix}$$

We will use QL method to find eigenvalues here.

QL ALGORITHM FOR EIGENVALUE PROBLEM

If a symmetric matrix A is considered, This matrix can be devided into two submatrix as Q orthogonal matrix and L lower triangular matrix.

A=Q L . Since Q is an orthogonal matrix $Q^T = Q^{-1}$. If both side of the eqaution is multiplied by Q^T , it becomes

$Q^T A = L$. A new B matrix can be created as:

$B=L\ Q = Q^T A\ Q$. Matrix Bis a similarity transformed form of matrix A, therefore eigenvalues of both matrices are the same.

QL algorithm is based on a series of orthogonal transformations. The recurrence relations used is $\mathbf{A_m=Q_m\ L_m}$ and $\mathbf{A_{m+1}= L_m\ Q_m = Q_m^T\ A_m\ Q_m}$

Because A is a symmetric matrix, eigenvalue will be the real numbers

```
import java.io.*;
import javax.swing.*;
//import Numeric;

class NA134
{

public static double[][] tridiagonal(double b[][], double d[], double e[])
{
//reduces matrix to tridiaonal form by using HOUSEHOLDER TRANSFORMATION
//this method is used by QL method to calculate eigen values
//and eigen  vectors of a symmetric matrix
        int l,k,j,i;
  int n=b.length;
        double scale,hh,h,g,f;
  double a[][]=new double[n+1][n+1];
  double c[][]=new double[n][n];
  for(i=0;i<n;i++)
   for(j=0;j<n;j++)
     a[i][j]=b[i][j];
        for (i=n;i>=2;i--) {
                l=i-1;
                h=scale=0.0;
                if (l > 1) {
                        for (k=1;k<=l;k++)
                                scale += Math.abs(a[i-1][k-1]);
                        if (scale == 0.0)
                                e[i-1]=a[i-1][l-1];
                        else {
                                for (k=1;k<=l;k++) {
                                        a[i-1][k-1] /= scale;
                                        h += a[i-1][k-1]*a[i-1][k-1];
                                }
                                f=a[i-1][l-1];
                                g=(f >= 0.0 ? -Math.sqrt(h) : Math.sqrt(h));
                                e[i-1]=scale*g;
                                h -= f*g;
                                a[i-1][l-1]=f-g;
                                f=0.0;
                                for (j=1;j<=l;j++) {
                                        a[j-1][i-1]=a[i-1][j-1]/h;
                                        g=0.0;
                                        for (k=1;k<=j;k++)
                                                g += a[j-1][k-1]*a[i-1][k-1];
                                        for (k=j+1;k<=l;k++)
                                                g += a[k-1][j-1]*a[i-1][k-1];
                                        e[j-1]=g/h;
                                        f += e[j-1]*a[i-1][j-1];
                                }
                                hh=f/(h+h);
```

```
                                        for (j=1;j<=l;j++) {
                                                f=a[i-1][j-1];
                                                e[j-1]=g=e[j-1]-hh*f;
                                                for (k=1;k<=j;k++)
                                                        a[j-1][k-1] -= (f*e[k-1]+g*a[i-1][k-1]);
                                        }
                                }
                        } else
                                e[i-1]=a[i-1][l-1];
                        d[i-1]=h;
                }
                d[1-1]=0.0;
                e[1-1]=0.0;
                /* Contents of this loop can be omitted if eigenvectors not
                                wanted except for statement d[i-1]=a[i-1][i-1]; */
                for (i=1;i<=n;i++) {
                        l=i-1;
                        if (d[i-1] != 0) {
                                for (j=1;j<=l;j++) {
                                        g=0.0;
                                        for (k=1;k<=l;k++)
                                                g += a[i-1][k-1]*a[k-1][j-1];
                                        for (k=1;k<=l;k++)
                                                a[k-1][j-1] -= g*a[k-1][i-1];
                                }
                        }
                        d[i-1]=a[i-1][i-1];
                        a[i-1][i-1]=1.0;
                        for (j=1;j<=l;j++) a[j-1][i-1]=a[i-1][j-1]=0.0;
                }
return  a;
}

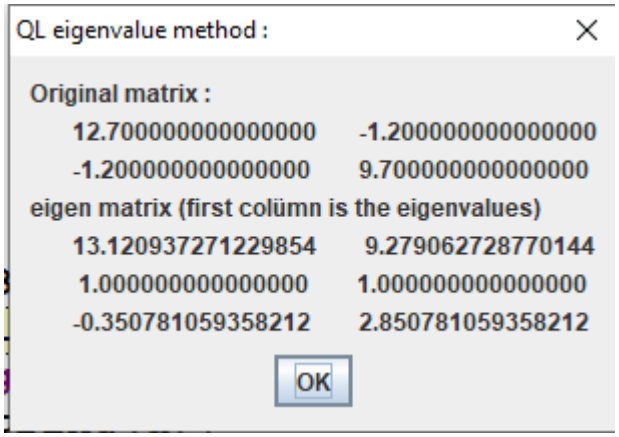
public static double pythag(double a, double b)
{
//this method is used by QL method
                double absa,absb;
                absa=Math.abs(a);
                absb=Math.abs(b);
                if (absa > absb) return absa*Math.sqrt(1.0+(absb/absa)*(absb/absa));
                else return (absb==0.0 ? 0.0 : absb*Math.sqrt(1.0+(absa/absb)*(absa/absb)));
}

public static double[][] QL(double d[], double e[], double a[][])
{
// QL algorithm : eigenvalues of a symmetric matrix reduced to tridiagonal
// form by using method tridiagonal
   int n=d.length;
                int m,l,iter,i,j,k;
                double s,r,p,g,f,dd,c,b;
                for (i=2;i<=n;i++) e[i-2]=e[i-1];
                e[n-1]=0.0;
   double z[][]=new double[n][n];
   for(i=0;i<n;i++)
     for(j=0;j<n;j++)
       z[i][j]=a[i][j];
                for (l=1;l<=n;l++) {
                        iter=0;
                        do {
                                for (m=l;m<=n-1;m++) {
                                        dd=Math.abs(d[m-1])+Math.abs(d[m]);
                                        if ((Math.abs(e[m-1])+dd) == dd) break;
                                }
                                if (m != l) {
            if (iter++ == 30) System.out.println("Too many iterations in QL");
            g=(d[l]-d[l-1])/(2.0*e[l-1]);
            r=Matrix.pythag(g,1.0);
            g=d[m-1]-d[l-1]+e[l-1]/(g+Matrix.SIGN(r,g));
            s=c=1.0;
            p=0.0;
            for (i=m-1;i>=l;i--) {
                                                f=s*e[i-1];
                                                b=c*e[i-1];
                                                e[i]=(r=Matrix.pythag(f,g));
                                                if (r == 0.0) {
                                                        d[i] -= p;
                                                        e[m-1]=0.0;
```

```
                                                                        break;
                                                                }
                                                                s=f/r;
                                                                c=g/r;
                                                                g=d[i ]-p;
                                                                r=(d[i-1]-g)*s+2.0*c*b;
                                                                d[i ]=g+(p=s*r);
                                                                g=c*r-b;
                                                                for (k=1;k<=n;k++) {
                                                                        f=z[k-1][i ];
                                                                        z[k-1][i ]=s*z[k-1][i-1]+c*f;
                                                                        z[k-1][i-1]=c*z[k-1][i-1]-s*f;
                                                                }
                                                        }
                                                        if (r == 0.0 && i >= l) continue;
                                                        d[l-1] -= p;
                                                        e[l-1]=g;
                                                        e[m-1]=0.0;
                                                }
                                        } while (m != l);
                                }
                        return z;
                }

                public static double[][] eigenQL(double a[][])
                {
                 // QL algoritm to solve eigen value problems
                 // symmetric matrices only (real eigen values)
                 // first column of the matrix returns eigen values
                 // second..n+1 column returns eigen vectors.
                 // Note : If matrix is not symmetric DO NOT use
                 // this method use eigenValue method (a QR algorithm)
                 int i,j;
                 int n=a.length;
                 double sum[]=new double[n];;
                 double d[]=new double[n];
                 double b[][]=new double[n][n];
                 double e[]=new double[n];
                 double z[][]=new double[n+1][n];
                 b=tridiagonal(a,d,e);
                 b=QL(d,e,b);
                 for(j=0;j<n;j++)
                    {
                    z[0][j]=d[j];
                    for(i=0;i<n;i++)
                      {
                      z[i+1][j]=b[i][j]/b[0][j];
                      if(z[i+1][j]<1e-13) z[i+1][i]=0;
                      }
                    }
                 // result: first row is eigenvalues, the rest is eigenvector
                 return z;
                }

                  public static void main(String args[]) throws IOException
                  {
                  //Eigen Value calculations
                  double c[][];
                  c=new double[3][3];
                double a[][]={{12.7,-1.2},{-1.2,9.7}};
                  String s="Original matrix : \n";
                  s+=Matrix.toString(a);
                  c=new double[3][3];
                  c=eigenQL(a);
                  s+="eigen matrix (first colümn is the eigenvalues) \n";
                  s+=Matrix.toString(c);
                  JOptionPane.showMessageDialog(null,s,
                     "QL eigenvalue method : ",JOptionPane.PLAIN_MESSAGE);
                  }
                }
```



From this principle thermal conductivity matrix

$$k = \begin{vmatrix} 13.12093 & 0 & 0 \\ 0 & 9.27906 & 0 \\ 0 & 0 & 22.5 \end{vmatrix}$$

And eigenvector(directional cosines are:

$$l_{ij} = \begin{vmatrix} 1 & -0.350781059 & 1 \\ 1 & 2.850781059 & 1 \\ 1 & 1 & 1 \end{vmatrix}$$

A further approximation can also be applied by defining a new coordinate system such as:

$$k = (k_1 k_2 k_3)^{1/2}$$

$$X = \left(\frac{k}{k_1}\right)^{1/2} u_1$$

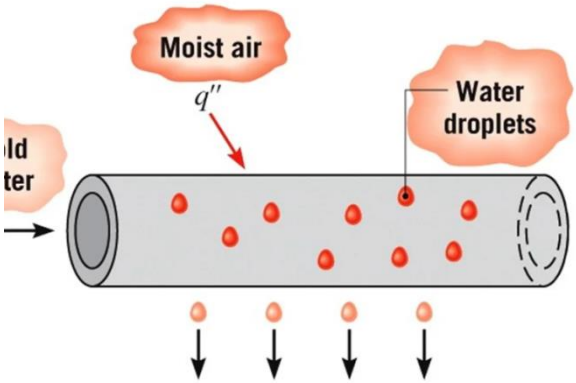
$$Y = \left(\frac{k}{k_2}\right)^{1/2} u_2$$

$$Z = \left(\frac{k}{k_3}\right)^{1/2} u_3$$

Then the conduction energy equation becomes

$$k\left(\frac{\partial^2 T(X,Y,Z,t)}{\partial X^2} + \frac{\partial^2 T(X,Y,Z,t)}{\partial Y^2} + \frac{\partial^2 T(X,Y,Z,t)}{\partial Z^2}\right) + g(X,Y,Z,t) = \rho C_p \frac{\partial T(X,Y,Z,t)}{\partial t}$$

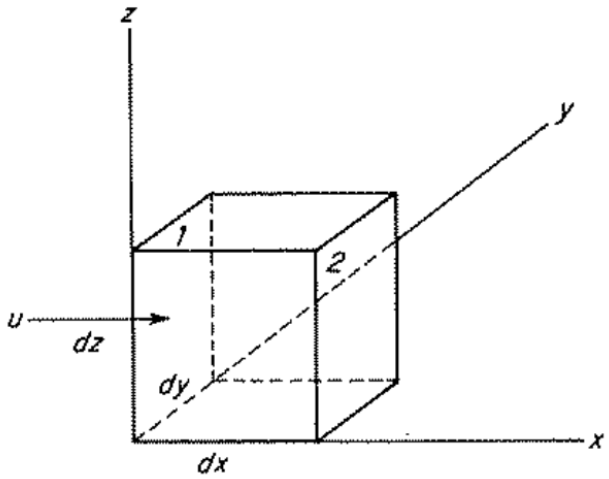
PART II HEAT CONVECTION



11 NAVIER STOKES EQUATIONS

11.1 CONSERVATION OF MASS

Let us consider a cartesian coordinate system differential element of dimensions dx,dy and dz.
Conservation of mass for this rectangular element can be written as:
Mass going in – mass going out = mass stored in the element



$$m_{in} - m_{out} = \Delta m_{stored}$$
$$m_x = \rho u dA_x = \rho u(dydz)$$
$$m_y = \rho v dA_y = \rho v(dxdz)$$
$$m_z = \rho w dA_z = \rho w(dxdy)$$

where
ρ density kg/m^3
u x direction velocity m/s
v y direction velocity m/s
w z direction velocity m/s
Taylor formula can be written as:

$$m_{x+dx} = m_x + \frac{dm_x}{dx}dx + \frac{1}{2!}\frac{d^2m_x}{dx^2}(dx)^2 + \frac{1}{3!}\frac{d^3m_x}{dx^3}(dx)^3 + ..$$

Because of small differential dimension of the cube higher derivatives of the Taylor's formula can be neglected and

$$m_{x+dx} = m_x + \frac{dm_x}{dx}dx$$

Similarly,

$$m_{y+dy} = m_y + \frac{dm_y}{dy}dy$$

$$m_{z+dz} = m_z + \frac{dm_z}{dz}dz$$

And

$$\Delta m_{stored} = \frac{dm}{dt} = \frac{d(\rho V)}{dt} = \frac{d(\rho dxdydz)}{dt} = \frac{d\rho}{dt}dxdydz$$

Total mass balance:

$$(m_x + m_y + m_z) - (m_{x+dx} + m_{y+dy} + m_{z+dz}) = \frac{dm}{dt}$$

$$(m_x + m_y + m_z) - \left(m_x + \frac{dm_x}{dx}dx + m_y + \frac{dm_y}{dy}dy + m_z + \frac{dm_z}{dz}dz\right) = \frac{dm}{dt}$$

$$-\left(\frac{dm_x}{dx}dx + \frac{dm_y}{dy}dy + \frac{dm_z}{dz}dz\right) = \frac{dm}{dt}$$

$$-\left(\frac{d[\rho u(dydz)]}{dx}dx + \frac{d[\rho v(dxdz)]}{dy}dy + \frac{d[\rho w(dxdy)]}{dz}dz\right) = \frac{d[\rho dxdydz]}{dt}$$

$$-\left(\frac{d[\rho u]}{dx}dxdydz + \frac{d[\rho v]}{dy}dxdydz + \frac{d[\rho w]}{dz}dxdydz\right) = \frac{d[\rho]}{dt}dxdydz$$

$$-\left(\frac{d[\rho u]}{dx} + \frac{d[\rho v]}{dy} + \frac{d[\rho w]}{dz}\right) = \frac{d[\rho]}{dt}$$

$$\frac{d[\rho]}{dt} + \frac{d[\rho u]}{dx} + \frac{d[\rho v]}{dy} + \frac{d[\rho w]}{dz} = 0$$

$$\frac{d\rho}{dt} + u\frac{d\rho}{dx} + v\frac{d\rho}{dy} + w\frac{d\rho}{dz} + \rho\left(\frac{du}{dx} + \frac{dv}{dy} + \frac{dw}{dz}\right) = 0$$

$$\frac{D\rho}{dt} = \frac{d\rho}{dt} + u\frac{d\rho}{dx} + v\frac{d\rho}{dy} + w\frac{d\rho}{dz}$$

Divergence V = $divV = \left(\frac{du}{dx} + \frac{dv}{dy} + \frac{dw}{dz}\right)$

$$\frac{D\rho}{dt} + divV = 0$$

If change of density is ignored then

$$\left(\frac{du}{dx} + \frac{dv}{dy} + \frac{dw}{dz}\right) = \frac{du_i}{dx_i} = 0$$

$$divV = 0$$

11.2 CONSERVATION OF MOMENTUM

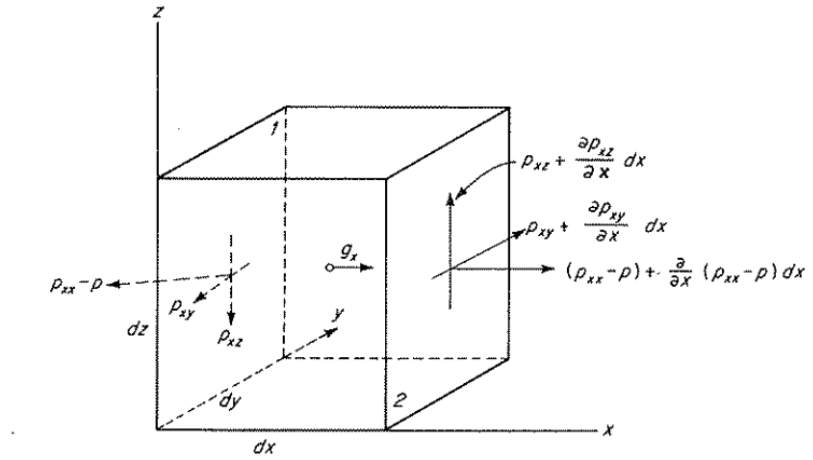
Momentum is the multiplication of mass and velocity. The basic equation fort he conservation of momentum can be expresses as total forces effected into the differential element is eqaul to total change of the momentum in the system. The total change of the momentum is the difference of momentum going into the element minus momemntum going out of the element minus momentum stored in the element. Since the definition of the momentum contains velocity, it is a vectoral quantity, therefore, in x,y and z direction conservation of momentum should be investigated seperately.

Total forces effected into control volume in the given direction=change of the momentum

Change of the momentum=momentum going in – momentum going out – momentum stored in the element

$$\sum F = M_{in} - M_{out} - \Delta M_{stored}$$

Total forces effected into control volume in x direction



In this term P_{xx} is x direction normal stress, P_{xy} and P_{xz} x direction shear stresses going into the differential element. P is the pressure effecting in x direction into the inlet surface. These are surface forces per unit area. There are also volume force effecting to element due to the accelaration of fluid in x direction. Volume force in x direction can be expressed as $F = mg_x$ according to Newton's law. Surface forces going into the element and going out of the element in x direction

$$\vec{F}_x = (P_{xx} - P)dA_x + P_{xy}dA_x + P_{xz}dA_x$$

$$\vec{F}_{x+dx} = \left[(P_{xx} - P)dA_x + \frac{d}{dx}(P_{xx} - P)dx\right]dA_x + \left[P_{xy} + \frac{d}{dy}(P_{xz})dx\right]dA_x + \left[P_{xz} + \frac{d}{dz}(P_{xz})dx\right]dA_x$$

$$\sum F_x = \vec{F}_x - \vec{F}_{x+dx} - mg_x\vec{\imath}$$

$$\sum F_x = -\left(\frac{d(P_{xx} - P)}{dx}dxdydz\right)\vec{\imath} - \left(\frac{dP_{xy}}{dy}dxdydz\right)\vec{\jmath} - \left(\frac{dP_{xz}}{dz}dxdydz\right)\vec{k} - \rho g_x dxdydz\vec{\imath}$$

$$\sum F_x = -\left[\left(\frac{d(P_{xx} - P)}{dx}\right)\vec{\imath} + \left(\frac{dP_{xy}}{dy}\right)\vec{\jmath} + \left(\frac{dP_{xz}}{dz}\right)\vec{k} + \rho g_x\vec{\imath}\right]dxdydz$$

Mass flow going into the element:

$$m_x = \rho u dA_x = \rho u(dydz)$$
$$m_y = \rho v dA_y = \rho v(dxdz)$$
$$m_z = \rho w dA_z = \rho w(dxdy)$$

X direction momentum going into the element

$$\vec{M}_x = m_x u = \rho u^2(dydz)$$
$$\vec{M}_y = m_y u = \rho uv(dxdz)$$
$$\vec{M}_z = m_z u = \rho uw(dxdy)$$

X direction momentum going out of the element

$$\vec{M}_{x+dx} = \vec{M}_x + \frac{d\vec{M}_x}{dx}dx$$

$$\vec{M}_{y+dy} = \vec{M}_y + \frac{d\vec{M}_y}{dy}dy$$

$$\vec{M}_{z+dz} = \vec{M}_z + \frac{d\vec{M}_z}{dz}dz$$

$$\Delta M_{stored_x} = \frac{dM}{dt} = \frac{d(mu)}{dt} = \frac{d(\rho dxdydzu)}{dt} = \frac{d(\rho u)}{dt}dxdydz$$

$$\sum M_x = \left(\vec{M}_x + \vec{M}_y + \vec{M}_z\right) - \left(\vec{M}_{x+dx} + \vec{M}_{y+dy} + \vec{M}_{z+dz}\right) - \frac{dM}{dt}$$

$$\sum M_x = \left(\vec{M}_x + \vec{M}_y + \vec{M}_z\right) - \left(\vec{M}_x + \frac{d\vec{M}_x}{dx}dx + \vec{M}_y + \frac{d\vec{M}_y}{dy}dy + \vec{M}_z + \frac{d\vec{M}_z}{dz}dz\right) - \frac{dM}{dt}$$

$$\sum M_x = -\left(\frac{d\vec{M}_x}{dx}dx + \frac{d\vec{M}_y}{dy}dy + \frac{d\vec{M}_z}{dz}dz\right) - \frac{dM}{dt}$$

$$\sum M_x = -\left(\frac{d\rho u^2(dydz)}{dx}dx + \frac{d\rho uv(dxdz)}{dy}dy + \frac{\rho uw(dxdy)}{dz}dz\right) - \frac{d(\rho u)}{dt}dxdydz$$

$$\sum M_x = -\left(\frac{d\rho u^2}{dx} + \frac{d\rho uv}{dy} + \frac{\rho uw}{dz}\right)dxdydz - \frac{d(\rho u)}{dt}dxdydz$$

$$\sum M_x = \sum F_x$$

$$\sum M_x = \sum F_x$$

$$-\left(\frac{d\rho u^2}{dx} + \frac{d\rho uv}{dy} + \frac{\rho uw}{dz}\right) - \frac{d(\rho u)}{dt} = -\left[\left(\frac{d(P_{xx} - P)}{dx}\right)\vec{i} + \left(\frac{dP_{xy}}{dy}\right)\vec{j} + \left(\frac{dP_{xz}}{dz}\right)\vec{k} + \rho g_x\vec{i}\right]$$

$$\left(\frac{d\rho u^2}{dx} + \frac{d\rho uv}{dy} + \frac{\rho uw}{dz}\right) + \frac{d(\rho u)}{dt} = \left(\frac{d(P_{xx} - P)}{dx}\right)\vec{i} + \left(\frac{dP_{xy}}{dy}\right)\vec{j} + \left(\frac{dP_{xz}}{dz}\right)\vec{k} + \rho g_x\vec{i}$$

If partial derivative is taken:

$$u\left(\frac{d\rho u}{dx} + \frac{d\rho v}{dy} + \frac{d\rho w}{dz} + \frac{d\rho}{dt}\right) + \rho u\frac{du}{dx} + \rho v\frac{dv}{dy} + \rho w\frac{dw}{dz} + \rho\frac{du}{dt} = -\frac{dP}{dx} + \frac{dP_{xx}}{dx} + \frac{dP_{xy}}{dy} + \frac{dP_{xz}}{dz} + \rho g_x$$

Remembering that conservation of mass equation:

$$\frac{d[\rho]}{dt} + \frac{d[\rho u]}{dx} + \frac{d[\rho v]}{dy} + \frac{d[\rho w]}{dz} = 0$$

Momentumun equation in x direction is reduces to:

$$\rho u\frac{du}{dx} + \rho v\frac{dv}{dy} + \rho w\frac{dw}{dz} + \rho\frac{du}{dt} = -\frac{dP}{dx} + \frac{dP_{xx}}{dx} + \frac{dP_{xy}}{dy} + \frac{dP_{xz}}{dz} + \rho g_x$$

Similarly momentum equations in y and z directions can be written as:

$$\rho u\frac{dv}{dx} + \rho v\frac{dv}{dy} + \rho w\frac{dv}{dz} + \rho\frac{dv}{dt} = -\frac{dP}{dy} + \frac{dP_{yy}}{dy} + \frac{dP_{yx}}{dx} + \frac{dP_{yz}}{dz} + \rho g_y$$

$$\rho u\frac{dw}{dx} + \rho v\frac{dw}{dy} + \rho w\frac{dw}{dz} + \rho\frac{dw}{dt} = -\frac{dP}{dz} + \frac{dP_{zz}}{dz} + \frac{dP_{zx}}{dx} + \frac{dP_{zy}}{dy} + \rho g_z$$

In short writing

$$\frac{Du}{dt} = u\frac{du}{dx} + v\frac{dv}{dy} + w\frac{dw}{dz} + \frac{du}{dt}$$

$$\frac{Dv}{dt} = u\frac{dv}{dx} + v\frac{dv}{dy} + w\frac{dv}{dz} + \frac{dv}{dt}$$

$$\frac{Dw}{dt} = u\frac{dw}{dx} + v\frac{dw}{dy} + w\frac{dw}{dz} + \frac{dw}{dt}$$

$$\rho\frac{Du}{dt} = -\frac{dP}{dx} + \frac{dP_{xx}}{dx} + \frac{dP_{xy}}{dy} + \frac{dP_{xz}}{dz} + \rho g_x$$

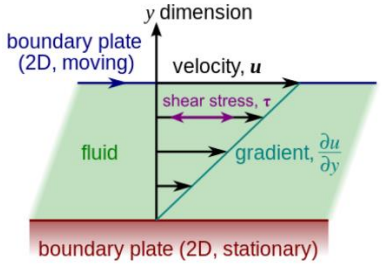
$$\rho\frac{Dv}{dt} == -\frac{dP}{dy} + \frac{dP_{yy}}{dy} + \frac{dP_{yx}}{dx} + \frac{dP_{yz}}{dz} + \rho g_y$$

$$\rho\frac{Dw}{dt} = -\frac{dP}{dz} + \frac{dP_{zz}}{dz} + \frac{dP_{zx}}{dx} + \frac{dP_{zy}}{dy} + \rho g_z$$

In order to solve these differential equation relation between stresses and velocities can be expressed. There are several different theories trying to express these relations. One of the most known relation between shear stresses and velocities are suggested by Newton. This equation assumes linear relation between change of velocity and shear stress is linear (in one directional flow assumption).

$$P_{xy} = \tau_{xy} = \mu \frac{du}{dy}$$

The proportionality constant in Newton's assumption (It is not a law, it is just an approximation curve fitting equation) is called dynamic viscosity. From the equation unit of the dynamic viscosity comes as: $Ns/m^2 = Pa.s$. Fluids that can be approximated to Newton's assumption is called Newtonian fluids. Basic derivation of the equation is coming from Couette flow



If you have a layer of fluid at top of a constant buttom layer and a moving plate is placed at the top and moved with a velocity u. The motion in x direction only and varies with y. Fluid velocity at the bottom plate is zero. If the velocity profile between buttom plate and moving plate is linearly change, we can assume that Newton's assumption is holding for this fluid and shear stress is also changing linearly

Another form of relation is suggested by Ostwald-de Waele has the form of $P_{xy} = \tau_{xy} = m\left(\frac{du}{dy}\right)^n$. m and n in this equation is constants. As it is seen from these equation, applying of these equation into momentum equation will make momentum equation quite a nonlinear equation. Fluids that can be approximated by using these equation is called Ostwald-de Waele fluids or some time as non-newtonian fluids.

Eyring suggested the following correlation: $P_{xy} = \tau_{xy} = Asinh^{-1}\left(\frac{1}{B}\frac{du}{dy}\right)$. Fluids that can be approximated by using these equation is called Eyring fluids. And Bingham suggested the following correletion

$P_{xy} = P_{xy} = \tau_{xy} = \mu\frac{du}{dy} + \tau_0$ if $P_{xy} = \tau_{xy} > \tau_0$

$\frac{du}{dy} = 0$ if $P_{xy} = \tau_{xy} < \tau_0$

As it is seen, in these correlation shear stress value is not going to zero when velocity is zero. For many fluids, it is assumed that fluid relation can be approximated by using Newton's formula. If these approximation is assumed normal and shear stresses can be given as:

$$P_{yx} = \mu\left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right)$$

$$P_{xx} = 2\mu\left(\frac{\partial u}{\partial x}\right) - \mu'\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right)$$

. μ' in these equation is taken value of 2/3 μ for monatomic gases, for the remaining of the fluids a bulk viscosity definition can be given and bulk viscosity values are listed in some references as

$$\kappa = \mu' - 2/3\mu$$

But some sources completely ignores these term. The knowledge about κ (or μ') is frequently not necessary in fluid dynamics and heat transfer problems. For incompressible fluids volumetric change is zero so the term contains Moreover κ is drops out from equations. For monotamic gases

$$\kappa = \frac{2}{3\mu} - \frac{2}{3\mu} =$$

So momentum equation of a Newtonian fluid is:

$$\rho\frac{Du}{dt}=-\frac{dP}{dx}+2\frac{d}{dx}\left(\mu\frac{\partial u}{\partial x}\right)+\frac{d}{dy}\left(\mu\left(\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}\right)\right)+\frac{d}{dz}\left(\mu\left(\frac{\partial u}{\partial z}+\frac{\partial w}{\partial x}\right)\right)-\frac{d}{dx}\left(\mu'\left(\frac{\partial u}{dx}+\frac{\partial v}{dy}+\frac{\partial w}{dz}\right)\right)+\rho g_x$$

$$\rho\frac{Dv}{dt}=-\frac{dP}{dy}+2\frac{d}{dy}\left(\mu\frac{\partial v}{\partial y}\right)+\frac{d}{dz}\left(\mu\left(\frac{\partial v}{\partial z}+\frac{\partial w}{\partial y}\right)\right)+\frac{d}{dx}\left(\mu\left(\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}\right)\right)-\frac{d}{dy}\left(\mu'\left(\frac{\partial u}{dx}+\frac{\partial v}{dy}+\frac{\partial w}{dz}\right)\right)+\rho g_y$$

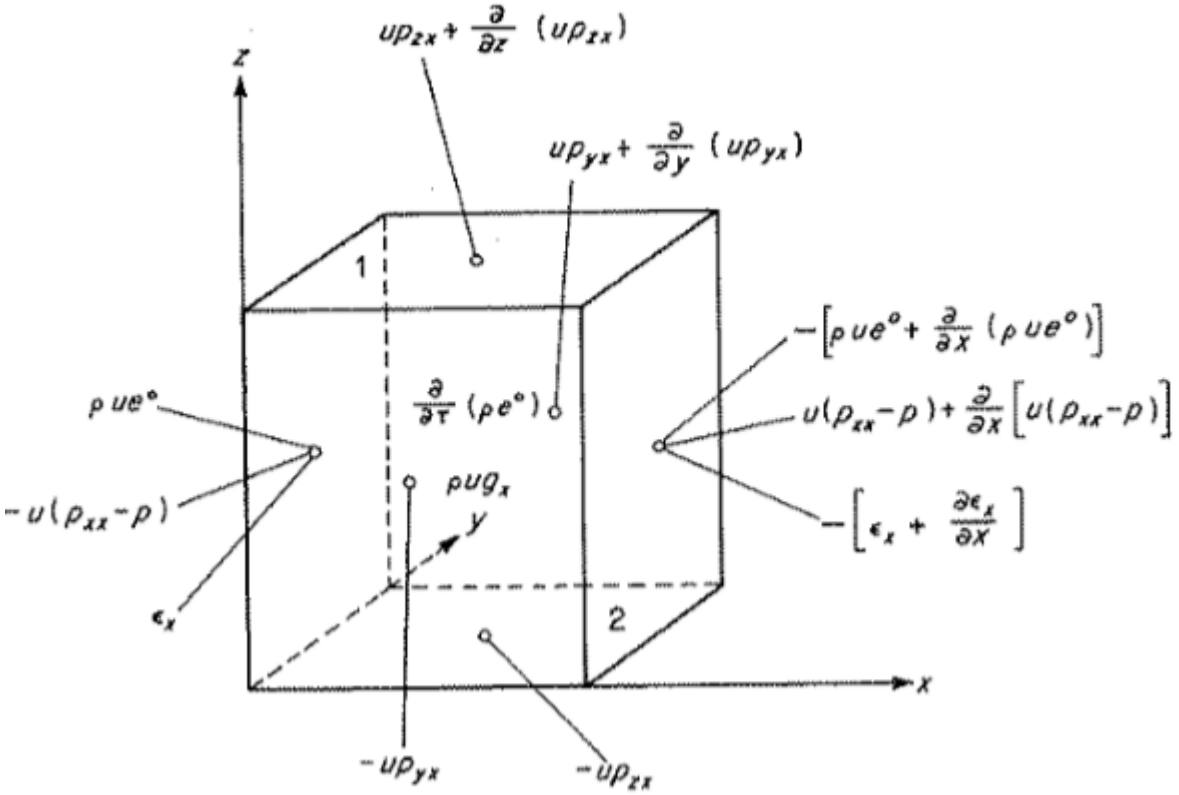
$$\rho\frac{Dw}{dt}=-\frac{dP}{dz}+2\frac{d}{dz}\left(\mu\frac{\partial w}{\partial y}\right)+\frac{d}{dx}\left(\mu\left(\frac{\partial w}{\partial x}+\frac{\partial u}{\partial z}\right)\right)+\frac{d}{dy}\left(\mu\left(\frac{\partial w}{\partial y}+\frac{\partial v}{\partial z}\right)\right)-\frac{d}{dz}\left(\mu'\left(\frac{\partial u}{dx}+\frac{\partial v}{dy}+\frac{\partial w}{dz}\right)\right)+\rho g_z$$

11.3 CONSERVATION OF ENERGY

The first law of thermodynamics (Energy Equation) can be considered as:

$$dW+dE_c+dE_M=dE_i$$

dW indicated work transfer in control volume (Energy transfer caused by forces acting on the control volume). dE_c is the energy transported with the fluids which crosses the boundaries. dE_M indicates all the other forms of energy transport across boundaries caused by molecular movements such as heat conduction.



Surface forces acting on x direction

$$\vec{F}_x=(P_{xx}-P)dA_x+P_{xy}dA_x+P_{xz}dA_x$$

$$\vec{F}_{x+dx}=\left[(P_{xx}-P)dA_x+\frac{d}{dx}(P_{xx}-P)dx\right]dA_x+\left[P_{xy}+\frac{d}{dy}(P_{xz})dx\right]dA_x+\left[P_{xz}+\frac{d}{dz}(P_{xz})dx\right]dA_x$$

$$\sum F_x=\vec{F}_x-\vec{F}_{x+dx}-mg_x\vec{\imath}$$

$$\sum F_x=-\left(\frac{d(P_{xx}-P)}{dx}dxdydz\right)\vec{\imath}-\left(\frac{dP_{xy}}{dy}dxdydz\right)\vec{\jmath}-\left(\frac{dP_{xz}}{dz}dxdydz\right)\vec{k}-\rho g_x dxdydz\vec{\imath}$$

$$\sum F_x = -\left[\left(\frac{d(P_{xx}-P)}{dx}\right)\vec{i} + \left(\frac{dP_{xy}}{dy}\right)\vec{j} + \left(\frac{dP_{xz}}{dz}\right)\vec{k} + \rho g_x \vec{i}\right] dxdydz$$

Work component in x direction then

$$W_x = -u(P_{xx}-P)dA_x - uP_{xy}dA_y - uP_{xz}dA_z$$

$$W_{x+dx} = \left[-u(P_{xx}-P) + \frac{d}{dx}\left(-u(P_{xx}-P)\right)\right]dA_x + \left[-uP_{xy} + \frac{d}{dy}\left(-uP_{xy}\right)\right]dA_y + \left[-uP_{xz} + \frac{d}{dy}\left(-uP_{xz}\right)\right]dA_z$$

$$\sum W_x = \left[\frac{d}{dx}\left(-u(P_{xx}-P)\right)\vec{i} + \frac{d}{dy}\left(-uP_{xy}\right)\vec{j} + \frac{d}{dy}\left(-uP_{xz}\right)\vec{k}\right] dxdydz + \rho u g_x dxdydz\,\vec{i}$$

$$\sum W_x = W_x - W_{x+dx} + umg_x\vec{i}$$

E_c component of energy transport in x direction

$$E_{cx} = (\rho u e^0)\, dA_x$$

$$E_{cx+dx} = \left(E_{cx} + \frac{dE_{cx}}{dx}dx\right) dA_x$$

$$\sum E_{cx} = E_{cx} - E_{cx+dx}) = -\frac{d(\rho u e^0)}{dx}dxdydz$$

E_M component of energy transport in x direction

$$E_{Mx} = (\varepsilon_x)\, dA_x$$

$$E_{Mx+dx} = \left(E_{Mx} + \frac{dE_{Mx}}{dx}dx\right) dA_x$$

$$\sum E_{Mx} = E_{Mx} - E_{Mx+dx}) = -\frac{d(\varepsilon_x)}{dx}dxdydz$$

E_i component of energy transport

$$dE_i = \frac{d}{dt}(\rho e^0)dxdydz$$

When all x,y,z and t component terms put together as one equation:

$$\left[\frac{d}{dx}\left(-u(P_{xx}-P)\right)\vec{i} + \frac{d}{dy}\left(-uP_{xy}\right)\vec{j} + \frac{d}{dy}\left(-uP_{xz}\right)\vec{k}\right] dxdydz + \rho u g_x dxdydz\vec{i} +$$
$$\left[\frac{d}{dy}\left(-v\left(P_{yy}-P\right)\right)j + \frac{d}{dx}\left(-uP_{yx}\right)i + \frac{d}{dz}\left(-uP_{yz}\right)\vec{k}\right] dxdydz + \rho v g_y dxdydz\vec{j} +$$
$$\left[\frac{d}{dz}\left(-w(P_{zz}-P)\right)\vec{k} + \frac{d}{dx}\left(-uP_{zx}\right)\vec{i} + \frac{d}{dz}\left(-vP_{zy}\right)\vec{j}\right] dxdydz + \rho w g_z dxdydz\vec{j} +$$
$$-\frac{d(\rho u e^0)}{dx}dxdydz - \frac{d(\rho v e^0)}{dy}dxdydz - \frac{d(\rho u e^0)}{dz}dxdydz$$
$$-\frac{d(\varepsilon_x)}{dx}dxdydz - \frac{d\left(\varepsilon_y\right)}{dy}dxdydz - \frac{d(\varepsilon_z)}{dz}dxdydz$$

$$+\frac{d}{dt}(\rho e^0)dxdydz = 0$$

$$\left[\frac{d}{dx}\left(-u(P_{xx}-P)\right)\vec{i}+\frac{d}{dy}\left(-uP_{xy}\right)\vec{j}+\frac{d}{dy}\left(-uP_{xz}\right)\vec{k}\right]+\rho u g_x\vec{i}+\left[\frac{d}{dy}\left(-v\left(P_{yy}-P\right)\right)j+\frac{d}{dx}\left(-uP_{yx}\right)i+\frac{d}{dz}\left(-uP_{yz}\right)\vec{k}\right]+$$
$$\rho v g_y\vec{j}+\left[\frac{d}{dz}\left(-w(P_{zz}-P)\right)\vec{k}+\frac{d}{dx}\left(-uP_{zx}\right)\vec{i}+\frac{d}{dz}\left(-vP_{zy}\right)\vec{j}\right]+\rho w g_z\vec{j}+-\frac{d(\rho u e^0)}{dx}-\frac{d(\rho v e^0)}{dy}-\frac{d(\rho u e^0)}{dz}-\frac{d(\varepsilon_x)}{dx}-\frac{d(\varepsilon_y)}{dy}-$$
$$\frac{d(\varepsilon_z)}{dz}+\frac{d}{dt}(\rho e^0)=0$$

$$\frac{d}{dt}(\rho e^0)+\frac{d(\rho u e^0)}{dx}+\frac{d(\rho v e^0)}{dy}+\frac{d(\rho u e^0)}{dz}+\frac{d(\varepsilon_x)}{dx}-\frac{d\left(\varepsilon_y\right)}{dy}-\frac{d(\varepsilon_z)}{dz}+\frac{duP}{dx}+\frac{dvP}{dy}+\frac{dwP}{dz}=$$
$$\rho u g_x+\rho v g_y+\rho w g_z+\frac{duP_{xx}}{dx}+\frac{duP_{xy}}{dy}+\frac{duP_{xz}}{dz}+\frac{dvP_{yx}}{dx}+\frac{dvP_{yy}}{dy}+\frac{dvP_{yz}}{dz}+\frac{dwP_{zx}}{dx}+\frac{dwP_{zy}}{dy}+\frac{dwP_{zz}}{dz}$$

In vectorel notation:

$$\frac{d}{dt}(\rho e^0)+\frac{d(\rho v_i e^0)}{dx_i}+\frac{d(\varepsilon_i)}{dx_i}+\frac{d(v_i P)}{dx_i}=\frac{d\left(v_i P_{ji}\right)}{dx_j}+\rho v_i g_i$$

Partial derivative:

$$\frac{d}{dt}(\rho e^0)+\frac{d(\rho v_i e^0)}{dx_i}=\rho\frac{d}{dt}(e^0)+\rho v_i\frac{d(e^0)}{dx_i}+e^0\left[\frac{d}{dt}(\rho)+\frac{d(\rho v_i)}{dx_i}\right]$$

The second part of the equation is conservation of mass therefore:

$$\frac{d}{dt}(\rho e^0)+\frac{d(\rho v_i e^0)}{dx_i}=\rho\frac{d}{dt}(e^0)+\rho v_i\frac{d(e^0)}{dx_i}=\rho\frac{De^0}{dt}$$

The energy balance takes the form

$$\rho\frac{De^0}{dt}+\frac{d(\varepsilon_i)}{dx_i}+\frac{d(v_i P)}{dx_i}=\frac{d\left(v_i P_{ji}\right)}{dx_j}+\rho v_i g_i$$

Total internal energy can be relaced with total enthalpy

$$e^0=h-P/\rho$$
$$\rho\frac{De^0}{dt}=\rho\frac{Dh^0}{dt}-\frac{DP}{dt}+\frac{P}{\rho}\frac{D\rho}{dt}$$

From conservation of mass:

$$\frac{D\rho}{dt}+\rho\frac{d(v_i)}{dx_i}=0$$
$$\frac{P}{\rho}\frac{D\rho}{dt}=\frac{P}{\rho}\left[-\rho\frac{d(v_i)}{dx_i}\right]=-P\frac{d(v_i)}{dx_i}$$
$$\rho\frac{De^0}{dt}=\rho\frac{Dh^0}{dt}-\frac{DP}{dt}+\frac{P}{\rho}\frac{D\rho}{dt}=\rho\frac{Dh^0}{dt}-\frac{dP}{dt}-v_i\frac{dP}{dx_i}-P\frac{dv_i}{dx_j}=\rho\frac{Dh^0}{dt}-\frac{dP}{dt}-\frac{dv_iP}{dx_i}$$

Introducing this term into the energy equation gives

$$\rho\frac{Dh^0}{dt}-\frac{DP}{dt}-\frac{dv_iP}{dx_i}+\frac{d(\varepsilon_i)}{dx_i}+\frac{dv_iP}{dx_i}=\frac{d\left(v_iP_{ji}\right)}{dx_j}+\rho v_i g_i$$

$$\rho\frac{Dh^0}{dt}-\frac{DP}{dt}+\frac{d(\varepsilon_i)}{dx_i}=\frac{d\left(v_iP_{ji}\right)}{dx_j}+\rho v_i g_i$$

$$\frac{Dh^0}{dt}-\frac{DP}{dt}+\frac{d(\varepsilon_i)}{dx_i}=v_i\frac{d\left(P_{ji}\right)}{dx_j}+P_{ji}\frac{d(v_i)}{dx_j}+\rho v_i g_i$$

Momentum equation was in the form of

$$\rho\frac{Dv_i}{dt}=-\frac{dP}{dx_i}+\frac{dP_{ji}}{dx_j}+\rho g_i$$

Multiplying these equation by v_i

$$\rho v_i\frac{Dv_i}{dt}+v_i\frac{dP}{dx_i}=v_i\frac{dP_{ji}}{dx_j}+\rho v_i g_i$$

$$\rho \frac{DV^2/2}{dt} + v_i \frac{dP}{dx_i} = v_i \frac{dP_{ji}}{dx_j} + \rho v_i g_i$$

Note that

$$h^0 = h + \frac{V^2}{2}$$

$$\frac{Dh^0}{dt} = \frac{Dh}{dt} + \frac{DV^2/2}{dt}$$

Substracting these from energy equation

$$\rho \frac{Dh^0}{dt} - \frac{dP}{dt} + \frac{d}{dx_i}(\varepsilon_i) - \rho \frac{DV^2/2}{dt} - v_i \frac{dP}{dx_i} = v_i \frac{dP_{ji}}{dx_j} +_i P_{ji} \frac{dv_i}{dx_j} + \rho v_i g_i - v_i \frac{dP_{ji}}{dx_j} - \rho v_i g_i$$

$$\rho \frac{Dh}{dt} - \frac{DP}{dt} + \frac{d}{dx_i}(\varepsilon_i) = P_{ji} \frac{dv_i}{dx_j}$$

A simpler from of the energy equation is created. The term $\Phi = P_{ji} \frac{dv_i}{dx_j}$ is called the dissipation function. For the Newtonian fluid

$$\Phi = P_{ji} \frac{dv_i}{dx_j} = \left(\mu \left[\frac{dv_i}{dx_j} + \frac{dv_j}{dx_i} \right] - \mu' \left[\frac{dv_k}{dx_k} \delta_{ij} \right] \right) \frac{dv_i}{dx_j}$$

Remembering also

$h = e + \frac{P}{\rho}$ and conservation of mass gives: $\frac{D\rho}{dt} = -\rho \frac{dv_i}{dx_i}$

$$\frac{Dh}{dt} = \frac{De}{dt} + \frac{1}{\rho}\frac{DP}{dt} - \frac{P}{\rho^2}\frac{D\rho}{dt} = \frac{De}{dt} + \frac{1}{\rho}\frac{DP}{dt} + \frac{P}{\rho}\frac{dv_i}{dx_i}$$

Substituting these into the energy equation gives:

$$\rho \left(\frac{De}{dt} + \frac{1}{\rho}\frac{DP}{dt} + \frac{P}{\rho}\frac{dv_i}{dx_i} \right) - \frac{DP}{dt} + \frac{d}{dx_i}(\varepsilon_i) = P_{ji} \frac{dv_i}{dx_j}$$

$$\rho \frac{De}{dt} + P \frac{dv_i}{dx_i} + \frac{d}{dx_i}(\varepsilon_i) = P_{ji} \frac{dv_i}{dx_j}$$

For a one-component fluid, the only mode of energy transfer which has to be considered in the term dE_M is the conduction heat transfer and it can be expressed by using Fourier's law of conduction

$\varepsilon_i = -$ so the term $\frac{d}{dx_i}(\varepsilon_i) = -\frac{d}{dx_i}\left(k_i \frac{dT}{dx_i}\right)$

$$\rho \frac{De}{dt} + P\frac{dv_i}{dx_i} - \frac{d}{dx_i}\left(k_i \frac{dT}{dx_i}\right) = P_{ji}\frac{dv_i}{dx_j}$$

11.4 TURBULENT FLOW EQUATIONS

Turbulent flow consist of irregular random motions in addition to mean motion. Navier-Stokes equations could have been used to solve all the complex motion and beheviour in turbulent flow, yet even for the most advanced computers available to us, it is an impractible afford to pursue. Instead we look at time average quantities in Turbulent flow. In 1895 Reynolds described. Instantaneous flow parameters are written as the sum of the time-averaged values and instantaneous deviations from these values.

$v_i = \bar{v}_i + v'_i$

$P = \bar{P} + P'$

It follows from the definition of the fluctuation quantities that their time avarage values are equal to zero

$\overline{v'_i} = 0 \quad \overline{P'} = 0$

Bu time averaged product values are not equal to zero

$\overline{v'_\iota v'_J} \neq 0$

In addition the following rule applies

$\frac{\overline{\partial v_\iota}}{\overline{\partial x_J}} = \frac{\partial \bar{v}_i}{\partial x_j}$

The continuity equation for a constant property fluid

$\frac{\overline{\partial v_\iota}}{\partial x_i} + \frac{\partial v'_i}{\partial x_i} = 0$ Time averaged form of the equation:

$\frac{\partial \bar{v}_i}{\partial x_i} + \frac{\partial \overline{v'_i}}{\partial x_i} = 0$ According to the definition the terms $\frac{\partial \overline{v'_i}}{\partial x_i} = 0$ so the continuity equation

$\frac{\partial \bar{v}_i}{\partial x_i} = 0$ will be same as the laminar version

The same procesure now can be applied to momentum equation for two dimensional flow

$$\rho\left(\frac{\partial \bar{u}}{\partial t} + \bar{u}\frac{\partial \bar{u}}{\partial x} + \bar{v}\frac{\partial \bar{u}}{\partial y} + \frac{\overleftarrow{\partial \overline{u'^2}}}{\partial x} + \frac{\partial \overline{u'v'}}{\partial y}\right) = -\frac{\partial \bar{P}}{\partial x} + \mu\frac{\partial^2 \bar{u}}{\partial y^2} \quad \text{(x direction)}$$

$$\rho\left(\frac{\overleftarrow{\partial \overline{v'^2}}}{\partial y}\right) = -\frac{\partial \bar{P}}{\partial x}$$

The y direction equation can be integrated to give:

$\bar{P} + \rho\overline{v'^2} =$ when the turblent flactuations are zero or negligable. Introducing P_s to x direction momentum equation gives

$$\rho\left(\frac{\partial \bar{u}}{\partial t} + \bar{u}\frac{\partial \bar{u}}{\partial x} + \bar{v}\frac{\partial \bar{u}}{\partial y}\right) = -\frac{\partial P_s}{\partial x} + \mu\frac{\partial^2 \bar{u}}{\partial y^2} - \rho\frac{\partial \overline{u'v'}}{\partial y}$$

The term $-\rho\frac{\partial(\overline{u'^2} - \overline{v'^2})}{\partial x}$ has been neglected due tosmaller order of magnitudes in normal boundary layers.

This equation assumes that variation of the velocity u is such that a meaningfull separation into a time-averaged and fluctuating contribution is possible. So the basic problem in the analysis of turbulent flow can be reduced to find a relation to connect turbulent stresses with a mean flow parameters. For boundary layersthis means finding a relation between turbulent shear $\tau_{turbulent} = -\rho u'v'$

and time mean values of velocity and presure. A general means of such relations can not be found so far, but several approximations, mostly based on experimental data is made. Some of them as follows
Boussinesq(1877)[86] wrote an analogy to Newton's law

$\tau_{turbulent} = \rho\varepsilon_m \frac{\partial \bar{u}}{\partial y}$

The term ε_m is called turbulent difisuvity of momentum. Yet actual relation of ε_m and average velocity gradient was not clearly defined. Prantl[87] introduced a possile relation between this two parameters.

$\varepsilon_m = l^2 \left|\frac{\partial \bar{u}}{\partial y}\right|$ The velocity gradient was multiplied by the square of length. This length l prandtl called the mixing length, and he interpreted it as the length which fluid particles travel in turbulent mixing process. He suggested from the results of his experimental data l=0.4y.

Karman[88] obtained another relation

$\varepsilon_m = \kappa^2 \frac{\left(\frac{\partial \bar{u}}{\partial y}\right)^3}{\left(\frac{\partial^2 \bar{u}}{\partial y^2}\right)^2}$ considering $\tau_w = \rho\varepsilon_m \frac{\partial \bar{u}}{\partial y}$ $\tau_w = \rho\kappa^2 \frac{\left(\frac{\partial \bar{u}}{\partial y}\right)^4}{\left(\frac{\partial^2 \bar{u}}{\partial y^2}\right)^2}$ assuming $z = \left(\frac{\partial \bar{u}}{\partial y}\right)$

$\tau_w = \rho\kappa^2 \frac{z^4}{\left(\frac{\partial z}{\partial y}\right)^2}$ $\left(\frac{\partial z}{\partial y}\right)^2 = \frac{\kappa^2}{\tau_w/\rho} z^4$ $\left(\frac{\partial z}{\partial y}\right) = \pm\frac{\kappa}{\sqrt{\tau_w/\rho}} z^2$ $\left(\frac{dz}{z^2}\right) = \pm\frac{\kappa}{\sqrt{\tau_w/\rho}} dy$ $-\frac{1}{z} = \pm\frac{\kappa}{\sqrt{\tau_w/\rho}} y + C$

Assume C=0 and reconsider $z = \left(\frac{\partial \bar{u}}{\partial y}\right)$

$\frac{\sqrt{\tau_w/\rho}}{\kappa}\frac{1}{y} = \frac{\partial \bar{u}}{\partial y}$ $\frac{\kappa}{\sqrt{\tau_w/\rho}} d\bar{u} = \frac{dy}{y}$ $\ln(y) + C' = \frac{\kappa}{\sqrt{\tau_w/\rho}}\bar{u}$ $\frac{\bar{u}}{\sqrt{\tau_w/\rho}} = \frac{1}{\kappa}\ln(y) + C''$ $u^+ = \frac{\bar{u}}{\sqrt{\tau_w/\rho}}$

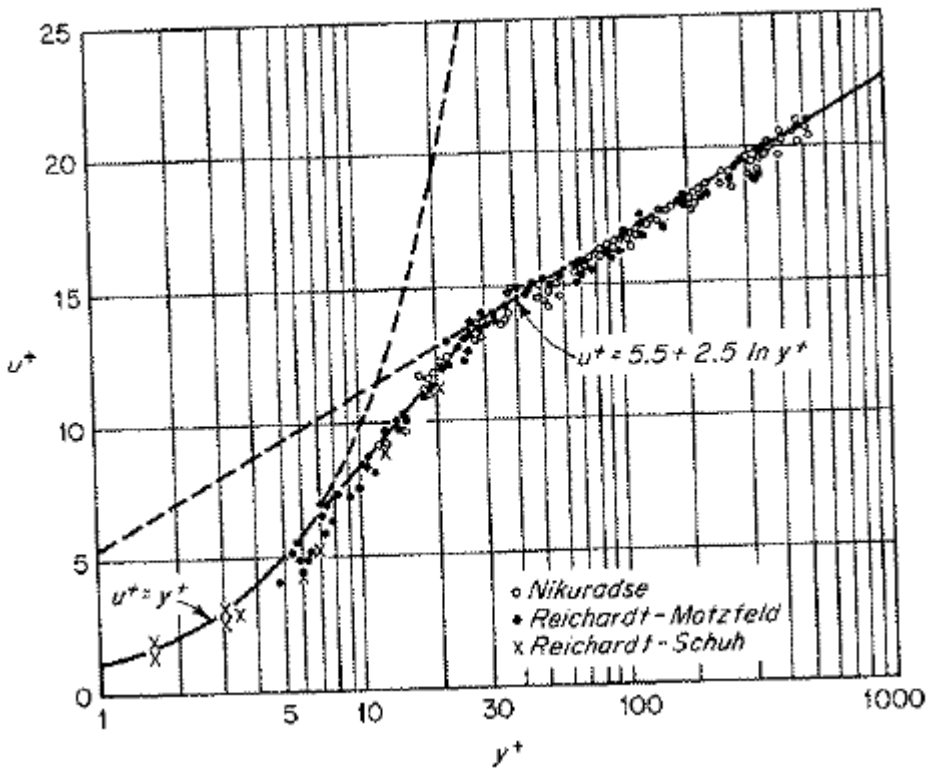
$u^* = \sqrt{\tau_w/\rho}$ is called shear velocity. Normalising of y value $y^+ = y\frac{\sqrt{\tau_w/\rho}}{\nu}$ so equation becomes

$$u^+ = \frac{1}{\kappa}\ln(y^+) + C$$

Karman suggested $\kappa = 0.4$ and C=5.5 Experiment shows for near the wall

$u^+ = y^+$ for $y^* < 30$

$u^+ = \frac{1}{\kappa}\ln(y^+) + C \ \ for \ \ y^* \geq 30$



Deisler[90] gives a relation for small distances as

$$l = \left(1 + \frac{\varepsilon_m}{\nu}\right)\frac{du^+}{dy^+}$$

$$\varepsilon_m = n^2 u^+ y^+ [1 - \exp(n^2 u^+ y^+) \ ]$$

Deisler found experimentally that best fit value fort he experiments for n=0.125 and limit value that equation holds is $y^+ = 26$

This results in $u^+ = 2.78\ln(y^+) + 3.8$

Van Driest[91] developed a smilar equation as

$$\varepsilon_m = K^2 (y^+)^2 \left[1 - \exp\left(\frac{y^+}{A^+}\right) \ \right]^2 \left|\frac{du^+}{dy^+}\right|$$

With the constants K=0.4 and $A^+ = 26$ (determined from experiments)

These equations are valid fort he flow close to the Wall . This is approximately 2/10ths to 3/10ths of the boundary layer.

The avilability of computers with high speed has sparked development of new approaches in which turbulent terms are described by differential equations similar to the conservation equations. One of these equations is a –averaged turbulent energy $k = v_i'^2/2$, an equation proposed by Prandtl, but not used then because of our inability to solve the equation. Energy generated per unit mass and time

$e_1 = \frac{1}{\rho}\tau_l \frac{du^+}{dy^+}$ where turbulent shear: $\tau_l = \rho \varepsilon_m \frac{du^+}{dy^+}$ turbuleny diffusivity: $\varepsilon_m = C_m \sqrt{k} L$

$$e_1 = C_m \sqrt{k} L \left(\frac{d\bar{u}}{dy}\right)^2$$

The distruction of turbulance by the resistance of the turbulance lumpsin the fluctuating velocity field. With L expressing scale of the tubulance (the size of turbulance lumps) the resistance W can be written as

$W = \rho C_D L^2 u'^2$ The work caused by this resistance per unit time and mass is then

$$e_2 = C_D \frac{u'^2}{L} = C_D \frac{k^{2/3}}{L}$$

Convection and diffusion of turbulent energy also have to be considered. The loss of turbulent energy in control volume by diffusion can be expressed in the form:

$$e_3 = \frac{\partial}{\partial y}\left(\varepsilon_m \frac{dk}{dy}\right) = \frac{\partial}{\partial y}\left(C_m \sqrt{k} L \frac{dk}{dy}\right)$$

Which is analogous to the equation describing laminar heat conduction. With this equation the turbulent energy equation takes the form

$$\frac{Dk}{dt} = C_m \frac{\partial}{\partial y}\left(\sqrt{k} L \frac{dk}{dy}\right) + C_m \sqrt{k} L \left(\frac{\partial \bar{u}}{\partial y}\right)^2 - C_D \frac{k^{3/2}}{L}$$

Another differential equation to be introduced fort he turbulance scale L or some parameter describing this scale. In 1951 Rotta developed an equation for the product kL

$$\frac{D(kL)}{dt} = \frac{\partial}{\partial y}\left\{\frac{\varepsilon_m}{\sigma kL}\left[\frac{d(kL)}{dy} + C_{kL} L \frac{dk}{dy}\right]\right\} + C_{B\varepsilon_m} L \left(\frac{\partial \bar{u}}{\partial y}\right)^2 - C_S k^{\frac{3}{2}} - C_w \left(\frac{L}{y}\right)^q \ k^{\frac{3}{2}}$$

This equation will be solved together with the other navier stokes equations.

$$\frac{\partial \bar{u}}{\partial x} + \frac{\partial \bar{v}}{\partial y} = 0$$

$$\rho \frac{D\bar{u}}{\partial t} = \rho\left(\frac{\partial \bar{u}}{\partial t} + \bar{u}\frac{\partial \bar{u}}{\partial x} + \bar{v}\frac{\partial \bar{u}}{\partial y}\right) = -\frac{\partial \bar{P}}{\partial x} + +\rho \frac{\partial}{\partial y}\left(\varepsilon_m \frac{d\bar{u}}{dy}\right)$$

Coefficients for different geometries could be different. Coefficients for boundary layers:

$C_m = 1$

$C_D = 0.1$

$C_s = 0.055$

$C_B = 0.84$

$C_{kl} = -0.4$
$C_w = 2.2$
$\sigma_k = 2$
$\sigma_{kL} = -0.4$
Turbulence parameter kL is not the only one that can be used. Various combination $k^m L^n$ can be considered

11.5 PROPERTIES OF GASES AND LIQUIDS

11.5 VISCOSITY

Informally, viscosity is the quantity that describes a fluid's resistance to flow. Fluids resist the relative motion of immersed objects through them as well as to the motion of layers with differing velocities within them. Formally, *viscosity* (represented by the symbol η "eta") is the ratio of the *shearing stress* (*F*/*A*) to the *velocity gradient* $\left(\frac{dV_x}{dz}\right)$ in a fluid.

$$\eta = \frac{F/A}{\left(\frac{dV_x}{dz}\right)} = \frac{\sigma}{\left(\frac{dV_x}{dz}\right)}$$

The more usual form of this relationship, called *Newton's equation*, states that the resulting shear of a fluid is directly proportional to the force applied and inversely proportional to its viscosity.

$$\frac{F}{A} = \sigma = \eta\left(\frac{dV_x}{dz}\right)$$

The SI unit of viscosity is the *pascal second* [Pa s],

There are actually two quantities that are called viscosity. The quantity defined above is sometimes called *dynamic viscosity*, *absolute viscosity*, or *simple viscosity* to distinguish it from the other quantity, but is usually just called viscosity. The other quantity called *kinematic viscosity* (represented by the Greek letter ν "nu") is the ratio of the viscosity of a fluid to its density.

$$\nu = \frac{\eta}{\rho}$$

The SI unit of kinematic viscosity is the *square meter per second* [m^2/s]

VISCOSITY OF LOW PRESSURE GASES [130][131]

Basic low pressure viscosity definitions are based on the quantum mechanics theories about molecular collisions

Chapman-Enskog[129] given the following equation for viscosity

$$\eta = \frac{26.69x10^{-7}(MT)^{1/2}}{\sigma^2\Omega_v}$$

Where σ (A=10^{-10} m) is hard-sphere diameter of the molecule, M molecular weight kg/kmol, Ω_v collision integral, which can be calculated as

$\Omega_v = 1.16145(T^*)^{-0.14874} + 0.52487\exp(-0.77320T^*) + 2.16178\exp(-2.43787T^*)$ where

$T^* = \frac{kT}{\epsilon}$ where k is Boltzmann's constant. The value of $\frac{k}{\epsilon}$ and σ is given in the table below for some gases.

Substance		σ A	ε/k K
Ar	Argon	3.542	93.3
He	Helium	2.551	10.22
Kr	Krypton	3.655	178.9
Ne	Neon	2.82	32.8

Xe	Xenon	4.047	231
Air	Air	3.711	78.6
CCl4	Carbontetracholoride	5.947	322.7
CH4	Methane	3.758	148.6
CO	Carbonmonoxide	3.69	336
CO2	Carbondioxide	3.941	195.2
C2H2	Ethylene	4.163	224.7
C2H6	Ethane	4.443	224.7
C2H5OH	Ethanol	4.53	362.6
CH5OCH5	Methylether	4.307	395
C3H8	Propane	5.118	237.1
n-C4H10	n-butane	4.687	531.4
iso-C4H10	iso-butane	5.278	330.1
n-C5H12	n-Pentane	5.784	341.1
C6H6	Benzene	5.349	412.3
C6H12	cyclohexane	6.182	287.1
n-C6H14	n-hexane	5.949	399.3
Cl2	Choline	4.217	316
F2	Florine	3.357	112.6
H2	Hydrogen	2.827	59.7
H2O	Water	2.641	809.1
H2O2	Hydrogenperoxide	4.196	289.3
H2S	Hydrogensulfide	3.623	301.1
Hg	Mercury	2.969	750
I2	Iodine	5.16	474.2
NH3	Ammonia	2.9	558.3
N2	Nitrogen	3.798	71.4
NO	nitricoxide	3.492	116.7
N2O	Nitricoxide	3.828	232.4
O2	Oxygen	3.467	106.7
SO2	sulfurdioxide	4.112	335.4

EXAMPLE : Calculate viscosity of methane at 300 ℃ , and compare it with the given data in class Gas.

```
 public class viscosityB
{
//Collusion integral
public static double omegav(double e_k,double T)
 {double Ts=T/e_k;
  double Ov=1.16145*Math.pow(Ts,-0.14874)+0.52487*Math.exp(-0.7732*Ts)+2.161178*Math.exp(-2.43787*Ts);
  return Ov;
 }

 //Chapman- Enskog
 //sigma m
  public static double visc(double M,double T,double sigma,double e_k)
 {  double Ov=omegav(e_k,T);
    double eta=26.69*1.0e-7*Math.sqrt(M*T)/(sigma*sigma*Ov);
    return eta;
 }

 public static void main(String arg[])
 {   //CH4 viscosity
  double M=16; //kg/kmol
  double T=300+273.15;  //Degree K
 double sigma=3.758;  //m
 double e_k=148.6;
 double eta=visc(M,T,sigma,e_k);
 Gas g=new Gas("ch4");
 double eta1=g.vis(T);
  System.out.println("viscosity = "+eta+" Pas");
  System.out.println("viscosity1 = "+eta1+" Pas");
  double error=(eta1-eta)/eta1*100;
  System.out.println("error = "+error+"%");
  }}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" viscosityB
viscosity = 1.8524950473508883E-5 Pas
viscosity1 = 1.8480925475941284E-5 Pas
error = -0.23821857636356744%

> Terminated with exit code 0.
```

For the case that data is not available, Chung at al. [127], [128]suggested that

$$\frac{\epsilon}{k} = \frac{T_c}{1.2503}$$

$T^* = 1.2593\frac{T}{T_c} = 1.2593T_r$ where T_c is critical temperature and T_r is reduced temperature.

$\Omega_v(T^*) = 1.16145(T^*)^{-0.14874} + 0.52487\exp(-0.77320T^*) + 2.16178\exp(-2.43787T^*)$

They also add a correction factor F_c into the equation and equation becomes:

$\eta = 40.785x10^{-7}\frac{F_c(MT)^{1/2}}{(V_c x10^3)^{2/3}\Omega_v}$ where η is viscosity in μP, M molecular weight kg/kmol, T temperature degree K, V_c critical volume m^3/kmole, $F_c = 1 - 0.2756\omega + 0.05903\mu_r^4 + \kappa$

In this last equation ω is called **Pitzer's accentric factor[130]**. This factor is calculated as

$$\omega = -log_{10}P_{saturated\ vapor}(at\ T_r = 0.7) - 1$$

To obtain values of ω, the reduced vapor pressure ($P_r = P/P_{crit}$) at $T_r = T/T_{crit} = 0.7$ is required. For example for water T_{crit}=647.3 K T=0.7*647.3=453.11 K=179.96 C so for this temperature P_v=10.01168 bar and P_{crit}=221.2 bar

$$P_r = P/P_{crit} = \frac{10.01168}{221.2} = 0.045261$$

$$\omega = -log_{10}P_{r\ saturated\ vapor}(0.045261) - 1 = 1.344278 - 1 = 0.344278$$

A list of Pitzer's acentric factor, critical properites etc. for some substances are given in the table below:

		molar mass	Boiling T	Critical T	Critical P	Critical comp.	Pitzer's acentric factor	specific heat
		M	Tb	Tc	Pc	Zc	ω	Cp
Substance	**Formula**	**kg/kmol**	**K**	**K**	**Mpa**			**J/kgK**
Acetone	C3H6O	58	329.2	508	4.7	0.233	0.309	1300
Acetylene	C2H2	26	189.5	309	6.2	0.271	0.184	1580
Air		29	82e	132g	3.75g	0.28g	0.035	1004
Ammonia	NH3	17	239.8	406	11.3	0.242	0.25	2200
Argon	Ar	40	87.4	151	4.86	0.291	0	523
Benzene	C6H6	78	353.3	563	4.92	0.271	0.212	1300
1,3-Butadiene	C4H6	54	268.5	425	4.33	0.27	0.193	1510
n-Butane	C4H10	58	272.6	425	3.8	0.274	0.193	1580
iso-Butane	C4H10	58	261.5	408	3.64	0.28	0.176	1580
Carbon dioxide	CO2	44	194.7f	304	7.38	0.274	0.225	840h
Carbon monoxide	CO	28	81.7	133	3.5	0.295	0.049	1100
Carbon tetracholoride	CCl4	154	349.7	556	4.56	0.272	0.194	862
Cyclohexane	C6H6	84	353.9	554	4.07	0.273	0.212	
n-Decane	C10H22	142	447.3	619	2.12	0.247	0.49	1680
n-Dodecane	C12H26	170	489.4	659	1.8	0.24	0.562	1690
DME(dimethylether)	C2H6O	46	250.6	400	5.37	0.271	0.274	1430
Ethane	C2H6	30	184.6	305	4.88	0.285	0.1	1700
Ethanol	C2H6O	46	351.5	516	6.39	0.248	0.635	1520

Ether(diethyl ether)	C4H10O	74	307.6	467	3.61	0.26	0.281	1600
ETBE(ethyl tert-butyl ether)	C6H14O	102	345	517	3.11	0.274	0.298	1550
Ethylene	C2H4	28	169.5	283	5.12	0.276	0.085	1470
Ethylene glycol	C2H6O2	62	471	645	7.53	0.268	1.137	1410
Helium(4 He)	He	4	4.2	5.3	0.23	0.301	-0.387	5190
Helium(3 He)	He	3	3.2	3.3	0.11	0.301	-0.46	
n-Heptane	C7H16	100	371	540	2.77	0.263	0.35	1650
n-Hexane	C6H14	86	342	508	3.03	0.263	0.296	1700
Hydrazine	N2H4	32	387	653	14.7	0.376	0.325	
Hydrogen	H2	2	20.1	33	1.32	0.305	-0.22	14200
(Hydrogen) Deuterium	D2	4	23.6	38	1.66	0.249	-0.16	14200
Mercuryi	Hg	201	630	736	104			
Methane	CH4	16	112	191	4.6	0.288	0.01	2180
Methanol	CH4O	32	338.1	513	8.08	0.224	0.559	1350
MTBE(methyl tert-butyl ether)	C5H12O	88	328	497	3.43	0.273	0.267	1500
Neon	Ne	20	26.2	44	2.7	0.301	0	1030
Nitrogen	N2	28	77.4	126	3.39	0.29	0.038	1040
Nitrogen dioxide	NO2	46	294.4	431	10.1	0.233		800
Nitrogen monooxide	NO	30	121.2	180	6.55	0.25	0.607	996
di-Nitrogen oxde	N2O	44	184.7	310	7.26	0.272	0.141	864
n-Octane	C8H18	114	399	569	2.49	0.259	0.394	1700
iso-Octane	C8H18	114	372	544	2.59	0.267	1650	
Ozone	O3	48	161.4	268	6.78	0.272		
Oxygen	O2	32	90.2	155	5.08	0.288	0.021	913
iso-Pentane	C5H12	72	301.3	461	3.33	0.268	0.227	1680
n-Pentane	C5H12	72	309.2	470	3.38	0.262	0.251	1680
Phenol	C6H6O	94	455	694	6.13	0.243	0.426	
Propane	C3H8	44	231.1	370	4.26	0.281	0.152	1570
iso-Propanol	C3H8O	60	355.4	508	4.76	0.248	0.669	1540
Propylene(propene)	C3H6	42	225.4	365	4.62	0.275	0.148	1460
Propylene glycol	C3H8O2	76	461.3	626	6.1	0.28	1.107	
R12(dichlorodifluoromethane)	CCl2F2	121	243	385	4.14	0.28	0.179	573
R134a(tetrafluoroethane)	CF3CH2F	102	246.6	374	4.07	0.258	0.33	840
R410Al	n.a.	73	221.8	345	4.9	0.271	0.296	820
Sulfurdioxide	SO2	64	263.2	430	7.87	0.264	0.251	607
Sulfurhexafluoride	SF6	146	204.9f	319	3.76	0.36	0.21	598
Toluene	C6F14	92	383.7	592	4.13	0.284	0.266	
Tetradecafluorohexane	hexafluoriden	338	329	449	1.83			
Uranium hexafluoride	UF6	352	329	503	4.6	0.282	0.092	370
Water	H2O	18	372.8	647.3	22.12	0.229	0.344	2050
Xenon	Xe	131	165	289.8	5.84	0.291	0	158

κ is a special correction factor for higly polar molecules such as alcohols and water. Some of the values are listed in the table below.

Compound	κ	compound	κ
Methanol	0.215	n-pentanol	0.122
Ethanol	0.175	n-hexanol	0.114
n-propanol	0.143	n-heptanol	0.109

i-propanol	0.143	Acetic acid	0.0916
n-butanol	0.132	Water	0.076
i- butanol	0.132		

μ_r is dimensionless dipole moment and μ is dipole moment in debyes.

$$\mu_r = 131.3 \frac{\mu}{(V_c T_c)^{1/2}}$$

Values of μ and other properties of gases are listed in the table below.

Thermophysical properties of some gases[130]

No	Formula	Name	MolWt	Tfp K	Tb K	Tc K	Pc kPa	Vc cm3/mole	Zc	ω	μ Dipm debby
0	Ar	argon	39.948	83.8	87.3	150.8	4870	74.9	0.29	0	0
1	NH3	ammonia	17.031	195.4	239.	405.5	1135	72.5	0.24	0.25	1.5
2	C6H5Cl	chlorobenzene	112.56	227.6	404.	632.4	4520	308	0.26	0.24	1.6
3	NO	nitricoxide	30.006	109.5	121.	180	6480	57.7	0.25	0.58	0.2
4	NO2	nitrogendioxide	46.006	261.9	294.	431	1010	167.8	0.47	0.83	0.4
5	N2	nitrogene	28.013	63.3	77.4	126.2	3390	89.8	0.29	0.03	0
6	N2O	nitrousoxide	44.013	182.3	184.	309.6	7240	97.4	0.27	0.16	0.2
7	O2	oxygen	31.999	54.4	90.2	154.6	5040	73.4	0.28	0.02	0
8	O3	ozone	47.998	80.5	181.	261.1	5570	88.9	0.22	0.69	0.6
9	H2	hydrogen	2.016	14	20.3	33	1290	64.3	0.30	-	0
10	H2O	water	18.015	273.2	373.	647.3	2212	57.1	0.23	0.33	1.8
11	CH4	methane	16.043	90.7	111.	190.4	4600	99.2	0.28	0.01	0
12	CH4O	methanol	32.042	175.5	337.	512.6	8090	118	0.22	0.55	1.7
13	C2H6	ethane	30.07	89.9	184.	305.4	4880	148.3	0.28	0.09	0
14	C3H8	propane	44.094	85.5	231.	369.8	4250	203	0.28	0.15	0
15	C3H6O	propionaldehyde	58.08	193	321	515.3	6330	0	0	0.31	2.7
16	C4H10	n-butane	58.124	134.8	272.	425.2	3800	255	0.27	0.19	0
17	C4H10	isobutane	58.124	113.6	261.	408.2	3650	263	0.28	0.18	0.1
18	C5H12	n-pentane	72.151	143.4	309.	469.7	3370	304	0.26	0.25	0
19	C6H6	benzene	78.114	278.7	353.	562.2	4890	259	0.27	0.21	0
20	C6H14	n-hexane	86.178	177.8	341.	507.5	3010	370	0.26	0.29	0
21	CCl2F2	dichlorodifloromethane	120.91	115.4	245.	385	4140	216.7	0.28	0.20	0.5
22	C7H16	n-heptane	100.21	182.6	371.	540.3	2740	27.4	432	0.26	0.349
23	C8H10	ethylbenzene	106.17	178.2	409.	617.2	3600	374	0.26	0.30	0.4
24	C8H18	n-octane	114.23	216.4	398.	568.8	2490	492	0.25	0.39	0
25	C9H20	n-nonane	128.26	219.7	424	594.6	2290	548	0.26	0.44	0
26	C10H8	naphtalene	128.17	353.5	491.	748.4	4050	413	0.26	0.30	0
27	C11H10	1-methylnaphtalene	142.2	242.7	517.	772	3600	462	0.23	0.31	0.5
28	C11H10	2-methylnaphtalene	142.2	307.7	514.	761	3500	462	0.26	0.38	0.4
29	C12H12	1-ethylnaphtalene	156.22	259	531.	774.9	3140	520	0.25	0.4	0
30	C13H14	1-propylnaphtalene	170.25	265	546	771.5	2760	580	0.25	0.49	0
31	C14H16	1-butylnaphtalene	184.28	253	562.	781.5	2500	630	0.24	0.53	0
32	C10H22	n-decane	142.29	243.5	447.	617.7	2120	603	0.24	0.48	0
33	C11H24	n-undecane	156.31	247.6	469.	638.8	1970	660	0.24	0.53	0
34	C12H26	n-dodecane	170.34	263.6	489.	658.2	1820	713	0.24	0.57	0
35	C13H28	n-tridecane	184.37	267.8	508.	676	1720	780	0.24	0.61	0
36	C14H30	n-tetradecane	198.39	279	526.	693	1440	830	0.23	0.58	0
37	C15H32	n-pentadecane	212.42	283	543.	707	1520	880	0.23	0.70	0
38	C16H34	n-hexadecane	226.45	291	560	722	1410	1030	0.24	0.74	0
39	C17H36	n-heptadecane	240.48	295	575.	733	1300	1000	0.22	0.77	0
40	C18H38	n-octadecane	254.5	301.3	589.	748	1200	1190	0.25	0.79	0
41	C19H40	n-nonadecane	268.53	305	603.	756	1110	1130	0.21	0.82	0
42	C20H42	n-eicosane	282.56	310	617	767	1110	1110	0.19	0.90	0
43	C3H6	propylene	42.081	87.9	225.	364.9	4600	181	0.27	0.14	0.4
44	C11H16	penthylbenzene	148.24	198	478.	679.9	2600	550	0.25	0.44	0

45	C12H18	hexylbenzene	162.27	212	499.	697.5	2380	620	0.25	0.48	0
46	C13H20	heptylbenzene	176.3	225	519.	713.5	2200	660	0.25	0.53	0
47	C14H22	octylbenzene	190.32	237	537.	728	2040	720	0.24	0.58	0
48	C15H24	nonylbenzene	204.25	249	555.	741	1900	790	0.24	0.63	0
49	C16H26	decylbenzene	218.38	259	571.	728	2040	720	0.24	0.58	0
50	C17H28	undecylbenzene	232.4	268	586.	764	1670	910	0.24	0.72	0
51	C18H30	dodecylbenzene	246.43	276	600.	774	1580	980	0.24	0.77	0
52	CO	carbonmonoxide	28.01	68.1	81.7	132.9	3500	93.2	0.29	0.06	0.1
53	CO2	carbondioxide	44.01	216.5	194.	304.1	7380	93.9	0.27	0.23	0
54	C8H18	2,3,3-trimethylpentane	114.23	172.5	387.	573.6	2820	455	0.26	0.29	0
55	C8H18	2,3,4-trimethylpentane	114.23	163.9	386.	566.4	2730	461	0.26	0.31	0
56	C8H18	2-methyl-3-ethylpentane	114.23	158.2	388.	567.1	2700	443	0.25	0.33	0
57	C8H18	3-methyl-3-ethylpentane	114.23	182.3	391.	576.6	2810	455	0.26	0.30	0
58	C6H11N	capronitrile	97.161	194	436.	622	3250	0	0	0.52	3.5
59	C6H12	cyclohexane	84.162	279.6	353.	553.5	4070	308	0.27	0.21	0.3
60	C6H12	methylcyclopentane	84.162	130.7	345	532.7	3780	319	0.27	0.23	0
61	C6H12	1-hexene	84.163	133.3	336.	504	3170	350	0.26	0.28	0.4
62	C6H12	2-hexene,cis	84.162	132	342	518	3280	351	0.27	0.25	0
63	C6H12	2-hexene,trans	84.162	140	341	516	3270	351	0.27	0.24	0
64	C6H12	3-hexene,cis	84.162	135.3	339.	517	3280	350	0.27	0.22	0.3
65	C6H12	3-hexene,trans	84.162	159.7	340.	519.9	3250	350	0.26	0.22	0
66	C6H12	2-methyl-2-pentene	84.162	138.1	340.	518	3280	351	0.27	0.22	0
67	C6H12	3-methyl-2-pentene,cis	84.162	138.3	340.	518	3280	351	0.27	0.26	0
68	C6H12	3-methyl-2-pentene,trans	84.162	134.7	343.	521	3290	350	0.27	0.20	0
69	C6H12	4-methyl-2-pentene,cis	84.162	139	329.	490	3040	360	0.27	0.29	0
70	C6H12	4-methyl-2-pentene,trans	84.162	132	331.	493	3040	360	0.27	0.29	0
71	C6H12	2,3-dimethyl-1-butene	84.162	115.9	328.	501	3240	343	0.27	0.22	0
72	C6H12	2,3-dimethyl-2-butene	84.162	198.9	364.	524	3360	351	0.27	0.23	0
73	C6H12	3,3-dimethyl-1-butene	84.162	158	314.	490	3250	340	0.27	0.12	0
74	C6H12O	cyclohexanol	100.16	298	434.	625	3750	0	0	0.52	1.7
75	C6H12O	ethylpropylketone	100.16	0	396.	582.8	3320	0	0	0.37	0
76	C6H12O	methylisobutylketone	100.16	189	389.	571	3270	0	0	0.38	2.8
77	C6H12O2	n-butylacetate	116.16	199.7	399.	579	3140	400	0.26	0.41	1.8
78	C6H12O2	isobutylacetate	116.16	174.3	389.	564	3020	414	0.26	0.45	1.9
79	C6H12O2	ethylbutyrate	116.16	180	394.	569	2960	421	0.26	0.46	1.8
80	C6H12O2	ethylisobutyrate	116.16	185	383.	555	2970	421	0.27	0.43	2.1
81	C6H12O2	n-propylpropionate	116.16	197.3	395.	571	3020	0	0	0	1.8
82	C6H14	2-methylpentane	86.178	119.5	333.	497.5	3010	367	0.26	0.27	0
83	C6H14	3-methylpentane	86.178	155	336.	504.5	3120	367	0.27	0.27	0
84	C6H14	2,2-dimethylbutane	86.178	173.3	322.	488.8	3080	359	0.27	0.23	0
85	C6H14	2,3-dimethylbutane	86.178	144.6	331.	500	3130	358	0.26	0.24	0
86	C6H14O	1-hexanol	102.18	229.2	430.	611	4050	381	0.3	0.56	1.8
87	C6H14O	ethylbutylether	102.18	170	365.	531	3040	390	0.27	0.4	1.2
88	C6H14O	dipropylether	102.18	151	363.	530.6	3030	0	0	0.36	1.2
89	C6H14O	diisopropylether	102.18	187.7	341.	500.3	2880	386	0.26	0.33	1.2
90	C6H15N	dipropylamine	101.19	233.6	382.	555.8	2990	0	0	0.47	1
91	C6H15N	diisopropylamine	101.19	212.2	357.	523.1	3020	0	0	0.36	1
92	C6H15N	triethylamine	101.19	158.4	362.	535	3030	389	0.26	0.32	0.9
93	C7F14	perfluoromethylcyclohexa	350.06	0	349.	486.8	2330	0	0	0.49	0
94	C7F16	perfluoro-n-heptane	388.05	195	355.	474.8	1620	664	0.27	0.55	0
95	C7H3F5	2,3,4,5,6-	182.09	0	390.	566.5	3130	384	0.25	0.41	0
96	C7H5N	benzonitrile	103.12	260	464.	699.4	4220	0	0	0.36	3.5
97	C7H6O	benzaldehyde	106.12	216	452.	694.8	4540	0	0	0.31	2.8
98	C7H6O2	benzoicacid	122.12	395.6	523	752	4560	341	0.25	0.62	1.7
99	C7H8	toluene	92.141	178	383.	591.8	4100	316	0.26	0.26	0.4
100	C7H8O	methylphenylether	108.14	235.7	426.	645.6	4250	0	0	0.34	1.2
101	C7H8O	benzylalcohol	108.14	257.8	478.	720.2	4400	0	0	0	1.7
102	C7H8O	o-cresol	108.14	304.1	464.	697.6	5010	0	0	0.43	1.6
103	C7H8O	m-cresol	108.14	285.4	475.	705.8	4560	309	0.24	0.45	1.8
104	C7H8O	p-cresol	108.14	307.9	475.	704.6	5150	0	0	0.50	1.6
105	C7H9N	n-methylaniline	107.16	216	469.	701	5200	0	0	0.47	1.7
106	C7H9N	o-toluidine	107.16	258.4	473.	694	3750	0	0	0.43	1.6

107	C7H9N	m-toluidine	107.16	242.8	476.	709	4150	0	0	0.41	1.5
108	C7H14	cycloheptane	98.189	265	391.	604.2	3810	353	0.26	0.23	0
109	C7H14	1,1-dimethylcyclopentane	98.189	203.4	361	547	3440	360	0.27	0.27	0
110	C7H14	1,2-	98.189	219.3	372.	564.8	3440	368	0.27	0.26	0
111	C7H14	1,2-	98.189	155.6	365	553.2	3440	362	0.27	0.26	0
112	C7H14	ethylcyclopentane	98.189	134.7	376.	569.5	3400	375	0.26	0.27	0
113	C7H14	methylcyclohexane	98.189	146.6	374.	572.2	3470	368	0.26	0.23	0
114	C7H14	1-heptene	98.189	154.3	366.	537.3	2830	440	0.28	0.35	0.3
115	C7H14	2,3,3-trimethyl-1-butene	98.189	163.3	351	533	2890	400	0.26	0.19	0
116	C7H14O2	n-propylbutyrate	130.19	176	416.	590	2710	0	0	0	1.8
117	C7H14O2	n-propylisobutyrate	130.19	0	408.	581	2830	0	0	0	0
118	C7H14O2	isobutylpropionate	130.19	201.8	410	583	2770	0	0	0	0
119	C7H16	3-methylhexane	100.21	100	365	535.3	2810	404	0.25	0.32	0
120	C7H16	2,2-dimethylpentane	100.21	149.4	352.	520.5	2770	416	0.26	0.28	0
121	C7H16	2,3-dimethylpentane	100.21	0	362.	537.4	2910	393	0.25	0.29	0
122	C7H16	2,4-dimethylpentane	100.21	154	353.	519.8	2740	418	0.26	0.30	0
123	C7H16	3,3-dimethylpentane	100.21	138.7	359.	536.4	2950	414	0.27	0.26	0
124	C7H16	3-ethylpentane	100.21	154.6	366.	540.6	2890	416	0.26	0.31	0
125	C7H16	2,2,3-trimethylbutane	100.21	248.3	354	531.2	2950	398	0.26	0.25	0
126	C7H16O	1-heptanol	116.2	239.2	449.	633	3040	435	0.25	0.56	1.7
127	C8H4O3	phthalicanhydride	148.12	404	560	810	4760	368	0.26	0	5.3
128	C8H8	styrene	104.15	242.5	418.	647	3990	0	0	0.25	0.1
129	C8H8O	methylphenylketone	120.15	292.8	474.	714	4060	376	0.25	0.42	3
130	C8H8O2	methylbenzoate	136.15	260.8	472.	692	3640	396	0.25	0.43	1.9
131	C8H10	o-xylene	106.17	248	417.	630.3	3730	369	0.26	0.31	0.5
132	C8H10	m-xylene	106.17	225.3	412.	617.1	3540	376	0.25	0.32	0.3
133	C8H10	p-xylene	106.17	286.4	411.	616.2	3510	379	0.26	0.32	0.1
134	C8H10O	ethylphenylether	122.17	243	443	647	3420	0	0	0.41	1.2
135	C8H11N	N,N-dimethylaniline	121.18	275.6	467.	687	3600	0	0	0.41	1.6
136	C8H16	1,1-dimethylcyclohexane	112.22	239.7	392.	591	2960	416	0.25	0.23	0
137	C8H16	1,2-dimethylcyclohexane-	112.22	223.1	402.	606	2960	0	0	0.23	0
138	C8H16	1,2-dimethylcyclohexane-	112.22	185	396.	596	2960	0	0	0.24	0
139	C8H16	1,3-dimethylcyclohexane-	112.22	197.6	393.	591	2960	0	0	0.22	0
140	C8H16	1,3-dimethylcyclohexane-	112.22	183	397.	598	2970	0	0	0.18	0
141	C8H16	1,4-dimethylcyclohexane-	112.22	185.7	397.	598	2970	0	0	0.23	0
142	C8H16	1,4-dimethylcyclohexane-	112.22	0	392.	587.7	2970	0	0	0.24	0
143	C8H16	ethylcyclohexane	112.22	161.8	404.	609	3000	450	0.27	0.24	0
144	C8H16	1,1,2-	112.22	0	386.	579.5	2940	0	0	0.25	0
145	C8H16	1,1,3-	112.22	0	378	569.5	2830	0	0	0.21	0
146	C8H16	1,2,4-	112.22	0	391	579	2900	0	0	0.27	0
147	C8H16	1,2,4-	112.22	0	382.	571	2800	0	0	0.24	0
148	C8H16	1-methyl-1-	112.22	0	394.	592	3000	0	0	0.25	0
149	C8H16	n-propylcyclopentane	112.22	155.8	404.	603	3000	425	0.25	0.33	0
150	C8H16	isopropylcyclopentane	112.22	160.5	399.	601	3000	0	0	0.24	0
151	C8H16	cyclooctane	112.22	287.6	422	647.2	3560	410	0.27	0.23	0
152	C8H16	1-octene	112.22	171.4	394.	566.7	2620	464	0.26	0.38	0.3
153	C8H16	2-octene-trans	112.22	185.4	398.	580	2770	0	0	0.35	0
154	C8H16O2	isobutylbutyrate	144.21	0	430.	603	2450	0	0	0	0
155	C8H16O2	isobutylisobutyrate	144.21	0	421.	594	2460	0	0	0	0
156	C8H16O2	n-propylisovalerate	144.21	0	429.	609	0	0	0	0	0
157	C8H18	2-methylheptane	114.23	164	390.	559.6	2480	488	0.26	0.37	0
158	C8H18	3-methylheptane	114.23	152.7	392.	563.7	2550	464	0.25	0.37	0
159	C8H18	4-methylheptane	114.23	152.2	390.	561.7	2540	476	0.25	0.37	0
160	C8H18	2,2-dimethylhexane	114.23	152	380	549.9	2530	478	0.26	0.33	0
161	C8H18	2,3-dimethylhexane	114.23	0	388.	563.5	2630	468	0.26	0.34	0
162	C8H18	2,4-dimethylhexane	114.23	0	382.	553.5	2560	472	0.26	0.34	0
163	C8H18	2,5-dimethylhexane	114.23	181.9	382.	550.1	2490	482	0.26	0.35	0
164	C8H18	3,3-dimethylhexane	114.23	147	385.	562	2650	443	0.25	0.32	0
165	C8H18	3,4-dimethylhexane	114.23	0	390.	568.9	2690	466	0.26	0.33	0
166	C8H18	3-ethylhexane	114.23	0	391.	565.5	2610	455	0.25	0.36	0
167	C8H18	2,2,3-trimethylpentane	114.23	160.9	383	563.5	2730	436	0.25	0.29	0
168	C8H18	2,2,4-trimethylpentane	114.23	165.8	372.	544	2570	468	0.26	0.30	0

169	C8H18	2,2,3,3-tetramethylbutane	114.23	374	379.	567.8	2870	461	0.28	0.25	0
170	C8H18O	1-octanol	130.23	257.7	468.	652.5	2860	490	0.25	0.58	2
171	C8H18O	2-octanol	130.23	241.2	452	637	2810	0	0	0	1.6
172	C8H18O	4-methyl-3-heptanol	130.23	0	443	623.5	2800	0	0	0	0
173	C8H18O	5-methyl-3-heptanol	130.23	0	445	621.2	0	0	0	0	0
174	C8H18O	2-ethyl-1-hexanol	130.23	203.2	457.	640.2	0	0	0	0	1.8
175	C8H18O	dibutylether	130.23	175	413.	580	2530	0	0	0.50	1.2
176	C8H18O	di-tert-butylether	130.23	0	382.	550	2420	0	0	0	0
177	C8H19N	dibutylamin	129.25	211	432.	607.5	2640	0	0	0.58	1.1
178	C8H19N	diisobutylamine	129.25	203	412.	584.4	2720	0	0	0.54	0
179	C9H7N	quinoline	129.16	258	510.	782	4500	402	0.27	0.31	0
180	C9H7N	isoquinoline	129.16	300	516.	803	4700	402	0.28	0	0
181	C9H10	indane	118.18	0	451.	684.9	3950	0	0	0.30	0
182	C9H10	alpha-methylstyrene	118.18	0	438.	654	3400	0	0	0	0
183	C9H10O2	ethylbenzoate	150.18	238.3	485.	668.7	2320	0	0	0.48	0
184	C9H12	n-propylbenzene	120.2	173.7	432.	638.2	3200	440	0.26	0.34	0
185	C9H12	isopropylbenzene	120.2	177.1	425.	631.1	3210	0	0	0.32	0
186	C9H12	1-methyl-3-ethylbenzene	120.2	177.6	434.	637	2840	490	0.26	0.36	0
187	C9H12	1-methyl-4-ethylbenzene	120.2	210.8	435.	640	2940	470	0.26	0.32	0
188	C9H12	1,2,3-trimethylbenzene	120.2	247.7	449.	664.5	3450	0	0	0.36	0
189	C9H12	1,2,4-trimethylbenzene	120.2	227	442.	649.2	3230	0	0	0.37	0
190	C9H12	1,3,5-trimethylbenzene	120.2	228.4	437.	637.3	3130	0	0	0.39	0.1
191	C9H13N	N,N-dimethyl-o-toluidine	135.21	212	467.	668	3120	0	0	0.48	0.9
192	C9H18	n-propylcyclohexane	126.24	178.7	429.	639	2800	0	0	0.25	0
193	C9H18	isopropylcyclohexane	126	243	183.	427.7	6400	28.3	0	0	0.237
194	C9H18	1,trans-3,5-	126.24	0	413.	602.2	2800	0	0	0	0
195	C9H18	1-nonene	126.24	191.8	420	592	2340	580	0.28	0.43	0
196	C9H18O	dibutylketone	142.24	267.3	461.	640	0	0	0	0	2.7
197	C9H20	2-methyloctane	128.24	192.8	416.	587	2310	0	0	0.42	0
198	C9H20	2,2-dimethylheptane	128.24	160	405.	576.8	2350	0	0	0.39	0
199	C9H20	2,2,3-trimethylhexane	128.26	0	406.	588	2490	0	0	0.33	0
200	C9H20	2,2,4-trimethylhexane	128.26	153	399.	573.7	2370	0	0	0.32	0
201	C9H20	2,2,5-trimethylhexane	128.26	167.4	397.	568	2330	519	0.26	0.35	0
202	C9H20	3,3-diethylpentane	128.26	240.1	419.	610	2670	0	0	0.33	0
203	C9H20	2,2,3,3-	128.26	263	413.	607.7	2740	0	0	0.30	0
204	C9H20	2,2,3,4-	128.26	152	406.	592.7	2600	0	0	0.31	0
205	C9H20	2,2,4,4-	128.26	206	395.	574.7	2490	0	0	0.31	0
206	C9H20	2,3,3,4-	128.26	171.1	414.	607.7	2720	0	0	0.31	0
207	C9H20O	1-nonanol	144.26	268	486.	671	0	546	0	0	1.7
208	C10F8	perfluoronapthalene	272.09	0	482	673.1	0	0	0	0	0
209	C10F18	perfluorodecalin	462.07	0	415	566	1520	0	0	0.39	0
210	C10H12	1,2,3,4-	132.21	242	480.	719	3510	0	0	0.30	0
211	C10H14	n-butylbenzene	134.22	185.2	456.	660.5	2890	497	0.26	0.39	0.4
212	C10H14	isobutylbenzene	134.22	221.7	445.	650	3140	480	0.28	0.38	0.3
213	C10H14	sec-butylbenzene	134.22	197.7	446.	664	2940	0	0	0.27	0.4
214	C10H14	tert-butylbenzene	134.22	215.3	442.	660	2960	0	0	0.26	0.5
215	C10H14	1-methyl-2-	134.22	0	451.	670	2890	0	0	0.27	0
216	C10H14	1-methyl-3-	134.22	0	448.	666	2930	0	0	0.27	0
217	C10H14	1-methyl-4-	134.22	200	450.	651	2730	0	0	0.27	0
218	C10H14	1,4-diethylbenzene	134.22	231	456.	657.9	2800	0	0	0.40	0.1
219	C10H14	1,2,3,5-	134.22	249	471.	679	0	0	0	0	0
220	C10H14	1,2,4,5-	134.21	352	470	675	2940	0	0	0.43	0
221	C10H14O	thymol	150.22	323	505.	698	0	0	0	0	0
222	C10H15N	n-butylaniline	149.24	259	513.	721	2830	0	0	0	0
223	C10H18	cis-decalin	138.25	230	468.	702.3	3200	0	0	0.28	0
224	C10H18	trans-decalin	138.25	242.8	460.	687.1	3140	0	0	0.27	0
225	C10H18	1,3-decadiene	138.25	0	442	615	0	0	0	0	0
226	C10H19N	caprylonitrile	153.27	255.3	516	622	3250	0	0	0	0
227	C10H20	butylcyclohexane	140.26	198.4	454.	667	3150	0	0	0.36	0
228	C10H20	isobutylcyclohexane	140.27	0	444.	659	3120	0	0	0.31	0
229	C10H20	sec-butylcyclohexane	140.27	0	452.	669	2670	0	0	0.26	0
230	C10H20	tert-butylcyclohexane	140.27	232	444.	659	2660	0	0	0.52	0

231	C10H20	1-decene	140.27	206.9	443.	615	2200	650	0.28	0.49	0
232	C10H20O	menthol	156.27	316	489.	694	0	0	0	0	0
233	C10H22	n-decane	144.29	243.5	447.	617.7	2120	603	0.24	0.48	0
234	C10H22	3,3,5-trimethylheptane	142.29	0	428.	609.7	2320	0	0	0.38	0
235	C10H22	2,2,3,3-tetramethylhexane	142.29	0	433.	623.2	2510	0	0	0.36	0
236	C10H22	2,2,5,5-tetramethylhexane	142.29	0	410.	581.6	2190	0	0	0.37	0
237	C10H22O	1-decanol	158.29	280.1	506.	687	2220	600	0.23	0	1.8
238	C11H14O	butylbenzoate	178.23	251	523	723	2600	561	0.25	0.58	0
239	C11H22	n-hexylcyclopentane	154.3	200.2	476.	660.1	2130	0	0	0.47	0
240	C11H22	1-undecene	154.3	224	465.	637	1990	0	0	0.51	0
241	C11H24	n-undecane	156.31	247.6	469.	638.8	1970	660	0.24	0.53	0
242	C12H10	diphenyl	154.21	342.4	529.	789	3850	502	0.29	0.37	0
243	C12H10O	diphenylether	170.21	300	531.	766	3140	0	0	0.44	1.1
244	C12H24	n-heptylcyclopentane	168.32	220	497.	679	1940	0	0	0.51	0
245	C12H24	1-dodecene	168.32	238	486.	657	1850	0	0	0.55	0
246	C12H26	dodecane	170.34	263.6	489.	658.2	1820	713	0.24	0.57	0
247	C12H26O	dihexylether	186.34	230	499.	657	1820	720	0.24	0.7	0
248	C12H26O	dodecanol	186.34	297.1	533.	679	1920	718	0.24	0	1.6
249	C12H27N	tributylamine	185.36	0	486.	643	1820	0	0	0	0.8
250	C13H12	diphenylmethane	168.24	300	538.	770	2860	0	0	0.44	0.4
251	C13H26	n-octylcyclopentane	182.35	229	516.	694	1790	0	0	0.56	0
252	C13H26	1-tridecene	182.35	250.1	505.	674	1700	0	0	0.59	0
253	C13H28	1-tridecane	184.37	267.8	508.	676	1720	780	0.24	0.61	0
254	C14H10	anthracane	178.23	489.7	613.	869.3	0	554	0	0	0
255	C14H10	phenanthrene	178.23	373.7	613	873	0	554	0	0	0
256	C14H28	n-nonylcyclopentane	196.38	244	535.	710.5	1650	0	0	0.61	0
257	C14H28	1-tetradecene	196.38	260.3	524.	689	1560	0	0	0.64	0
258	C14H30	n-tetradecane	198.39	279	526.	693	1440	830	0.23	0.58	0
259	C15H30	n-decyclopentane	210.41	251.1	552.	723.8	1520	0	0	0.65	0
260	C15H30	1-pentadecene	210.41	269.4	541.	704	1450	0	0	0.68	0
261	C15H32	n-pentadecane	212.42	283	543.	707	1520	880	0.23	0.70	0
262	C16H22O	dibutyl-o-phthalate	278.35	238	608	0	0	0	0	0	0
263	C16H32	n-decyclohexane	224.43	271	570.	750	1350	0	0	0.58	0
264	C16H32	1-hexadecene	224.43	277.3	558	717	1330	0	0	0.72	0
265	C16H34	hexadecane	226.45	291	560	722	1410	0	0	0.74	0
266	C17H34	n-dodecyclopentane	238.46	268	584.	750	1290	0	0	0.71	0
267	C17H36O	heptadecanol	256.47	327	597	736	1410	0	0	0	0
268	C18H14	o-terphenyl	230.31	330	605	891	3900	753	0.39	0.43	0
269	C18H14	m-terphenyl	230.31	360	638	924.9	3510	768	0.35	0.44	0
270	C18H14	p-terphenyl	230.31	485	649	926	3320	763	0.32	0.52	0.7
271	C18H36	1-octadecene	252.49	290.8	588	739	1130	0	0	0.80	0
272	C18H36	n-tridecyclopentane	252.49	278	598.	761	1200	0	0	0.75	0
273	C18H38	octadecane	254.5	301.3	589.	748	1200	0	0	0.79	0
274	C18H38O	1-octadecanol	270.5	331	608	747	1410	0	0	0	1.7
275	C19H38	1-cyclopentyltetradecane	266.51	282	599	772	1120	0	0	0.78	0
276	C19H40	n-nonadecane	268.53	305	603.	756	1110	0	0	0.82	0
277	C20H40	1-cyclopentylpentadecane	280.54	290	625	780	1020	0	0	0.83	0
278	C20H42O	1-eicosanol	298.56	339	629	770	1200	0	0	0	0
279	C6F6	perfluorobenzene	186.06	0	353.	516.7	3300	335	0.25	0.39	0
280	C6F12	perfluorocyclohexane	300.05	0	326	457.2	2430	459	0.27	0.43	0
281	C6F14	perfluoro-n-hexane	338.04	186	329.	448.8	1870	606	0.30	0.51	0
282	C6F14	perfluoro-2-	338.04	0	330.	453	1820	550	0.26	0.46	0
283	C6F14	perfluoro-3-	338.04	0	331.	450	1690	0	0	0.47	0
284	C6F14	perfluoro-2,3-	338.04	0	332.	463	1870	525	0.25	0.39	0
285	C6HF5	pentafluorobenzene	168.06	0	358.	531	3530	324	0.26	0.37	0
286	C6HF5O	pentafluorophenol	184.06	0	418.	609	4000	348	0.27	0.50	0
287	C6H2F4	1,2,3,4-tetrafluorobenzene	150.07	0	367.	550.8	3790	313	0.25	0.34	0
288	C6H2F4	1,2,3,5-tetrafluorobenzene	150.07	0	357.	535.5	3750	0	0	0.34	0
289	C6H2F4	1,2,4,5-tetrafluorobenzene	150.07	0	363.	543.4	3800	0	0	0.35	0
290	C6H4Cl2	o-dichlorobenzene	147	256.1	452	729	4100	360	0.24	0.27	2.3
291	C6H4F2	1,4-difluorobenzene	114.09	0	362	556	4400	0	0	0.29	0
292	C6H5Br	bromobenzene	157.01	242.3	429.	670	4520	324	0.26	0.25	1.5

293	C6H5F	fluorobenzene	96.104	234	357.	560.1	4550	269	0.26	0.24	1.4
294	C6H5I	iodobenzene	204.01	241.8	461.	721	4520	351	0.26	0.24	1.4
295	C6H6	benzene	78.114	278.7	353.	562.2	4890	259	0.27	0.21	0
296	C6H6O	phenol	94.113	314	455	694.2	6130	229	0.24	0.43	1.6
297	C6H7N	aniline	93.129	267	457.	699	5310	274	0.25	0.38	1.6
298	C6H7N	2-methylpyridine	93.129	207	402.	621	4600	0	0	0.29	1.9
299	C6H7N	3-methylpyridine	93.129	0	417.	645	0	0	0	0	2.4
300	C6H7N	4-methylpyridine	93.129	276.9	418.	646	4460	311	0.26	0.30	0
301	C6H10	1,5-hexadiene	82.146	132	332.	507	3440	0	0	0.16	0
302	C6H10	cyclohexene	82.146	169.7	356.	560.5	4340	0	0	0.21	0.6
303	C5H12O	2,2-dimethyl-1-propanol	88.15	327	386.	549	0	0	0	0	0
304	C5H12O	ethylpropylether	88.15	146.4	336.	500.2	3370	339	0.27	0.33	1.2
305	C5H12O	butylmethylether	88.15	157.7	343.	512.8	3370	329	0.26	0.31	1.3
306	C5H12O	tert-butylmethylether	88.15	0	328.	496.4	3370	0	0	0.26	1.2
307	C6BrF5	bromopentafluorobenzene	246.96	0	410	601	3040	0	0	0.35	0
308	C6ClF5	chloropentafluorobenzene	202.51	0	391.	570.8	3240	376	0.25	0.4	0
309	C6Cl2F4	dichlorotetrafluorobenzen	218.96	0	430.	626	5320	0	0	0.62	0
310	C6Cl3F3	1,3,5-	235.42	0	471.	684.9	3270	448	0.25	0.42	0
311	C3H9N	n-propylamine	59.112	190	321.	497	4810	233	0.27	0.30	1.3
312	C3H9N	isopropylamine	59.112	177.9	305.	471.8	4540	221	0.22	0.29	0
313	C3H9N	trimethylamine	59.112	156	276	433.3	4090	254	0.28	0.20	0.6
314	C4F8	perfluorocyclobutane	200.03	0	267.	388.5	2780	324	0.27	0.35	0
315	C4F10	perfluorobutane	238.02	145	271.	386.4	2320	378	0.27	0.37	0
316	C4H4	vinylacetylene	52.076	227.6	278.	455	4960	202	0.26	0.09	0
317	C4H4O	furan	68.075	187.5	304.	490.2	5500	218	0.29	0.20	0.7
318	C4H4N	thiophene	84.136	234.9	357.	579.4	5690	219	0.25	0.19	0.5
319	C4H5N	allylcyanide	67.091	186.7	392	585	3950	265	0.22	0.39	3.4
320	C4H5N	pyrrole	67.091	0	403	639.8	0	0	0	0	1.8
321	C4H6	1-butyne	54.092	147.4	281.	463.7	4710	220	0.27	0.05	0.8
322	C4H6	2-butyne	54.092	240.9	300.	488.7	5080	221	0.27	0.12	0.8
323	C4H6	1,2-butadiene	54.092	137	284	443.7	4490	219	0.26	0.25	0.4
324	C4H6	1,3-butadiene	54.092	164.2	268.	425	4330	221	0.27	0.19	0
325	C4H6O2	vinylacetate	86.091	173	346	525	4350	265	0.26	0.34	1.7
326	C4H6O3	aceticanhydride	102.09	199	413.	569	4680	0	0	0.90	3
327	C4H6O4	dimethyloxalate	118.09	327	436.	628	3980	0	0	0.55	0
328	C4H6O4	succinicacid	118.09	456	508	0	0	0	0	0	2.2
329	C4H7N	butylonitrile	69.107	161	391.	582.2	3790	0	0	0.37	3.8
330	C4H7O2	methylacrylate	86.091	196.7	353.	536	4300	265	0.25	0.35	0
331	C4H8	1-butene	56.108	87.8	266.	419.6	4020	240	0.27	0.19	0.3
332	C4H8	2-butene,cis	56.108	134.3	276.	435.6	4200	234	0.27	0.20	0.3
333	C4H8	2-butene,trans	56.108	167.6	274	428.6	3990	238	0.26	0.20	0
334	C4H8	cyclobutane	56.108	182.4	285.	460	4990	210	0.27	0.18	0
335	C4H8	isobutylene	56.108	132.8	266.	417.9	4000	239	0.27	0.19	0.5
336	C4H8O	n-butyraldehyde	72.107	176.8	348	545.4	5380	0	0	0.35	2.6
337	C4H8O	isobutyraldehyde	72.107	208.2	337	513	4150	274	0.27	0.35	0
338	C4H8O	methylethylketone	72.107	186.5	352.	536.8	4210	267	0.25	0.32	3.3
339	C4H8O	tetrahydrofuran	72.107	164.7	338	540.1	5190	224	0.25	0.21	1.7
340	C4H8O	vinylethylether	72.107	157.9	308.	475	4070	0	0	0.26	1.3
341	C4H8O2	n-butyricacid	88.107	267.9	437.	628	5270	290	0.29	0.68	1.5
342	C4H8O2	isobutyricacid	88.107	227.2	427.	609	4050	292	0.23	0.62	1.3
343	C4H8O2	1,4-dioxane	88.107	285	374.	587	5210	238	0.25	0.28	0.4
344	C4H8O2	ethylacetate	88.107	189.6	350.	523.2	3830	286	0.25	0.36	1.9
345	C4H8O2	methylpropionate	88.107	185.7	352.	530.6	4000	282	0.25	0.35	1.7
346	C4H8O2	n-propylformate	88.107	180.3	354.	538	4060	285	0.25	0.31	1.9
347	C4H8S	tetrahydrothiophene	88.172	177	394.	632	0	0	0	0	1.9
348	C4H9Cl	1-chlorobutane	92.569	150.1	351.	542	3680	312	0.25	0.21	2
349	C4H9Cl	2-chlorobutane	92.569	141.8	341.	520.6	3950	305	0.28	0.3	2.1
350	C4H9Cl	tert-butylchloride	92.569	247.8	324	507	3950	295	0.28	0.19	2.1
351	C4H9N	pyrrolidine	71.123	0	359.	568.6	5610	249	0.29	0.27	1.6
352	C4H10	isobutane	58.124	113.6	261.	408.2	3650	263	0.28	0.18	0.1
353	C4H10O	n-butanol	74.123	183.9	390.	563.1	4420	275	0.25	0.59	1.8
354	C4H10O	2-butanol	74.123	158.5	372.	536.1	4180	269	0.25	0.57	1.7

355	C4H10O	isobutanol	74.123	165.2	381	547.8	4300	273	0.25	0.59	1.7
356	C4H10O	tert-butanol	74.123	298.8	355.	506.2	3970	275	0.25	0.61	1.7
357	C4H10O	diethylether	74.123	156.9	307.	466.7	3640	280	0.26	0.28	1.3
358	C4H10O	methylpropylether	74.123	0	311.	476.3	3800	0	0	0.27	1.2
359	C4H10O	methylisopropylether	74.123	0	303.	464.5	3760	0	0	0.26	0
360	C4H10O2	1,2-dimethoxyethane	90.123	202	358	536	3870	271	0.23	0.35	0
361	C4H10O3	diethyleneglycol	106.12	265	519	681	4700	0	0	0	0
362	C4H10S	diethylsulfide	90.184	169.2	365.	557	3960	318	0.27	0.29	1.6
363	C4H10S2	diethyldisulfide	122.24	171.7	427.	642	0	0	0	0	2
364	C4H11N	n-butylamine	73.139	224.1	349.	531.9	4200	0	0	0.32	1.3
365	C4H11N	isobutylamine	73.139	188	336.	514.3	4100	0	0	0.36	1.2
366	C4H11N	diethylamine	73.139	223.4	328.	496.5	3710	301	0.27	0.29	1.1
367	C5F12	perfluoropentane	288.03	0	302.	420.6	2050	473	0.27	0.43	0
368	C5H2F6O	hexafluoroacetylacetone	208.06	0	327.	485.1	2770	0	0	0.27	0
369	C5H4O2	furfural	96.085	234.5	434.	670	5890	0	0	0.38	3.6
370	C5H5N	pyridine	79.102	231.5	388.	620	5630	254	0.27	0.24	2.3
371	C5H6N2	2-methylpyrazine	94.117	0	410	634.3	5010	283	0.26	0.31	0
372	C5H6O	2-methylfuran	82.102	0	338	527	4720	247	0.26	0.27	0.7
373	C5H8	cyclopentene	68.119	138.1	317.	506	0	0	0	0	0.9
374	C5H8	1,2-pentadiene	68.119	135.9	318	503	4070	276	0.26	0.17	0
375	C5H8	1,3-pentadiene,trans	68.119	185.7	315.	496	3990	275	0.26	0.17	0.7
376	C5H8	1,4-pentadiene	68.119	124.9	299.	478	3790	276	0.26	0.10	0.4
377	C5H8	pentyne	68.119	167.5	313.	493.5	4050	278	0.27	0.16	0.9
378	C5H8	2-methyl-1,3-butadiene	68.119	127.2	307.	484	3850	276	0.26	0.16	0.3
379	C5H8	3-methyl-1,2-butadiene	68.119	159.5	314	496	4110	267	0.26	0.16	0
380	C5H8O	cyclopentanone	84.118	222.5	403.	634.6	5110	268	0.26	0.35	3
381	C5H8O	dihydropyran	84.118	0	359	561.7	4560	268	0.26	0.24	1.4
382	C5H8O2	ethylacrylate	100.12	201	373	552	3740	320	0.26	0.4	0
383	C5H10	cyclopentane	70.135	179.3	322.	511.7	4510	260	0.27	0.19	0
384	C5H10	1-pentene	70.135	107.9	303.	464.8	3530	300	0.31	0.23	0.4
385	C5H10	2-pentene,cis	70.135	121.7	310.	476	3650	0	0	0.25	0
386	C5H10	2-pentene,trans	70.135	132.9	309.	475	3660	0	0	0.25	0
387	C5H10	2-methyl-1-butene	70.135	135.6	304.	465	3450	0	0	0.23	0.5
388	C5H10	2-methyl-2-butene	70.135	139.3	311.	470	3450	0	0	0.24	0
389	C5H10	3-methyl-1-butene	70.135	104.7	293.	450	3510	0	0	0.20	0
390	C5H10O	valeraldehyde	86.134	182	376	554	3540	333	0.26	0.4	2.6
391	C5H10O	methyl-n-propyl-ketone	86.134	196	375.	561.1	3690	301	0.23	0.34	2.5
392	C5H10O	methyl	86.134	181	367.	553.4	3850	310	0.25	0.33	2.8
393	C5H10O	diethyl-ketone	86.134	234.2	375.	561	3730	336	0.26	0.34	2.7
394	C5H10O	2-methyl-tetrahydrofuran	86.134	0	351	537	3760	267	0.22	0.26	0
395	C5H10O	tetrahydropyran	86.134	0	361	572.2	4770	263	0.26	0.21	1.6
396	C5H10O2	n-valeric-acid	102.13	239	459.	651	0	0	0	0	0
397	C5H10O2	isovaleric-acid	102.13	0	449.	634	0	0	0	0	1
398	C5H10O2	isobutyl-formate	102.13	178	371.	554	3730	352	0.28	0.39	1.9
399	C5H10O2	n-propyl-acetate	102.13	178	374.	549.4	3330	345	0.25	0.39	1.8
400	C5H10O2	ethyl-propionate	102.13	199.3	372.	546	3360	345	0.25	0.39	1.8
401	C5H10O2	methyl-butyrate	102.13	188.4	375.	554.4	3480	340	0.25	0.38	1.7
402	C5H10O2	methyl-isobutyrate	102.13	185.4	365.	540.8	3430	339	0.25	0.36	2
403	C5H11N	piperidine	85.15	262.7	379.	594	4760	289	0.28	0.25	1.2
404	C5H12	n-pentane	72.151	143.4	309.	469.7	3370	304	0.26	0.25	0
405	C5H12	2-methyl-butane	72.151	113.3	301	460.4	3390	306	0.27	0.22	0.1
406	C5H12	2,2-dimethylpropane	72.151	256.6	282.	433.8	3200	303	0.26	0.19	0
407	C5H12O	1-pentanol	88.15	195	411.	588.2	3910	326	0.26	0.57	1.7
408	C5H12O	2-methyl-1-butanol	88.15	203	401.	571	3340	0	0	0	0
409	C5H12O	3-methyl-1-butanol	88.15	156	405.	579.4	0	0	0	0	1.8
410	C5H12O	2-methyl-2butanol	88.15	264.4	375.	545	3950	0	0	0	1.9
411	C5H12O	2,2-dimethyl-1-propanol	88.15	327	386.	549	0	0	0	0	0
412	C4H9N0	morpholine	87.122	268.4	401.	618	5470	253	0.27	0.37	1.5
413	AlBr3	aliminiumtribromide	266.69	370.7	528	763	2890	310	0.41	0.39	5
414	C6H10O	cyclohexanone	98.145	242	428.	629	3900	0	0	0	3.1
415	C6H12O	methylbutylketone	100.16	216	400.	587	3320	0	0	0.39	0
416	C6H12O2	isoamylformate	116.16	0	396.	578	0	0	0	0	0

417	C6H14O	2-hexanol	102.18	0	411	586.2	0	0	0	0	0
418	C6H14O	methylamylether	102.18	0	372	546.5	3040	392	0.26	0.34	0
419	C7F8	perfluorotoluene	236.06	0	377.	534.5	2710	428	0.26	0.47	0
420	C7H9N	2,3-dimethylpyridine	107.16	0	434.	655.4	0	0	0	0	2.2
421	C7H9N	2,4-dimethylpyridine	107.16	0	431.	647	0	0	0	0	2.3
422	C7H9N	2,5-dimethylpyridine	107.16	0	430.	644.2	0	0	0	0	2.2
423	C7H9N	2,6-dimethylpyridine	107.16	267	417.	623.8	0	0	0	0	1.7
424	C7H9N	3,4-dimethylpyridine	107.16	0	452.	683.8	0	0	0	0	1.9
425	C7H9N	3,5-dimethylpyridine	107.16	0	445.	667.2	0	0	0	0	2.6
426	C7H9N	p-toluidine	107.16	316.9	473.	667	2380	0	0	0.44	1.6
427	C7H14O	methylamylketone	114.19	0	424.	611.5	3440	0	0	0.48	0
428	C7H14O2	isoamylacetate	130.19	194.7	415.	599	0	0	0	0	1.8
429	C7H16	2-methylhexane	100.21	154.9	363.	530.4	2730	421	0.26	0.32	0
430	C6H12O2	n-amylformate	116.16	199.7	403.	576	3460	1	0	0.53	0

EXAMPLE: Estimate the viscosity of sulfur dioxide (SO_2) gas at atmospheric pressure(101.325 kPa) and 573.15 K temperature by using Chung method. The experimental viscosity is $0.246x10^{-5}$ Pas

```
public class viscosityA
{
//Collusion integral
public static double omegav(double e_k,double T)
 {double Ts=T/e_k;
  double Ov=1.16145*Math.pow(Ts,-0.14874)+0.52487*Math.exp(-0.7732*Ts)+2.161178*Math.exp(-2.43787*Ts);
  return Ov;
 }
 public static double omegav(double Ts)
 {
  double Ov=1.16145*Math.pow(Ts,-0.14874)+0.52487*Math.exp(-0.7732*Ts)+2.161178*Math.exp(-2.43787*Ts);
  return Ov;
 }
//reduced dipole moment Chun at al
 public static double mur(double Vc,double Tc,double mu)
 {  //mu : is dipole moment in debyes
    //Vc cm^3/mole
    //Tc degree K
                double m=131.3*mu/Math.sqrt(Vc*1e3*Tc);
                return m;
 }
 public static double Fc(double Vc,double Tc,double mu,double omega)
 {// omega pitser' acentycity factor
 double mr=mur(Vc,Tc,mu);
  double F=1.0-0.2756*omega+0.059035*mr*mr*mr*mr;
  System.out.println("mur="+mr+"F="+F);
  return F;
 }
 //Method of Chung et. al.
 public static double visc(double M,double T,double Vc,double Tc,double mu,double w)
 { double e_k=Tc/1.2503;
    return visc(M,T,Vc,Tc,mu,w,e_k);
 }
  public static double visc(double M,double T,double Vc,double Tc,double mu,double w,double e_k)
 {  double Tr=T/Tc;
     double Ts=1.2593*Tr;
                double Ov=omegav(Ts);
                double F=Fc(Vc,Tc,mu,w);
    double eta=40.785e-7*F*Math.sqrt(M*T)/Math.pow(Vc*1e3,(2.0/3.0))/Ov;
    System.out.println("Ts="+Ts+"omega="+Ov+"eta="+eta);
    return eta;
 }

 public static void main(String arg[])
 {double Tc=430.8;  //Degree K
  double Vc=122e-3;  //m^3/kmol
  double M=64.065; //kg/kmol
  double mu=1.6;   //Deby
  double T=300+273.15;  //Degree K
  double w=0.257;
double eta=visc(M,T,Vc,Tc,mu,w);
  double  eta1=2.46e-5;  //Pas
  System.out.println("viscosity = "+eta+" Pas");
```

```
 double error=(eta1-eta)/eta1*100;
  System.out.println("error = "+error+"%");
 }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" viscosityA
mur=0.9163617355542482F=0.9707980146023764
Ts=1.6754127089136492omega=1.2557116160647797eta=2.4562775801970916E-5
viscosity = 2.4562775801970916E-5 Pas
error = 0.15131787816701 45%

> Terminated with exit code 0.
```

Lucas[124] suggested the following correlations:

$$\eta\xi = f(T_r) = [0.807T_r^{0.618} - 0.357\exp(-0.449T_r) + 0.340\exp(-4.058T_r) + 0.018]F_p^0 F_Q^0$$

Where $\xi = 0.01\left[\frac{(RT_c)(N_0)^2}{M^3P_c^4}\right]^{1/6}$ where R=8314.46261815324 J/(kmolK) is universal gas constant N_0=6.02214076x10^{26} 1/(kmol) is Avagadro's constant. Pc is the critical pressure in kPa, T_c is the critical temperature in degree K, M is molecular weight in kg/kmol, $T_r = \frac{T}{T_c}$ is reduced temperature. μ_r is dimensionless dipole moment and μ is dipole moment in debyes. Lucas et al defined μ_r as

$$\mu_r = 0.5246\frac{\mu^2 P_c}{T_c}$$

Correction factor F_p^0 defined as:

$$F_p^0 = 1 \qquad 0 \le \mu_r < 0.022$$
$$F_p^0 = 1 + 30.55(0.292 - Z_c)^{1.72} \qquad 0.022 \le \mu_r < 0.075$$
$$F_p^0 = 1 + 30.55(0.292 - Z_c)^{1.72}[0.96 + 0.1(T_r - 0.7)] \qquad 0.075 \le \mu_r$$

Correction factor F_p^0 is used only for the quantum gases He, H_2 and D_2

$$F_Q^0 = 1.22Q^{0.15}\{1 + 0.00385[(T_r - 12)^2]^{1/M} sign(T_r - 12)\}$$

Where Q=1.38(He), Q=0.76(H_2), Q=0.52(D_2)

Viscosity methods discussed so far is given in the following program:

```
 public class viscosity
{
 //Method of .Lucas
 public static double visc(double T,double Tc,double Pc,double Zc,double M,double mu)
 {//Enter Pc in kPa T and Tc in degree Kelvin M molecular mass in kg/kmol
  //mu is dipole moment in deby
 double N0=6.02214076e26;// avagadro number    1/kmol
  double R=8314.46261815324; // Universal gas constant  J/(kmolK)
  double eps=0.01*Math.pow((R*Tc*N0*N0/(M*M*M*Pc*Pc*Pc*Pc)),(1.0/6.0));
  double Tr=T/Tc;
  double mur=0.5246*mu*mu*Pc/(Tc*Tc);
  double FP=0;
  if(mur<0.022 && mur>=0) FP=1.0;
  else if(mur>=0.022 && mur<0.075) FP=1.0+30.55*Math.pow((0.292-Zc),1.72);
  else FP=1.0+30.55*Math.pow((0.292-Zc),1.72)*(0.96+0.1*(Tr-0.7));
  double eta=(0.807*Math.pow(Tr,0.618)-0.357*Math.exp(-0.449*Tr)+0.34*Math.exp(-4.058*Tr)+0.018)*FP/eps;
  return eta;
 }

 public static void main(String arg[])
 {double Tc=512.64;  //Degree K
  double Pc=80.97e2;  //kPa
  double Zc=0.224;
  double M=32.042; //kg/kmol
  double mu=1.7;   //Deby
  double T=550;  //Degree K
```

```
 double eta=visc(T,Tc,Pc,Zc,M,mu);
 double  eta1=1.81e-5;  //Pas
 System.out.println("viscosity = "+eta+" Pas");
 double error=(eta1-eta)/eta1*100;
  System.out.println("error = "+error+"%");
 }
}
```

EXAMPLE:

Estimate the viscosity of methanol vapor at a temperature T=550 K and P=100 kPa by using Lucas method. The experimental value is 1.81×10^{-5} Pas. Calculate the error

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" viscosity
viscosity = 1.7788626008221592E-5 Pas
error = 1.7202982971182734%

> Terminated with exit code 0.
```

VİSCOSİTY OF GAS MIXTURES

When viscosity of gas mixtures are required Method of Wilke [133] can be used

$$\eta_m = \sum_{i=1}^{n} \frac{y_i \eta_i}{\sum_{j=1}^{n} y_i \phi_{ij}}$$

$$\phi_{ij} = \frac{\left[1 + \left(\frac{\eta_i}{\eta_j}\right)^{1/2} \left(\frac{M_j}{M_i}\right)^{1/4}\right]}{\left[8\left(1 + \left(\frac{M_j}{M_i}\right)\right)\right]^{1/2}}$$

EXAMPLE: Predict viscosity of the exhaust gas mixtures given below at 800 K

Gas	%
N_2	72
CO_2	10
H_2O	10
O_2	8

```
public class viscosity_mix
{
public Gas gasList[];
public double N;
public int ngas;

public viscosity_mix(String gasname[],double Ni[])
{ngas=gasname.length;
 gasList=new Gas[ngas];
 N=0;
 for(int i=0;i<ngas;i++)
 { gasList[i]=new Gas(gasname[i],Ni[i]);
   N+=Ni[i];
 }
 N/=ngas;
}
public double  vis(double T)
   {
   // dynamic viscosity of the mixture
   // Wilke method
      double vmix=0;
   for(int i=0;i<ngas;i++)
     {
     T=T;
     double fij;
```

```
    double xj=0;
    double p1;
    double c1,c2,c3;
    for(int j=0;j<ngas;j++)
     {
     double vratio;
     double xi=gasList[i].M/gasList[j].M;
     if(gasList[j].vis(T)!=0)
       {
                 c1=gasList[i].vis(T)/gasList[j].vis(T);
                 vratio=Math.sqrt(c1);
                 }
     else
                 vratio=0;
               c2=Math.pow(xi,0.25);
               p1=(1+vratio/c2);
               c3=8.0+8.0*xi;
               fij=p1*p1 / Math.sqrt(c3);
     xj+=fij*gasList[j].N/N;
     }
    vmix+=gasList[i].N/N*gasList[i].vis(T)/xj;
    }
  return vmix;
  }
public static void main(String arg[])
{String gasname[]={"n2","co2","h2o","o2"};
 double N[]={0.71,0.1,0.1,0.08};
 viscosity_mix vm=new viscosity_mix(gasname,N);
 double T=800;  //degree K
 double vis=vm.vis(T);
 System.out.println("vis="+vis+" Pas");
}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" viscosity_mix
vis=3.497000369572745E-5 Pas

> Terminated with exit code 0.
```

Where η_m is the viscosity of the mixture η_i, pure component viscosities, y_i mole fractions (kg/kmol)
As a second method for gas mixtures Lucas[124] Method is defined below:

$$T_{cm} = \sum_i y_i T_{ci}$$

$$P_{cm} = RT_{cm} \frac{\sum_i y_i Z_{ci}}{\sum_i y_i V_{ci}}$$

$$M_m = \sum_i y_i M_i$$

$$F_{Pm}^0 = \sum_i y_i F_{Pi}^0$$

$$F_{Qm}^0 = \left(\sum_i y_i F_{Qi}^0 \right) A$$

$$A = 1 - 0.01 \left(\frac{M_H}{M_L} \right)^{0.87} \quad for \quad \left(\frac{M_H}{M_L} \right) > 9 \; and \; 0.05 < y_H < 0.7$$

Where M_H is the highest molecular weight and M_L is the lowest molecular weight.

VISCOSITY OF HIGH PRESSURE GASES

If the gas pressure is high, low prossure estimations will not be valid anymore and effect of pressure in viscosity should be taken into the account. Reichenberg[123] suggested the following correlation:

$$\frac{\eta}{\eta^0} = 1 + Q\frac{AP_r^{3/2}}{BP_r + (1 + CP_r^D)^{-1}}$$

Where

$A = \frac{\alpha_1}{T_r}\exp(\alpha_2 T_r^a) \quad B = A(\beta_1 T_r - \beta_2)$

$C = \frac{\gamma_1}{T_r}\exp(\gamma_2 T_r^c) \quad D = \frac{\delta_1}{T_r}\exp(\delta_2 T_r^d)$

α_1=1.9824x10^{-3} α_2=5.2683 a=-0.5767

β_1=1.6552 β_2=1.276

γ_1=0.1319 γ_2=3.7035 c=-79.8678

δ_1=2.9496 δ_2=2.919 d=-16.6169

$Q = (1 - 5.655\mu_r)$ where $\mu_r = 0.5246\frac{\mu^2 P_c}{T_c}$

For non-polar materials Q=1

PROBLEM: Use Reichenberg method to estimate the viscosity of n-pentane vapor at T=500 K and P=101 bar. Experimental value is 5.46x10^{-5} Pas.

```
public class viscosity1
{  //high pressure viscosity
public static double vis(String name,double T,double P,double Tc,double Pc,double mu)
{//P kPA  T degree K
double alpha1=1.9824e-3;
double alpha2=5.2683;
double a=-0.5767;
double beta1=1.6552;
double beta2=1.276;
double gamma1=0.1319;
double gamma2=3.7035;
double c=-79.8678;
double delta1=2.9496;
double delta2=2.919;
double d=-16.6169;
double Tr=T/Tc;
double Pr=P/Pc;
double A=alpha1/Tr*Math.exp(alpha2*Math.pow(Tr,a));
double B=A*(beta1*Tr-beta2);
double C=gamma1/Tr*Math.exp(gamma2*Math.pow(Tr,c));
double D=delta1/Tr*Math.exp(delta2*Math.pow(Tr,d));
Gas g=new Gas(name);
double vis0=g.vis(T);
System.out.println("vis0="+vis0);
double mur=0.5246*mu*mu*Pc/Tc;
double Q=1.0-5.655*mur;
double ratio=1+Q*A*Math.pow(Pr,1.5)/(B*Pr+1.0/(1.0+C*Math.pow(Pr,D)));
double vis=vis0*ratio;
return vis;
}
 public static void main(String arg[])
 {double Tc=469.7;  //Degree K
  double Pc=33.7e2;  //kPa
  double mu=0;   //Deby
  double T=500;  //Degree K
  double P=101e2;
  double eta=vis("c5h12",T,P,Tc,Pc,mu);
  double  eta1=5.46e-5;  //Pas
  System.out.println("viscosity = "+eta+" Pas");
  double error=(eta1-eta)/eta1*100;
   System.out.println("error = "+error+"%");
  } }
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" viscosity1
Pr=2.9970326409495547Tr=1.0645092612305727
A=0.29992897584825234B=0.14575820274271756C=0.12706048708828802D=7.784556283273705
vis0=1.1557614648554248E-5
ratio=4.549878004832756
viscosity = 5.258573667778984E-5 Pas
error = 3.6891269637548687%

> Terminated with exit code 0.
```

In a similar manner **Lucas**[124],[125][126] described high pressure gas viscosity by using the following equations

$$Z_1 = \eta^0\xi = [0.807T_r^{0.618} - 0.357\exp(-0.449T_r)\,0.34\exp(-4.058T_r) + 0.018]F_p^0F_Q^0$$

Where η^0 refres the low pressure viscosity.

If $T_r \leq 1$ and $P_r < \left(\frac{P_{vp}}{P_c}\right)$ $then$

$$Z_2 = 0.6 + 0.76P_r^{\alpha} + (6.99P_r^{\beta} - 0.6)(1 - T_r)$$

With $\alpha = 3.262 + 14.98P_r^{5.508}$

$\beta = 1.390 + 5.746P_r$

If $1 \leq T_r \leq 1$ and $0 \leq P_r \leq 100$ $then$

$$Z_2 = Z_1\left[1 + \frac{aP_r^e}{b + P_r^f + \left(1 + cP_r^d\right)^{-1}}\right]$$

$a = \frac{a_1}{T_r}\exp(\alpha_2 T_r^a)$ $\quad b = a(\beta_1 T_r - \beta_2)$

$c = \frac{c_1}{T_r}\exp(c_2 T_r^c)$ $\quad d = \frac{d_1}{T_r}\exp(d_2 T_r^{\varepsilon})$

e=1.3088

f=$f_1 \exp\left(f_1 T_r^{\xi}\right)$

$a_1 = 1.245x10^{-3}$	$a_2 = 5.1726$	$\gamma = -0.3286$
$b_1 = 1.6553$	$b_2 = 1.2723$	
$c_1 = 0.4489$	$c_2 = 3.0578$	$\delta = -37.7332$
$d_1 = 1.7368$	$d_2 = 2.2310$	$\varepsilon = -7.6351$
$f_1 = 0.9425$	$f_2 = -0.1853$	

After computing Z_1 and Z_2 calculate correction factors F_P and F_Q as:

$$Y = \frac{Z_2}{Z_1}$$

$$F_P = \frac{1 + \left(F_p^0 - 1\right)Y^{-3}}{F_p^0}$$

$$F_Q = \frac{1 + \left(F_Q^0 - 1\right)[Y^{-1} - 0.007(\ln(Y))^4]}{F_Q^0}$$

F_p^0 and F_Q^0 are low pressure polarity and quantum factors. Low pressure polarity F_p^0 defined as:

$$F_p^0 = 1 \qquad 0 \leq \mu_r < 0.022$$

$$F_p^0 = 1 + 30.55(0.292 - Z_c)^{1.72} \qquad 0.022 \leq \mu_r < 0.075$$

$$F_p^0 = 1 + 30.55(0.292 - Z_c)^{1.72}[0.96 + 0.1(T_r - 0.7)] \qquad 0.075 \leq \mu_r$$

Quantumr F_Q^0 is used only for the quantum gases He, H_2 and D_2

$$F_Q^0 = 1.22Q^{0.15}\left\{1 + 0.00385[(T_r - 12)^2]^{1/M}sign(T_r - 12)\right\}$$

Where Q=1.38(He), Q=0.76(H_2), Q=0.52(D_2)

Finally, the dense gas viscosity is calculated as:

$$\eta=\frac{Z_2 F_P F_Q}{\xi}$$

Where $\xi = 0.01\left[\frac{(RT_c)(N_0)^2}{M^3 P_c^4}\right]^{1/6}$, R=8314.46261815324 J/(kmolK) is universal gas constant N_0=6.02214076x10^{26} 1/(kmol) is Avagadro's constant. Pc is the critical pressure in kPa, T_c is the critical temperature in degree K, M is molecular weight in kg/kmol

PROBLEM: Estimate the viscosity of ammonia gas at 420 K and 30000 kPa pressure by using Lucas method. The experimental value of η is 5.71x10^{-5} Pas.

```
public class viscosity2
{  //high pressure viscosity  Method of .Lucas
 public static double visc(String name,double T,double P,double Tc,double Pc,double Zc,double M,double mu)
 {//Enter Pc in kPa T and Tc in degree Kelvin M molecular mass in kg/kmol
  //mu is dipole moment in deby
  double N0=6.02214076e26;// avagadro number    1/kmol
   double R=8314.46261815324; // Universal gas constant  J/(kmolK)
   double eps=0.01*Math.pow((R*Tc*N0*N0/(M*M*M*Pc*Pc*Pc*Pc)),(1.0/6.0));
   System.out.println("eps="+eps);
   double Tr=T/Tc;
   double Pr=P/Pc;
   System.out.println("Tr="+Tr+"Pr="+Pr);
   double mur=0.5246*mu*mu*Pc/(Tc*Tc);
   System.out.println("mur="+mur);
   double FP0=0;
   if(mur<0.022 && mur>=0) FP0=1.0;
   else if(mur>=0.022 && mur<0.075) FP0=1.0+30.55*Math.pow((0.292-Zc),1.72);
   else FP0=1.0+30.55*Math.pow((0.292-Zc),1.72)*(0.96+0.1*(Tr-0.7));
   System.out.println("FP0="+FP0);
   double FQ0=1.0;
   double Q=0;
   if(name.equals("h2"))
   {Q=0.76;FQ0=1.22*Math.pow(Q,0.15)*(1.0+0.00385*Math.pow(((Tr-12.0)*(Tr-12.0)),1.0/M)*(Tr-12.0)/Math.abs(Tr-12.0));}
   else if (name.equals("he"))
   {Q=1.38;FQ0=1.22*Math.pow(Q,0.15)*(1.0+0.00385*Math.pow(((Tr-12.0)*(Tr-12.0)),1.0/M)*(Tr-12.0)/Math.abs(Tr-12.0));}
   else if(name.equals("d2"))
   {Q=0.52;;FQ0=1.22*Math.pow(Q,0.15)*(1.0+0.00385*Math.pow(((Tr-12.0)*(Tr-12.0)),1.0/M)*(Tr-12.0)/Math.abs(Tr-12.0));}
   else
   {Q=0;FQ0=1.0;}
   double Z1=(0.807*Math.pow(Tr,0.618)-0.357*Math.exp(-0.449*Tr)+0.34*Math.exp(-4.058*Tr)+0.018)*FP0*FQ0;
   double eta0=Z1/eps;
   System.out.println("Z1="+Z1+"eta0="+eta0);
   double Z2=Z1;
   double alpha=0;
   double beta=0;
   double a,b,c,d,e,f,a1,a2,b1,b2,c1,c2,d1,d2,f1,f2,gamma,delta,epsilon,xsi,Y;
   if(Tr<=1.0)
   {
                alpha=3.262+14.98*Math.pow(Pr,5.508);
                beta=1.390+5.746*Pr;
      Z2=0.6+0.76*Math.pow(Pr,alpha)+(6.99*Math.pow(Pr,beta)-0.6)*(1.0-Tr);
   }
   else if(Tr>=1  && Tr<40 &&  Pr>0  && Pr<=100.0)
   {a1=1.245e-3;a2=5.1726;gamma=-0.3286;
    b1=1.653;b2=1.2723;
    c1=0.4489;c2=3.0578;delta=-37.7332;
    d1=1.7368;d2=2.2310;epsilon=-7.6351;
    f1=0.9425;f2=-0.1853;xsi=0.4489;
    a=a1/Tr*Math.exp(a2*Math.pow(Tr,gamma));
    b=a*(b1*Tr-b2);
    c=c1/Tr*Math.exp(c2*Math.pow(Tr,delta));
    d=d1/Tr*Math.exp(d2*Math.pow(Tr,epsilon));
    e=1.3088;
    f=f1*Math.exp(f2*Math.pow(Tr,xsi));
    System.out.println("a="+a+"b="+b+"c="+c+"d="+d+"e="+e+"f="+f);
    Z2=Z1*(1.0+a*Math.pow(Pr,e)/(b*Math.pow(Pr,f)+1.0/(1.0+c*Math.pow(Pr,d))));
   }
   Y=Z2/Z1;
   System.out.println("Z2="+Z2+"Y="+Y);
   double FP=(1.0+(FP0-1.0)/(Y*Y*Y))/FP0;
```

```
 double lnY=Math.log(Y);
 double FQ=(1.0+(FQ0-1.0)*(1.0/Y-0.007*lnY*lnY*lnY*lnY))/FQ0;
 double eta=Z2*FP*FQ/eps;
 System.out.println("FP="+FP+"FQ="+FQ+"eta="+eta);
 return eta;
}
public static void main(String arg[])
{double Tc=405.5;  //Degree K
 double Pc=113.53e2;  //kPa
 double Zc=0.244;
 double mu=1.47;   //Debyes
 double M=17.031;//kg/kmol
 double T=420;  //Degree K
 double P=300e2;//kPa
 double eta=visc("c5h12",T,P,Tc,Pc,Zc,M,mu);
 double  eta1=5.71e-5;  //Pas
 System.out.println("viscosity = "+eta+" Pas");
 double error=(eta1-eta)/eta1*100;
  System.out.println("error = "+error+"%");
 }}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" viscosity2
eps=49605.21180786296
Tr=1.0357583230579532Pr=2.6424733550603365
mur=0.0782693769146492
FP0=1.163661739215932
Z1=0.7256183338078463eta0=1.4627864842476655E-5
a=0.1997821513499511b=0.08786588991320828c=0.9763879591012788d=9.234925678658891e=1.3088f=0.7807775467359662
Z2=3.479691717204067Y=4.7954848369715215
FP=0.8606315906763139FQ=1.0eta=6.0371330118296836E-5
viscosity = 6.0371330118296836E-5 Pas
error = -5.72912455043229%

> Terminated with exit code 0.
```

PROBLEM: Use Reichenberg method to estimate the viscosity of n-pentane vapor at T=500 K and P=101 bar. Experimental value is 5.46×10^{-5} Pas.

```
public class viscosity2
{  //high pressure viscosity  Method of .Lucas
 public static double visc(String name,double T,double P,double Tc,double Pc,double Zc,double M,double mu)
 {//Enter Pc in kPa T and Tc in degree Kelvin M molecular mass in kg/kmol
  //mu is dipole moment in deby
  double N0=6.02214076e26;// avagadro number    1/kmol
  double R=8314.46261815324; // Universal gas constant  J/(kmolK)
  double eps=0.01*Math.pow((R*Tc*N0*N0/(M*M*M*Pc*Pc*Pc)),(1.0/6.0));
  System.out.println("eps="+eps);
  double Tr=T/Tc;
  double Pr=P/Pc;
  System.out.println("Tr="+Tr+"Pr="+Pr);
  double mur=0.5246*mu*mu*Pc/(Tc*Tc);
  System.out.println("mur="+mur);
  double FP0=0;
  if(mur<0.022 && mur>=0) FP0=1.0;
  else if(mur>=0.022 && mur<0.075) FP0=1.0+30.55*Math.pow((0.292-Zc),1.72);
  else FP0=1.0+30.55*Math.pow((0.292-Zc),1.72)*(0.96+0.1*(Tr-0.7));
  System.out.println("FP0="+FP0);
  double FQ0=1.0;
  double Q=0;
  if(name.equals("h2"))
  {Q=0.76;FQ0=1.22*Math.pow(Q,0.15)*(1.0+0.00385*Math.pow(((Tr-12.0)*(Tr-12.0)),1.0/M)*(Tr-12.0)/Math.abs(Tr-12.0));}
  else if (name.equals("he"))
  {Q=1.38;FQ0=1.22*Math.pow(Q,0.15)*(1.0+0.00385*Math.pow(((Tr-12.0)*(Tr-12.0)),1.0/M)*(Tr-12.0)/Math.abs(Tr-12.0));}
  else if(name.equals("d2"))
  {Q=0.52;;FQ0=1.22*Math.pow(Q,0.15)*(1.0+0.00385*Math.pow(((Tr-12.0)*(Tr-12.0)),1.0/M)*(Tr-12.0)/Math.abs(Tr-12.0));}
  else
  {Q=0;FQ0=1.0;}
  double Z1=(0.807*Math.pow(Tr,0.618)-0.357*Math.exp(-0.449*Tr)+0.34*Math.exp(-4.058*Tr)+0.018)*FP0*FQ0;
  double eta0=Z1/eps;
  Gas g=new Gas("name");
  System.out.println("Z1="+Z1+"eta0="+eta0);
  System.out.println("viscosity value by using class Gas="+g.vis(T));
  double Z2=Z1;
```

```
   double alpha=0;
   double beta=0;
   double a,b,c,d,e,f,a1,a2,b1,b2,c1,c2,d1,d2,f1,f2,gamma,delta,epsilon,xsi,Y;
   if(Tr<=1.0)
   {
                alpha=3.262+14.98*Math.pow(Pr,5.508);
                beta=1.390+5.746*Pr;
      Z2=0.6+0.76*Math.pow(Pr,alpha)+(6.99*Math.pow(Pr,beta)-0.6)*(1.0-Tr);
   }
   else if(Tr>=1  && Tr<40 &&  Pr>0  && Pr<=100.0)
   {a1=1.245e-3;a2=5.1726;gamma=-0.3286;
    b1=1.653;b2=1.2723;
    c1=0.4489;c2=3.0578;delta=-37.7332;
    d1=1.7368;d2=2.2310;epsilon=-7.6351;
    f1=0.9425;f2=-0.1853;xsi=0.4489;
    a=a1/Tr*Math.exp(a2*Math.pow(Tr,gamma));
    b=a*(b1*Tr-b2);
    c=c1/Tr*Math.exp(c2*Math.pow(Tr,delta));
    d=d1/Tr*Math.exp(d2*Math.pow(Tr,epsilon));
    e=1.3088;
    f=f1*Math.exp(f2*Math.pow(Tr,xsi));
    System.out.println("a="+a+"b="+b+"c="+c+"d="+d+"e="+e+"f="+f);
    Z2=Z1*(1.0+a*Math.pow(Pr,e)/(b*Math.pow(Pr,f)+1.0/(1.0+c*Math.pow(Pr,d))));
   }
   Y=Z2/Z1;
   System.out.println("Z2="+Z2+"Y="+Y);
   double FP=(1.0+(FP0-1.0)/(Y*Y*Y))/FP0;
   double lnY=Math.log(Y);
   double FQ=(1.0+(FQ0-1.0)*(1.0/Y-0.007*lnY*lnY*lnY*lnY))/FQ0;
   double eta=Z2*FP*FQ/eps;
   System.out.println("FP="+FP+"FQ="+FQ+"eta="+eta);
   return eta;
 }
public static void main(String arg[])
 {double Tc=469.7;  //Degree K
  double Pc=33.7e2;  //kPa
  double Zc=0.26;
  double mu=0;   //Debyes
  double M=72.151;//kg/kmol
  double T=500;  //Degree K
  double P=101e2;//kPa
  double eta=visc("c5h12",T,P,Tc,Pc,Zc,M,mu);
  double  eta1=5.46e-5;  //Pas
  System.out.println("viscosity = "+eta+" Pas");
  double error=(eta1-eta)/eta1*100;
   System.out.println("error = "+error+"%");
  }}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" viscosity2
eps=55503.23718079557
Tr=1.0645092612305727Pr=2.9970326409495547
mur=0.0
FP0=1.0
Z1=0.6399546013525853eta0=1.153004101847258E-5
viscosity value by using class Gas=1.3087179810183268E-5
a=0.18568461945210255b=0.09049039283579664c=0.5630321346079307d=6.512794946711124e=1.3088f=0.7789620201174583
Z2=2.973712898540491Y=4.646756023404405
FP=1.0FQ=1.0eta=5.357728755268732E-5
viscosity = 5.357728755268732E-5 Pas
error = 1.8730997203528892%

> Terminated with exit code 0.
```

VISCOSITY OF LIQUIDS

Viscosity of liquids does not have any general theory like collision theory of gases, Therefore equations to describe liquid viscosities are usually based on a reference point viscosity and a form of curve fitting.

Przezdziecki and Sridar Method[132]:

$$\eta_L = \frac{V_0}{E(V - V_0)}$$

Where η_L liquid viscosity, Pas

T_c critical temperature, degree K

V_c critical volume m^3/kmol

M molecular weight kg/kmol

T_{fP} freezing temperature degree K (at 101.325 kPa)

ω acentric factor

V_m liquid molar volume at T_{fP} m3/kmol

V^R reference liquid molar volume m3/kmol at reference temperature T^R degree K

At any other temperature

$$V(T) = \frac{f(T)}{f(T^R)} V^R$$

Where

$$f(T) = H_1(1 - \omega H_2)$$

$$H_1 = 0.33593 - 0.33953T_r + 1.51941T_r^2 - 2.02512T_r^3 + 1.11422T_r^4$$

$$H_2 = 0.29607 - 0.0904T_r - 0.048421T_r^2$$

V_0 m^3/kmol

$$V_0 = (0.0085\omega T_c - 2.02) * 10^{-3} + \frac{V_m}{0.342\frac{T_{fp}}{T_c} + 0.894}$$

$$E = -1.12x10^{-3} + \frac{V_c}{12.94 + 0.1M - 0.23x10^{-2}P_c + 0.0424T_{fP} - 11.58\frac{T_{fP}}{T_c}}$$

EXAMPLE: Estimate viscosity of Toluene at 383 K. Experimental value is 0.249e-5

```
public class viscosity_liquid
{ public static double viscosity(double T, double Tfp,double Tc,double  Pc,double Vc,double VL,double TL,double w,double M)
  { // T temperature
    // Tc,Pc,Vc specific temperature degree K, specific pressure kPa, specific volume m^3/kmol
    // w acentric factor
    // Vm liquid molar volume
    // Tfp freezing point degree K
    // VL(TL) volume m^3/kmolat the reference temperature TL
  if_x f=x->{double Tr=x/Tc;
              double H1=0.33593-0.33953*Tr+1.51941*Tr*Tr-2.02512*Tr*Tr*Tr+1.11422*Tr*Tr*Tr*Tr;
              double H2=0.29607-0.09045*Tr-0.04842*Tr*Tr;
              double ff= H1*(1.0-w*H2);
              System.out.println("T="+x+"Tr="+Tr+"H1="+H1+"H2="+H2+"ff="+ff);
              return ff;
             };
             double fL=f.func(TL);
             double ff=f.func(T);
             double ff1=f.func(Tfp);
             System.out.println("fL="+fL+"ff="+ff+"ff1="+ff1);
             double Vr=VL/Vc;
             double Vm=ff1/fL*VL;
             double V=ff/fL*VL;
             double V0=(0.0085*w*Tc-2.02)*1e-3+Vm/(0.342*(Tfp/Tc)+0.894);
             double E=-1.12*1e-3+Vc/(12.94+0.1*M-0.23*Pc*1e-2+0.0424*Tfp-11.58*Tfp/Tc);
             double eta=V0/(E*(V-V0));
             System.out.println("Vm="+Vm+"Vr="+Vr+"V0="+V0+"V="+V+"E="+E+"eta="+eta*1e-5);
            return eta*1e-5;
```

```
  }
  public static void main(String arg[])
  { double T=383;
    double Tc=591.75;
    double Pc=41.08e2; //kPa
    double Vc=316e-3; //m^3/kmol
    double Tfp=178.0;  //degree K
    double M=92.14; //kg/kmol
    double w=0.264;
    //reference liquid volume VL at temperature TL
    double VL=106.87e-3;  //m^3/kmol
    double TL=298.15;//Degree K

    viscosity(T, Tfp,Tc,Pc,Vc,VL,TL,w,M);
  }
}
```

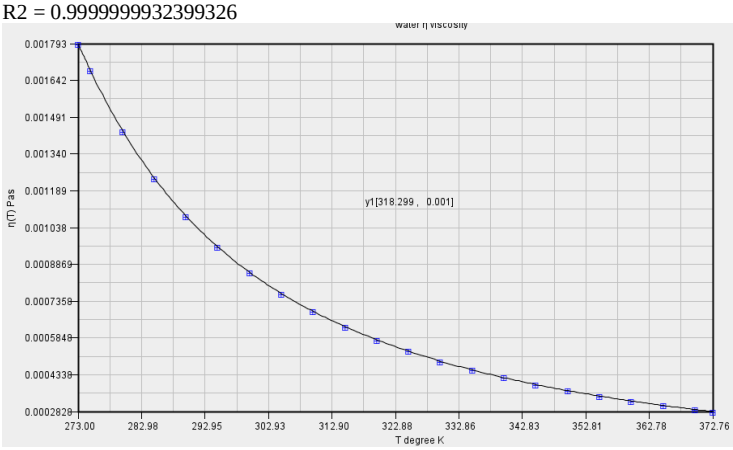
```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" viscosity_liquid
T=298.15Tr=0.503844528939586H1=0.3633572080734229H2=0.23820539459905768ff=0.34050704523122394
T=383.0Tr=0.6472327841149134H1=0.3991255498803614H2=0.21724415907254532ff=0.37623471854605606
T=178.0Tr=0.30080270384452895H1=0.3252819633210632H2=0.2644812440865449ff=0.302569785039813
fL=0.34050704523122394ff=0.37623471854605606ff1=0.302569785039813
Vm=0.09496318322943087Vr=0.33819620253164556V0=0.0945688058770237V=0.11808332583460446E=0.0177237288907065 5eta=0.002269115956445087

> Terminated with exit code 0.
```

Viscosity of some liquids utilised as heat transfer fluids are listed below as adta and polynomial least square curve fitting coefficients

Water $\eta(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa

c[0] = 3.2369615827675666
c[1] = -0.04741447156439535
c[2] = 2.5006856595267827E-4
c[3] = -3.8808225971981284E-7
c[4] = -1.4438559160868313E-9
c[5] = 7.629200587234908E-12
c[6] = -1.3833661670702365E-14
c[7] = 1.0858773920300192E-17
c[8] = -2.644656439331438E-21
R2 = 0.9999999932399326

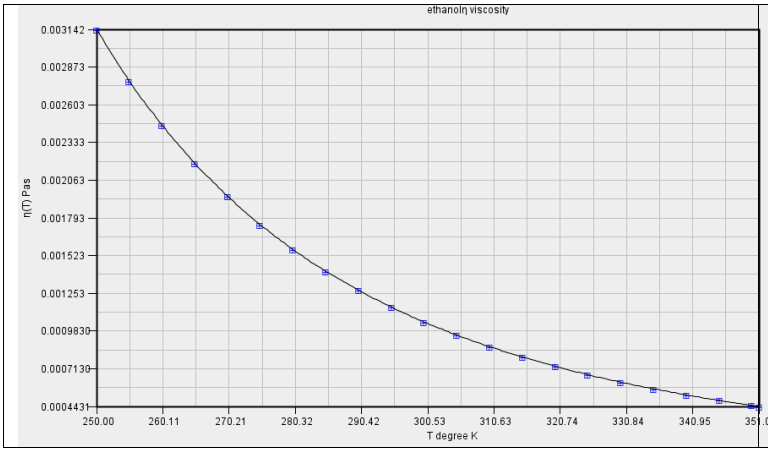


water	
T K	η Pas
273	0.0017912
275	0.0016819
280	0.0014336
285	0.0012392
290	0.001084
295	0.00095781
300	0.00085374
305	0.00076679
310	0.00069333
315	0.00063066
320	0.00057673
325	0.00052997
330	0.00048915
335	0.0004533
340	0.00042163
345	0.00039353
350	0.00036847
355	0.00034603
360	0.00032586
365	0.00030765
370	0.00029117
372.76	0.00028275

Ethanol $\eta(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa

c[0] = 0.8587749945994518
c[1] = -0.01151949693488959
c[2] = 5.237066298050750 4E-5
c[3] = -4.868894667132381E-8
c[4] = -2.7272544174105575E-10
c[5] = 3.930444294481756E-13
c[6] = 1.9957794870453912E-15
c[7] = -5.99395752171059E-18
c[8] = 4.714167908605965E-21
R2 = 0.9999999776274365

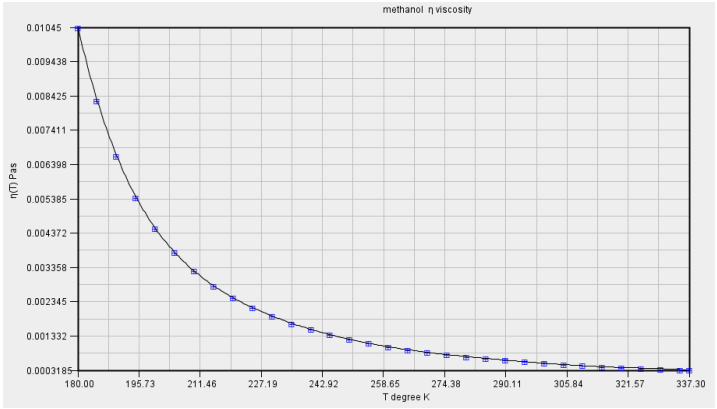
Ethanol	
T K	η Pas
250	0.0031425
255	0.0027725
260	0.0024557
265	0.0021831
270	0.0019476
275	0.0017432
280	0.0015653
285	0.0014097



290	0.0012733
295	0.0011532
300	0.0010472
305	0.00095329
310	0.0008699
315	0.00079561
320	0.00072926
325	0.00066983
330	0.00061645
335	0.0005684
340	0.00052503
345	0.00048578
350	0.00045018
351.05	0.00044311

Methanol $\eta(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa

c[0] = 8.787212577205471
c[1] = -0.2092606755080141
c[2] = 0.002057479893776877
c[3] = -1.0420086172935875E-5
c[4] = 2.6119007262147922E-8
c[5] = -1.2783581716733907E-11
c[6] = -9.33147226638734E-14
c[7] = 2.1864816507056146E-16
c[8] = -1.54414274924783E-19
R2 = 0.9999766176756644

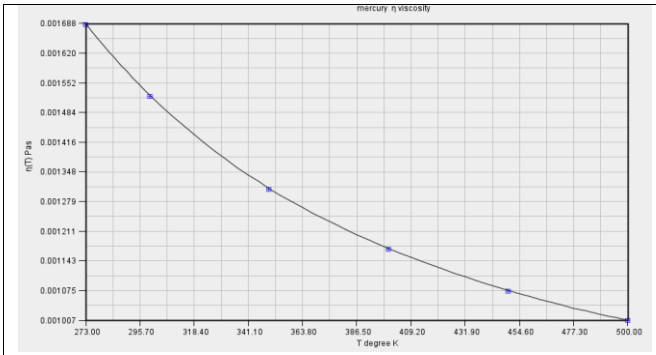


Methanol	
T K	ηPas
180	0.010451
185	0.00828
190	0.006646
195	0.005428
200	0.00451
205	0.003805
210	0.003254
215	0.002813
220	0.002455
225	0.00216
230	0.001912
235	0.001703
240	0.001524
245	0.001371
250	0.001237
255	0.001121
260	0.001019
265	0.00093
270	0.000851
275	0.000781
280	0.000719
285	0.000663
290	0.000614
295	0.000569
300	0.000529
305	0.000493
310	0.000461
315	0.000431
320	0.000404
325	0.000379
330	0.000357
335	0.000336
337.3	0.000327

Mercury $\eta(T) = \sum_{i=0}^{5} c[i]T^i$ P=100 kPa

c[0] = 0.008505006995799922
c[1] = -5.40488715893246E-5
c[2] = 1.5317086852741352E-7
c[3] = -1.9124205008472784E-10
c[4] = 6.518688865086575E-14
c[5] = 3.4073196380036E-17
R2 = 1.0

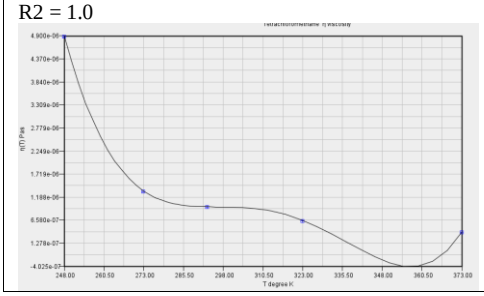
Mercury	
T K	ηPas
273	0.001688
300	0.001523
350	0.001309
400	0.001171
450	0.001075
500	0.001007
550	9.53E-04
600	9.11E-04



Tetrachloromethane $\eta(T) = \sum_{i=0}^{5} c[i]T^i$

P=100 kPa

c[0] = 0.002006081246907952
c[1] = -2.2961698656332422E-5
c[2] = 8.836082985402947E-8
c[3] = -9.621928279095551E-11
c[4] = -1.3984952656049176E-13
c[5] = 2.722702487799705E-16
R2 = 1.0



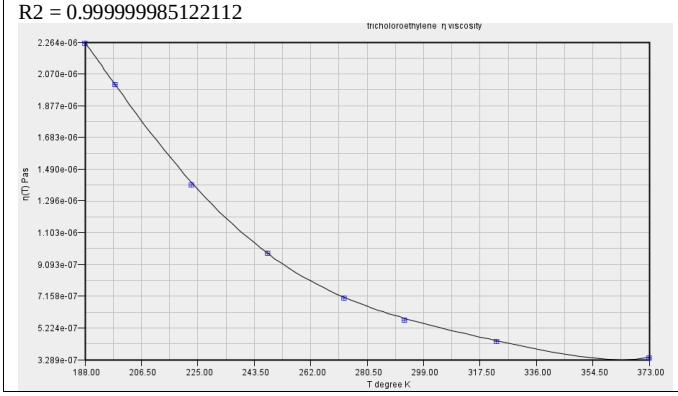
tetrachloromethane	
T K	ηPas
248	4.90E-06
273	1.33E-06
293	9.65E-07
323	6.51E-07
373	3.83E-07

Caution: minimum number of data, it is not given a reliable curve fitting

tricholoroethylene $\eta(T) = \sum_{i=0}^{5} c[i]T^i$

P=100 kPa

c[0] = -4.535756933253732E-5
c[1] = 1.0411625211245255E-6
c[2] = -8.417103562485085E-9
c[3] = 3.207435189116408E-11
c[4] = -5.899381170955896E-14
c[5] = 4.235999293526989E-17
R2 = 0.999999985122112

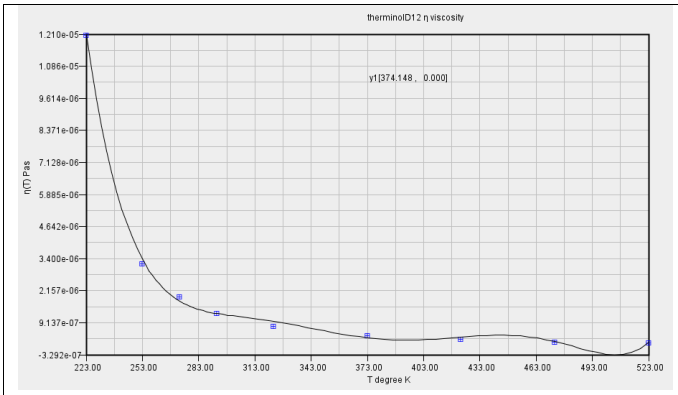


tricholoroethylene	
T K	ηPas
188	2.26E-06
198	2.01E-06
223	1.40E-06
248	9.81E-07
273	7.10E-07
293	5.76E-07
323	4.46E-07
373	3.43E-07

therminolD12 $\eta(T) = \sum_{i=0}^{6} c[i]T^i$ P=100 kPa

c[0] = 0.003656505934251059
c[1] = -5.931052164281876E-5
c[2] = 3.9740280627509773E-7
c[3] = -1.406391140255445E-9
c[4] = 2.771119168866921E-12
c[5] = -2.881775188185996E-15
c[6] = 1.2356590909473694E-18
R2 = 0.9999984310265598

therminolD12	
T K	ηPas
188	2.96E-04
223	1.21E-05
253	3.23E-06
273	1.93E-06
293	1.29E-06
323	8.00E-07
373	4.50E-07



423	2.90E-07
473	2.00E-07
523	1.50E-07

therminolVLT $\eta(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa

c[0] = 0.0197443635549003 3

c[1] = -5.135647340116697E-4

c[2] = 5.769374224336889E-6

c[3] = -3.654442953889881E-8

c[4] = 1.427396798688836E-10

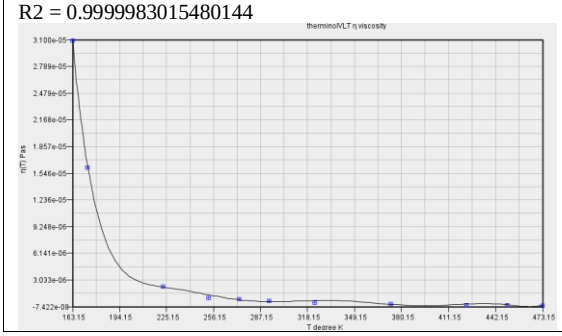
c[5] = -3.520871171897029E-13

c[6] = 5.357391053339772E-16

c[7] = -4.599354638935805E-19

c[8] = 1.706426123631801E-22

R2 = 0.9999983015480144



therminolVLT	
T K	ηPas
163.15	3.10E-05
173.15	1.62E-05
223.15	2.32E-06
253.15	1.03E-06
273.15	8.76E-07
293.15	6.61E-07
323.15	4.60E-07
373.15	2.77E-07
423.15	1.77E-07
473.15	1.18E-07

glycerine $\eta(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa

c[0] = -3.235967041975914

c[1] = 0.073125126401012

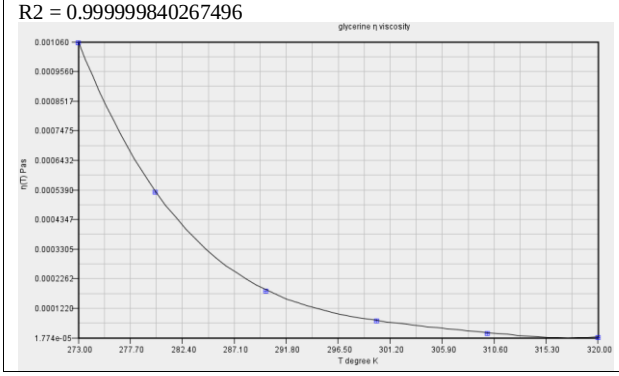
c[2] = -6.087265216698797E-4

c[3] = 2.411932747732771E-6

c[4] = -4.624252159889307E-9

c[5] = 3.4642605018161598E-12

R2 = 0.999999840267496



glycerine	
T K	ηPas
273	0.00106
280	5.34E-04
290	1.85E-04
300	7.99E-05
310	3.52E-05
320	2.10E-05

xceltherm_MK1 $\eta(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa

c[0] = -3.551838283353483

c[1] = 0.0660782511039661

c[2] = -5.223568245930484E-4

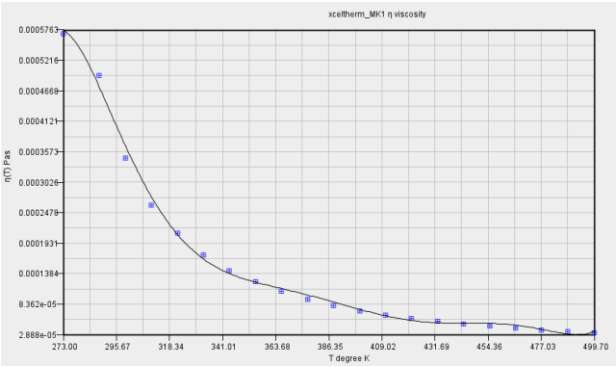
c[3] = 2.2773256165820594E-6

c[4] = -5.922275731055917E-9

c[5] = 9.209456156026099E-12

xceltherm_MK1	
T K	ηPas
273	5.71E-04
288.6	4.96E-04
299.7	3.47E-04
310.8	2.63E-04

c[6] = -7.979618512654582E-15
c[7] = 3.0582146124692673E-18
c[8] = -1.0951200853428187E-22
R2 = 0.9995688341054612

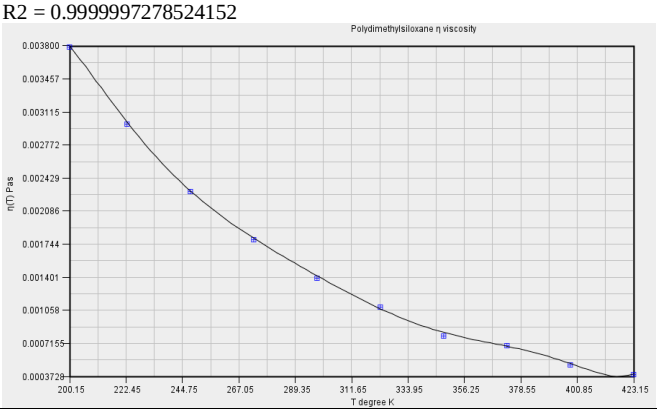


321.9	2.12E-04
333	1.73E-04
344.1	1.45E-04
355.2	1.25E-04
366.3	1.08E-04
377.4	9.36E-05
388.6	8.22E-05
399.7	7.30E-05
410.8	6.54E-05
421.9	5.91E-05
433	5.37E-05
444.1	4.90E-05
455.2	4.49E-05
466.3	4.14E-05
477.4	3.83E-05
488.6	3.54E-05
499.7	3.30E-05

Polydimethylsiloxane

$\eta(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa

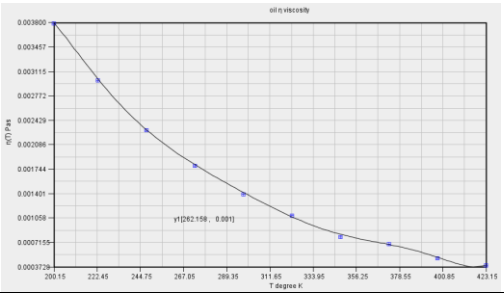
c[0] = 0.2670546672322028
c[1] = -0.012612199866618972
c[2] = 2.1819514146226673E-4
c[3] = -1.942134413979056E-6
c[4] = 1.0088386887160388E-8
c[5] = -3.192903280917403E-11
c[6] = 6.081159766939502E-14
c[7] = -6.418368139937804E-17
c[8] = 2.888263566503347E-20
R2 = 0.9999997278524152



Polydimethylsiloxane	
T K	η Pas
200.15	0.0038
223.15	0.003
248.15	0.0023
273.15	0.0018
298.15	0.0014
323.15	0.0011
348.15	8.00E-04
373.15	7.00E-04
398.15	5.00E-04
423.15	4.00E-04

Oil $\eta(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa

c[0] = 0.2670546672322028
c[1] = -0.012612199866618972
c[2] = 2.1819514146226673E-4
c[3] = -1.942134413979056E-6
c[4] = 1.0088386887160388E-8
c[5] = -3.192903280917403E-11
c[6] = 6.081159766939502E-14
c[7] = -6.418368139937804E-17
c[8] = 2.888263566503347E-20
R2 = 0.9999997278524152



oil	
T K	ηPas
200.15	0.0038
223.15	0.003
248.15	0.0023
273.15	0.0018
298.15	0.0014
323.15	0.0011
348.15	8.00E-04
373.15	7.00E-04
398.15	5.00E-04
423.15	4.00E-04

11.5.2 THERMAL CONDUCTIVITY

Thermal conductivity is defined by Fourier's low of conduction as:

$$q = -k(T)\frac{\partial T}{\partial x}$$

Therefore the SI unit of thermal conductivity is W/(mK)

THERMAL CONDUCTIVITY OF LOW PRESSURE GASES

Method of Chung et. Al.[121],[122]

$$\frac{k(T)M}{\eta(T)C_v(T)} = \frac{3.75\psi}{C_v(T)/R}$$

Where

k Thermal conductivity W/(mK)

M molecular weight kg/kmol

η low pressure gas viscosity Ns/m^2 Pas

C_v heat capacity at constant volume kJ/(kmolK)

R universal gas constant 8.3145 kJ/(kmolK)

$$\psi = 1 + \alpha\{[0.215 + 0.28288\alpha - 1.061\beta + 0.26665Z]/[0.6366 + \beta Z + 1.061\alpha\beta]\}$$

$$\alpha = \frac{C_v(T)}{R} - 3/2$$

$$\beta = 0.7862 - 0.7109\omega + 1.3168\omega^2$$

$$Z = 2.0 + 10.5T_r^2$$

Term β is an emprical correlation and apply only for non-polar gases.

EXAMPLE: Use Chung Method to estimate the thermal conductivity of isopentane vapor at 100 kPa and 373.15 K. The tabulated value is 0.022 W/(mK)

```
public class thermal_conductivity
{
//Collusion integral
public static double omegav(double e_k,double T)
 {double Ts=T/e_k;
  double Ov=1.16145*Math.pow(Ts,-0.14874)+0.52487*Math.exp(-0.7732*Ts)+2.161178*Math.exp(-2.43787*Ts);
  return Ov;
 }
 public static double omegav(double Ts)
 {
  double Ov=1.16145*Math.pow(Ts,-0.14874)+0.52487*Math.exp(-0.7732*Ts)+2.161178*Math.exp(-2.43787*Ts);
  return Ov;
 }
//reduced dipole moment Chun at al
 public static double mur(double Vc,double Tc,double mu)
 {  //mu : is dipole moment in debyes
    //Vc cm^3/mole
    //Tc degree K
              double m=131.3*mu/Math.sqrt(Vc*1e3*Tc);
              return m;
 }
 public static double Fc(double Vc,double Tc,double mu,double omega)
 {// omega pitser' acentycity factor
 double mr=mur(Vc,Tc,mu);
  double F=1.0-0.2756*omega+0.059035*mr*mr*mr*mr;
  return F;
 }
 //Method of Chung et. al.
 public static double visc(double M,double T,double Vc,double Tc,double mu,double w)
 { double e_k=Tc/1.2503;
   return visc(M,T,Vc,Tc,mu,w,e_k);
 }
  public static double visc(double M,double T,double Vc,double Tc,double mu,double w,double e_k)
```

```
 {  double Tr=T/Tc;
    double Ts=1.2593*Tr;
              double Ov=omegav(Ts);
              double F=Fc(Vc,Tc,mu,w);
   double eta=40.785e-7*F*Math.sqrt(M*T)/Math.pow(Vc*1e3,(2.0/3.0))/Ov;
   return eta;
 }
 //Method of chang
 public static double thermal_conductivity(String name,double T,double P,double Tc,double Pc,double Vc,double Zc,double M,double
mu,double w)
 {   double R=8.31446261815324; // Universal gas constant  J/(kmolK)
 double visc=visc(M,T,Vc,Tc,mu,w);
 double Tr=T/Tc;
  Gas g=new Gas(name);
  double Cv=g.Cv(T); //kJ/(kmolK);
  double alpha=Cv/R-3.0/2.0;
  double beta=0.7862-0.7109*w+1.3168*w*w;
  double Z=2.0+10.5*Tr*Tr;
  double fi=1.0+alpha*((0.215+0.28288*alpha-1.061*beta+0.26665*Z)/(0.6366+beta*Z+1.061*alpha*beta));
  double k2=visc*Cv/(M*1.0e-3)*3.75*fi/(Cv/R);
  return k2;
 }
 public static void main(String arg[])
 {//isopentane
   double mu=0.29;  //dipole momentdouble Tc=460.39;
   double Tc=460.39; //degree  K
   double Pc=33.81e2;  //kPa
   double Vc=308.3e-3;// m^3/kmol
   double Zc=0.272;
   double w=0.229;
   double M=72.151; //kg/kmol
   double T=100+273.15;  //Degree K
   double P=100.0;  //kPa
   double eta=visc(M,T,Vc,Tc,mu,w);
   System.out.println("eta  ="+eta+"Pas");
   String name="isoc5h12";
   double k1=0.022;
   double k= thermal_conductivity(name,T,P,Tc,Pc,Vc,Zc,M,mu,w);
   System.out.println("k="+k+"W/mK");
   double error=(k-k1)/k1*100.0;
    System.out.println("error="+error+"%");
 }}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" thermal_conductivity
eta  =8.718304039219677E-6Pas
k=0.022878825484233485W/mK
error=3.994661291970393%

> Terminated with exit code 0.
```

Method of Stiel and Thodos[120]

In this approach a reference thermal conductivity $k^0(T_0)$ is used

$$k - k^0 = \frac{1.22x10^{-2}[\exp(0.535\rho_r) - 1]}{\Gamma Z_c^5} \qquad \rho_r < 0.5$$

$$k - k^0 = \frac{1.14x10^{-2}[\exp(0.67\rho_r) - 1.069]}{\Gamma Z_c^5} \qquad 0.5 \le \rho_r < 2.0$$

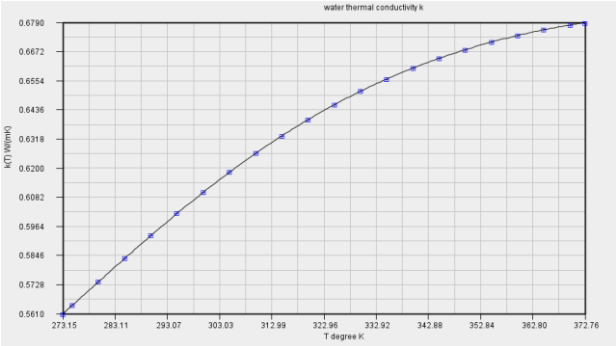
$$k - k^0 = \frac{2.6x10^{-3}[\exp(1.1555\rho_r) - 2.016]}{\Gamma Z_c^5} \qquad 2.0 \le \rho_r < 2.8$$

Where Z_c is critical compressibility $\rho_r = \rho/\rho_c$ is reduced density

Thermal conductivities of some liquids utilised as heat transfer fluids are listed below as adta and polynomial least square curve fitting coefficients

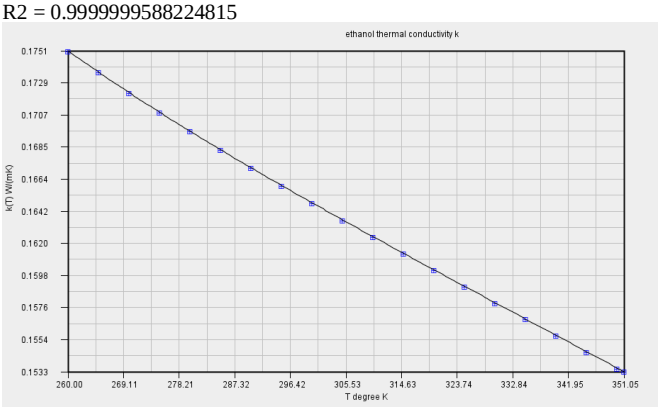
Water $k(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa c[0] = 11.229944896858548	water	
	T K	k W/(mK)

c[1] = -0.12861656653185466
c[2] = 4.147012296697086E-4
c[3] = 8.022325466138565E-7
c[4] = -8.82687351121519E-9
c[5] = 2.6895062231979395E-11
c[6] = -4.7889333260608144E-14
c[7] = 5.55299667486651E-17
c[8] = -3.21361483887385E-20
R2 = 0.9999999667614299



273.15	0.561
275	0.56459
280	0.57409
285	0.58352
290	0.59277
295	0.60174
300	0.61032
305	0.61845
310	0.62609
315	0.63319
320	0.63975
325	0.64575
330	0.65121
335	0.65615
340	0.66058
345	0.66453
350	0.66803
355	0.6711
360	0.67378
365	0.67608
370	0.67803
372.76	0.67897

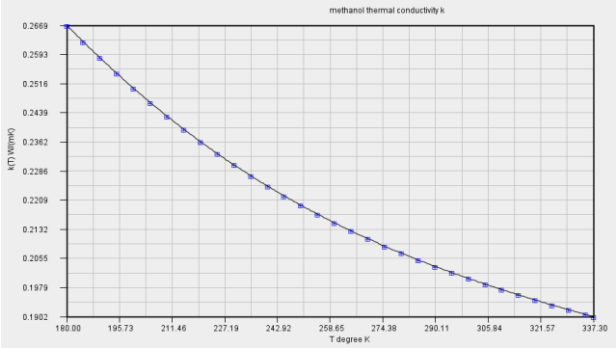
Ethanol $k(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa
c[0] = 1.0760566691313826
c[1] = -0.011213876773231022
c[2] = 5.1168526714428544E-5
c[3] = -7.687011921684678E-8
c[4] = -1.1561943522927112E-10
c[5] = 3.713139069106359E-13
c[6] = 3.3529187255852976E-16
c[7] = -1.7117423428994304E-18
c[8] = 1.3310081197651189E-21
R2 = 0.9999999588224815



Ethanol	
T K	k W/(mK)
260	0.17508
265	0.17363
270	0.17224
275	0.17091
280	0.16961
285	0.16835
290	0.16712
295	0.16592
300	0.16474
305	0.16358
310	0.16243
315	0.1613
320	0.16017
325	0.15905
330	0.15794
335	0.15683
340	0.15572
345	0.15461
350	0.15349
351.05	0.15326

Methanol $k(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa
c[0] = -1.1676656921063362
c[1] = 0.04237909284592859
c[2] = -4.7696819194285265E-4
c[3] = 2.725189360502254E-6
c[4] = -8.308081263375004E-9
c[5] = 1.0841950428513975E-11
c[6] = 6.728551557261495E-15
c[7] = -3.642098433903119E-17
c[8] = 3.064879214173367E-20
R2 = 0.9999999840080913

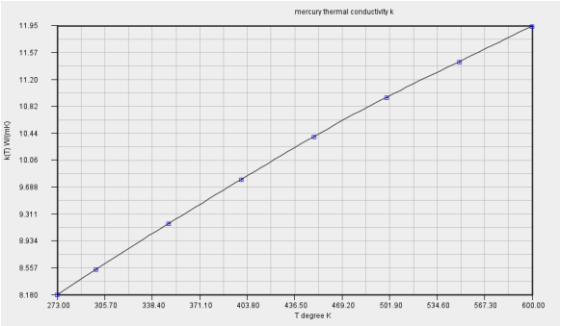
Methanol	
T K	k W/(mK)
180	0.26692
185	0.26257
190	0.25833
195	0.25425
200	0.25033
205	0.24657
210	0.24298
215	0.23954
220	0.23626
225	0.23312
230	0.23013
235	0.22728
240	0.22456



245	0.22197
250	0.21949
255	0.21714
260	0.21489
265	0.21274
270	0.2107
275	0.20875
280	0.20689
285	0.20512
290	0.20342
295	0.20179
300	0.20023
305	0.19873
310	0.19729
315	0.1959
320	0.19455
325	0.19324
330	0.19198
335	0.19074
337.3	0.19018

Mercury $k(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa

c[0] = 40.40652157105432
c[1] = -0.5862364200619486
c[2] = 0.004007675096782129
c[3] = -1.316525515167458E-5
c[4] = 1.8853441012468423E-8
c[5] = 5.842730080054613E-12
c[6] = -5.750321096006963E-14
c[7] = 7.214238680206064E-17
c[8] = -3.037447526199056E-20
R2 = 0.9999999999999988

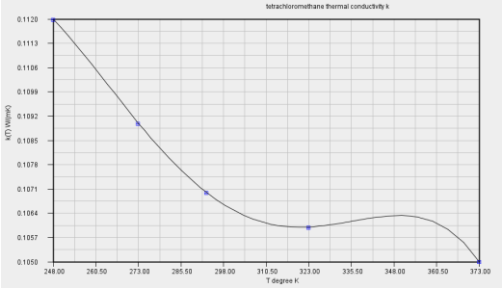


Mercury	
T K	k W/(mK)
273	8.18
300	8.54
350	9.18
400	9.8
450	10.4
500	10.95
550	11.45
600	11.95

tetrachloromethane $k(T) = \sum_{i=0}^{8} c[i]T^i$

P=100 kPa

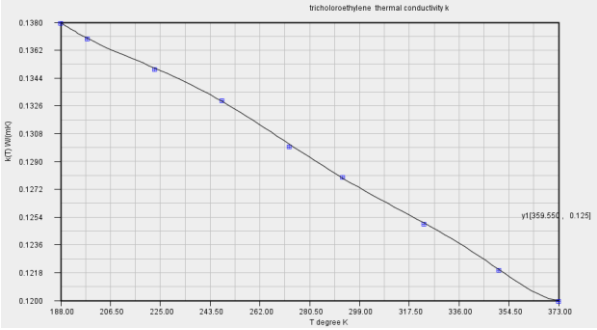
c[0] = -0.8141158478233231
c[1] = 0.011706418712803857
c[2] = -4.8521156838611155E-5
c[3] = 5.314836814018988E-8
c[4] = 9.912751222541551E-11
c[5] = -1.902487938086665E-13
R2 = 1.0



tetrachloromethane	
T K	k W/(mK)
248	0.112
273	0.109
293	0.107
323	0.106
373	0.105

tricholoroethylene $k(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa

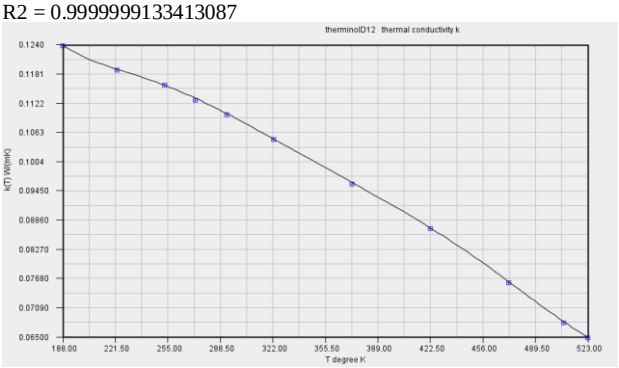
c[0] = -2.7835600553588056
c[1] = 0.08737307094119164
c[2] = -0.0010746658959795397
c[3] = 7.039275598099097E-6
c[4] = -2.6217012004493298E-8
c[5] = 5.3385933759648315E-11
c[6] = -4.6024274605252756E-14
c[7] = -1.2799107795263896E-17
c[8] = 3.565263495233575E-20
R2 = 0.9999999837055702



tricholoroethylene	
T K	k W/(mK)
188	0.138
198	0.137
223	0.135
248	0.133
273	0.13
293	0.128
323	0.125
373	0.12

therminolD12 $k(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa

c[0] = 1.7785726184755815
c[1] = -0.03442917140888797
c[2] = 3.017183037983274E-4
c[3] = -1.4466393887201425E-6
c[4] = 4.1254104297888803E-9
c[5] = -7.139065259533781E-12
c[6] = 7.27705265830623E-15
c[7] = -3.962467634780938E-18
c[8] = 8.746409524467002E-22
R2 = 0.9999999133413087

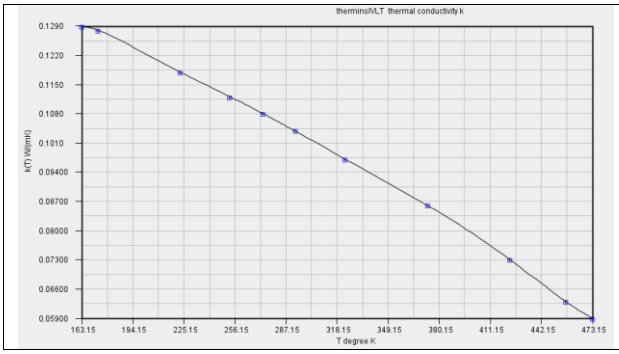


therminolD12	
T K	k W/(mK)
188	0.124
223	0.119
253	0.116
273	0.113
293	0.11
323	0.105
373	0.096
423	0.087
473	0.076
523	0.065

therminolVLT $k(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa

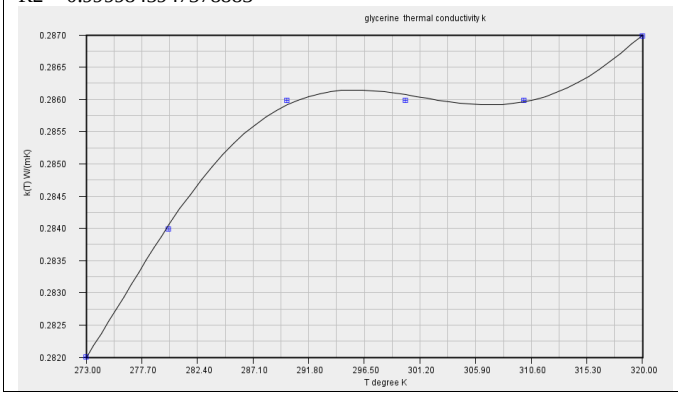
c[0] = -3.3094565127749
c[1] = 0.09323713485579183
c[2] = -0.0010746196365681155
c[3] = 6.897308599909250 5E-6
c[4] = -2.701773230564594E-8
c[5] = 6.613510996986178E-11
c[6] = -9.878632735162683E-14
c[7] = 8.230579952675473E-17
c[8] = -2.927426916859929E-20
R2 = 0.9999999831738312

therminolVLT	
T K	k W/(mK)
163.15	0.129
173.15	0.128
223.15	0.118
253.15	0.112
273.15	0.108
293.15	0.104
323.15	0.097
373.15	0.086
423.15	0.073
473.15	0.059



glycerine $k(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa

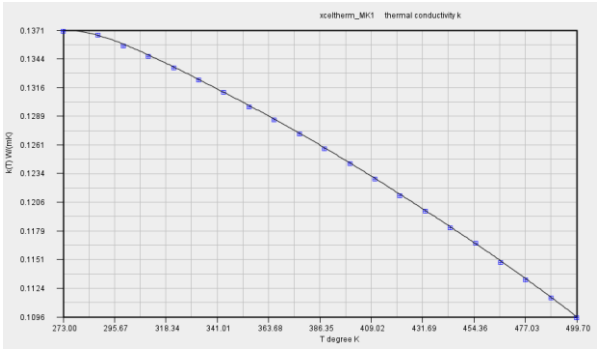
c[0] = 446.96203734260894
c[1] = -7.507937677189395
c[2] = 0.05037181103909041
c[3] = -1.6862762409047797E-4
c[4] = 2.81686036890841E-7
c[5] = -1.878472017767546E-10
R2 = 0.9999845947578883



glycerine	
T K	k W/(mK)
273	0.282
280	0.284
290	0.286
300	0.286
310	0.286
320	0.287

xceltherm_MK1 $k(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa

c[0] = -8.461978722910265
c[1] = 0.1439600498115204 7
c[2] = -9.940429024835088E-4
c[3] = 3.5705151640896126E-6
c[4] = -6.640839806301446E-9
c[5] = 4.1129742690572595E-12
c[6] = 6.287145352105836E-15
c[7] = -1.2488555902561936E-17
c[8] = 6.307862356753185E-21
R2 = 0.999999112050507



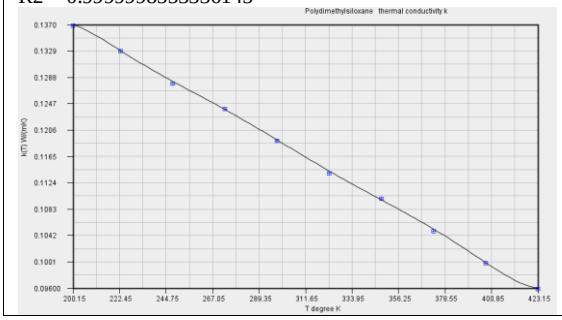
xceltherm_MK1	
T K	k W/(mK)
273	0.1371
288.6	0.1367
299.7	0.1357
310.8	0.1347
321.9	0.1336
333	0.1324
344.1	0.1312
355.2	0.1298
366.3	0.1286
377.4	0.1272
388.6	0.1258
399.7	0.1244
410.8	0.1229
421.9	0.1213
433	0.1198
444.1	0.1182
455.2	0.1167
466.3	0.1149
477.4	0.1132
488.6	0.1115
499.7	0.1096

Polydimethylsiloxane $k(T) = \sum_{i=0}^{8} c[i]T^i$
P=100 kPa

c[0] = -24.37423722838234
c[1] = 0.6366551764891148
c[2] = -0.0071145694130173125

Polydimethylsiloxane	
T K	k W/(mK)
200.15	0.137
223.15	0.133
248.15	0.128

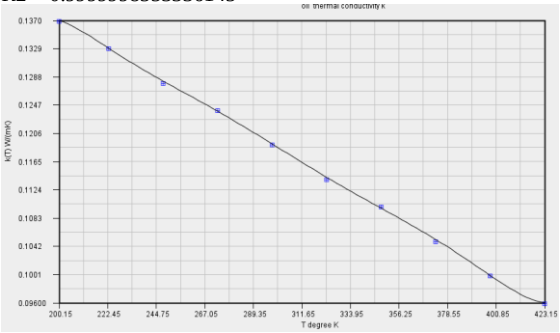
c[3] = 4.4674817550186125E-5
c[4] = -1.7231993282655374E-7
c[5] = 4.176604722370718E-10
c[6] = -6.203654402531216E-13
c[7] = 5.15350803245629E-16
c[8] = -1.8284968858948645E-19
R2 = 0.9999998553336145



273.15	0.124
298.15	0.119
323.15	0.114
348.15	0.11
373.15	0.105
398.15	0.1
423.15	0.096

Oil $k(T) = \sum_{i=0}^{8} c[i]T^i$ P=100 kPa

c[0] = -24.37423722838234
c[1] = 0.6366551764891148
c[2] = -0.007114569413017 3125
c[3] = 4.4674817550186125E-5
c[4] = -1.7231993282655374E-7
c[5] = 4.176604722370718E-10
c[6] = -6.203654402531216E-13
c[7] = 5.15350803245629E-16
c[8] = -1.8284968858948645E-19
R2 = 0.9999998553336145



oil	
T K	k W/(mK)
200.15	0.137
223.15	0.133
248.15	0.128
273.15	0.124
298.15	0.119
323.15	0.114
348.15	0.11
373.15	0.105
398.15	0.1
423.15	0.096

11.5.3 SURFACE TENSION

The boundary between a liquid phase and a gas phase can be considered a third phase with properties distinct from those of the liquid and gas. A qualitative picture of the microscopic surface layer shows that there are unequal forces acting upon the molecules; i.e., at low gas densities, the surface molecules are attracted sidewise and toward the bulk liquid but experience little attraction in the direction of the bulk gas. Thus the surface layer is in tension and tends to contract to the smallest area compatible with the mass of material, container restraints, and external forces,e.g., gravity.
This tension can be presented in various quantitative ways; the most common is the surface tension σ, defined as the force exerted in the plane of the surface per unit length. Unit of surface tension is N/m. Surface tension is a property between two phases of fluids: liquids and gases, therefore it is a proprety specially important in phase change heat transfer.



Water droplet shaped as a sphere due to surface tension between water and air

Pitzer and Curl equation[119]:

$$\sigma = P_c^{2/3} T_c^{1/3} \frac{1.86 + 1.18\omega}{19.05} \left[\frac{3.74 + 0.91\omega}{0.291 - 0.08\omega}\right]^{2/3} (1 - T_r)^{11/9}$$

EXAMPLE Check surface tension of ethyl mercaptan at 303 K. The experimental value is 22.60x10^{-3} N/m

```
public class surface_tension
{
public static double sigma(double T,double Tc,double Pc,double w)
{ double Tr=T/Tc;
  double x1=(1.86+1.18*w)/19.05;
  double x2=(3.74+0.91*w)/(0.291-0.08*w);
  double x3=1.0-Tr;
   double sigma=1.0e-3*Math.pow((Pc*1.0e-2),(2.0/3.0))*Math.pow(Tc,(1.0/3.0))*x1*Math.pow(x2,(2.0/3.0))*Math.pow(x3,(11.0/9.0));
   return sigma;
}
/*
 public static void main(String arg[])
 {//water H2O
   double Tc=647.3; //degree  K
   double Pc=22.12e3;  //kPa
   double w=0.344;
   double T=100+273.15; //degree K
  steamIAPWS_IF97 st=new steamIAPWS_IF97();
  double sigma1=st.sigma(T);
  double sigma2=sigma(T,Tc,Pc,w);
  System.out.println("sigma1="+sigma1+"\nsigma2="+sigma2);
 }
 */
  public static void main(String arg[])
 {//ethyl metkaptan
   double Tc=499; //degree  K
   double Pc=54.9e2;  //kPa
   double w=0.192;
   double T=303; //degree K
  double sigma1=22.68e-3;
  double sigma2=sigma(T,Tc,Pc,w);
  System.out.println("sigma1="+sigma1+"\nsigma2="+sigma2);
 }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" surface_tension
sigma1=0.02268
sigma2=0.023486834242258378

> Terminated with exit code 0.
```

EXAMPLE Check surface tension of water at 373.15 K. Use steam EOS to find exact value

```
public static void main(String arg[])
 {//water H2O
   double Tc=647.096; //degree  K
    double Pc=22.064e3; //kPa critical pressure
   double T=100+273.15; //degree K
  steamIAPWS_IF97 st=new steamIAPWS_IF97();
   double Ts=0.7*Tc;
   double Ps=st.Ps(Ts);
   double Pr=Ps/Pc;
   double w=-Math.log10(Pr)-1;
   System.out.println("w="+w+"Ts="+Ts+"Ps="+Ps);
  double sigma1=st.sigma(T);
  double sigma2=sigma(T,Tc,Pc,w);
  System.out.println("sigma1="+sigma1+"\nsigma2="+sigma2);
 }
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" surface_tension
w=0.3443682785677622Ts=452.9672Ps=998.4262030460582
sigma1=0.058911868587664076
sigma2=0.08128871881092409

> Terminated with exit code 0.
```

As you see from the result. Equation is sucsessful for ethyl mercaptan, but result is nor correct for water. Water is a polar component and does not

Brock and Bird Equation[118]:

$$\sigma = P_c^{2/3} T_c^{1/3} Q (1 - T_r)^{11/9}$$

$$Q = 0.116 \left[1 + \frac{T_{br} ln(P_c/101.325)}{1 - T_{br}} \right] - 0.279$$

```
public class surface_tension1
{
public static double sigma(double T,double Tc,double Pc,double Tb)
{ double Tr=T/Tc;
  double Tbr=Tb/Tc;
  double x1=(Tbr*Math.log(Pc/101.325))/(1.0-Tbr);
  double x2=1.0+x1;
  double x3=0.1196*x2;
  double x4=x2-0.279;
  System.out.println("x1="+x1+"x2="+x2+"x3="+x3+"x4="+x4);
  double Q=0.1196*(1.0+(Tbr*Math.log(Pc/101.325))/(1.0-Tbr))-0.279;
  double x5=1.0-Tr;
   double sigma=1.0e-3*Math.pow((Pc*1.0e-2),(2.0/3.0))*Math.pow(Tc,(1.0/3.0))*Q*Math.pow(x5,(11.0/9.0));
   return sigma;
}

public static void main(String arg[])
 {//ethyl metkaptan
   double Tc=499; //degree  K
   double Tb=308.15;
   double Pc=54.9e2;  //kPa
   double T=303; //degree K
  double sigma1=22.68e-3;
  double sigma2=sigma(T,Tc,Pc,Tb);
  System.out.println("sigma1="+sigma1+"\nsigma2="+sigma2);
 }

}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" surface_tension1
x1=6.446123992910521x2=7.446123992910521x3=0.8905564295520982x4=7.167123992910521
sigma1=0.02268
sigma2=0.022360738672092047

> Terminated with exit code 0.
```

Zuo-Stenby Eqs[117]

In this approach two reference fluid is calculated and linear interpolation formulation applies to find actual chemicals surface tension value.

Two reference fluid is used. The first one is methane(CH_4), The second one is n-octane(C_8H_{18}).

Equations are used as follows: $T_r = \frac{T}{T_c}$

$$\sigma_r = ln\left(1 + \frac{\sigma x 10^3}{T_c^{1/3}(P_c x 10^{-2})^{2/3}}\right)$$

methane(CH_4) $\sigma^{(1)} = 40.52x10^{-3}(1 - T_r)^{1.287}$

n-octane(C_8H_{18}) $\sigma^{(2)} = 52.095x10^{-3}(1 - T_r)^{1.21458}$

$$\sigma_r = \sigma_r^{(1)} + \frac{\omega - \omega^{(1)}}{\omega^{(2)} - \omega^{(1)}}\left(\sigma_r^{(2)} - \sigma_r^{(1)}\right)$$

```
public class surface_tension2
{
public static double sigma(double T,double Tc,double Pc,double w)
{double Tr=T/Tc;
double x5=1.0-Tr;
// reference fluid: Methane CH4
 double Tc1=190.56; //degree K
 double Pc1=45.99e2;
 double w1=0.011;
//reference fluid n-octane
 double Tc2=568.70;
 double Pc2=24.90e2;
 double w2=0.399;
 double x1=(1.0-Tr);
  double sigma1=40.52*Math.pow(x1,1.287)*1.0e-3;
 double sigma2=52.095*Math.pow(x1,1.21548)*1.0e-3;
 double sigmar1=Math.log(1.0+sigma1*1.0e3/Math.pow(Tc1,(1.0/3.0))/Math.pow(Pc1*1.0e-2,(2.0/3.0)));
 double sigmar2=Math.log(1.0+sigma2*1.0e3/Math.pow(Tc2,(1.0/3.0))/Math.pow(Pc2*1.0e-2,(2.0/3.0)));
 double sigmar=sigmar1+(w-w1)/(w2-w1)*(sigmar2-sigmar1);
 double sigma= (Math.exp(sigmar)-1.0)*Math.pow(Tc,(1.0/3.0))*Math.pow(Pc*1.0e-2,(2.0/3.0))*1e-3;
 return sigma;
}

public static void main(String arg[])
 {//ethyl metkaptan
   double Tc=499; //degree  K
   double Tb=308.15;
   double Pc=54.9e2;  //kPa
   double w=0.192;
   double T=303; //degree K
  double sigma1=22.68e-3;
  double sigma2=sigma(T,Tc,Pc,w);
  System.out.println("sigma1="+sigma1+"\nsigma2="+sigma2);
 }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" surface_tension2
sigma1=0.02268
sigma2=0.022667516900779443

> Terminated with exit code 0.
```

11.5.4 SPECIFIC HEAT

Enthalpy is a part of Navier-Stokes equations, but in the differential form, it usually reduced to specific geat at constant pressure. Specific heat, Cp can be calculated from equation of states, or specially for ideal gases and liquids can be given as a curve fitting function of temperature. Equation of states (EOS) can usually be given as a function of temperature and specific volume(density). Most common forms of the equation of states are:

Pressure form $\boldsymbol{P(T,v)or\ P(T,\rho)}$

Helmholtz free energy form: $A(T,v)or\ A(T,\rho)$

Gibbs free energy form: $G(T,v)or\ G(T,\rho)$

Some of the Pressure form equations:

The most famous and simplest one is the **ideal gas EOS**

$P(T,v) = \frac{RT}{v} = \rho RT$ or $P(T,v) = \frac{\left(\frac{R}{M}\right)T}{v} = \rho\left(\frac{R}{M}\right)T$ where R is the universal gas constant (J/kmolK), M molecular weight (kg/kmol), v is specific volume (m^3/kmol or m^3/kg), ρ is density (kmol/ m^3 or kg/m^3)

Cubic Equation of State has a general form of equation

$P = \frac{RT}{V-b} - \frac{a}{V^2+ubV+wb^2}$ (2.2.1)

The equation can also be written in the following form:

$Z^3 - (1 + B^* - uB^*)Z^2 + (A^* + wB^{*2} - uB^* - uB^{*2})Z - A^*B^* - B^{*2} - wB^{*2} - wB^{*3} = 0$ (2.2.2)

where $A^* = \frac{aP}{R^2T^2}$ (2.2.3) and $B^* = \frac{bP}{RT}$ (2.2.4) $Z = \frac{PV}{RT}$ (2.2.5)

Some of the most common equation of states are: van der Waals , Redlich-Kwong (RK) ,Soave (SRK) , ve Peng-Robinson (PR). All these equations are the same form, but structures of the coefficients *u*, *w*, *a*, *b* are different

Tablo Coefficients of cubic equation of states

Denklem	u	w	b	a	fω
van der Waals	**0**	**0**	$\frac{RT_{crit}}{8P_{crit}}$	$\frac{27}{64}\frac{R^2T_{crit}^2}{P_{crit}}$	
Redlich Kwong	**1**	**0**	$\frac{0.08664RT_{crit}}{P_{crit}}$	$0.42748\frac{R^2T_{crit}^{2.5}}{P_{crit}T^{1/2}}$	
Soawe	**1**	**0**	$\frac{0.08664RT_{crit}}{P_{crit}}$	$\frac{0.42748R^2T_{crit}^2}{P_{crit}}[1+f\omega(1-T_r^{0.5})]^2$	$0.48 + 1.574\omega - 0.176\omega^2$
Peng Robinson	**2**	**-1**	$\frac{0.0780RT_{crit}}{P_{crit}}$	$\frac{0.45724R^2T_{crit}^2}{P_{crit}}[1+f\omega(1-T_r^{0.5})]^2$	$0.37464 + 1.54226\omega - 0.269992\omega^2$

ωin Peng-Robinson and Soawe equation of states coefficient is called **Pitzer's accentric factor**. This factor is calculated as

$$\omega = -log_{10}P_{saturated\ vapor}(at\ T_r = 0.7) - 1$$

To obtain values of ω, the reduced vapor pressure ($P_r = P/P_{crit}$) at $T_r = T/T_{crit} = 0.7$ is required. For example for water T_{crit}=647.3 K T=0.7*647.3=453.11 K=179.96 C so for this temperature P_v=10.01168 bar and P_{crit}=221.2 bar

$$P_r = P/P_{crit} = \frac{10.01168}{221.2} = 0.045261$$

$$\omega = -log_{10}P_{saturated\ vapor}(0.045261) - 1 = 1.344278 - 1 = 0.344278$$

Some of the Helmholtz form equations:

ISO 17584-2005 EOS for refrigerants

$\emptyset = \frac{A}{RT} = \emptyset_{id} + \emptyset_r$

"id" in the equation indicates ideal gas and r indicates real gas part of the equation of state.

$\emptyset_r = \sum_k N_k/\tau^{t_k}\,\delta^{d_k}exp[-\alpha_k(\delta-\varepsilon_k)^{l_k}]exp[-\beta_k(\tau-\gamma_k)^{m_k}]$

Ideal gas term of the equation :

$\emptyset_{id} = \frac{h_{ref}}{RT} - \frac{s_{ref}}{R} - 1 + ln\left(\frac{RT\rho}{P_{ref}}\right) + \frac{1}{RT}\int_{T_{ref}}^{T} C_{p,id}dT\,\frac{1}{R} - \int_{T_{ref}}^{T}\frac{C_{p,id}}{T}dT$

Helmholtz EOS for water and steam

$$A(\rho,T) = A_{id}(T) + A_r(\rho,T) = A_{id}(T) + RT[\ln(\rho) + \rho Q(\rho,\tau)]$$

$$A_{id}(T) = \sum_{i=1}^{6}\frac{C_i}{\tau^{i-1}} + C_7\ln(T) + C_8\ln T/\tau$$

$$Q(\rho,\tau) = (\tau-\tau_{crit})\sum_{j=1}^{7}(\tau-\tau_{aj})^{j-2}\left[\sum_{i=1}^{8}B_{ij}(\rho-\rho_{aj})^{i-1} + \exp(-E\rho)\sum_{i=1}^{8}B_{ij}\rho^{i-9}\right]$$

Details of equation of states are not given here for futher details please refer to thermodynamic books.

Data on specific heat of gases can be found in two excellent books, JANAF Thermochemical Tables[1], and Ihsan Barin Thermochemical Data of Pure Substances[2]. Nowadays JANAF tables can be accessed through internet as well. JANAF table list through internet adress **http://kinetics.nist.gov/janaf** is shown In figure 2.1.1. **http://webbook.nist.gov/chemistry/** adress is also an important source to get Cp data. JANAF tables can also be found as a text format[]. Another good data source for Cp values is **http://www.chem.msu.su/rus/handbook/ivtan/**
Knowing the table values Cp(T) data can be created by using curve fitting methods. Some of the common equations used can be summerized as follows:
Polynomial curve fitting equations:

$$C_p(T) = \sum_{i=0}^{n} a_i T^i \quad (2.1.12)$$

This type of coefficient of equations are listed for approximately 600 species in "The properties of Gases and Liquids"[130].

$$C_p(T) = a_0 + a_1 T + a_2 T^2 + a_3 T^3 \ \ kJ/kmolK$$

ref	formula	name	a_0	a_1	a_2	a_3
0	NH3	ammonia	27.3100	2.383E-02	1.707E-05	-1.185E-08
1	Ar	argon	20.8000	0.000E+00	0.000E+00	0.000E+00
2	C6H5Cl	chlorobenzene	-33.8900	5.631E-01	-4.522E-04	1.426E-07
3	NO	nitricoxide	29.3500	-9.378E-04	9.747E-06	-4.187E-09
4	NO2	nitrogendioxide	24.2300	4.836E-02	-2.081E-05	2.930E-10
5	N2	nitrogene	31.1500	-1.357E-02	2.680E-05	-1.168E-08
6	N2O	nitrousoxide	21.6200	7.281E-02	-5.778E-05	1.830E-08
7	O2	oxygen	28.1100	-3.680E-06	1.746E-05	-1.065E-08
8	O3	ozone	20.5400	8.009E-02	-6.243E-05	1.697E-08
9	H2	hydrogen	27.1400	9.274E-03	-1.381E-05	7.645E-09
10	H2O	water	32.2400	1.924E-03	1.055E-05	-3.596E-09
11	CH4	methane	19.2500	5.213E-02	1.197E-05	-1.132E-08
12	CH4O	methanol	21.1500	7.092E-02	2.587E-05	-2.852E-08
13	C2H6	ethane	5.4090	1.781E-01	-6.938E-05	8.713E-09
14	C3H8	propane	-4.2240	3.063E-01	-1.586E-04	3.215E-08
15	C3H6O	propionaldehyde	11.7200	2.614E-01	-1.300E-04	2.126E-08
16	C4H10	n-butane	9.4870	3.313E-01	-1.108E-04	-2.822E-09
17	C4H10	isobutane	-1.3900	3.847E-01	-1.846E-04	2.895E-08
18	C5H12	n-pentane	-3.6260	4.873E-01	-2.580E-04	5.305E-08
19	C6H6	benzene	-33.9200	4.739E-01	-3.017E-04	7.130E-08
20	C6H14	n-hexane	-4.4130	5.820E-01	-3.119E-04	6.494E-08
21	CCl2F2	dichlorodifloromethane	31.6000	1.782E-01	-1.509E-04	4.342E-08
22	C7H16	n-heptane	-5.1460	6.762E-01	-3.651E-04	7.658E-08
23	C8H10	ethylbenzene	-43.1000	7.072E-01	-4.811E-04	1.301E-07
24	C8H18	n-octane	-6.0960	7.712E-01	-4.195E-04	8.855E-08
25	C9H20	n-nonane	-8.3470	8.729E-01	-4.823E-04	1.031E-07
26	C10H8	naphtalene	-68.8000	8.499E-01	-6.506E-04	1.981E-07
27	C11H10	1-methylnaphtalene	-64.8200	9.387E-01	-6.942E-04	2.016E-07
28	C11H10	2-methylnaphtalene	-56.5200	8.997E-01	-6.469E-04	1.840E-07
29	C12H12	1-ethylnaphtalene	-63.8190	1.040E+00	-8.000E-04	2.000E-07
30	C13H14	1-propylnaphtalene	-60.2350	1.116E+00	-8.000E-04	2.000E-07
31	C14H16	1-butylnaphtalene	15.0660	8.659E-01	4.000E-04	6.000E-08
32	C10H22	n-decane	-7.9130	9.609E-01	-5.288E-04	1.131E-07
33	C11H24	n-undecane	-8.3950	1.054E+00	-5.799E-04	1.237E-07
34	C12H26	n-dodecane	-9.3280	1.149E+00	-6.347E-04	1.359E-07
35	C13H28	n-tridecane	-10.4600	1.245E+00	-6.912E-04	1.490E-07
36	C14H30	n-tetradecane	-10.9800	1.338E+00	-7.423E-04	1.598E-07
37	C15H32	n-pentadecane	-11.9200	1.433E+00	-7.972E-04	1.720E-07
38	C16H34	n-hexadecane	-13.0200	1.529E+00	-8.537E-04	1.850E-07
39	C17H36	n-heptadecane	-13.9700	1.624E+00	-9.081E-04	1.972E-07
40	C18H38	n-octadecane	-14.4700	1.717E+00	-9.592E-04	2.078E-07

41	C19H40	n-nonadecane	-15.4900	1.812E+00	-1.015E-03	2.205E-07
42	C20H42	n-eicosane	-22.3800	1.939E+00	-1.117E-03	2.528E-07
43	C3H6	propylene	3.7100	2.345E-01	-1.160E-04	2.205E-08
44	C11H16	penthylbenzene	-57.4000	1.036E+00	-7.000E-04	2.090E-07
45	C12H18	hexylbenzene	-0.9436	9.028E-01	-5.000E-04	9.000E-08
46	C13H20	heptylbenzene	1.5097	9.848E-01	-5.000E-04	1.000E-07
47	C14H22	octylbenzene	3.7589	1.068E+00	-6.000E-04	1.000E-07
48	C15H24	nonylbenzene	-28.7370	1.293E+00	-8.000E-04	2.000E-07
49	C16H26	decylbenzene	9.1501	1.230E+00	-6.000E-04	1.000E-07
50	C17H28	undecylbenzene	-31.6240	1.494E+00	-9.000E-04	2.000E-07
51	C18H30	dodecylbenzene	33.6640	1.307E+00	-6.000E-04	9.000E-08
52	CO	carbonmonoxide	30.8700	-1.285E-02	2.789E-05	-1.272E-08
53	CO2	carbondioxide	19.8000	7.344E-02	-5.602E-05	1.715E-08
54	C8H18	2,3,3-trimethylpentane	-9.2150	7.859E-01	-4.400E-04	9.697E-08
55	C8H18	2,3,4-trimethylpentane	-9.2150	7.859E-01	-4.400E-04	9.697E-08
56	C8H18	2-methyl-3-ethylpentane	-9.2150	7.859E-01	-4.400E-04	9.697E-08
57	C8H18	3-methyl-3-ethylpentane	-9.2150	7.859E-01	-4.400E-04	9.697E-08
59	C6H12	cyclohexane	-54.5400	6.113E-01	-2.523E-04	1.321E-08
60	C6H12	methylcyclopentane	-50.1100	6.381E-01	-3.642E-04	8.014E-08
61	C6H12	1-hexene	-1.7460	5.309E-01	-2.903E-04	6.054E-08
62	C6H12	2-hexene,cis	-9.8100	5.309E-01	-2.717E-04	4.827E-08
63	C6H12	2-hexene,trans	-32.9200	6.929E-01	-5.619E-04	2.005E-07
64	C6H12	3-hexene,cis	-21.7300	5.811E-01	-3.362E-04	7.457E-08
65	C6H12	3-hexene,trans	-4.3380	5.510E-01	-3.282E-04	8.047E-08
66	C6H12	2-methyl-2-pentene	-14.7500	5.669E-01	-3.341E-04	7.963E-08
67	C6H12	3-methyl-2-pentene,cis	-14.7500	5.669E-01	-3.341E-04	7.963E-08
68	C6H12	3-methyl-2-pentene,trans	-14.7500	5.669E-01	-3.341E-04	7.963E-08
69	C6H12	4-methyl-2-pentene,cis	-1.6750	5.376E-01	-3.044E-04	6.753E-08
70	C6H12	4-methyl-2-pentene,trans	12.6300	5.154E-01	-3.007E-04	7.327E-08
71	C6H12	2,3-dimethyl-1-butene	7.0250	5.585E-01	-3.696E-04	1.063E-07
72	C6H12	2,3-dimethyl-2-butene	2.2940	4.827E-01	-2.199E-04	3.042E-08
73	C6H12	3,3-dimethyl-1-butene	-12.5600	5.485E-01	-2.915E-04	5.208E-08
74	C6H12O	cyclohexanol	-55.5300	7.214E-01	-4.086E-04	8.235E-08
76	C6H12O	methylisobutylketone	3.8940	5.656E-01	-3.318E-04	8.231E-08
77	C6H12O2	n-butylacetate	13.6200	5.489E-01	-2.278E-04	7.910E-10
78	C6H12O2	isobutylacetate	7.3100	5.740E-01	-2.576E-04	1.101E-08
79	C6H12O2	ethylbutyrate	21.5100	4.928E-01	-1.938E-04	3.559E-09
82	C6H14	2-methylpentane	-10.5700	6.184E-01	-3.573E-04	8.058E-08
83	C6H14	3-methylpentane	-2.3860	5.690E-01	-2.870E-04	5.033E-08
84	C6H14	2,2-dimethylbutane	-16.6300	6.293E-01	-3.481E-04	6.850E-08
85	C6H14	2,3-dimethylbutane	-14.6100	6.150E-01	-3.376E-04	6.820E-08
86	C6H14O	1-hexanol	4.8110	5.891E-01	-3.010E-04	5.426E-08
87	C6H14O	ethylbutylether	23.6300	5.367E-01	-2.528E-04	4.157E-08
88	C6H14O	dipropylether	18.6200	5.335E-01	-2.285E-04	2.442E-08
89	C6H14O	diisopropylether	7.5050	5.849E-01	-3.027E-04	5.845E-08
90	C6H15N	dipropylamine	6.4600	6.293E-01	-3.390E-04	7.072E-08
92	C6H15N	triethylamine	-18.4300	7.155E-01	-4.392E-04	1.092E-07
96	C7H5N	benzonitrile	-26.0500	5.732E-01	-4.430E-04	1.349E-07
97	C7H6O	benzaldehyde	-12.1400	4.961E-01	-2.845E-04	5.167E-08
98	C7H6O2	benzoicacid	-51.2900	6.293E-01	-4.237E-04	1.062E-07
99	C7H8	toluene	-24.3500	5.125E-01	-2.765E-04	4.911E-08
101	C7H8O	benzylalcohol	-7.3980	5.481E-01	-3.357E-04	7.771E-08
102	C7H8O	o-cresol	-32.2800	7.055E-01	-5.924E-04	2.124E-07
103	C7H8O	m-cresol	-45.0100	7.264E-01	-6.029E-04	2.077E-07
104	C7H8O	p-cresol	-40.6300	7.055E-01	-5.757E-04	1.967E-07
107	C7H9N	m-toluidine	-15.9900	5.681E-01	-3.033E-04	4.643E-08
108	C7H14	cycloheptane	-76.1900	7.867E-01	-4.204E-04	7.561E-08
109	C7H14	1,1-	-57.8900	7.670E-01	-4.501E-04	1.010E-07
110	C7H14	1,2-	-55.6400	7.616E-01	-4.484E-01	1.014E-07
111	C7H14	1,2-	-54.5200	7.591E-01	-4.480E-04	1.017E-07
112	C7H14	ethylcyclopentane	-55.3100	7.511E-01	-4.396E-04	1.004E-07
113	C7H14	methylcyclohexane	-61.9200	7.842E-01	-4.438E-04	9.366E-08

114	C7H14	1-heptene	-3.3030	6.297E-01	-3.512E-04	7.607E-08
119	C7H16	3-methylhexane	-7.0460	6.837E-01	-3.734E-04	7.834E-08
120	C7H16	2,2-dimethylpentane	-50.1000	8.956E-01	-6.360E-04	1.736E-07
121	C7H16	2,3-dimethylpentane	-7.0460	6.837E-01	-3.734E-04	7.834E-08
122	C7H16	2,4-dimethylpentane	-7.0460	6.837E-01	-3.734E-04	7.834E-08
123	C7H16	3,3-dimethylpentane	-7.0460	6.837E-01	-3.734E-04	7.834E-08
124	C7H16	3-ethylpentane	-7.0460	6.837E-01	-3.734E-04	7.834E-08
125	C7H16	2,2,3-trimethylbutane	-22.9400	7.519E-01	-4.421E-04	1.005E-07
126	C7H16O	1-heptanol	49.0700	6.778E-01	-3.447E-04	6.046E-08
127	C8H4O3	phthalicanhydride	-4.4550	6.540E-01	-4.283E-04	1.009E-07
128	C8H8	styrene	-28.2500	6.159E-01	-4.023E-04	9.935E-08
129	C8H8O	methylphenylketone	-29.5800	6.410E-01	-4.071E-04	9.722E-08
130	C8H8O2	methylbenzoate	-21.2100	5.501E-01	-1.799E-04	4.425E-08
131	C8H10	o-xylene	-15.8500	5.962E-01	-3.443E-04	7.528E-08
132	C8H10	m-xylene	-29.1700	6.297E-01	-3.747E-04	8.478E-08
133	C8H10	p-xylene	-25.0900	6.042E-01	-3.374E-04	6.820E+00
136	C8H16	1,1-dimethylcyclohexane	-72.1100	8.997E-01	-5.020E-04	1.030E-07
137	C8H16	1,2-	-68.3700	8.972E-01	-5.137E-04	1.099E-07
138	C8H16	1,2-	-68.4800	9.123E-01	-5.355E-04	1.181E-07
139	C8H16	1,3-	-65.1600	8.838E-01	-4.932E-04	1.020E-07
140	C8H16	1,3-	-64.1500	8.826E-01	-5.016E-04	1.068E-07
141	C8H16	1,4-	-64.1500	8.826E-01	-5.016E-04	1.068E-07
142	C8H16	1,4-	-70.3600	9.131E-01	-5.309E-04	1.155E-07
143	C8H16	ethylcyclohexane	-63.8900	8.893E-01	-5.108E-04	1.103E-07
149	C8H16	n-propylcyclopentane	-55.9700	8.449E-01	-4.924E-04	1.117E-07
152	C8H16	1-octene	-4.0990	7.239E-01	-4.036E-04	8.675E-08
153	C8H16	2-octene-trans	-12.8200	7.532E-01	-4.442E-04	1.050E-07
157	C8H18	2-methylheptane	-89.7000	1.242E+00	-1.176E-03	4.618E-07
158	C8H18	3-methylheptane	-9.2150	7.859E-01	-4.400E-04	9.697E-08
159	C8H18	4-methylheptane	-9.2150	7.859E-01	-4.400E-04	9.697E-08
160	C8H18	2,2-dimethylhexane	-9.2150	7.859E-01	-4.400E-04	9.697E-08
161	C8H18	2,3-dimethylhexane	-9.2150	7.859E-01	-4.400E-04	9.697E-08
162	C8H18	2,4-dimethylhexane	-9.2150	7.859E-01	-4.400E-04	9.697E-08
163	C8H18	2,5-dimethylhexane	-9.2150	7.859E-01	-4.400E-04	9.697E-08
164	C8H18	3,3-dimethylhexane	-9.2150	7.859E-01	-4.400E-04	9.697E-08
165	C8H18	3,4-dimethylhexane	-9.2150	7.859E-01	-4.400E-04	9.697E-08
166	C8H18	3-ethylhexane	-9.2150	7.859E-01	-4.400E-04	9.697E-08
167	C8H18	2,2,3-trimethylpentane	-9.2150	7.859E-01	-4.400E-04	9.697E-08
168	C8H18	2,2,4-trimethylpentane	-7.4610	-7.779E-01	-4.287E-04	9.173E-08
170	C8H18O	1-octanol	6.1710	7.607E-01	-3.797E-04	6.263E-08
171	C8H18O	2-octanol	25.8800	7.641E-01	-4.224E-04	9.064E-08
174	C8H18O	2-ethyl-1-hexanol	-14.9900	8.645E-01	-5.280E-04	1.285E-07
175	C8H18O	dibutylether	6.0540	7.729E-01	-4.085E-04	8.085E-08
177	C8H19N	dibutylamin	9.7640	8.081E-01	-4.329E-04	9.249E-08
182	C9H10	alpha-methylstyrene	-24.3300	6.933E-01	-4.530E-04	1.181E-07
183	C9H10O2	ethylbenzoate	20.6700	6.887E-01	-3.608E-04	5.062E-08
184	C9H12	n-propylbenzene	-31.2900	7.468E-01	-4.601E-04	1.081E-07
185	C9H12	isopropylbenzene	-39.3600	7.842E-01	-5.078E-04	1.291E-07
186	C9H12	1-methyl-3-ethylbenzene	-29.0000	7.293E-01	-4.363E-04	9.998E-08
187	C9H12	1-methyl-4-ethylbenzene	-27.3100	7.176E-01	-4.224E-04	9.542E-08
188	C9H12	1,2,3-trimethylbenzene	-6.9420	6.335E-01	-3.326E-04	6.611E-08
189	C9H12	1,2,4-trimethylbenzene	-4.6680	6.238E-01	-3.263E-04	6.376E-08
190	C9H12	1,3,5-trimethylbenzene	-19.5900	6.724E-01	-3.629E-04	7.700E-08
192	C9H18	n-propylcyclohexane	-62.5200	9.889E-01	-5.795E-04	1.291E-07
195	C9H18	1-nonene	-3.7180	8.122E-01	-4.509E-04	9.705E-08
196	C9H18O	dibutylketone	0.0000	0.000E+00	0.000E+00	0.000E+00
197	C9H20	2-methyloctane	-10.1100	8.805E-01	-4.936E-04	1.083E-07
198	C9H20	2,2-dimethylheptane	-20.8900	9.668E-01	-6.120E-04	1.570E-07
199	C9H20	2,2,3-trimethylhexane	-45.6300	1.055E+00	-7.172E-04	1.987E-07
200	C9H20	2,2,4-trimethylhexane	-60.3100	1.104E+00	-7.712E-04	2.188E-07
201	C9H20	2,2,5-trimethylhexane	-54.1100	1.095E+00	-7.746E-04	2.255E-07
202	C9H20	3,3-diethylpentane	-67.2700	1.126E+00	-7.988E-04	2.306E-07

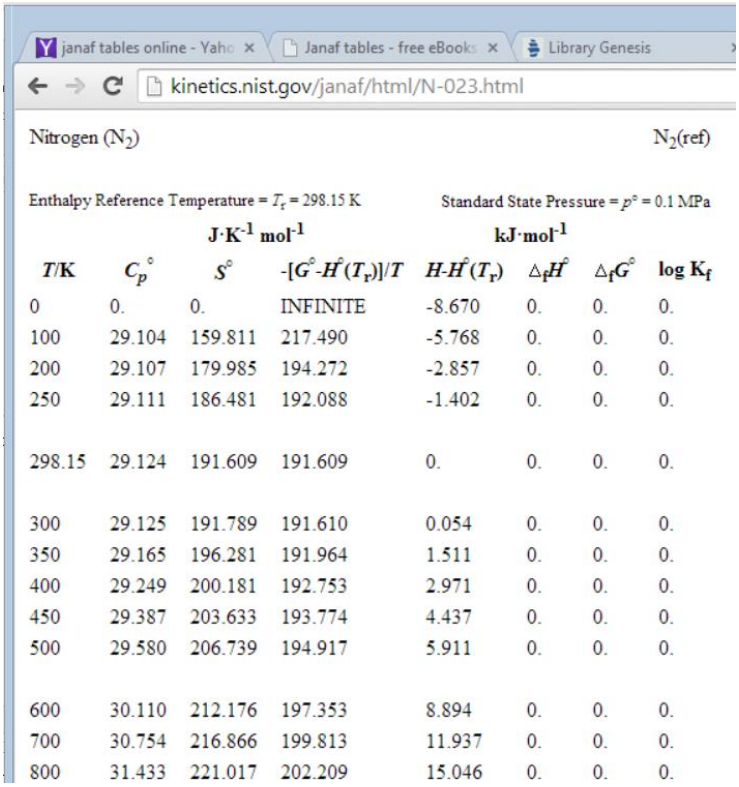
203	C9H20	2,2,3,3-	-54.5800	1.089E+00	-7.570E-04	2.142E-07
204	C9H20	2,2,3,4-	-54.5800	1.089E+00	-7.570E-04	2.142E-07
205	C9H20	2,2,4,4-	-67.4000	1.168E+00	-8.612E-04	2.574E-07
206	C9H20	2,3,3,4-	-54.9200	1.091E+00	-7.603E-04	2.158E-07
207	C9H20O	1-nonanol	1.2800	8.817E-01	-4.791E-04	9.801E-08
211	C10H14	n-butylbenzene	-22.9900	7.934E-01	-4.396E-04	8.570E-08
213	C10H14	sec-butylbenzene	-65.1500	9.893E-01	-7.214E-04	2.152E-07
214	C10H14	tert-butylbenzene	-86.0000	1.102E+00	-8.746E-04	2.827E-07
216	C10H14	1-methyl-3-	-48.7600	9.064E-01	-6.054E-04	1.627E-07
218	C10H14	1,4-diethylbenzene	-37.4200	8.671E-01	-5.560E-04	1.411E-07
219	C10H14	1,2,3,5-	3.9230	7.131E-01	-3.711E-04	6.840E-08
220	C10H14	1,2,4,5-	16.2500	6.519E-01	-2.879E-04	3.257E-08
222	C10H15N	n-butylaniline	-34.0700	9.144E-01	-5.560E-04	1.287E-07
223	C10H18	cis-decalin	-	1.118E+00	-6.607E-04	1.437E-07
224	C10H18	trans-decalin	-97.6700	1.045E+00	-5.476E-04	8.981E-08
227	C10H20	butylcyclohexane	-62.9600	1.081E+00	-6.305E-04	1.400E-07
231	C10H20	1-decene	-4.6640	9.077E-01	-5.058E-04	1.095E-07
233	C10H22	n-decane	-7.9130	9.609E-01	-5.288E-04	1.131E-07
234	C10H22	3,3,5-trimethylheptane	-70.3700	1.232E+00	-8.646E-04	2.455E-07
235	C10H22	2,2,3,3-	-58.8300	1.231E+00	-8.834E-04	2.585E-07
236	C10H22	2,2,5,5-	-62.3400	1.245E+00	-8.956E-04	2.618E-07
237	C10H22O	1-decanol	14.5700	8.947E-01	-3.921E-04	3.451E-08
238	C11H14O2	butylbenzoate	-17.3700	8.675E-01	-4.610E-04	7.235E-08
239	C11H22	n-hexylcyclopentane	-58.3200	1.128E+00	-6.536E-04	1.473E-07
240	C11H22	1-undecene	-5.5850	1.003E+00	-5.602E-04	1.216E-07
241	C11H24	n-undecane	-8.3950	1.054E+00	-5.799E-04	1.237E-07
242	C12H10	diphenyl	-97.0700	1.106E+00	-8.855E-04	2.790E-07
243	C12H10O	diphenylether	-60.7300	9.282E-01	-5.870E-04	1.359E-07
244	C12H24	n-heptylcyclopentane	-59.2600	1.223E+00	-7.084E-04	1.596E-07
245	C12H24	1-dodecene	-6.5440	1.098E+00	-6.155E-04	1.341E-07
246	C12H26	dodecane	-9.3280	1.149E+00	-6.347E-04	1.359E-07
247	C12H26O	dihexylether	33.5400	1.073E+00	-5.535E-04	1.678E-07
248	C12H26O	dodecanol	9.2240	1.103E+00	-5.338E-04	7.779E-08
249	C12H27N	tributylamine	7.9930	1.198E+00	-6.703E-04	1.449E-07
251	C13H26	n-octylcyclopentane	-59.9500	1.317E+00	-7.612E-04	1.708E-07
252	C13H26	1-tridecene	-7.1180	1.191E+00	-6.674E-04	1.451E-07
253	C13H28	1-tridecane	-10.4600	1.245E+00	-6.912E-04	1.490E-07
254	C14H10	anthracane	-58.9800	1.006E+00	-6.594E-04	1.606E-07
255	C14H10	phenanthrene	-58.9800	1.006E+00	-6.594E-04	1.606E-07
256	C14H28	n-nonylcyclopentane	-60.8100	1.412E+00	-8.156E-04	1.830E-07
257	C14H28	1-tetradecene	-7.9670	1.286E+00	-7.210E-04	1.569E-07
258	C14H30	n-tetradecane	-10.9800	1.338E+00	-7.423E-04	1.598E-07
259	C15H30	n-decyclopentane	-61.9200	1.508E+00	-8.717E-04	1.959E-07
260	C15H30	1-pentadecene	-9.2030	1.382E+00	-7.783E-04	1.703E-07
261	C15H32	n-pentadecane	-1.1920	1.433E+00	-7.972E-04	1.720E-07
262	C16H22O4	dibutyl-o-phthalate	1.8800	1.254E+00	-6.121E-04	6.971E-08
263	C16H32	n-decyclohexane	-69.0200	1.654E+00	-9.613E-04	2.143E-07
264	C16H32	1-hexadecene	-9.7050	1.475E+00	-8.298E-04	1.810E-07
265	C16H34	hexadecane	-13.0200	1.529E+00	-8.537E-04	1.850E-07
266	C17H34	n-dodecyclopentane	-63.2600	1.695E+00	-9.768E-04	2.186E-07
267	C17H36O	heptadecanol	-7.7920	1.653E+00	-9.345E-04	2.044E-07
271	C18H36	1-octadecene	-11.3300	1.664E+00	-9.374E-04	2.049E-07
272	C18H36	n-tridecyclopentane	-64.2100	1.790E+00	-1.032E-03	2.309E-07
273	C18H38	octadecane	-14.4700	1.717E+00	-9.592E-04	2.078E-07
274	C18H38O	1-octadecanol	-8.7040	1.748E+00	-9.881E-04	2.157E-07
275	C19H38	1-cyclopentyltetradecane	-64.9300	1.884E+00	-1.085E-03	2.426E-07
276	C19H40	n-nonadecane	-15.4900	1.812E+00	-1.015E-03	2.205E-07
277	C20H40	1-	-66.0900	1.980E+00	-1.140E-03	2.550E-07
278	C20H42O	1-eicosanol	-12.5800	1.950E+00	-1.118E-03	2.516E-07
279	C6F6	perfluorobenzene	36.2800	5.267E-01	-4.547E-04	1.456E-07
290	C6H4Cl2	o-dichlorobenzene	-14.3000	5.506E-01	-4.513E-04	1.429E-07
291	C6H4F2	1,4-difluorobenzene	-25.9600	5.722E-01	-4.677E-04	1.475E-07

292	C6H5Br	bromobenzene	-28.8100	5.351E-01	-4.080E-04	1.212E-07
293	C6H5F	fluorobenzene	-38.7300	5.669E-01	-4.434E-04	1.355E-07
294	C6H5I	iodobenzene	-29.2700	5.564E-01	-4.509E-04	1.443E-07
295	C6H6	benzene	-33.9200	4.739E-01	-3.017E-04	7.130E-08
296	C6H6O	phenol	-35.8400	5.983E-01	-4.827E-04	1.527E-07
297	C6H7N	aniline	-40.5200	6.385E-01	-5.133E-04	1.633E-07
298	C6H7N	2-methylpyridine	-36.2600	5.584E-01	-3.704E-04	9.663E-08
299	C6H7N	3-methylpyridine	-37.0900	5.600E-01	-3.719E-04	9.685E-08
300	C6H7N	4-methylpyridine	-17.4300	4.882E-01	-2.798E-04	5.451E-08
302	C6H10	cyclohexene	-68.6500	7.252E-01	-5.414E-04	1.644E-07
303	C5H12O	2,2-dimethyl-1-propanol	12.1500	5.397E-01	-3.160E-04	7.122E-08
306	C5H12O	tert-butylmethylether	2.5340	5.136E-01	-2.596E-04	4.303E-08
311	C3H9N	n-propylamine	6.6910	3.498E-01	-1.822E-04	3.586E-08
312	C3H9N	isopropylamine	-7.4860	4.175E-01	-2.826E-04	8.348E-08
313	C3H9N	trimethylamine	-8.2060	3.972E-01	-2.219E-04	4.622E-08
316	C4H4	vinylacetylene	6.7570	2.841E-01	-2.265E-04	7.461E-08
317	C4H4O	furan	-35.5300	4.321E-01	-3.455E-04	1.047E-07
318	C4H4N	thiophene	-30.6100	4.480E-01	-3.772E-04	1.253E-07
319	C4H5N	allylcyanide	21.7000	2.572E-01	-1.192E-04	1.229E-08
321	C4H6	1-butyne	12.5500	2.744E-01	-1.545E-04	3.450E-08
322	C4H6	2-butyne	15.9300	2.381E-01	-1.070E-04	1.753E-08
323	C4H6	1,2-butadiene	11.2000	2.724E-01	-1.468E-04	3.089E-08
324	C4H6	1,3-butadiene	-1.6870	3.419E-01	-2.340E-04	6.335E-08
325	C4H6O2	vinylacetate	15.1600	2.795E-01	-8.805E-05	-1.660E-08
326	C4H6O3	aceticanhydride	-23.1300	5.087E-01	-3.580E-04	9.835E-08
328	C4H6O4	succinicacid	15.0700	4.689E-01	-3.143E-04	7.938E-08
329	C4H7N	butylonitrile	15.2100	3.206E-01	-1.638E-04	2.982E-08
330	C4H7O2	methylacrylate	15.1600	2.796E-01	-8.805E-05	-1.660E-08
331	C4H8	1-butene	-2.9940	3.532E-01	-1.990E-04	4.463E-08
332	C4H8	2-butene,cis	0.4396	2.953E-01	-1.108E-04	-6.160E-10
333	C4H8	2-butene,trans	18.3200	2.564E-01	-7.013E-05	-8.989E-09
334	C4H8	cyclobutane	-50.2500	5.024E-01	-3.558E-04	1.047E-07
335	C4H8	isobutylene	16.0500	2.804E-01	-1.091E-04	9.098E-09
336	C4H8O	n-butyraldehyde	14.0800	3.457E-01	-1.743E-04	2.887E-08
337	C4H8O	isobutyraldehyde	24.4600	3.356E-01	-2.057E-04	6.368E-08
338	C4H8O	methylethylketone	10.9400	3.559E-01	-1.900E-04	3.920E-08
339	C4H8O	tetrahydrofuran	19.1000	5.162E-01	-4.132E-04	1.454E-07
340	C4H8O	vinylethylether	17.2800	3.236E-01	-1.471E-04	2.150E-08
341	C4H8O2	n-butyricacid	11.7400	4.137E-01	-2.430E-04	5.531E-08
342	C4H8O2	isobutyricacid	9.8140	4.668E-01	-3.720E-04	1.350E-07
343	C4H8O2	1,4-dioxane	-53.5700	5.987E-01	-4.085E-04	1.062E-07
344	C4H8O2	ethylacetate	7.2350	4.072E-01	-2.092E-04	2.855E-08
345	C4H8O2	methylpropionate	18.2000	3.140E-01	-9.353E-05	-1.828E-08
348	C4H9Cl	1-chlorobutane	-2.6130	4.497E-01	-2.937E-04	8.018E-08
349	C4H9Cl	2-chlorobutane	-3.4330	4.559E-01	-2.981E-04	8.256E-08
350	C4H9Cl	tert-butylchloride	-3.9310	4.652E-01	-2.886E-04	7.871E-08
351	C4H9N	pyrrolidine	-51.5300	5.338E-01	-3.240E-04	7.528E-08
352	C4H10	isobutane	-1.3900	3.847E-01	-1.846E-04	2.895E-08
353	C4H10O	n-butanol	3.2660	4.180E-01	-2.242E-04	4.685E-08
354	C4H10O	2-butanol	5.7530	4.245E-01	-2.328E-04	4.773E-08
355	C4H10O	isobutanol	-7.7080	4.689E-01	-2.884E-04	7.231E-08
356	C4H10O	tert-butanol	-46.8100	7.172E-01	-7.084E-04	2.920E-07
357	C4H10O	diethylether	21.4200	3.359E-01	-1.035E-04	-9.357E-09
358	C4H10O	methylpropylether	21.3100	3.390E-01	-1.127E-04	-2.855E-09
359	C4H10O	methylisopropylether	13.5300	3.697E-01	-1.481E-04	1.205E-08
360	C4H10O2	1,2-dimethoxyethane	32.2300	3.567E-01	-1.336E-04	8.399E-09
361	C4H10O3	diethyleneglycol	73.0600	3.461E-01	-1.468E-04	1.846E-08
362	C4H10S	diethylsulfide	13.5900	3.959E-01	-1.780E-04	2.649E-08
363	C4H10S2	diethyldisulfide	26.9000	4.601E-01	-2.710E-04	5.970E-08
364	C4H11N	n-butylamine	5.0790	4.476E-01	-2.407E-04	7.599E-08
365	C4H11N	isobutylamine	9.4910	4.430E-01	-2.110E-04	2.333E-08
366	C4H11N	diethylamine	2.0390	4.430E-01	-2.183E-04	3.653E-08

370	C5H5N	pyridine	39.7900	4.928E-01	-3.558E-04	1.004E-07
373	C5H8	cyclopentene	-41.5100	4.631E-01	-2.579E-04	5.434E-08
374	C5H8	1,2-pentadiene	8.8260	3.880E-01	-2.280E-04	5.246E-08
375	C5H8	1,3-pentadiene,trans	30.6900	2.811E-01	-6.711E-05	-2.352E-08
376	C5H8	1,4-pentadiene	6.9960	3.952E-01	-2.374E-04	5.598E-08
377	C5H8	pentyne	18.0700	3.511E-01	-1.913E-04	4.098E-08
378	C5H8	2-methyl-1,3-butadiene	-3.4120	4.585E-01	-3.337E-04	1.000E-07
379	C5H8	3-methyl-1,2-butadiene	14.6900	3.598E-01	-1.976E-04	4.262E-08
380	C5H8O	cyclopentanone	-40.6400	5.255E-01	-3.124E-04	7.130E-08
382	C5H8O2	ethylacrylate	16.8100	3.690E+00	-1.382E-04	-5.732E-09
383	C5H10	cyclopentane	-53.6200	5.426E-01	-3.031E-04	6.485E-08
384	C5H10	1-pentene	-0.1340	4.329E-01	-2.317E-04	4.681E-08
385	C5H10	2-pentene,cis	-14.2900	4.601E-01	-2.541E-04	5.455E-08
386	C5H10	2-pentene,trans	1.9470	4.182E-01	-2.178E-04	4.405E-08
387	C5H10	2-methyl-1-butene	10.5700	3.997E-01	-1.946E-04	3.314E-08
388	C5H10	2-methyl-2-butene	11.8000	3.509E-01	-1.117E-04	-5.807E-09
389	C5H10	3-methyl-1-butene	21.7400	3.890E-01	-2.007E-04	4.011E-08
390	C5H10O	valeraldehyde	14.2400	4.329E-01	-2.107E-04	3.162E-08
391	C5H10O	methyl-n-propyl-ketone	1.1470	4.802E-01	-2.818E-04	6.661E-08
392	C5H10O	methyl	-2.9140	4.991E-01	-2.935E-04	6.665E-08
393	C5H10O	diethyl-ketone	30.0100	3.939E-01	-1.907E-04	3.398E-08
396	C5H10O2	n-valeric-acid	13.3900	5.033E-01	-2.931E-04	6.619E-08
398	C5H10O2	isobutyl-formate	19.8500	4.034E-01	-1.436E-04	-7.402E-09
399	C5H10O2	n-propyl-acetate	15.4200	4.501E-01	-1.686E-04	-1.439E-08
400	C5H10O2	ethyl-propionate	19.8500	4.034E-01	-1.437E-04	-7.394E-09
403	C5H11N	piperidine	-53.0700	6.289E-01	-3.358E-04	6.427E-08
404	C5H12	n-pentane	-3.6260	4.873E-01	-2.580E-04	5.305E-08
405	C5H12	2-methyl-butane	-9.5250	5.066E-01	-2.729E-04	5.723E-08
406	C5H12	2,2-dimethylpropane	-16.5900	5.552E-01	-3.306E-04	7.633E-08
407	C5H12O	1-pentanol	3.8690	5.045E-01	-2.639E-04	5.120E-08
408	C5H12O	2-methyl-1-butanol	-9.4830	5.677E-01	-3.481E-04	8.637E-08
409	C5H12O	3-methyl-1-butanol	-9.5420	5.681E-01	-3.485E-04	8.650E-08
410	C5H12O	2-methyl-2butanol	-12.0900	6.096E-01	-4.204E-04	1.228E-07
411	C5H12O	2,2-dimethyl-1-propanol	12.1500	5.397E-01	-3.160E-04	7.122E-08
412	C4H9N0	morpholine	-42.8000	5.388E-01	-2.666E-04	4.199E-08
413	AlBr3	aliminiumtribromide	64.9400	6.098E-02	-7.306E-05	2.978E-08
414	C6H10O	cyclohexanone	-37.8100	5.539E-01	-1.953E-04	-1.534E-08
429	C7H16	2-methylhexane	-39.3900	8.642E-01	-6.289E-04	1.836E-07

It should be note that taking C_p value constant (only first term of the polynomial) are also used commonly in thermodynamics. Polynomial coefficients and table values of C_p of most common gases are usually given in most of the thermodynamic text books. We can work an example. If a polynomial least square curve fitting equation for nitrogen data is required to be created

Figure 2.1.1 JANAF Tables at internet adress kinetics.nist.gov/janaf



janaf tables online - Yaho × | Janaf tables - free eBooks × | Library Genesis

kinetics.nist.gov/janaf/html/N-023.html

Nitrogen (N_2) N_2(ref)

Enthalpy Reference Temperature = T_r = 298.15 K Standard State Pressure = $p°$ = 0.1 MPa

	J·K^{-1} mol^{-1}			kJ·mol^{-1}			
T/K	$C_p°$	$S°$	$-[G°-H°(T_r)]/T$	$H-H°(T_r)$	$\Delta_f H°$	$\Delta_f G°$	log K_f
0	0.	0.	INFINITE	-8.670	0.	0.	0.
100	29.104	159.811	217.490	-5.768	0.	0.	0.
200	29.107	179.985	194.272	-2.857	0.	0.	0.
250	29.111	186.481	192.088	-1.402	0.	0.	0.
298.15	29.124	191.609	191.609	0.	0.	0.	0.
300	29.125	191.789	191.610	0.054	0.	0.	0.
350	29.165	196.281	191.964	1.511	0.	0.	0.
400	29.249	200.181	192.753	2.971	0.	0.	0.
450	29.387	203.633	193.774	4.437	0.	0.	0.
500	29.580	206.739	194.917	5.911	0.	0.	0.
600	30.110	212.176	197.353	8.894	0.	0.	0.
700	30.754	216.866	199.813	11.937	0.	0.	0.
800	31.433	221.017	202.209	15.046	0.	0.	0.

Input data 2.1 Least square curve fitting input data , Nitrogen Cp(T), source : NIST Janaf Tables

T	Cp	T	Cp	T	Cp	T	Cp	T	Cp
100	29.104	1100	33.241	2500	36.616	3900	37.508	5300	38.013
200	29.107	1200	33.723	2600	36.713	4000	37.55	5400	38.046
250	29.111	1300	34.147	2700	36.801	4100	37.59	5500	38.08
298.15	29.124	1400	34.518	2800	36.883	4200	37.629	5600	38.116
300	29.125	1500	34.843	2900	36.959	4300	37.666	5700	38.154
350	29.165	1600	35.128	3000	37.03	4400	37.702	5800	38.193
400	29.249	1700	35.378	3100	37.096	4500	37.738	5900	38.234
450	29.387	1800	35.6	3200	37.158	4600	37.773	6000	38.276
500	29.58	1900	35.796	3300	37.216	4700	37.808		
600	30.11	2000	35.971	3400	37.271	4800	37.843		
700	30.754	2100	36.126	3500	37.323	4900	37.878		
800	31.433	2200	36.268	3600	37.373	5000	37.912		
900	32.09	2300	36.395	3700	37.42	5100	37.947		
1000	32.697	2400	36.511	3800	37.465	5200	37.981		

Now by using a least suare curve fitting program we can create a curve fitting equation.

Program 2.1.1 Least square curve fitting program

```
import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.Color;

class SCO11B4
```

```
{
  public static double[] pivotlugauss(double a[][],double b[])
  { //kısmi pivotlu gauss eleme yöntemi
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss eleme
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivotlama
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
              //gauss elemeyi çözme
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
              }
  }
  //geriye doğru yerine koyma
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}

public static double[] EKK(double xi[],double yi[],int n)
{
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
```

```
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  for(j=0;j<n+1;j++)
     {if(i==0 && j==0) A[i][j]=l;
     else  for(k=0;k<l;k++) A[i][j] += Math.pow(xi[k],(i+j));
     }
     for(k=0;k<l;k++) { if(i==0)  B[i]+= yi[k];
                  else      B[i] += Math.pow(xi[k],i)*yi[k];}
  }
//System.out.println(Matrix.toString(A));
//System.out.println(Matrix.toStringT(B));
X=pivotlugauss(A,B);
//X=B/A;
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
Text.printT(X);
 return X;
}

public static double funcEKK(double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=e.length;
double ff;
if(n!=0.0)
  { ff=e[n-1];
  for(int i=n-2;i>=0;i--)
   { ff=ff*x+e[i]; }
  }
else
  ff=0;
return ff;
}

public static double hata(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcEKK(e,x[k]))*(y[k]-funcEKK(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

public static double[][]  funcEKK(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
 {
             //aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
             // ara değer sayısı 2 dir
              int n=xi.length;
             int nn=(n-1)*(aradegersayisi+1)+1;
```

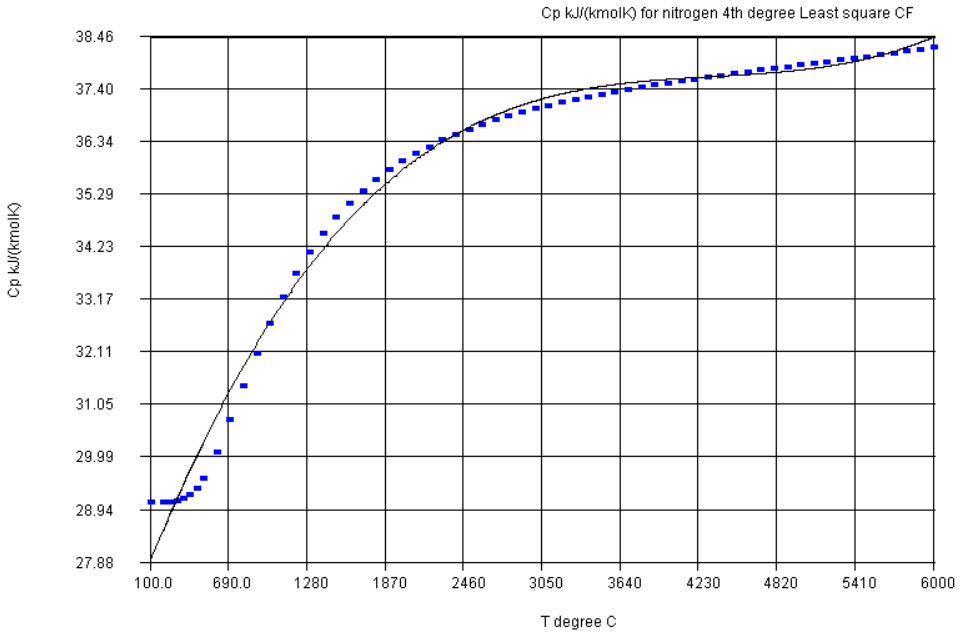
```
            double z[][]=new double[2][nn];
    double E[]=EKK(xi,yi,polinomkatsayisi);
    double hata=hata(xi,yi,E);
    //System.out.println("hata="+hata+"\nkatsayılar :\n"+Matrix.toStringT(E));
    double dx=0;
    int k=0;
    int i;
    for(i=0;i<(n-1);i++)
      {z[0][k]=xi[i];z[1][k]=funcEKK(E,z[0][k]);k++;
      for(int j=0;j<aradegersayisi;j++)
      {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
      z[0][k]=z[0][k-1]+dx;z[1][k]=funcEKK(E,z[0][k]);k++;}
      }
      z[0][k]=xi[i];z[1][k]=funcEKK(E,z[0][k]);
    return z;
  }
  public static double yavg(double y[])
  {
  int n=y.length;
  double total=0;
  for(int i=0;i<n;i++) total+=y[i];
  return total/n;
  }
  public static double R2(double y[],double f[])
  {
  double yavg=yavg(y);
  int n=y.length;
  double SStot=0;
  double SSerr=0;
  for(int i=0;i<n;i++)
  {SStot=(y[i]-yavg)*(y[i]-yavg);
   SSerr=(y[i]-f[i])*(y[i]-f[i]);
  }
  return (1-SSerr/SStot);
  }

  public static double R2(double x[],double y[],double e[])
  { int n=y.length;
  double f[]=new double[n];
    for(int k=0;k<n;k++)
     {f[k]=funcEKK(e,x[k]);}
    return R2(y,f);
  }

   public static void main(String args[]) throws IOException
   {
   double x[];
   double y[];
   String s1=JOptionPane.showInputDialog("dosya adı (R134a_Cp.txt)");
   //JFileChooser fc=new JFileChooser();
   //if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file =
fc.getSelectedFile();s1=file.getName(); }
   double a[][]=Text.readDoubleT(s1);
   x=a[0];
   y=a[1];
   int n=Integer.parseInt(JOptionPane.showInputDialog("polinom derecesi n:"));
   double z[][]=funcEKK(x,y,n,10);
   Text.printT(z);
   System.out.println("E.K.K.\n"+Matrix.toStringT(z));
```

```
  Plot pp=new Plot(a[0],a[1]);
  pp.setPlabel("Cp kJ/(kmolK) for nitrogen 4th degree Least square CF");
  pp.setXlabel("T degree C");
  pp.setYlabel("Cp kJ/(kmolK)");
  pp.setPlotType(0,23);
  pp.addData(z[0],z[1]);
  pp.setGrid(1,1);
  pp.setColor(0,Color.BLUE);
  pp.plot();
  }
}
```

The program output:



$C_p(T)$= 27.19872924529033+0.006943211110342824*T-1.568881107785314E-6*T^2+1.211840254124337E-10*T^3-7.162042075128584E-17*T^4

Fitted data into the curve:

100	27.8775	750	31.5747	1800	35.3193	2850	37.0442	3900	37.5865	4950	37.7812
150	28.2053	800	31.8112	1850	35.4406	2900	37.0902	3950	37.5971	5000	37.796
200	28.5256	850	32.0413	1900	35.5574	2950	37.1337	4000	37.6069	5050	37.812
225	28.6829	900	32.2651	1950	35.6698	3000	37.1746	4050	37.6162	5100	37.8292
250	28.8384	950	32.4827	2000	35.778	3050	37.2131	4100	37.6249	5150	37.8479
274.1	28.9863	1000	32.6942	2050	35.8818	3100	37.2493	4150	37.6332	5200	37.868
298.2	29.1326	1050	32.8996	2100	35.9816	3150	37.2833	4200	37.6411	5250	37.8896
299.1	29.1382	1100	33.0991	2150	36.0773	3200	37.3151	4250	37.6489	5300	37.9129
300	29.1438	1150	33.2928	2200	36.1691	3250	37.3449	4300	37.6564	5350	37.9379
325	29.2937	1200	33.4807	2250	36.257	3300	37.3727	4350	37.6639	5400	37.9647
350	29.4419	1250	33.6629	2300	36.3412	3350	37.3987	4400	37.6714	5450	37.9935
375	29.5882	1300	33.8395	2350	36.4217	3400	37.4228	4450	37.679	5500	38.0242
400	29.7327	1350	34.0107	2400	36.4986	3450	37.4453	4500	37.6869	5550	38.057
425	29.8755	1400	34.1765	2450	36.572	3500	37.4662	4550	37.695	5600	38.092
450	30.0165	1450	34.3369	2500	36.642	3550	37.4856	4600	37.7035	5650	38.1293
475	30.1558	1500	34.4922	2550	36.7086	3600	37.5035	4650	37.7124	5700	38.1689
500	30.2933	1550	34.6423	2600	36.7721	3650	37.5202	4700	37.722	5750	38.211
550	30.5631	1600	34.7874	2650	36.8324	3700	37.5355	4750	37.7322	5800	38.2556
600	30.826	1650	34.9276	2700	36.8897	3750	37.5498	4800	37.7431	5850	38.3028
650	31.0822	1700	35.0629	2750	36.944	3800	37.563	4850	37.7548	5900	38.3528
700	31.3318	1750	35.1934	2800	36.9955	3850	37.5752	4900	37.7675	5950	38.4056
										6000	38.4612

As it is seen from the least square cure fitting results, using a single equation is always cause errors, and error grows even further close to the critical point of gases. In this context when modelling a more accurate partially continious curve fitting method will be utilized in our modelling of ideal gas C_p

$Cp_i(T) = A_i + B_i*10^{-3}*T + C_i*10^5/T^2 + D_i*10^{-6}*T^2 \qquad T_{Li} >= T > T_{Hi} \qquad KJ/kmol\ K \qquad (2.1.13)$

By using this type of partially continuous equation gives us oppurtunity to adjust error range of the equations. In order to find coefficients of equation Least square curve fitting method will be applied. Let us define the method of Least sqauare It is assumed that T_i, C_{pi} i=0...(n-1) data is divided into L data region and data for each set becomes T_i,C_{pi} i=(n-1)/L*k...(n-1)/L*(k+1) k=0…(L-1). If a curve fitting function

$$C_{pk}(T) = \sum_{j=0}^{m} a_{jk}^{(m)} \phi_j(T), \qquad T_{Li} \le T \le T_{Hi} \quad k = 0 \dots (L-1) \quad (2.1.14)$$

general linear function is defined with multiplication functions $\phi_j(T)$ for each linear coefficient of equation. For the equation 2.1.13 this coefficients will be: $\phi_0=1$, $\phi_1=10^{-3}*T$, $\phi_2=10^5/T^2$, $\phi_3=10^{-6}*T^2$

In order to apply least square curve fitting minimisation of the following function will be carried out.

$$H\left(a_{0k}^{(m)},..,a_{mk}^{(m)}\right) = \sum_{i=\frac{n-1}{L}*k}^{\frac{n-1}{L}*(k+1)} w_k(T_i)\left[\frac{C_{pi} - \sum_{j=0}^{m} a_{jk}^{(m)}\phi_j(T_i)}{(L-1)}\right]^2 \qquad (2.1-15)$$

$w_k(T)$ is called the weight function. The value of the weight function can be assumed to be equal to unity to simplify the process. The minimum of the function is the roots of its derivatives with respect to coefficients of the equation..

$$\frac{\partial H_k\left(a_{0k}^{(m)},..,a_{mk}^{(m)}\right)}{\partial a_p^{(m)}} = \frac{2}{(L-1)}\sum_{i=1}^{n} w(T_i)\left[\frac{C_{pi} - \sum_{j=0}^{m} a_j^{(m)}\phi_j(T_i)}{(L-1)}\right]\phi_p(T_i) = 0 \quad p = 0,\dots,m \qquad (2.1.16)$$

This equation is m+1 linear system of equation and can be solved by using any system of equation solving methods, such as Gauss elimination process. For weight function $w(x_i)$=1, general least square equation becomes:

$$\begin{bmatrix} \sum_{i=1}^{n} w(T_i)\phi_0(T_i)\phi_0(T_i) & \sum_{i=1}^{n} w(T_i)\phi_0(T_i)\phi_1(T_i) & \sum_{i=1}^{n} w(T_i)\phi_0(T_i)\phi_2(T_i) & \cdots & \sum_{i=1}^{n} w(T_i)\phi_0(T_i)\phi_n(T_i) \\ \sum_{i=1}^{n} w(T_i)\phi_1(T_i)\phi_0(T_i) & \sum_{i=1}^{n} w(T_i)\phi_1(T_i)\phi_1(T_i) & \sum_{i=1}^{n} w(T_i)\phi_1(T_i)\phi_2(T_i) & \cdots & \sum_{i=1}^{n} w(T_i)\phi_2(T_i)\phi_n(T_i) \\ \sum_{i=1}^{n} w(T_i)\phi_2(T_i)\phi_0(T_i) & \sum_{i=1}^{n} w(T_i)\phi_2(T_i)\phi_1(T_i) & \sum_{i=1}^{n} w(T_i)\phi_2(T_i)\phi_2(T_i) & \cdots & \sum_{i=1}^{n} w(T_i)\phi_2(T_i)\phi_n(T_i) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \sum_{i=1}^{n} w(T_i)\phi_n(T_i)\phi_0(T_i) & \sum_{i=1}^{n} w(T_i)\phi_n(T_i)\phi_1(T_i) & \sum_{i=1}^{n} w(T_i)\phi_n(T_i)\phi_2(T_i) & \cdots & \sum_{i=1}^{n} w(T_i)\phi_n(T_i)\phi_n(T_i) \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \cdots \\ a_n \end{Bmatrix} =$$

$$\begin{Bmatrix} \sum_{i=1}^{n} w(T_i)\phi_0(T_i)C_p(T_i) \\ \sum_{i=1}^{n} w(T_i)\phi_1(T_i)C_p(T_i) \\ \sum_{i=1}^{n} w(T_i)\phi_2(T_i)C_p(T_i) \\ \cdots \\ \sum_{i=1}^{n} w(T_i)\phi_n(T_i)C_p(T_i) \end{Bmatrix} (2.1.17)$$

A curve fitting program is developed in Java language to carry out this procedure. The list of the program is given below.

Program 2.1.2 Partially continuous curve fitting program

```
import java.io.*;
import java.util.*;
import javax.swing.*;
abstract class f_xr
{abstract double func(double x,int equation_ref);}

class fh extends f_xr
{
            double func(double t,int i)
            {
    double T=t+273.15;
             double xx=0.0;
    if(i==0) xx=T;
```

```
    else if(i==1) xx=1.0e-3/2.0*T*T;
    else if(i==2) xx=-1.0e5/T;
    else if(i==3) xx=1.0e-6/3.0*T*T*T;
    return xx;
            }
}

class fcp extends f_xr
{
            double func(double t,int i)
            { // Cp(T) = xa[i]+xb[i]*1e-3*T+xc[i]*1.0e5/T^2+xd[i]*1e-6*T^2
    double T=t;
             double xx=0.0;
    if(i==0) xx=1.0;
    else if(i==1) xx=1.0e-3*T;
    else if(i==2) xx=1.0e5/(T*T);
    else if(i==3) xx=1.0e-6*T*T;
    return xx;
            }
}

class Gas_GEN_Data
{
//this class reads and store and proses ideal gas data
public double M;       // kg/kmol
public double h0;      // KJ/kmol at temperature a[0][0]=Tref
public double s0;      // KJ/kmolK at temperature a[0][0]=Tref
public double hf;      // KJ/kmol at 298 K
public double R=8.3145; // KJ/kgmolK
public double a[][];
public String GasName;
public String openName;
public String ekbilgi;
int fitoption;
f_xr f1;
f_xr f2;

public double cpl[][];  // ideal gas heat capacitiy
public double visl[][]; // saturated liquid viscosity
public double kl[][]; // saturated liquid thermal conductivity

//General partial continious least square method
  public Gas_GEN_Data(int fitoptioni,String GasNamei,String openNamei,double ai[][],int ngroup,double Mi, double
h0i,double s0i,double hfi)
  {
  //fitoption : 0   Cp
  //other         h
  fitoption=fitoptioni;
  GasName=GasNamei;
  a=ai;
  ekbilgi=" ";
  openName=openNamei;
  int n=a.length;
  int i,j;
  if(fitoption==0) {f1=new fcp(); }
  else           {f1=new fh(); }
  f2=new fcp();
  cpl=EKKgeneral(f1,a[0],a[2],ngroup);
  visl=EKKgeneral(f2,a[0],a[3],ngroup);
  kl=EKKgeneral(f2,a[0],a[4],ngroup);
  M=Mi;
  h0=h0i;
  s0=s0i;
  hf=hfi;
  }

  public static double[] pivotlugauss(double a[][],double b[])
```

```
  { //gauss eliminiation with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gausselimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivoting
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
              //solving of gauss elimination
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
              }
  }
  //back substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}

public static double[][] EKKgeneral(f_xr f, double xi[],double yi[],int ngroup)
{
              //ngroup : number of data in each group
              //negri  : degree of curve fitting
              //groupnumber : number of group
              int negri=3;
              int n=negri;
              double y[]=new double[xi.length];
              for(int i=0;i<xi.length;i++)
                 {y[i]=yi[i];}
if(ngroup<negri) ngroup=negri;
int Nxi=xi.length; //total number of data
int groupnumber=Nxi/ngroup;
int ngroup1=ngroup+Nxi%ngroup;
//
int i,j,k;
int np1=4;
int np5=np1+2;
double A[][];
```

```
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[][];
X=new double[groupnumber][np5];
System.out.println("group number = "+ groupnumber);
double Y[]=new double[np1];
int l=0;
double max=0;
//
for(l=0;l<groupnumber-1;l++)
{
for(i=0;i<np1;i++)
  {  B[i]=0;
                for(j=0;j<np1;j++)
    {
                A[i][j]=0.0;
    for(k=l*ngroup;k<=(l+1)*ngroup;k++) {A[i][j]+=f.func(xi[k],i)*f.func(xi[k],j);}
    }
    for(k=l*ngroup;k<=(l+1)*ngroup;k++) B[i]+= f.func(xi[k],i)*y[k];
  }
Y=pivotlugauss(A,B);//pivotlugauss(A,B);

int ii;
for(ii=0;ii<Y.length;ii++) {X[l][ii]=Y[ii];}
X[l][ii]=xi[l*ngroup];
X[l][ii+1]=xi[(l+1)*ngroup];
for(i=0;i<n+1;i++)
 if(Math.abs(X[l][i]) > max) max = Math.abs(X[l][i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[l][i]/max) > 0) && (Math.abs(X[l][i]/max) < 1.0e-100))X[l][i]=0;
}
l=groupnumber-1;
int k1=l*ngroup;
int k2=(l+1)*ngroup+Nxi%ngroup;
for(i=0;i<np1;i++)
  {  B[i]=0;
                for(j=0;j<np1;j++)
    {
                A[i][j]=0.0;
    for(k=k1;k<k2;k++) {A[i][j]+=f.func(xi[k],i)*f.func(xi[k],j);}
    }
    for(k=k1;k<k2;k++) {B[i]+= f.func(xi[k],i)*y[k];}
  }
Y=pivotlugauss(A,B);
int ii;
for(ii=0;ii<Y.length;ii++) {X[l][ii]=Y[ii];}
X[l][ii]=xi[k1];
X[l][ii+1]=xi[k2-1];

return X;
}

public static double funcEKKgeneral(f_xr f,double e[][],double x)
{
// this function calculates the value of
// least square curve fitting function
int n1=e.length;
int n2=e[0].length;
int n3=n2-2;
System.out.println("n1="+n1+"n2="+n2+"n3="+n3);
double ff=0;
double xlow,xhigh;
for(int j=0;j<n1;j++)
{
xlow=e[j][n3];
xhigh=e[j][n3+1];
```

```
System.out.println("xlow="+xlow+"xhigh="+xhigh);
if((x>=xlow) && (x<=xhigh))
{
ff=0;
for(int i=0;i<n3;i++)
  {ff+=e[j][i]*f.func(x,i);System.out.println("ff="+ff+"i="+i);}
break;
}
}
return ff;
}

public static double[][] xyfit(f_xr f,double x[],double yi[],double e[][])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 double yy=0;
 double z[][]=new double[2][x.length];
 for(k=0;k<n;k++)
 {   z[0][k]=x[k];
                z[1][k]=funcEKKgeneral(f,e,x[k]);
 }
return z;
}


public static double[][] dxyfit(f_xr f,double x[],double yi[],double e[][])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 double yy=0;
 double z[][]=new double[2][x.length];
 for(k=0;k<n;k++)
 {   z[0][k]=x[k];
                z[1][k]=yi[k]-funcEKKgeneral(f,e,x[k]);
 }
return z;
}

public static double error(f_xr f,double x[],double yi[],double e[][])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 double yy=0;
 for(k=0;k<n;k++)
 {
                yy=yi[k]-funcEKKgeneral(f,e,x[k]);
 total+=yy*yy;
 }
total=Math.sqrt(total/(n-1));
return total;
}
 public static double[][]  errorEKKPlot(f_xr f,double xi[],double yi[],int ngroup)
 {
                //ngroup : Number of data in each group
                int n=xi.length;
                int l;
                double z[][];
    double E[][];//=EKKgeneral(xi,yi,polinomkatnumber,ngroup);
    double dx=0;
    int k=0;
```

```
  int i;
            E=EKKgeneral(f,xi,yi,ngroup);
  z=dxyfit(f,xi,yi,E);
  Plot pp=new Plot(z[0],z[1]);
  pp.setPlabel("Cp=a0+a1*1e-3*T+a2*1e5/T^2+a3*1e-5*T^2  Tl<=T<=Th");
  pp.setXlabel("T degree K");
  pp.setYlabel("Error Cp KJ/kmolK");
  pp.plot();
  return z;
}

public static double[][]  funcEKKPlot(f_xr f,double xi[],double yi[],int ngroup)
{
            int n=xi.length;
            int l;
            double z[][];
  double E[][];
  double dx=0;
  int k=0;
  int i;
            E=EKKgeneral(f,xi,yi,ngroup);
  z=xyfit(f,xi,yi,E);
  Plot pp=new Plot(z[0],z[1]);
    pp.setPlabel("Cp=a0+a1*1e-3*T+a2*1e5/T^2+a3*1e-5*T^2  Tl<=T<=Th");
    pp.setXlabel("T temperature degree K");
    pp.setYlabel("Cp specific heat KJ/kmolK");
   pp.addData(xi,yi,1);
  pp.plot();
  return z;
}

public static double[][]  funcEKKgeneral(f_xr f,double xi[],double yi[],int ngroup,int subintervalnumber)
{
            int n=xi.length;
            int nn=(n-1)*(subintervalnumber+1)+1;
            double z[][]=new double[2][nn];
  double E[][]=EKKgeneral(f,xi,yi,ngroup);
  Text.print(E,"EfuncEKK");
  double dx=0;
  int k=0;
  int i;
  for(i=0;i<(n-1);i++)
    {z[0][k]=xi[i];z[1][k]=funcEKKgeneral(f,E,z[0][k]);
    for(int j=0;j<subintervalnumber;j++)
    {dx=(xi[i+1]-xi[i])/((double)subintervalnumber+1.0);
    k++;
    z[0][k]=z[0][k-1]+dx;z[1][k]=funcEKKgeneral(f,E,z[0][k]);k++;}
    }
    z[0][k]=xi[i];z[1][k]=funcEKKgeneral(f,E,z[0][k]);
  return z;
}

 public double Cp_l(double t)
 { return funcEKKgeneral(f2,cpl,t);}

 public double viscosity_l(double t)
 {return funcEKKgeneral(f2,visl,t);}

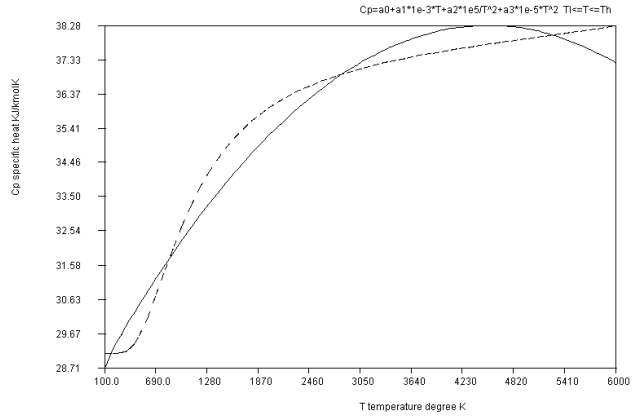
 public double k_l(double t)
 {return funcEKKgeneral(f2,kl,t);}

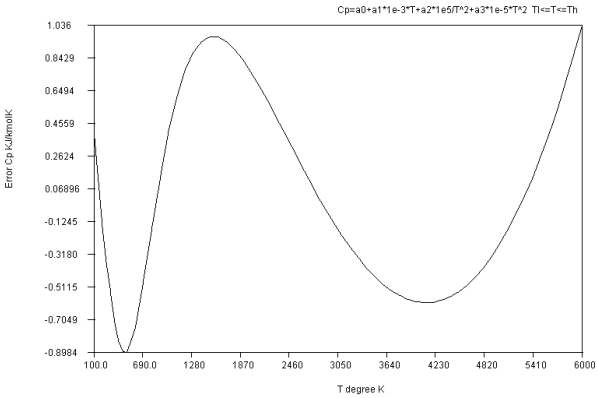

 public static void main(String args[]) throws IOException
 {
 String s1=JOptionPane.showInputDialog("name of the input file : ");
 double c[][]=Text.readDoubleT(s1);
 int n=c[0].length;
```

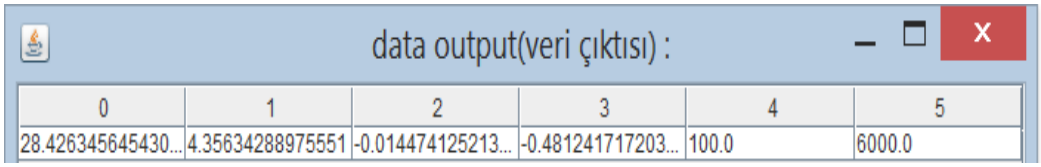
```
  Text.print(Text.T(c));
  double d[][]=EKKgeneral(new fcp(), c[0],c[1],n);
  funcEKKPlot(new fcp(), c[0],c[1],n);
  Text.print(d);
  errorEKKPlot(new fcp(), c[0],c[1],n);
  System.out.println(Matrix.toString(d));
  }

}
```

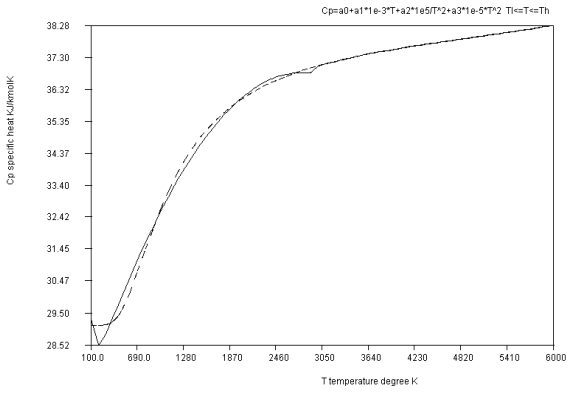
Output 2.1-1 Least square curve fitting, Nitrogen Cp(T) one equation for the full range (n=1)

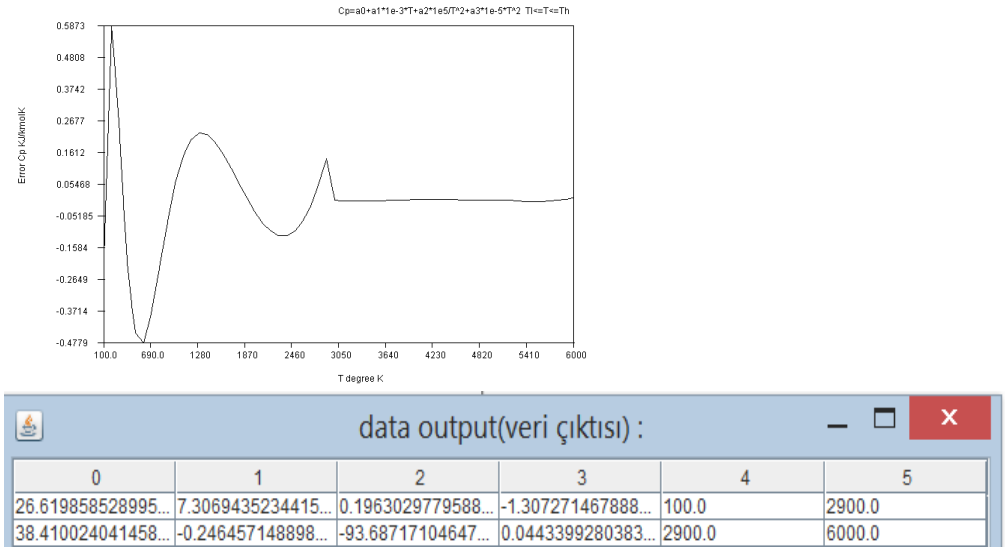






Output 2. 1-2 Least square curve fitting, Nitrogen Cp(T) one equation for the full range (n=2)

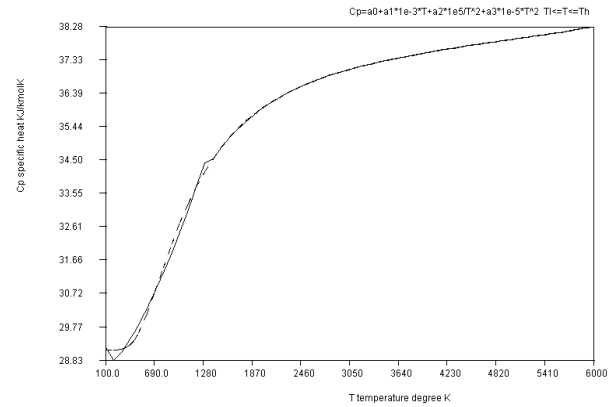


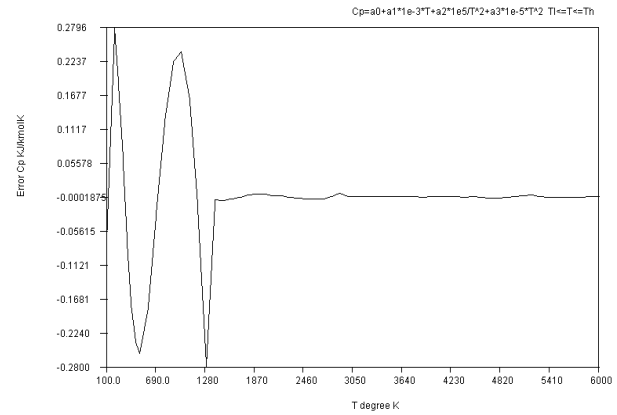


data output(veri çıktısı) :

0	1	2	3	4	5
26.619858528995...	7.3069435234415...	0.1963029779588...	-1.307271467888...	100.0	2900.0
38.410024041458...	-0.246457148898...	-93.68717104647...	0.0443399280383...	2900.0	6000.0

Output 2. 1-2 Least square curve fitting, Nitrogen Cp(T) one equation for the full range (n=4)

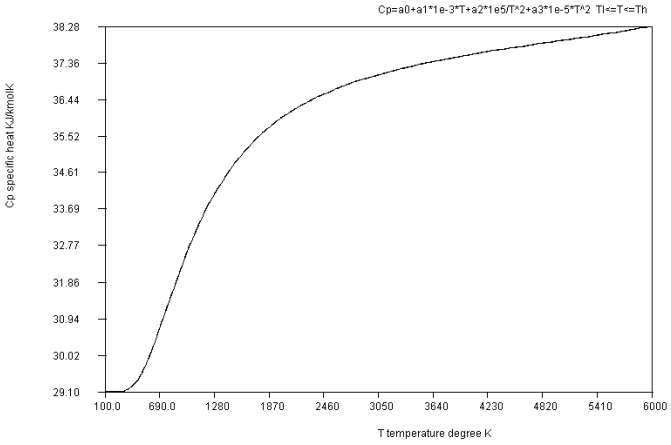


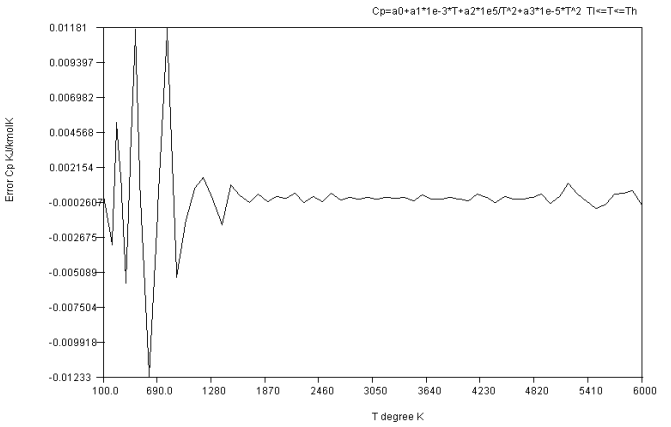


data output(veri çıktısı) :

0	1	2	3	4	5
27.95585931143741	2.890045926186791	0.09175734463285376	1.6027541706504944	100.0	1300.0
34.129733722662024	1.815381004555621	-32.73483985828752	-0.243935897854228...	1300.0	2900.0
37.10276237021983	0.2150440463728742	-63.93913586682272	-8.392331032663524...	2900.0	4500.0
45.43488884308771	-2.0861200239998876	-392.3837574823669	0.17908400732262333	4500.0	6000.0

Output 2. 1-2 Least square curve fitting, Nitrogen Cp(T) one equation for the full range (n=10)





data output(veri çıktısı) :

0	1	2	3	4	5
29.555385254879834	-3.232642098568939	-0.019052612012152012	6.224236424391002	100.0	400.0
23.988366007007876	9.959151874366158	2.417913410404676	-1.4254648553558849	400.0	900.0
25.86164354664174	9.448074436200576	-3.782953765636655	-2.232690628715062	900.0	1500.0
32.88912500566657	2.803699246843971	-27.35073189104521	-0.4604684103153939	1500.0	2100.0
35.979623210734	0.749609529291484	-48.70788838331627	-0.07328997910652715	2100.0	2700.0
36.175006858688064	0.6214704973224496	-49.66177442961639	-0.05085601729249234	2700.0	3300.0
35.978446045142014	0.6414315086898076	-40.934634506230836	-0.046214975739819024	3300.0	3900.0
40.933267819293036	-0.9995199973148224	-176.40761457591654	0.10733882875920711	3900.0	4500.0
35.901894859933094	0.4422039657867144	5.059559515400795	-0.008831649055007006	4500.0	5100.0
48.26707554027542	-2.8182573871402785	-514.3623038423627	0.23188115973340007	5100.0	6000.0

Program outputs clearly indicates how the error level of the equations can be improved with further division of the curve fitting data ranges. In table 2.1 coefficients calculates for most common gases are given.

Table 2. 1 Cp specific energy at constant pressure coefficients for partially continuous equation 2.23 for some selected gases (KJ/kmol K)

O2(OXYGEN)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
22.2733042	20.09559	1.5767767	-7.443438	298	600
26.2330328	13.48713	-2.0281281	-4.647975	600	1000
35.7634821	0.5428001	-17.710825	0.3348063	1000	1400
34.5568094	1.5534385	-12.681988	0.097656	1400	1800
32.682339	2.7805744	-0.2328397	-0.124124	1800	2200
34.1893024	2.0032457	-16.591274	-0.012322	2200	2600
23.1341719	7.7683414	105.06076	-0.860499	2600	3000
35.3620341	2.1744887	-71.654038	-0.136367	3000	3400
37.7138695	1.3732901	-128.8807	-0.061344	3400	3800
49.4863253	-2.649491	-428.22144	0.3255739	3800	4200
57.8026664	-5.225491	-684.17715	0.5497138	4200	4600
67.3497053	-7.837178	-1057.8244	0.7497392	4600	5000

O(OXYGEN)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
21.671447	-2.000487	0.6392959	1.3201975	298.2	600
20.9035553	-0.139446	1.0331238	0.0475465	600	1000
20.4553758	0.3849957	1.9839752	-0.123782	1000	1400
21.6687811	-0.766509	-2.1607524	0.1875427	1400	1800
17.6656043	1.9100293	24.481016	-0.317612	1800	2200
24.9178388	-2.537163	-32.696431	0.4494757	2200	2600
21.487001	-0.625	-1.33E-05	0.1500001	2600	3000
17.0117402	1.3203316	70.107043	-0.087746	3000	3400
8.95294307	4.2042415	252.79664	-0.375537	3400	3800
20.0977517	0.1079969	7.5767905	0.0482227	3800	4200
-1.3420264	6.7095788	678.62469	-0.523824	4200	4600
724.70512	-195.5212	-26984.851	15.305527	4600	5000

CH4 (METHANE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
0.26362115	103.39063	6.665204	-33.10402	298	600
6.78991069	89.603901	2.2331944	-24.83552	600	1000
27.7451351	61.806127	-33.708994	-14.39854	1000	1400
80.8115508	12.886412	-220.78185	-1.66196	1400	2000

CO2 (CARBONDIOXIDE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
25.4671973	51.023411	-1.3120972	-23.31607	298.2	600
34.6006273	30.277362	-6.7786053	-9.892535	600	1000
48.1290242	11.835579	-28.908503	-2.765834	1000	1400
54.2079272	5.7117773	-46.594799	-1.032728	1400	1800
59.0590366	2.0250122	-71.9214	-0.240514	1800	2200
62.0435167	0.2785119	-98.089295	0.0484271	2200	2600
65.5225099	-1.394249	-142.79412	0.2749848	2600	3000

C2H6 (ETHANE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
7.22082458	169.49999	-0.310051	-54.76307	298.1	600
15.8323038	151.00556	-6.0042494	-43.46719	600	1000

C4H10(BUTHANE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
8.84111125	333.98478	0.1153296	-115.5945	298.1	600
6.01218614	339.97468	2.0345276	-119.2002	600	1000
-3.8865264	352.83922	19.725427	-123.9349	1000	1500

C6H6(BENZENE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
-18.481746	402.83801	-2.687926	-180.804	298.1	600
53.4042651	249.95242	-51.078552	-88.3468	600	1000
302.997008	-74.83644	-496.35487	31.372196	1000	1500

C3H8(PROPANE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
2.7092291	274.00712	-1.2065294	-104.1566	298	600
35.0742571	205.13062	-22.964953	-62.47967	600	1000
147.689338	58.572903	-223.8476	-8.450814	1000	1500

C5H12(PENTANE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
7.82987398	434.08356	-1.9963999	-168.1041	298	600
61.2821127	320.36074	-37.953456	-99.30518	600	1000

NO(NITROUX OXIDE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}

26.0647325	6.3213237	1.472044	2.6988662	298	600
22.8517346	16.355181	1.876206	-5.412385	600	1000
30.8676094	5.6277872	-11.7589	-1.33654	1000	1400
35.035619	1.6318768	-25.363977	-0.254688	1400	1800
36.4013997	0.6632486	-33.453572	-0.061003	1800	2200
34.6513029	1.6332455	-16.591279	-0.212322	2200	2600
34.0190004	1.7499998	-5.73E-06	-0.2	2600	3000

NO2(NITROUSOXIDE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
35.7317582	22.766346	-4.715792	-6.218409	298	600
35.771235	22.757046	-4.7810842	-6.262163	600	1000
35.7679926	22.813891	-4.8874622	-6.305161	1000	1400
177.159804	-96.36465	-581.69215	21.69875	1400	1800
53.8006783	1.2561487	-0.4656812	0.0017521	1800	2200
55.3567682	0.3959388	-15.619017	0.1359161	2200	2600
53.8428231	1.2484724	-1.8087021	0.0017687	2600	3000

He(HELIUM)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
20.786	0	0	0	298	3000

CO(KARBONMONOXIDE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
28.6560206	-2.085128	0.337893	8.1811747	298.2	600
20.3829755	18.160934	4.52649	-5.81348	600	1000
26.7303654	9.2194438	-4.8584718	-2.281008	1000	1400
33.941026	2.1828711	-27.618905	-0.341279	1400	1800
34.8280427	1.5459515	-32.522207	-0.214507	1800	2200
33.3217675	2.2909392	-15.619011	-0.314084	2200	2600
32.8390896	2.2832633	0.9043359	-0.275884	2600	3000

H2O(STEAM)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
28.6877265	12.039738	1.1463393	0.25555	298.2	600
27.3108165	14.029766	2.5637497	-0.329693	600	1000
19.4614944	25.197732	14.032779	-4.795708	1000	1400
26.780184	18.605935	-12.587814	-3.128211	1400	1800
37.0771036	10.778625	-65.742941	-1.451386	1800	2200
44.5695869	6.3876967	-130.29958	-0.727978	2200	2600
46.3095925	5.3240178	-141.88969	-0.5509	2600	3000
51.008483	3.1345329	-205.6806	-0.26442	3000	3400
41.4946818	6.6320807	2.4818852	-0.625887	3400	3800
59.8614036	0.0025212	-416.85494	0.0479064	3800	4200
68.5382723	-2.651393	-689.72852	0.2756027	4200	4600
50.0986401	2.6143779	-15.736572	-0.148221	4600	5000

H(HIDROGEN)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
20.786	0.00E+00	0.00E+00	0.00E+00	298.2	5000

NH3(AMMONIA)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
20.3177799	48.018943	1.8656171	-12.18994	298.2	600
22.1735304	44.053981	0.6126955	-9.769835	600	1000
26.5786168	38.208896	-6.8714374	-7.581379	1000	1400
36.5302097	28.798806	-40.206718	-5.069489	1400	1800
50.6669644	18.375459	-117.49906	-2.90565	1800	2200
72.7680843	4.9328427	-295.24014	-0.602956	2200	2600
108.872152	-13.48839	-707.63993	2.0437324	2600	3000

Ar(ARGON)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
20.786	0	0	0	298.2	5000

Another way for the interpolation process is to fit a different polynomial in between each two point. If a third degree polinomial is considered:

$$r_k(x) = a_k(x - x_k)^3 + b_k(x - x_k)^2 + c(x - x_k)^3 + y_k \quad 1 \le k \le n \quad (2.1.18)$$

In the interpolation proses polinoms should be passing through all data points

$$r_k(x_{k+1}) = y_{k+1} \quad 1 \le k \le n \quad (2.1.19)$$

In the same time the first derivative of the polynomial should also be continious while passing from one polynomial to the next one at the data point

$$r'_{k-1}(x_k) = r'_k(x_k) \quad 1 \le k \le n \quad (2.1.20)$$

For the third degree polinomial second derivative of the polynomial should also be continious while passing from one polynomial to the next one at the data point

$$r''_{k-1}(x_k) = r''_k(x_k) \quad 1 \le k \le n \quad (2.1.21)$$

All these conditions are not enough to solve the coefficients of the polinomials. Two more conditions are required. This two additional conditions (A and B of the following equation) can be given by user

$$r''_1(x_1) = A \qquad r''_{n-1}(x_n) = B \quad (2.1.22)$$

They are the second derivatives at the both hand of the series of polinomials. If A and B values are taken equals to 0, it is called a **natural cubic spline**. Other end conditions such as the ones depends one the first derivatives can also be set to solve the system of equations.

Defining $h_k = x_{k+1} - x_k \quad 1 \le k \le n \quad (2.1.23)$

System of equations become:

$$a_k h_k^3 + b_k h_k^2 + b_k h_k = y_{k+1} - y_k \quad 1 \le k \le n \quad (2.1.24)$$
$$3a_{k-1}h_{k-1}^2 + 2b_{k-1}h_{k-1} + c_{k-1} - c_k = 0$$
$$6a_{k-1}h_{k-1} + 2b_{k-1} + 2b_k = 0$$
$$3b_0 = 0$$
$$6a_{n-1}h_{n-1} + 2b_{n-1} = 0$$

This set contains 3n-3 equations. This could a considerable load to the system of equation solving programs. To make calculation load simpler a special third degree polinomial can be considered. If our cubic polinomial is in the form of:

$$s_k(x) = a_k(x - x_k) + b_k(x_{k+1} - x) + [(x - x_k)^3 c_{k+1} + (x_{k+1} - x)^3 c_k]/(6h_k) \quad 1 \le k \le n \quad (2.1.25)$$

then derivative equations becomes

$$s'_k(x) = a_k - b_k + [(x - x_k)^2 c_{k+1} - (x_{k+1} - x)^2 c_k]/h_k \quad 1 \le k \le n \quad (2.1.26)$$
$$s''_k(x) = [(x - x_k)c_{k+1} - (x_{k+1} - x)c_k]/h_k \quad 1 \le k \le n$$

a_k ve b_k coefficients can be expressed as a function of c_k

$$b_k = \frac{[6y_k - h_k c_k]}{6h_k} \quad 1 \le k \le n \quad (2.1.27)$$

$$a_k = \frac{[6y_{k+1} - h_k^2 c_{k+1}]}{6h_k} \quad 1 \le k \le n \quad (2.1.28)$$

In this case only c_k terms left in the system of equations to be solved.

$$h_{k-1}c_{k-1} + 2(h_{k-1} - h_k)c_{k-1} + h_k c_{k+1} c_{k+1} = 6\left[\frac{y_{k+1} - y_k}{h_k} - \frac{y_k - y_{k-1}}{h_{k-1}}\right] \quad 1 \le k \le n \quad (2.1.29)$$

This system of equation has only n-2 terms to be solved. By making definition

$$w_k = \frac{y_{k+1} - y_k}{h_k}, \quad 1 \le k \le n \quad (2.1.31)$$

System of equation becomes

$$\begin{bmatrix} 1 & & & & & & & \\ h_1 & 2(h_1+h_2) & h_2 & & & & & \\ & h_2 & 2(h_2+h_3) & \cdots & & & & \\ & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ & & & \cdots & 2(h_{n-3}+h_{n-2}) & h_{n-2} & & \\ & & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & \\ & & & & & 1 & & \end{bmatrix} \begin{Bmatrix} c_0 \\ c_1 \\ c_2 \\ \cdots \\ c_{n-2} \\ c_{n-1} \\ c_n \end{Bmatrix} = \begin{Bmatrix} A \\ 6(w_2-w_1) \\ 6(w_3-w_2) \\ \cdots \\ 6(w_{n-2}-w_{n-3}) \\ 6(w_{n-1}-w_{n-2}) \\ B \end{Bmatrix} \quad (2.1.30)$$

Where A and B are the second derivative end conditions. A and B should be defined by user. Another important property of the above matrix is that it is a band matrix, therefore less amount of calculation is required to solve it (by using band matrix algorithms such as Thomas algorithm). Now by using cubic spline, the same Nitrogen Cp data will be used for the curve fitting.

Program 2.1.3 Cubic spline curve fitting program

```
 //Cubic spline curve fitting
import java.io.*;
import javax.swing.*;

class NA49
{
  public static double [] thomas(double f[],double e[],double g[],double r[])
  {
  // 3 band matrix system of equation solving algorithm
  int n=f.length;
  double x[]=new double[n];
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
     }
  for(int k=1;k<n;k++)
     {r[k]=r[k]-e[k]*r[k-1];
     }
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
     {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
  return x;
  }


  public static double [][] cubic_spline(double xi[],double yi[],double c0,double cn)
  {
  int n=xi.length;
  double h[]=new double[n];
  double w[]=new double[n];
  double f[]=new double[n];
  double e[]=new double[n];
  double g[]=new double[n];
  double d[]=new double[n];
  double x[]=new double[n];
  double S[][]=new double[4][n];
  int k;
  for(k=0;k<(n-1);k++)
     {h[k]=xi[k+1]-xi[k];
      w[k]=(yi[k+1]-yi[k])/h[k];
           }
  d[0]=c0;
  d[n-1]=cn;
  for(k=1;k<(n-1);k++)
     {d[k]=6.0*(w[k]-w[k-1]);}
  f[0]=1.0;
  f[n-1]=1.0;
```

```
 g[0]=0.0;
 g[n-1]=0.0;
 e[0]=0.0;
 e[n-1]=0.0;
 for(k=1;k<(n-1);k++)
 {f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];}
 S[2]=thomas(f,e,g,d);
 S[3]=xi;
 for(k=0;k<(n-1);k++)
   {S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
    S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
   }
 return S;
 }

 public static double funcSpline(double S[][],double x)
 {
 int n=S[0].length;
 double xx1=0;
 double xx2=0;
 double y=0;
 double hk=0;
 for(int k=0;k<(n-1);k++)
    {if(S[3][k]<=x && x<=S[3][k+1])
     {hk=(S[3][k+1]-S[3][k]);
      xx1=(x-S[3][k]);
      xx2=(S[3][k+1]-x);
      y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
      break;
      }
    }
      if(y==0 && S[3][n-2]<=x )
      {
               int k=n-2;
               hk=(S[3][k+1]-S[3][k]);
      xx1=(x-S[3][k]);
      xx2=(S[3][k+1]-x);
      y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
      }
 return y;
 }

public static double[][]  funcSpline(double xi[],double yi[],int numberofmidpoints)
{
            //numberofmidpoints :  in x--o--o--x--o--o--x chain if x's are esxperimental points
            // numberofmidpoints is 2
            int n=xi.length;
            int nn=(n-1)*(numberofmidpoints+1)+1;
            double z[][]=new double[2][nn];
   double S[][]=cubic_spline(xi,yi,0,0);
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
      {  z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);k++;
         for(int j=0;j<numberofmidpoints;j++)
            {dx=(xi[i+1]-xi[i])/((double)numberofmidpoints+1.0);
         z[0][k]=z[0][k-1]+dx;z[1][k]=funcSpline(S,z[0][k]);k++;}
      }
```

```
      z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);
    return z;
  }
   // Türev formülleri
// =================
   public static double dfSpline(double xi[],double yi[],double c0,double cn,double x)
   { //kübik şerit türev formülü
            double S[][]=cubic_spline(xi,yi,c0,cn);
    return dfSpline(S,x);
   }

   public static double dfSpline(double S[][],double x)
   {
     //kübik şerit türev formülü
   int n=S[0].length;
   double xx1=0;
   double xx2=0;
   double y=0;
   double hk=0;
   for(int k=0;k<(n-1);k++)
      {if(S[3][k]<=x && x<=S[3][k+1])
       {hk=(S[3][k+1]-S[3][k]);
        xx1=(x-S[3][k]);
        xx2=(S[3][k+1]-x);
        y=S[0][k]-S[1][k]+(xx1*xx1*S[2][k+1]-xx2*xx2*S[2][k])/(2.0*hk);
        break;
        }
      }
        if(y==0 && S[3][n-2]<=x )
        {
                int k=n-2;
                hk=(S[3][k+1]-S[3][k]);
        xx1=(x-S[3][k]);
        xx2=(S[3][k+1]-x);
        y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
        }
   return y;
   }

  public static double  dfSpline(double xi[],double yi[],double x)
  {
     //doğal kübik şerit türev formülü
            double S[][]=cubic_spline(xi,yi,0,0);
    return dfSpline(S,x);
  }

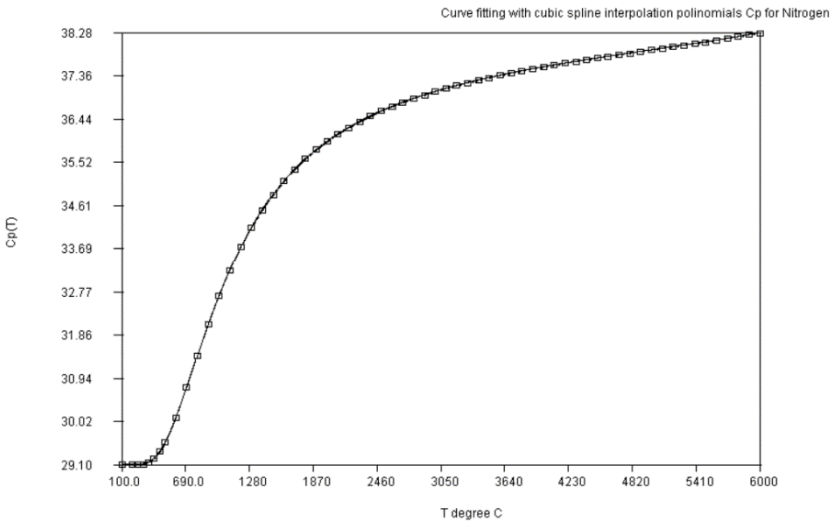
   public static double intSpline(double S[][],double a,double b)
   {
     //kübik şerit integral[S(x)]dx  formülü
   int n=S[0].length;
   double xx1=0;
   double xx2=0;
   double y1,y2;
   double hk=0;
   double toplam=0;
   y1=0;y2=0;
   for(int k=0;k<(n-1);k++)
      { hk=(S[3][k+1]-S[3][k]);
        if(a>S[3][k+1])
```

```
      {toplam=0;
      }
               else if(S[3][k]<=a && a<=S[3][k+1] && b>S[3][k+1])//şart 2
       {
       xx1=(a-S[3][k]);
       xx2=(S[3][k+1]-a);
       y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-
xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       xx1=hk;
       xx2=0;
       y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-
xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       toplam+=(y2-y1);
       }
       else if(S[3][k]<=a && a<=S[3][k+1] && S[3][k]<=b && b<=S[3][k+1])//şart 3
       {
       xx1=(a-S[3][k]);
       xx2=(S[3][k+1]-a);
       y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-
xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       xx1=(b-S[3][k]);
       xx2=(S[3][k+1]-b);
       y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-
xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       toplam+=(y2-y1);
       }
       else if(a<S[3][k] && b>=S[3][k+1]) //şart 4
       {
       xx1=0;
       xx2=hk;
       y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-
xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       xx1=hk;
       xx2=0;
       y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-
xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       toplam+=(y2-y1);
       }
       else if(S[3][k]<=b && b<=S[3][k+1])//şart 5
       {xx1=0;
        xx2=hk;
        y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-
xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
        xx1=(b-S[3][k]);
        xx2=(S[3][k+1]-b);
        y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-
xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
        toplam+=(y2-y1);
        }
        else break;
     }
  return toplam;
  }
public static double golden(double S[][],double a,double b)
{
//  find the minimum of the function
//  note  maximum f(x) = minimum (-f(x))
    double epsilon=1.0e-10;
    double delta=1.0e-8;
```

```
    int print=0;
    double r1 = (Math.sqrt(5.0)-1.0)/2.0; // golden ratio
    double r2 = r1*r1;
    double h = b - a;
    double ya = -funcSpline(S,a);
    double yb = -funcSpline(S,b);
    double c = a + r2*h;
    double d = a + r1*h;
    double yc = -funcSpline(S,c);
    double yd = -funcSpline(S,d);
    int k = 1;
    double dp,dy,p,yp;
    while ((Math.abs(yb-ya)>epsilon) || (h>delta))
    {
      k++;
      if (yc<yd)
        {
        b = d;
        yb = yd;
        d = c;
        yd = yc;
        h = b - a;
        c = a + r2 * h;
        yc = -funcSpline(S,c);
        }
      else
       {
         a = c;
         ya = yc;
         c = d;
         yc = yd;
         h = b - a;
         d = a + r1 * h;
         yd = -funcSpline(S,d);
       }//end of if
    }//end of while
    dp = Math.abs(b-a);
    dy = Math.abs(yb-ya);
    p = a;
    yp = ya;
    if (yb<ya)
    {
      p = b;
      yp = yb;
    }
    if(print==1)
    {System.out.println("x min = "+p+"ymin = "+yp+"error of x ="+dp+"error of y"+dy); }
    return p;
}

  public static void main(String args[]) throws IOException
  {
  String s1=JOptionPane.showInputDialog("file name : ");
  double c[][]=Text.readDoubleT(s1);
  double z3[][]=funcSpline(c[0],c[1],20);
  System.out.println("Cubic spline \n"+Matrix.toStringT(z3));
  Text.print(Text.T(z3),"Curve fitting with cubic spline interpolation polinomials");
  Plot pp=new Plot(c[0],c[1]);
  pp.setPlabel("Curve fitting with cubic spline interpolation polinomials");
```

```
  pp.setPlotType(0,20);
  pp.addData(z3[0],z3[1]);
  pp.plot();
  }
}
```



As it is seen from the graphic, cubic spline will give us almost perfect fitting. The only problem with this kind of fitting is that there are too many eqautions and coeficients for the curve fitting. But since computer programs can keep the coefficients in memory, it will not cause any problem.

We would like to investigate also how good single equations given by the reseachers would behave with respect to actual behaviors. Aly-Lee[41] equation, which is based on statistical thermodynamics has the form:

$$C_{p(T)} = a_0 + a_1\left(\frac{a_2/T}{\sinh\,(a_2/T)}\right) + a_3\left(\frac{a_4/T}{\sinh\,(a_4/T)}\right) \quad (2.1.31)$$

This equation is non-linear, so in order to curve-fit data into this equation non-linear equation fitting method is required. The following program utilises Nelder-Mead Curve fitting Method.

Program 2.1.4 Non linear Nelder-Mead optimisation curve fitting program for Aly-Lee equation

```
 import java.io.*;
import javax.swing.*;

    class yy implements if_xj
    {
            double xi[]; // independent variable data set
    double yi[]; //dependent variable data set
    double a[];  //fit function coefficient set
    int nn;
            public yy(String filename,double ia[])
    {
      //read the data to curvefit
      //get the data file and initial fit coefficient when class is defined
               double[] Xi=new double[500];
      double[] Yi=new double[500];
      int n=ia.length;
      a=new double[n];
      seta(ia);
      int i=-1;
      try{
               BufferedReader fin=new BufferedReader(new FileReader(filename));
```

```
      try {
      while(fin != null)
      {
      i++;
      Xi[i]=Text.readDouble(fin);
      Yi[i]=Text.readDouble(fin);
      }
      } catch(EOFException e_eof) {System.out.println("end of file"); }
      }catch(IOException e_io) {System.out.println("dosya bulunamadı"); }
    nn=i;
    xi=new double[nn];
    yi=new double[nn];
    for(int j=0;j<nn;j++){xi[j]=Xi[j];yi[j]=Yi[j];}
    a=ia;
    }

    public void seta(double ia[])
    {
                //assign new fit coefficient set
                for(int ii=0;ii<nn;ii++)
          a[ii]=ia[ii];
             }
             public double[] geta()
    {
               // return fit coefficient set
               return a;
             }

             double Cp(double T,double ai[])
    {a=ai;
    double A1=ai[2]/T/Math.sinh(ai[2]/T);
    double A2=a[3]/T/Math.cosh(ai[4]/T);
    double cp=ai[0]+ai[1]*A1*A1+ai[3]*A2*A2;
    return cp; //kJ/kgK
    }

    public double func(double ai[])
    {
    double ff=0;
    double w;
    double yy;
     for(int i=0;i<nn;i++)
               {w= Cp(xi[i],ai)-yi[i];
      ff+=w*w;
               }
    return ff;
               }
}
public class nelderi
{//2. Nelder ve Mead method(simpleks
method)_____
public static double[] nelder(if_xj fnelder,double a[],double da[],int maxiteration,double tolerance,int
printlist)
{
double x[][]=new double[a.length+1][a.length];
for(int i=0;i<x.length;i++)
  {for(int j=0;j<x[0].length;j++)
   {if(i==j){x[i][j]=a[j]+da[j];}
    else    {x[i][j]=a[j]; }
```

```
  }
}
// Nelder mead multidimensional simplex minimisation method
//  Nelder & Mead 1965 Computer J, v.7, 308-313.

// Giriş değişkenleri tanımlaması
// fnelder : abstract çok boyutlu fonksiyon f(x)
// x : for n boyutlu n+1 simlex noktasını içerir bağımsız değişken seti
// maxiteration : maximum iterasyon sayısı
// tolerance :
    int NDIMS = x.length-1;
    int NPTS = x.length;
    int FUNC = NDIMS;
    int ncalls = 0;
      ////// başlangıç simplexini oluştur ///////////////////
      double p[][]=new double[NPTS][NPTS]; // [row][col] = [whichvx][coord,FUNC]
      double z[]=new double[NDIMS];
      double best = 1E99;
      ///////////////// iilk fonksiyon değerlerini hesapla /////////////////
      for (int i=0; i<NPTS; i++)
      {
       for (int j=0; j<NDIMS; j++)
          {p[i][j] = x[i][j];}
       p[i][NDIMS] = fnelder.func(p[i]);
      }
      int iter=0;
      for (iter=1; iter<maxiteration; iter++)
      {
        ///////////  lo, nhi, hi noktalarını tanımla ///////////////
        int  ilo=0, ihi=0, inhi = -1; // -1 means missing
        double flo = p[0][FUNC];
        double fhi = flo;
        double pavg,sterr;
        for (int i=1; i<NPTS; i++)
        {
          if (p[i][FUNC] < flo)
            {flo=p[i][FUNC]; ilo=i;}
          if (p[i][FUNC] > fhi)
            {fhi=p[i][FUNC]; ihi=i;}
        }
        double fnhi = flo;
        inhi = ilo;
        for (int i=0; i<NPTS; i++)
         if ((i != ihi) && (p[i][FUNC] > fnhi))
           {fnhi=p[i][FUNC]; inhi=i;}
        ////////// çıkış kriteri ///////////////
        if ((iter % 4*NDIMS) == 0)
        {
          //yi nin standart hata kriteri set değerinden (tolerance)
          // küçük olmalı
          // ortalama değeri hesapla (en büyük değer de dahil olmak üzere)
          pavg=0;
          for(int i=0;i<NPTS;i++)
            pavg+=p[i][FUNC];
          pavg/=NPTS;
          double tot=0;
          if(printlist!=0)
          {  System.out.print(iter);
            for (int j=0; j<=NDIMS; j++)
```

```
          { System.out.print(p[ilo][j]+" ");}
        System.out.println("");
      }
      for(int i=0;i<NPTS;i++)
      { tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg);}
      sterr=Math.sqrt(tot/NPTS);
      //if(sterr < tolerance)
        { for (int j=0; j<NDIMS; j++)
            { z[j]=p[ilo][j];}
          //break;
        }
      best = p[ilo][FUNC];
    }

    ///// ave[] vektorünü en büyük değeri hariç tutarak hesapla //////

    double ave[] = new double[NDIMS];
    for (int j=0; j<NDIMS; j++)
      ave[j] = 0;
    for (int i=0; i<NPTS; i++)
      if (i != ihi)
        for (int j=0; j<NDIMS; j++)
          ave[j] += p[i][j];
    for (int j=0; j<NDIMS; j++)
      ave[j] /= (NPTS-1);


    ///////// yansıt ////////////////

    double r[] = new double[NDIMS];
    for (int j=0; j<NDIMS; j++)
      r[j] = 2*ave[j] - p[ihi][j];
    double fr = fnelder.func(r);

    if ((flo <= fr) && (fr < fnhi))  // in zone: accept
    {
      for (int j=0; j<NDIMS; j++)
        p[ihi][j] = r[j];
      p[ihi][FUNC] = fr;
      continue;
    }

    if (fr < flo)  //// genişlet, else kabul et
    {
      double e[] = new double[NDIMS];
      for (int j=0; j<NDIMS; j++)
        e[j] = 3*ave[j] - 2*p[ihi][j];
      double fe = fnelder.func(e);
      if (fe < fr)
      {
        for (int j=0; j<NDIMS; j++)
          p[ihi][j] = e[j];
        p[ihi][FUNC] = fe;
        continue;
      }
      else
      {
        for (int j=0; j<NDIMS; j++)
          p[ihi][j] = r[j];
```

```
                p[ihi][FUNC] = fr;
                continue;
              }
            }

            ////////////// daralt:

            if (fr < fhi)
            {
              double c[] = new double[NDIMS];
              for (int j=0; j<NDIMS; j++)
                c[j] = 1.5*ave[j] - 0.5*p[ihi][j];
              double fc = fnelder.func(c);
              if (fc <= fr)
              {
                for (int j=0; j<NDIMS; j++)
                  p[ihi][j] = c[j];
                p[ihi][FUNC] = fc;
                continue;
              }
              else   /////// daralt
              {
                for (int i=0; i<NPTS; i++)
                  if (i != ilo)
                  {
                    for (int j=0; j<NDIMS; j++)
                      p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                    p[i][FUNC] = fnelder.func(p[i]);
                  }
                continue;
              }
            }

            if (fr >= fhi)   ///
            {
              double cc[] = new double[NDIMS];
              for (int j=0; j<NDIMS; j++)
                cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
              double fcc = fnelder.func(cc);
              if (fcc < fhi)
              {
                for (int j=0; j<NDIMS; j++)
                  p[ihi][j] = cc[j];
                p[ihi][FUNC] = fcc;
                continue;
              }
              else    /////////
              {
                for (int i=0; i<NPTS; i++)
                  if (i != ilo)
                  {
                    for (int j=0; j<NDIMS; j++)
                      p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                    p[i][FUNC] = fnelder.func(p[i]);
                  }
              }
            }
          }
        return z;
```

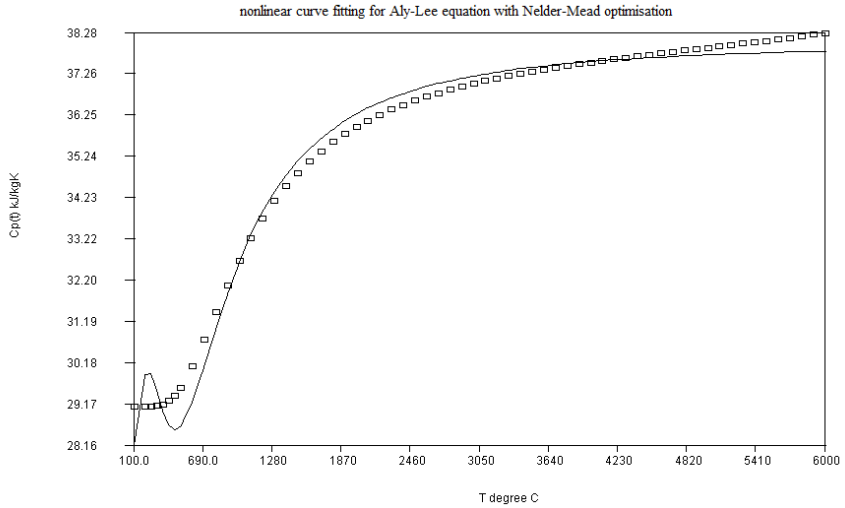
```
}
     public static double[] nelder(if_xj fnelder,double a[],double da[],double tolerance)
     {return nelder(fnelder,a,da,300,tolerance,0);}

     public static double[] nelder(if_xj fnelder,double a[],double da[])
     {return nelder(fnelder,a,da,500,1.0e-20,0);}

    public static void main(String args[]) throws IOException
    {
    String in_name=JOptionPane.showInputDialog(" veri dosyasının ismini giriniz : ");
    //String in_name="D1.txt";
    double a[];
    a=new double[5];
    a[0] = 1.0;
    a[1] = 1.0;
    a[2] = 1.0;
    a[3] = 1.0;
    a[4] = 1.0;
    double da[];
    da=new double[5];
    da[0] =0.3;
    da[1] =0.3;
    da[2] =0.3;
    da[3] =0.3;
    da[4] =0.3;
    yy f=new yy(in_name,a);
    int i=0;
    double p[]=nelder(f,a,da,1000,1e-15,0);
    String s1=" optimisation value  : \n"+Matrix.toStringT(p)+"\n";
    s1+="original function value = "+f.func(a)+"\n";
    s1+="function value = "+f.func(p)+"\n";
    s1+="standard division= "+Math.sqrt(f.func(p)/(f.nn-1));
    System.out.println(s1);
    Plot pp=new Plot(f.xi,f.yi);
    pp.setPlotType(0,20);
    double y[]=new double[f.xi.length];
    for(i=0;i<f.xi.length;i++)
    {y[i]=f.Cp(f.xi[i],f.a);}
    pp.addData(f.xi,y);
    pp.plot();
    }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" nelder
end of file
 optimisation value  :
       27.950645606873834
       10.078840538226359
     -400.883987151156600
     -193.009818119868000
      645.817054212756800

original function value = 72086.49376191813
function value = 8.585551215087992
standard division= 0.3691593026264981
> Terminated with exit code 0.
```



PPDS equation is also widely used to define Cp values for ideal gas specific heat data for gases[40].

$$C_{p(T)} = a_1 + (a_2 - a_1)\left(\frac{T}{a_0 + T}\right)^2 \left[a_3 + a_4\left(\frac{T}{a_0 + T}\right) + a_5\left(\frac{T}{a_0 + T}\right)^2 + a_6\left(\frac{T}{a_0 + T}\right)^3\right] \quad (2.1.32)$$

This equation is also non-linear with respect to coefficients so the same Nelder-Mead nonlinear least square program is used.

Program 2.1.5 Non linear Nelder-Mead optimisation curve fitting program for PPDS equation

```
import java.io.*;
import javax.swing.*;
    class yy implements if_xj
    {
            double xi[]; // independent variable data set
    double yi[]; //dependent variable data set
    double a[];  //fit function coefficient set
    int nn;
            public yy(String filename,double ia[])
    {
      //read the data to curvefit
      //get the data file and initial fit coefficient when class is defined
              double[] Xi=new double[500];
      double[] Yi=new double[500];
      int n=ia.length;
      a=new double[n];
      seta(ia);
      int i=-1;
      try{
              BufferedReader fin=new BufferedReader(new FileReader(filename));
      try {
      while(fin != null)
      {
      i++;
      Xi[i]=Text.readDouble(fin);
      Yi[i]=Text.readDouble(fin);
      }
      } catch(EOFException e_eof) {System.out.println("end of file"); }
      }catch(IOException e_io) {System.out.println("dosya bulunamadı"); }
    nn=i;
    xi=new double[nn];
    yi=new double[nn];
    for(int j=0;j<nn;j++){xi[j]=Xi[j];yi[j]=Yi[j];}
    a=ia;
    }
```

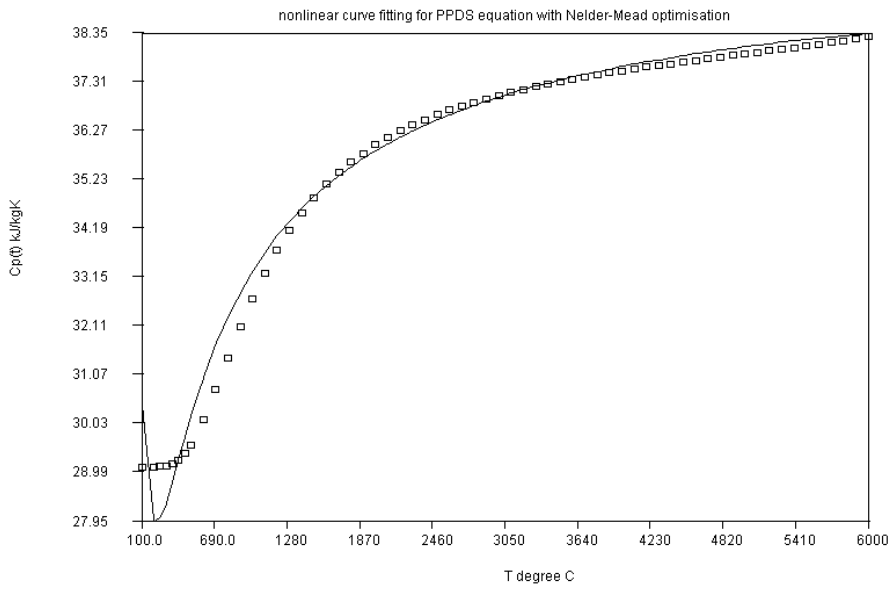
```
    public void seta(double ia[])
    {
                //assign new fit coefficient set
                for(int ii=0;ii<nn;ii++)
          a[ii]=ia[ii];
            }
            public double[] geta()
    {
              // return fit coefficient set
              return a;
            }
            double Cp(double T,double ai[])
    {a=ai;
    double R=8.3145;
    double A1=T/(ai[0]+T);
    double A2=ai[0]/(ai[0]+T);
    double cp=R*(ai[1]+(ai[2]-ai[1])*A1*A1*(1.0-A2*(ai[3]+ai[4]*A1+ai[5]*A1*A1+ai[6]*A1*A1*A1)));
    return cp; //kJ/kgK
    }

    public double func(double ai[])
    {
    double ff=0;
    double w;
    double yy;
     for(int i=0;i<nn;i++)
              {w= Cp(xi[i],ai)-yi[i];
      ff+=w*w;
              }
    return ff;
              }
}
public class nelder1i
{
   public static void main(String args[]) throws IOException
   {
   String in_name=JOptionPane.showInputDialog(" veri dosyasının ismini giriniz : ");
   //String in_name="D1.txt";
   double a[];
   a=new double[7];
   a[0] = 1.0;
   a[1] = 1.0;
   a[2] = 1.0;
   a[3] = 1.0;
   a[4] = 1.0;
   a[5] = 1.0;
   a[6] = 1.0;
   double da[];
   da=new double[7];
   da[0] =0.3;
   da[1] =0.3;
   da[2] =0.3;
   da[3] =0.3;
   da[4] =0.3;
   da[5] =0.3;
   da[6] =0.3;
   yy f=new yy(in_name,a);
   int i=0;
   double p[]=nelderi.nelder(f,a,da,1000,1e-15,0);
```

```
    String s1=" optimisation value  : \n"+Matrix.toStringT(p)+"\n";
    s1+="original function value = "+f.func(a)+"\n";
    s1+="function value = "+f.func(p)+"\n";
    s1+="standard division= "+Math.sqrt(f.func(p)/(f.nn-1));
    System.out.println(s1);
    Plot pp=new Plot(f.xi,f.yi);
    pp.setPlotType(0,20);
    double y[]=new double[f.xi.length];
    for(i=0;i<f.xi.length;i++)
    {y[i]=f.Cp(f.xi[i],f.a);}
    pp.addData(f.xi,y);
    pp.plot();
    }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" nelder1
end of file
 optimisation value  :
     218.014481532370040
       4.734899316748212
       4.813940589168753
     345.654306606462340
    -534.657797593246000
     239.168194745319400
      26.384727444067180

original function value = 47629.457639999986
function value = 10.34871648952423
standard division= 0.40529660879849366
```



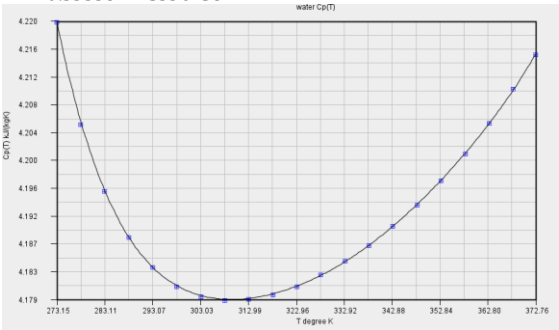
Specific heat values of liquids can be obtained as curve fitting values as well. The values for some of the heat transfer liquids are given in here

Water P=100 kPa $Cp(T) = \sum_{i=0}^{8} c[i]T^i$

c[0] = 270.5776803286873
c[1] = -3.510258807321355
c[2] = 0.01498354254279384
c[3] = -7.589055981721059E-6
c[4] = -1.0150786406519273E-7
c[5] = 1.3719023509450224E-10

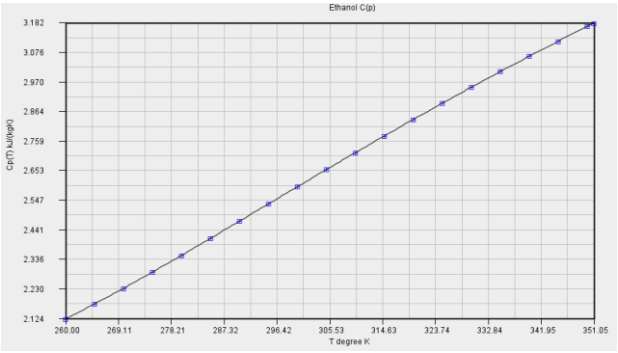
T K	Cp kJ/kgK
273.15	4.22
278.15	4.205
283.15	4.1952
288.15	4.1885
293.15	4.1841

c[6] = 5.659017501899996E-13
c[7] = -1.6574147190101998E-15
c[8] = 1.247400559628774E-18
R^2 = 0.999987245996238



298.15	4.1813
303.15	4.1798
308.15	4.1793
313.15	4.1794
318.15	4.1801
323.15	4.1813
328.15	4.183
333.15	4.185
338.15	4.1873
343.15	4.1901
348.15	4.1932
353.15	4.1968
358.15	4.2007
363.15	4.2052
368.15	4.2102
372.76	4.2152

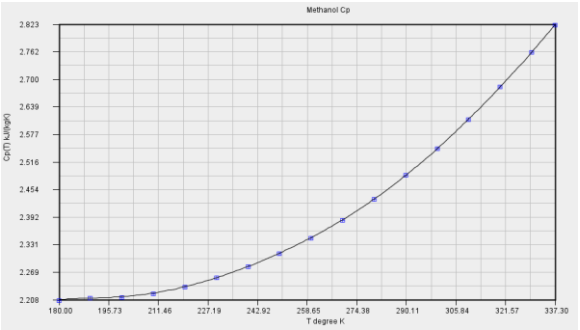
Ethanol P=100kPa $\boldsymbol{Cp(T) = \sum_{i=0}^{8} c[i]T^i}$
c[0] = 99.97500702573771
c[1] = -1.4790642217335723
c[2] = 0.008507238670942352
c[3] = -2.277794099110996E-5
c[4] = 2.9173950044190898E-8
c[5] = -4.30075812225333E-11
c[6] = 1.727344802645507E-13
c[7] = -3.6180817253065646E-16
c[8] = 2.605870896016471E-19
R2 = 0.9999999948019586



Ethanol

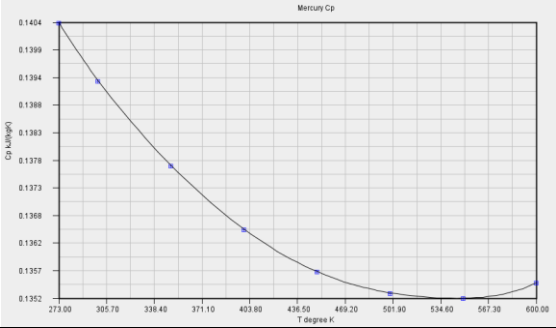
T K	Cp kJ/kgK
260	2.124
265	2.1767
270	2.2325
275	2.2907
280	2.3506
285	2.4115
290	2.4731
295	2.5348
300	2.5965
305	2.6578
310	2.7184
315	2.7784
320	2.8374
325	2.8955
330	2.9526
335	3.0087
340	3.0637
345	3.1177
350	3.1707
351.05	3.1818

Methanol $\boldsymbol{Cp(T) = \sum_{i=0}^{8} c[i]T^i}$
c[0] = -107.09543749764357
c[1] = 2.601611452749273
c[2] = -0.023549716436274053
c[3] = 8.59460826457016E-5
c[4] = 6.538343462420786E-8
c[5] = -1.6997167075625877E-9
c[6] = 6.130872797665774E-12
c[7] = -9.786566704292899E-15
c[8] = 6.092210693963097E-18
R2 = 0.9999997465309409



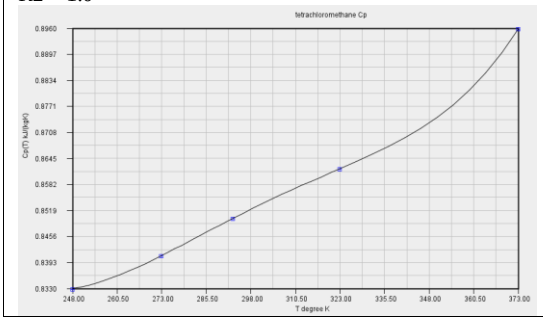
Methanol	
T K	Cp kJ/kgK
180	2.2079
190	2.2124
200	2.2141
210	2.2225
220	2.2376
230	2.2578
240	2.2826
250	2.312
260	2.3466
270	2.387
280	2.4336
290	2.4866
300	2.546
310	2.6119
320	2.6839
330	2.7621
337.3	2.8229

Mercury $\boldsymbol{Cp(T) = \sum_{i=0}^{8} c[i]T^i}$
c[0] = 0.2033272761593221
c[1] = -5.920844922772481E-4
c[2] = 2.0330837741077026E-6
c[3] = -2.772439767853268E-9
c[4] = 2.586268783476067E-12
c[5] = -2.0306289721643128E-14
c[6] = 7.156293968815732E-17
c[7] = -9.800486533233482E-20
c[8] = 4.7800532315135916E-23
R2 = 0.9999999999999997



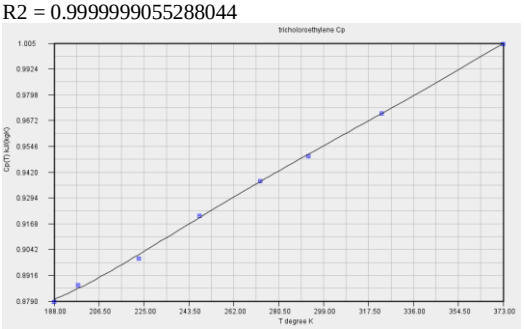
Mercury	
T K	Cp kJ/kgK
273	0.1404
300	0.1393
350	0.1377
400	0.1365
450	0.1357
500	0.1353
550	0.1352
600	0.1355

Tetrachloromethane $\boldsymbol{Cp(T) = \sum_{i=0}^{5} c[i]T^i}$
c[0] = 5.012297747700416
c[1] = -0.05053869667575192
c[2] = 2.0046965626768594E-4
c[3] = -2.0002902590766625E-7
c[4] = -4.446399307337629E-10
c[5] = 8.07585972589885E-13
R2 = 1.0



tetrachloromethane	
T K	Cp kJ/kgK
248	0.833
273	0.841
293	0.85
323	0.862
373	0.896

Tricholoroethylene $\boldsymbol{Cp(T) = \sum_{i=0}^{5} c[i]T^i}$
c[0] = 2.021413497778984
c[1] = -0.0216728733150517
c[2] = 1.5419846154678853E-4
c[3] = -5.218540177452986E-7
c[4] = 8.675697775141513E-10
c[5] = -5.675330725866384E-13
R2 = 0.9999999055288044

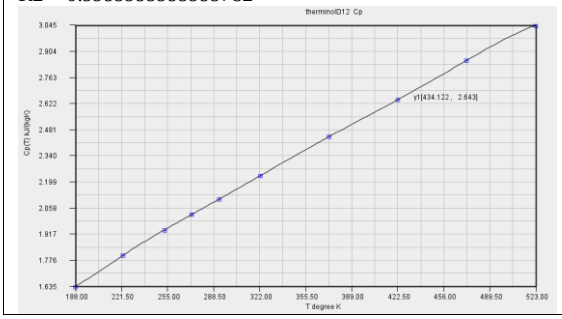


tricholoroethylene	
T K	Cp kJ/kgK
188	0.879
198	0.887
223	0.9
248	0.921
273	0.938
293	0.95
323	0.971
373	1.005

therminolD12 $\boldsymbol{Cp(T) = \sum_{i=0}^{8} c[i]T^i}$
c[0] = 60.526927558517734
c[1] = -1.4926682191903955
c[2] = 0.016013080520282986
c[3] = -9.553446964431934E-5
c[4] = 3.4806695017489665E-7

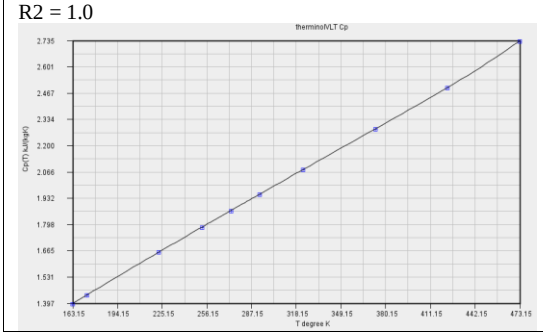
therminolD12	
T K	Cp kJ/kgK
188	1.635
223	1.805
253	1.941

c[5] = -7.937286137062844E-10
c[6] = 1.107162854924502E-12
c[7] = -8.642729926908821E-16
c[8] = 2.8924494422517614E-19
R2 = 0.9999999999999782



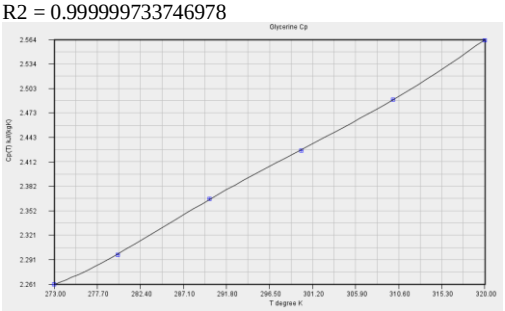
273	2.025
293	2.108
323	2.235
373	2.445
423	2.645
473	2.857
523	3.045

therminolVLT $\boldsymbol{Cp(T) = \sum_{i=0}^{8} c[i]T^i}$
c[0] = -0.02438801684043873
c[1] = 0.029830746122743928
c[2] = -3.8372497177788565E-4
c[3] = 3.1590538390345377E-6
c[4] = -1.5534921909351174E-8
c[5] = 4.683611618989939E-11
c[6] = -8.486188979967341E-14
c[7] = 8.472860840476584E-17
c[8] = -3.5745056585159034E-20
R2 = 1.0



therminolVLT	
T K	Cp kJ/kgK
163.15	1.397
173.15	1.441
223.15	1.659
253.15	1.787
273.15	1.871
293.15	1.955
323.15	2.08
373.15	2.287
423.15	2.497
473.15	2.735

Glycerine $\boldsymbol{Cp(T) = \sum_{i=0}^{5} c[i]T^i}$
c[0] = 5461.782629499105
c[1] = -90.65285425337981
c[2] = 0.6011914499449433
c[3] = -0.0019906504885026286
c[4] = 3.2911488782269877E-6
c[5] = -2.1734451391880267E-9
R2 = 0.999999733746978

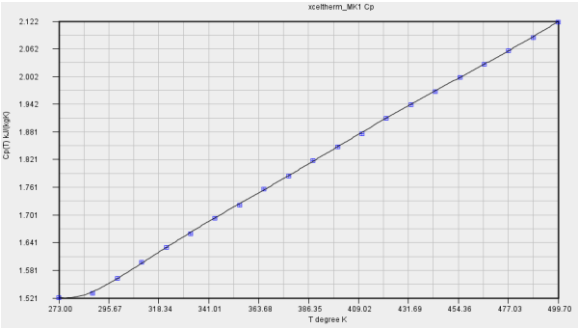


glycerine	
T K	Cp kJ/kgK
273	2.261
280	2.298
290	2.367
300	2.427
310	2.49
320	2.564

xceltherm_MK1 $\boldsymbol{Cp(T) = \sum_{i=0}^{8} c[i]T^i}$
c[0] = 655.3892408685903
c[1] = -11.887391675305821
c[2] = 0.09210892285990327
c[3] = -3.9508786804483384E-4
c[4] = 1.0162289049825188E-6
c[5] = -1.5770767978041933E-9
c[6] = 1.3908434773222344E-12
c[7] = -5.792395008000518E-16

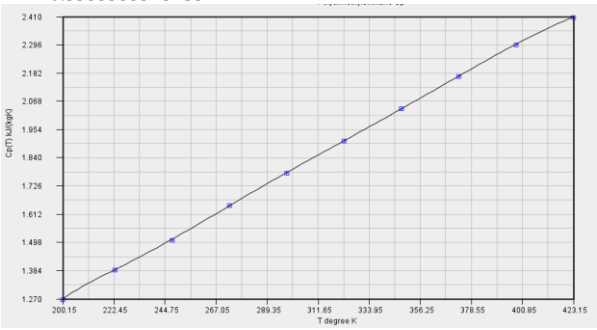
xceltherm_MK1	
T K	Cp kJ/kgK
273	1.523
288.6	1.532
299.7	1.565
310.8	1.599
321.9	1.632

c[8] = 5.424750831845585E-20
R2 = 0.999996629189612



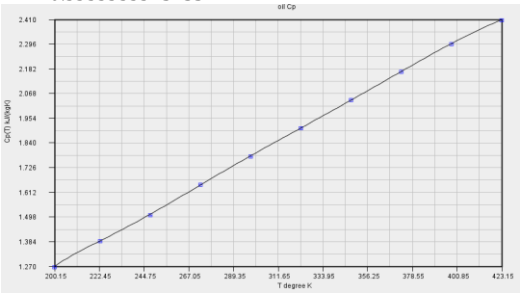
333	1.662
344.1	1.695
355.2	1.724
366.3	1.758
377.4	1.787
388.6	1.821
399.7	1.85
410.8	1.879
421.9	1.913
433	1.942
444.1	1.971
455.2	2.001
466.3	2.03
477.4	2.059
488.6	2.088
499.7	2.122

Polydimethylsiloxane $\boldsymbol{Cp(T) = \sum_{i=0}^{8} c[i]T^i}$
c[0] = -247.42609839724008
c[1] = 6.137524512999184
c[2] = -0.06514258699078336
c[3] = 3.87517827429489E-4
c[4] = -1.4109055471195397E-6
c[5] = 3.2161765885520272E-9
c[6] = -4.476013677272247E-12
c[7] = 3.470065528728324E-15
c[8] = -1.1439083662355205E-18
R2 = 0.999999991375347



Polydimethylsiloxane	
T K	Cp kJ/kgK
200.15	1.27
223.15	1.39
248.15	1.51
273.15	1.65
298.15	1.78
323.15	1.91
348.15	2.04
373.15	2.17
398.15	2.3
423.15	2.41

Oil
c[0] = -247.42609839724008
c[1] = 6.137524512999184
c[2] = -0.06514258699078336
c[3] = 3.87517827429489E-4
c[4] = -1.4109055471195397E-6
c[5] = 3.2161765885520272E-9
c[6] = -4.476013677272247E-12
c[7] = 3.470065528728324E-15
c[8] = -1.1439083662355205E-18
R2 = 0.999999991375347



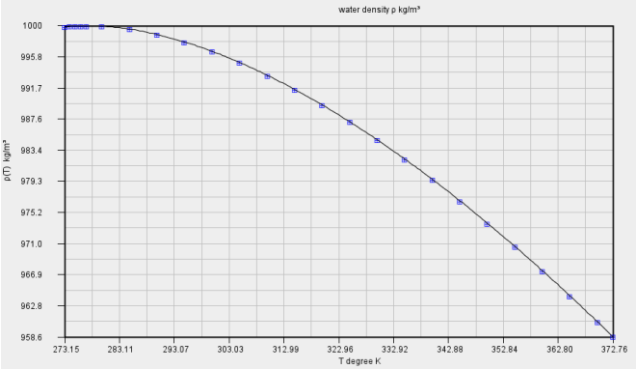
oil	
T K	Cp kJ/kgK
200.15	1.27
223.15	1.39
248.15	1.51
273.15	1.65
298.15	1.78
323.15	1.91
348.15	2.04
373.15	2.17
398.15	2.3
423.15	2.41

11.5.5 DENSITY

For density of low pressure gases ideal gas law might be sufficent, In higher pressures utilisation of a real equation of state is recommended. Some of the liquid properties are given as data and curvefitting values below:

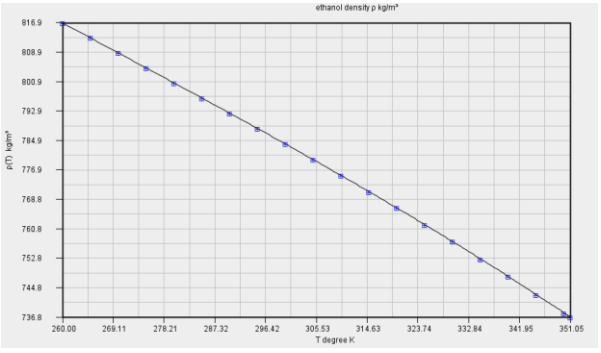
water $\boldsymbol{\rho(T) = \sum_{i=0}^{8} c[i]T^i}$

c[0] = -14094.004223625658
c[1] = 187.0634705255266
c[2] = -0.39800684599059927
c[3] = -0.0055928045422936066
c[4] = 4.623590488099427E-5
c[5] = -1.6140046519422862E-7
c[6] = 3.033066555401628E-10
c[7] = -3.0054514672565815E-13
c[8] = 1.2359140372220347E-16
R2 = 0.9999999958646062



water	
T K	ρ kg/m^3
273.15	999.8
274.15	999.9
275.15	999.94
276.15	999.97
277.15	999.97
280	999.91
285	999.52
290	998.8
295	997.81
300	996.56
305	995.08
310	993.38
315	991.5
320	989.43
325	987.19
330	984.79
335	982.23
340	979.54
345	976.7
350	973.73
355	970.63
360	967.4
365	964.06
370	960.59
372.76	958.63

ethanol $\boldsymbol{\rho(T) = \sum_{i=0}^{8} c[i]T^i}$
c[0] = 2382.5398728005543
c[1] = -21.84826796586477
c[2] = 0.12736747261133124
c[3] = -3.677438006916865E-4
c[4] = 5.19515543627176E-7
c[5] = -7.604622858666759E-10
c[6] = 3.0377998300334107E-12
c[7] = -6.8764195718127415E-15
c[8] = 5.350076455570246E-18
R2 = 0.9999999994513425

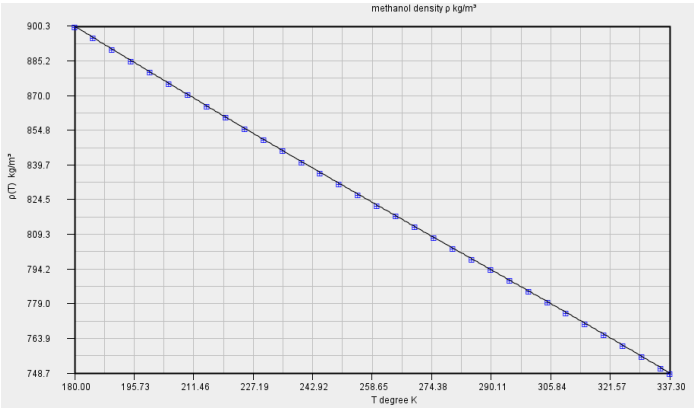


Ethanol	
T K	ρ kg/m^4
260.00000000	816.94
265.00000000	812.84
270.00000000	808.74
275.00000000	804.65
280.00000000	800.56
285.00000000	796.44
290.00000000	792.3
295.00000000	788.12
300.00000000	783.9
305.00000000	779.64
310.00000000	775.31
315.00000000	770.92
320.00000000	766.46
325.00000000	761.93
330.00000000	757.3
335.00000000	752.59
340.00000000	747.78
345.00000000	742.87
350.00000000	737.85
351.05000000	736.78

methanol $\boldsymbol{\rho(T) = \sum_{i=0}^{8} c[i]T^i}$
c[0] = -465.58693946551927
c[1] = 39.87079474066188
c[2] = -0.45187767639373677
c[3] = 0.00267900018416731

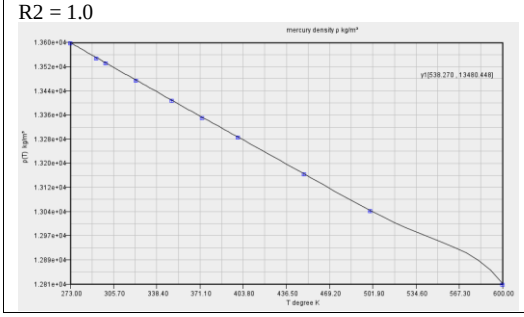
Methanol	
T K	ρ kg/m^3
180	900.32
185	895.35

c[4] = -8.94123682243866E-6
c[5] = 1.5266095017496084E-8
c[6] = -5.867069011453614E-12
c[7] = -1.8912737879574196E-14
c[8] = 2.081451057688526E-17
R2 = 0.9999999984730404



190	890.35
195	885.34
200	880.34
205	875.36
210	870.41
215	865.48
220	860.57
225	855.68
230	850.82
235	845.98
240	841.16
245	836.37
250	831.59
255	826.84
260	822.11
265	817.39
270	812.69
275	807.99
280	803.31
285	798.64
290	793.96
295	789.28
300	784.59
305	779.88
310	775.16
315	770.4
320	765.62
325	760.79
330	755.91
335	750.98
337.3	748.69

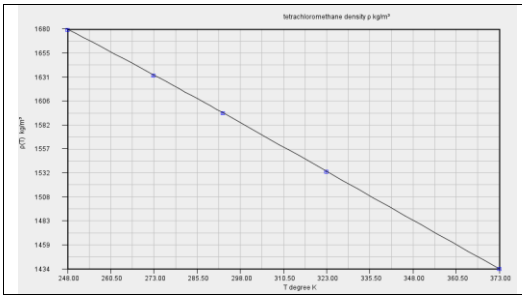
mercury $\boldsymbol{\rho(T) = \sum_{i=0}^{8} c[i]T^i}$
c[0] = 26031.804378214994
c[1] = -141.0945358709674
c[2] = 0.25687356179268567
c[3] = 0.0039388639437045025
c[4] = -2.9573184624687727E-5
c[5] = 9.450367638069545E-8
c[6] = -1.6189039758620223E-10
c[7] = 1.454878787678996E-13
c[8] = -5.401993183180348E-17
R2 = 1.0



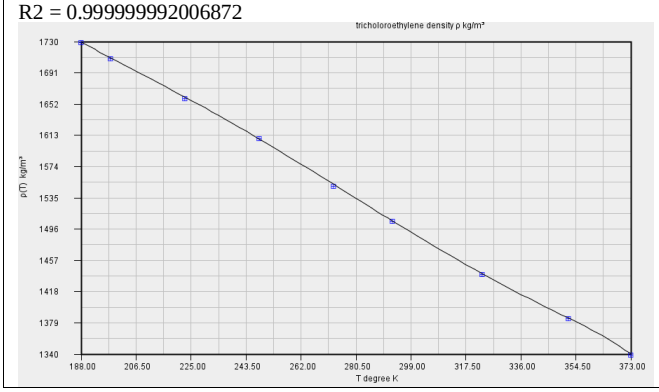
Mercury	
T K	ρ kg/m^4
273	13595
293	13545
300	13529
323	13472
350	13407
400	13287
450	13167
500	13048
600	12809

tetrachloromethane $\boldsymbol{\rho(T) = \sum_{i=0}^{8} c[i]T^i}$
c[0] = 2449.4084261663975
c[1] = -5.766194142666515
c[2] = 0.011354652942705339
c[3] = 2.6900743052415214E-5
c[4] = -1.129721483600801E-7
c[5] = -1.8931581051282716E-10
c[6] = 5.016207311086058E-13
c[7] = 1.3621144381614608E-15
c[8] = -2.6846930034050322E-18

tetrachloromethane	
T K	ρ kg/m^4
248	1680
273	1633
293	1594
323	1534
373	1434

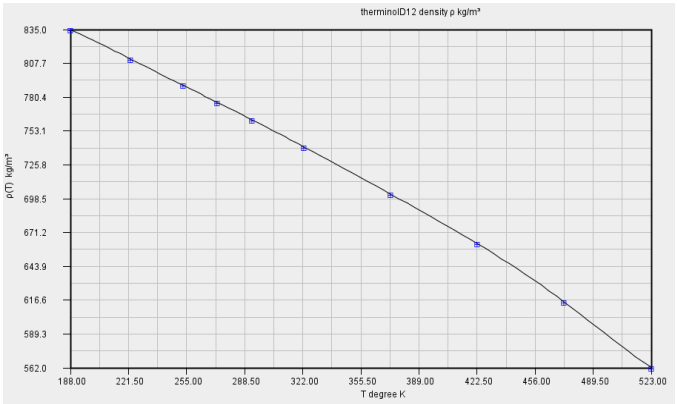


tricholoroethylene $\boldsymbol{\rho(T) = \sum_{i=0}^{8} c[i]T^i}$
c[0] = -58182.21780542969
c[1] = 1978.2696166811854
c[2] = -27.947254389218195
c[3] = 0.2217342755341111
c[4] = -0.0010836359339685944
c[5] = 3.346121576288167E-6
c[6] = -6.385871126693301E-9
c[7] = 6.896441261968005E-12
c[8] = -3.2305633061161147E-15
R2 = 0.999999992006872



tricholoroethylene	
T K	ρ kg/m^4
188	1730
198	1710
223	1660
248	1610
273	1550
293	1507
323	1440
373	1340

therminolD12 $\boldsymbol{\rho(T) = \sum_{i=0}^{8} c[i]T^i}$
[0] = -3448.1930906324665
c[1] = 106.83938833281174
c[2] = -1.1171731625331536
c[3] = 0.0064574943605130546
c[4] = -2.268707999927618E-5
c[5] = 4.954377378383254E-8
c[6] = -6.556516787544722E-11
c[7] = 4.795167000505446E-14
c[8] = -1.4791998030145217E-17
R2 = 0.9999999999999961

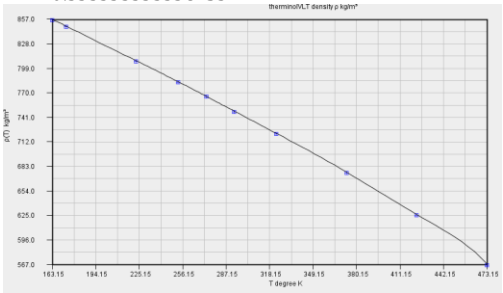


therminolD12	
T K	ρ kg/m^4
188	835
223	811
253	790
273	776
293	762
323	740
373	702
423	662
473	615
523	562

therminolVLT $\boldsymbol{\rho(T) = \sum_{i=0}^{8} c[i]T^i}$
c[0] = -6789.504166034223

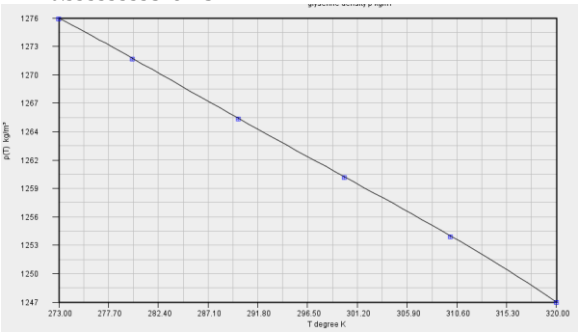
therminolVLT	

c[1] = 233.9757618886139
c[2] = -3.0380035438302166
c[3] = 0.02200953726212739
c[4] = -9.766244093027687E-5
c[5] = 2.71898136910363E-7
c[6] = -4.640909049431486E-10
c[7] = 4.443172763189479E-13
c[8] = -1.8282010943776378E-16
R2 = 0.9999999999982397



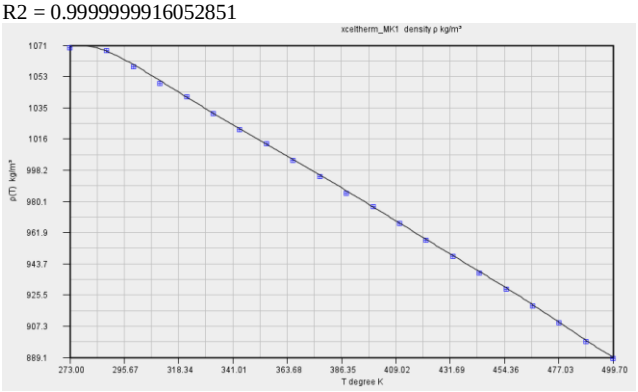
T K	ρ kg/m^4
163.15	857
173.15	849
223.15	808
253.15	783
273.15	766
293.15	748
323.15	722
373.15	676
423.15	626
473.15	567

glycerine $\boldsymbol{\rho(T) = \sum_{i=0}^{8} c[i]T^i}$
c[0] = -73551.93848032346
c[1] = 1215.2099695702009
c[2] = -7.853115698146029
c[3] = 0.025250910050717265
c[4] = -4.039810475614355E-5
c[5] = 2.5711657127401364E-8
R2 = 0.9999999954822341



glycerine	
T K	ρ kg/m^3
273	1276
280	1271.9
290	1265.8
300	1259.9
310	1253.9
320	1247.2

xceltherm_MK1 $\boldsymbol{\rho(T) = \sum_{i=0}^{8} c[i]T^i}$
c[0] = -160055.06641301597
c[1] = 2786.2892013610285
c[2] = -20.02883169011768
c[3] = 0.0761213562906084
c[4] = -1.5698402018355333E-4
c[5] = 1.4312195253196847E-7
c[6] = 4.247407894003096E-11
c[7] = -1.9144441231721183E-13
c[8] = 1.1027966866757059E-16
R2 = 0.9999999916052851

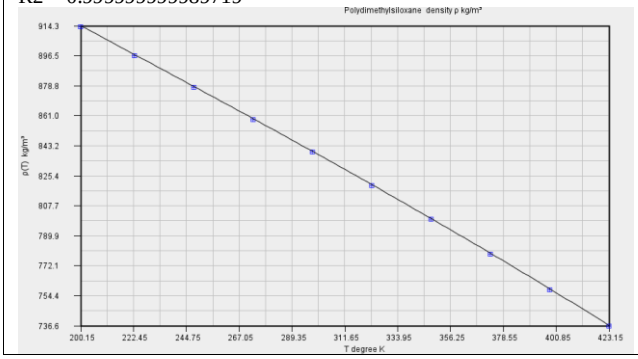


xceltherm_MK1	
T K	ρ kg/m^3
273	1070.1
288.6	1068.5
299.7	1058.9
310.8	1049.3
321.9	1041.3
333	1031.7
344.1	1022.1
355.2	1014.1
366.3	1004.5
377.4	994.8
388.6	985.2
399.7	977.2
410.8	967.6
421.9	958
433	948.4
444.1	938.8
455.2	929.2
466.3	919.5
477.4	909.9
488.6	898.7
499.7	889.1

Polydimethylsiloxane $\boldsymbol{\rho(T) = \sum_{i=0}^{8} c[i]T^i}$

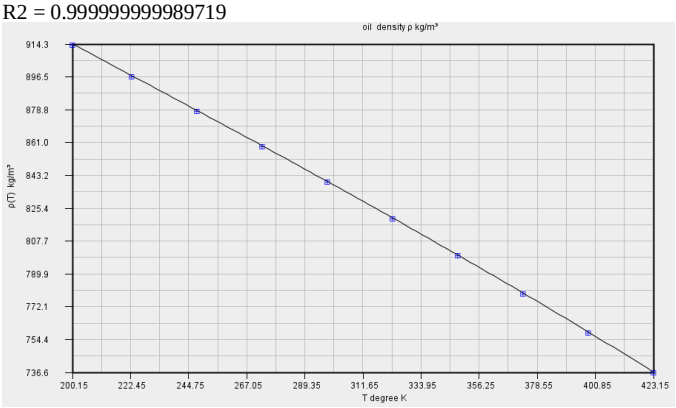
Polydimethylsiloxane	

c[0] = 1897.5801512251794
c[1] = -21.68460230756459
c[2] = 0.2276692944216558
c[3] = -0.0014044555926167376
c[4] = 5.392027484612206E-6
c[5] = -1.3221006573398557E-8
c[6] = 2.0239455061673124E-11
c[7] = -1.769911295132396E-14
c[8] = 6.768446165570133 5E-18
R2 = 0.999999999989719



T K	ρ kg/m^3
200.15	914.3
223.15	897.1
248.15	878.3
273.15	859.2
298.15	839.8
323.15	820.1
348.15	800
373.15	779.5
398.15	758.4
423.15	736.6

oil $\boldsymbol{\rho(T) = \sum_{i=0}^{8} c[i]T^i}$
c[0] = 1897.5801512251794
c[1] = -21.68460230756459
c[2] = 0.2276692944216558
c[3] = -0.0014044555926167376
c[4] = 5.392027484612206E-6
c[5] = -1.3221006573398557E-8
c[6] = 2.0239455061673124E-11
c[7] = -1.769911295132396E-14
c[8] = 6.768446165570133 5E-18
R2 = 0.999999999989719



oil	
T K	ρ kg/m^4
200.15	914.3
223.15	897.1
248.15	878.3
273.15	859.2
298.15	839.8
323.15	820.1
348.15	800
373.15	779.5
398.15	758.4
423.15	736.6

11.5.6 THERMAL EXPANSION COEFFICIENT

Thermal expansion coefficient is a thermodynamic property defined as

Thermal expansion coefficient= $\beta = \frac{1}{v}\left(\frac{\partial v}{\partial T}\right)_P = \frac{1}{\rho}\left(\frac{\partial P}{\partial T}\right)_\rho \left(\frac{\partial \rho}{\partial P}\right)_T$

The unit is 1/K. Naturally the value should be obtained form equation of states. For perfect gas: $v = RT/P$

$$\left(\frac{\partial v}{\partial T}\right)_P = \frac{R}{P}$$

$$\beta = \frac{1}{v}\left(\frac{\partial v}{\partial T}\right)_P = \left(\frac{P}{RT}\right)\frac{R}{P} = \frac{1}{T}$$

For the remaining values equation of state should be utilised to find the value. In order to calculate β in liquid region, your EOS should cover data of liquid region.

11.5.7 PRANDTL NUMBER

Prandtl number is defined as: $Pr = \frac{C_p \mu}{k}$ It is not a new property but depends on the other properties we already defined, therefore we are not defined how to calculate it, but due to fact at it shows almost in all heat convection formulations, it is usually listed as a seperate entity to avoid extra calculations.



Ludwig Prandtl

11.5.8 THERMAL DIFFUSIVITY

Thermal diffusivity is defined as: $\alpha = \frac{k}{\rho C_p}$. Like Prandtl number, it is not a new property but depends o the other properties we already defined. And like Prandtl number it is usually list as a seperate identity due to make easier heat trasnfer calculations.

11.5.9 HEAT OF VAPORISATION

Heat of vaporisation is also a thermodynamic property of phase change. It is difference of saturation entalpy of gas and liquid phases. If gas/liquid has a single molecule
$h_{fg}(T_s) = h_g(T_s) - h_f(T_s)$ where T_s is saturation pressure, h_g enthalpy of saturated vapor at saturation temperature(or pressure, itt is the same thing for pure substances), and h_f enthalpy of saturated liqıid at saturation temperature. If it is a mixture, saturation temperature of saturated liquid and saturated vapor will not be same (due to the difference of saturation temperature of each entity in mixture and molecular interactions). Saturation temperature of the vapor is called dew point temperature and saturation temperature of the liquid is called the buble point. Therefore enthalpy of veporation for a mixture can be given as: $h_{fg}(P_s) = h_g(T_{dew}) - h_f(T_{buble})$
In some equations of state saturation temperature relations are given seperately. In some it is merged as a part of EOS. Enthalpy of evaporation then can be calculated from Classius clapeyron eqaution.
When thermodynamic properties are considered, phase change region is always considered as additional important region. The same equations are developed so far can be adopted to phase change region. Consider the equation
$dh = Tds + vdP$. In the phase change of pure substances pressure remains constant for a given temperature. So the equation becomes: $dh = Tds$. This equation can be written as
$(h_g - h_f) = T(s_g - s_f)$ In this equation h_g entalphy of saturated vapour and h_f is the enthalpy of saturated liquid. Similarly, s_g entropy of saturated vapour and s_f is the entropy of saturated liquid. If equation
$\left(\frac{\partial S}{\partial V}\right)_T = \left(\frac{\partial P}{\partial T}\right)_v$
is applied to this region
$\left(\frac{s_g - s_f}{v_g - v_f}\right)_T = \left(\frac{\partial P}{\partial T}\right)_v$

$\left(\frac{h_g - h_f}{T(v_g - v_f)}\right)_T = \left(\frac{\partial P}{\partial T}\right)_v$ This equation is called **Classius –Clapeyron** equation.

11.5.10 THERMOPHYSICAL PROPERTIES OF HEAT TRANSFER FLUIDS AND SECONDARY COOLANTS

Air is one of the most use heat transnfer fluid, therefore all the properties discussed above sections are given as one complete table for air. In order to calculate this table NIST equation of state[] is used. The details of the equation of state is :

Formulation of Equation of State for dry air

Helmholtz energy equation for air is given as[134]

$$\alpha(\delta,\tau) = \frac{a(\rho,T)}{RT} = \alpha^0(\delta,\tau) + \alpha^r(\delta,\tau)$$

Where α^0 is the ideal-gas contribution to the Helmholtz energy and $\alpha^r$is the residial contribution to the helmholtz energy. $\delta = \rho/\rho_j$ is the reduced density, $\tau = \frac{T_j}{T}$ is reciprocal reduced temperature. T_j is maxcondentherm which is the maximum temperature existed before mixture of gases become supercritical, or state point of maximum temperature along the saturation line. The same definition apply also for ρ_j. The values are: Tj=132.6312±0.004 K (uncertainty K=2) and ρ_j=10.4477±0.05 kmol/m^3 (uncertainty K=2). Ideal gas part of the reduced Helmholtz energy equation :

$$\alpha^0(\delta,\tau) = ln\delta + \sum_{i=1}^{5} N_i\tau^{i-4} + N_6\tau^{1.5} + N_7\ln(\tau) + N_8\ln[1-\exp(-N_{11}\tau)] + N_9\ln[1-\exp(-N_{12}\tau)] + N_{10}\ln[2/3-\exp(-N_{13}\tau)]$$

The coefficients of the ideal gas part of EOS is given in table 1.

Table 2.1 coefficients of the ideal gas p art of Helmholtz EOS

Table 2.1 Coefficients of ideal gas part of the Helmholtz EOS

i	Ni
1	6.0571940000E-08
2	-2.1027476900E-05
3	-1.5886071600E-04
4	-1.3841928076E+01
5	1.7275266575E+01
6	-1.9536342000E-04
7	2.4908880320E+00
8	7.9130950900E-01
9	2.1223676800E-01
10	-1.9793890400E-01
11	2.5363650000E+01
12	1.6907410000E+01
13	8.7312790000E+01

The residual part of the helmholtz energy contribution to the equation of state given as:

$$\alpha^r(\delta,\tau) = \sum_{k=1}^{10} N_k\delta^{i_k}\tau^{j_k} + \sum_{k=11}^{19} N_k\delta^{i_k}\tau^{j_k}\exp(-\delta^{l_k})$$

Table 2.2 Coefficients of residual part of the Helmholtz EOS

k	Nk	ik	jk	lk
1	1.18160747229E-01	1	0	0
2	7.13116392079E-01	1	0.33	0
3	-1.61824192067E+00	1	1.01	0
4	7.14140178971E-02	2	0	0
5	-8.65421396646E-02	3	0	0
6	1.34211176704E-01	3	0.15	0
7	1.12626704218E-02	4	0	0
8	-4.20533228842E-02	4	0.2	0
9	3.49008431982E-02	4	0.35	0
10	1.64957183186E-04	6	1.35	0
11	-1.01365037912E-01	1	1.6	1
12	-1.73813690970E-01	3	0.8	1
13	-4.72103183731E-02	5	0.95	1
14	-1.22523554253E-02	6	1.25	1
15	-2.01466296097E+01	1	3.6	2
16	-3.16055879821E-02	3	6	2
17	2.33594806142E-04	11	3.25	2
18	1.48287891978E-02	1	3.5	3
19	-9.38782884667E-03	3	15	3

Thermodynamic properties can be calculated by using derivaties of the EOS

Compressibility factor- Pressure	$Z = \frac{P}{\rho RT} = 1 + \delta\left(\frac{\partial \alpha^r}{\partial \delta}\right)_\tau$
Internal energy kJ/kmol	$\frac{u}{RT} = \tau\left[\left(\frac{\partial \alpha^0}{\partial \tau}\right)_\delta + \left(\frac{\partial \alpha^r}{\partial \tau}\right)_\delta\right]$
Enthalpy kJ/kmol	$\frac{h}{RT} = \tau\left[\left(\frac{\partial \alpha^0}{\partial \tau}\right)_\delta + \left(\frac{\partial \alpha^r}{\partial \tau}\right)_\delta\right] + \delta\left(\frac{\partial \alpha^r}{\partial \delta}\right)_\tau + 1$
Entropy kJ/kmolK	$\frac{s}{R} = \tau\left[\left(\frac{\partial \alpha^0}{\partial \tau}\right)_\delta + \left(\frac{\partial \alpha^r}{\partial \tau}\right)_\delta\right] - \alpha^0 - \alpha^r$
Specific heat at constat volume, isovolumetric specific heat kJ/kmolK	$\frac{C_v}{R} = -\tau^2\left[\left(\frac{\partial^2 \alpha^0}{\partial \tau^2}\right)_\delta + \left(\frac{\partial^2 \alpha^r}{\partial \tau^2}\right)_\delta\right]$
Specific heat at constant pressure, isochoric specific heat kJ/kmolK	$\frac{C_p}{R} = \frac{C_v}{R} + \frac{\left[1 + \delta\left(\frac{\partial \alpha^r}{\partial \delta}\right)_\tau - \delta\tau\left(\frac{\partial^2 \alpha^r}{\partial \delta \partial \tau}\right)_\delta\right]^2}{\left[1 + 2\delta\left(\frac{\partial \alpha^r}{\partial \delta}\right)_\tau + \delta^2\left(\frac{\partial^2 \alpha^r}{\partial \delta^2}\right)_\delta\right]}$
Speed of sound m/s	$\frac{w^2 M}{RT} = \frac{C_p}{C_v}\left[1 + 2\delta\left(\frac{\partial \alpha^r}{\partial \delta}\right)_\tau + \delta^2\left(\frac{\partial^2 \alpha^r}{\partial \delta^2}\right)_\delta\right]$

Universal gas constant is taken as R=8.31451±0.00007 (uncertainity K=2). Molar mass of air is taken as: M=28.9586 kg/kmol
Saturation region equations of the mixture are also existed. Model assumed that air is a mixture of nitrogen, oxygen an air with the fraction of the following mole fraction composition

Component	Mole Fraction
N_2	0.7812
O_2	0.2096
Ar	0.0092

	$\theta = \left[1 - \frac{T}{T_j}\right]$
Bubble and dew point pressures kPA	$ln\left(\frac{P}{P_j}\right) = \left(\frac{T_j}{T}\right)\sum_{i=1}^{8} N_i \theta^{i/2}$
Bubble point density kmol/m^3	$\left(\frac{\rho}{\rho_j}\right) = 1 + N_1\theta^{0.65} + N_2\theta^{0.85} + N_3\theta^{0.95} + N_4\theta^{1.1} + N_5 ln\left(\frac{T}{T_j}\right)$ (2.13)
Dew point density kmol/m^3	$\ln\left(\frac{\rho}{\rho_j}\right) = N_1\theta^{0.41} + N_2\theta + N_3\theta^{2.8} + N_4\theta^{6.5}$
Freezing liquid pressure kPa	$\frac{P}{P_s} = 1 - 35493.5\left[\left(\frac{T}{T_s}\right)^{1.78963} - 1\right]$

Where P_j=3785.02± kPa (uncertainty K=2) and T_s=59.75 K P_s=5.265 kPa
Table 2.3 Coefficients of saturation properties

	P buble point	P dew point	□□buble point	□ dew point
	Ni	Ni	Ni	Ni
1	0.22607240	-0.15672660	44.34130000	-2.04660000
2	-7.08049900	-5.53963500	-240.07300000	-4.75200000
3	5.70028300	0.00000000	285.13900000	-13.25900000
4	-12.44017000	0.00000000	-88.33660000	-47.65200000
5	17.81926000	0.75672120	-0.89218100	0.00000000
6	-10.81364000	0.00000000	0.00000000	0.00000000
7	0.00000000	0.00000000	0.00000000	0.00000000
8	0.00000000	-3.51432200	0.00000000	0.00000000

2 Formulations of other thermophysical and thermodynamic properties

In order to calculate thermopysical properties (thermal conductivity and viscosity) of dry air Kadoya et al[135] equations are used. This equations has the following form:

$\square_0(T_r) = A_0 T_r + A_1 T_r^{0.5} + A_2 + \frac{A_3}{T_r} + \frac{A_4}{T_r^2} + \frac{A_5}{T_r^3} + \frac{A_6}{T_r^4}$

$\Delta\square(\rho_r) = \sum_{i=1}^{4} B_i \rho_r^i$ 3.2

$\square\square T_r, \rho_r) = H[\square_0\square T_r) + \square\square\square\rho_r)]$ 3.3

$k_0(T_r) = C_0 T_r + C_1 T_r^{0.5} + C_2 + \frac{C_3}{T_r} + \frac{C_4}{T_r^2} + \frac{C_5}{T_r^3} + \frac{C}{T_r^4}$

$\Delta k(\rho_r) = \sum_{i=1}^{4} D_i \rho_r^i$ 3.5

$k(T_r, \rho_r) = \square[k_0(T_r) + \Delta k(\rho_r)]$ 3.6

Where $\rho_r = \rho/\rho^*$ $T_r = T/T^*$ 3.7

Coefficients of the equations are given in Table

Table Coefficients of equations

$T^* = 132.5$ K	$\rho^* = 314.3$ kg/m3	□ = 25.9778 ($10^{-3}W$	H=6.1609 (10^{-6} Pas)	
i	A_i	B_i	C_i	D_i
0	0.128517	0.465601	0.239503	0.402287
1	2.60661	1.26469	0.00649768	0.356603
2	-1	-0.511425	1	-0.163159
3	-0.709661	0.2746	-1.92615	0.138059
4	0.662534		2.00383	-0.0201725
5	-0.197846		-1.07553	
6	0.00770147		0.229414	

Prandtl number: $Pr = \frac{c_p \mu}{k}$

Thermal expansion coefficient= $\beta = \frac{1}{v}\left(\frac{\partial v}{\partial T}\right)_P = \frac{1}{\rho}\left(\frac{\partial P}{\partial T}\right)_\rho \left(\frac{\partial \rho}{\partial P}\right)_T$

$$\left(\frac{\partial P}{\partial T}\right)_\rho = R\rho\left[1 + \delta\left(\frac{\partial \alpha^r}{\partial \delta}\right)_\tau - \delta\tau\left(\frac{\partial^2 \alpha^r}{\partial \delta \partial \tau}\right)_\delta\right]$$

$$\left(\frac{\partial P}{\partial \rho}\right)_T = \frac{RT}{\rho}\left[1 + 2\delta\left(\frac{\partial \alpha^r}{\partial \delta}\right)_\tau + \delta^2\left(\frac{\partial^2 \alpha^r}{\partial \delta^2}\right)_\delta\right]$$

$$\left(\frac{\partial \rho}{\partial P}\right)_T = \frac{1}{\left(\frac{\partial P}{\partial \rho}\right)_T}$$

Isothermal compressibility: □ $= \frac{1}{\rho\left(\frac{\partial P}{\partial \rho}\right)_T}$

```
public class air_NIST_test
{public static void main(String arg[])
 { air_NIST air=new air_NIST();
   double P=100.0; //kPa
   for(double T=110;T<3001;T+=10)
   {
   double ro=air.rom(T,P);
   double Cp=air.Cp_Trom(T,ro);
   double vis=air.vis(T,P);
   double k=air.k(T,P);
   double Pr=air.Pr_Tro(T,ro);
    double beta=air.beta(T,ro);
   System.out.println(T+" "+P+" "+ro+" "+Cp+" "+vis+" "+k+" "+Pr+" "+beta);
   } }}
```

For program code air-NIST please check www.turhancoban.com (SCO1.jar)

Properties of dry air at P=100 kPa

T degree K	ρ kg/m^3	Cp kJ/(kgK)	η Pas	k	Pr	β 1/K
110	3.216524	1.02904805	7.7843E-06	0.010376	0.923677	0.083515
120	2.937777	1.02219022	8.4450E-06	0.011333	0.883391	0.02955
130	2.704749	1.01763992	9.0927E-06	0.012266	0.854432	0.018919
140	2.506732	1.01448673	9.7294E-06	0.013176	0.832938	0.014242
150	2.336213	1.01222362	1.0356E-05	0.014068	0.816393	0.011558
160	2.187733	1.01055235	1.0971E-05	0.014947	0.803204	0.0098
170	2.057216	1.00929	1.1577E-05	0.015814	0.792364	0.008553
180	1.941548	1.00832067	1.2171E-05	0.016671	0.783226	0.00762
190	1.838305	1.00756906	1.2756E-05	0.017518	0.775362	0.006892
200	1.745567	1.00698547	1.3330E-05	0.018356	0.768488	0.006307
210	1.661798	1.00653675	1.3894E-05	0.019185	0.762403	0.005825
220	1.585747	1.00620079	1.4449E-05	0.020005	0.756966	0.00542
230	1.516388	1.00596297	1.4994E-05	0.020816	0.752075	0.005075
240	1.452869	1.00581383	1.5529E-05	0.021617	0.747649	0.004776
250	1.394481	1.00574752	1.6056E-05	0.022409	0.743631	0.004515
260	1.340622	1.00576068	1.6575E-05	0.023191	0.739971	0.004284
270	1.290782	1.0058517	1.7085E-05	0.023965	0.736633	0.004078
280	1.244528	1.00602017	1.7587E-05	0.024729	0.733584	0.003893
290	1.201483	1.00626646	1.8081E-05	0.025485	0.7308	0.003725
300	1.161325	1.0065914	1.8569E-05	0.026232	0.728256	0.003573
310	1.12377	1.00699611	1.9049E-05	0.02697	0.725936	0.003434
320	1.088574	1.00748177	1.9522E-05	0.027701	0.723822	0.003307
330	1.055521	1.00804954	1.9989E-05	0.028423	0.721901	0.003189
340	1.024419	1.0087004	2.0450E-05	0.029137	0.720159	0.00308

350	0.995102	1.00943515	2.0905E-05	0.029844	0.718585	0.002979
360	0.967418	1.01025428	2.1354E-05	0.030544	0.717169	0.002885
370	0.941236	1.011158	2.1797E-05	0.031237	0.715902	0.002797
380	0.916436	1.01214617	2.2235E-05	0.031923	0.714775	0.002715
390	0.892911	1.01321833	2.2668E-05	0.032603	0.71378	0.002637
400	0.870565	1.01437366	2.3096E-05	0.033276	0.71291	0.002565
410	0.849312	1.01561101	2.3519E-05	0.033943	0.712158	0.002496
420	0.829073	1.01692891	2.3937E-05	0.034605	0.711516	0.002431
430	0.809778	1.01832559	2.4351E-05	0.035261	0.71098	0.00237
440	0.791361	1.01979899	2.4761E-05	0.035911	0.710543	0.002312
450	0.773764	1.02134679	2.5166E-05	0.036557	0.710198	0.002257
460	0.756933	1.02296646	2.5567E-05	0.037197	0.709941	0.002204
470	0.74082	1.02465525	2.5965E-05	0.037833	0.709767	0.002154
480	0.725379	1.02641023	2.6358E-05	0.038464	0.709669	0.002107
490	0.710569	1.02822834	2.6748E-05	0.03909	0.709644	0.002061
500	0.696352	1.03010641	2.7135E-05	0.039712	0.709686	0.002018
510	0.682694	1.03204114	2.7518E-05	0.040331	0.709791	0.001976
520	0.669561	1.03402921	2.7897E-05	0.040945	0.709955	0.001936
530	0.656924	1.03606723	2.8274E-05	0.041555	0.710172	0.001898
540	0.644756	1.03815179	2.8647E-05	0.042162	0.71044	0.001861
550	0.633031	1.04027949	2.9017E-05	0.042765	0.710754	0.001826
560	0.621725	1.04244695	2.9384E-05	0.043365	0.711109	0.001792
570	0.610816	1.04465081	2.9748E-05	0.043961	0.711504	0.001759
580	0.600283	1.04688777	3.0109E-05	0.044555	0.711934	0.001728
590	0.590108	1.04915459	3.0468E-05	0.045145	0.712395	0.001698
600	0.580272	1.05144811	3.0824E-05	0.045732	0.712885	0.001669
610	0.570759	1.05376525	3.1177E-05	0.046317	0.713401	0.00164
620	0.561552	1.05610301	3.1528E-05	0.046899	0.713939	0.001613
630	0.552639	1.05845851	3.1876E-05	0.047478	0.714498	0.001587
640	0.544004	1.06082896	3.2222E-05	0.048054	0.715074	0.001562
650	0.535634	1.06321169	3.2565E-05	0.048628	0.715664	0.001537
660	0.527519	1.06560412	3.2906E-05	0.0492	0.716268	0.001513
670	0.519645	1.0680038	3.3245E-05	0.049769	0.716881	0.00149
680	0.512004	1.07040839	3.3582E-05	0.050336	0.717503	0.001468
690	0.504584	1.07281567	3.3916E-05	0.050901	0.718132	0.001446
700	0.497376	1.07522353	3.4249E-05	0.051464	0.718765	0.001425
710	0.490371	1.07762996	3.4579E-05	0.052025	0.719401	0.001404
720	0.483561	1.08003309	3.4907E-05	0.052584	0.720038	0.001385
730	0.476938	1.08243114	3.5234E-05	0.053141	0.720674	0.001365
740	0.470493	1.08482245	3.5558E-05	0.053696	0.72131	0.001347
750	0.46422	1.08720544	3.5880E-05	0.054249	0.721942	0.001328
760	0.458113	1.08957867	3.6201E-05	0.0548	0.72257	0.001311
770	0.452164	1.09194076	3.6520E-05	0.05535	0.723193	0.001293
780	0.446368	1.09429047	3.6837E-05	0.055899	0.723809	0.001277
790	0.440718	1.0966266	3.7152E-05	0.056445	0.724419	0.00126
800	0.43521	1.09894809	3.7466E-05	0.056991	0.72502	0.001244
810	0.429838	1.10125392	3.7778E-05	0.057534	0.725613	0.001229
820	0.424597	1.1035432	3.8088E-05	0.058076	0.726195	0.001214
830	0.419482	1.10581506	3.8397E-05	0.058617	0.726768	0.001199
840	0.414489	1.10806876	3.8704E-05	0.059157	0.72733	0.001185
850	0.409613	1.11030359	3.9010E-05	0.059695	0.727881	0.001171
860	0.404851	1.11251893	3.9314E-05	0.060232	0.728419	0.001157
870	0.400199	1.11471422	3.9616E-05	0.060767	0.728946	0.001143
880	0.395652	1.11688896	3.9917E-05	0.061302	0.72946	0.00113
890	0.391207	1.1190427	4.0217E-05	0.061835	0.729961	0.001118
900	0.386861	1.12117505	4.0515E-05	0.062367	0.730448	0.001105
910	0.38261	1.12328566	4.0812E-05	0.062898	0.730922	0.001093
920	0.378452	1.12537426	4.1108E-05	0.063428	0.731383	0.001081
930	0.374384	1.12744059	4.1402E-05	0.063957	0.73183	0.001069
940	0.370402	1.12948445	4.1695E-05	0.064484	0.732262	0.001058
950	0.366503	1.13150568	4.1986E-05	0.065011	0.732681	0.001047
960	0.362686	1.13350415	4.2276E-05	0.065537	0.733085	0.001036

970	0.358948	1.13547976	4.2565E-05	0.066062	0.733475	0.001025
980	0.355286	1.13743246	4.2853E-05	0.066586	0.733851	0.001014
990	0.351698	1.13936222	4.3140E-05	0.067109	0.734212	0.001004
1000	0.348182	1.14126904	4.3425E-05	0.067631	0.73456	9.94E-04
1010	0.344735	1.14315294	4.3709E-05	0.068152	0.734893	9.84E-04
1020	0.341356	1.14501397	4.3992E-05	0.068672	0.735212	9.75E-04
1030	0.338042	1.14685222	4.4274E-05	0.069192	0.735516	9.65E-04
1040	0.334793	1.14866776	4.4555E-05	0.069711	0.735807	9.56E-04
1050	0.331605	1.15046071	4.4835E-05	0.070229	0.736084	9.47E-04
1060	0.328477	1.1522312	4.5113E-05	0.070746	0.736347	9.38E-04
1070	0.325408	1.15397939	4.5391E-05	0.071262	0.736596	9.29E-04
1080	0.322395	1.15570542	4.5667E-05	0.071778	0.736832	9.20E-04
1090	0.319438	1.15740947	4.5942E-05	0.072293	0.737055	9.12E-04
1100	0.316535	1.15909173	4.6217E-05	0.072808	0.737264	9.04E-04
1110	0.313684	1.16075241	4.6490E-05	0.073321	0.73746	8.95E-04
1120	0.310884	1.16239169	4.6762E-05	0.073834	0.737643	8.87E-04
1130	0.308133	1.16400981	4.7034E-05	0.074347	0.737813	8.80E-04
1140	0.305431	1.16560699	4.7304E-05	0.074858	0.737971	8.72E-04
1150	0.302775	1.16718346	4.7573E-05	0.07537	0.738116	8.64E-04
1160	0.300166	1.16873946	4.7842E-05	0.07588	0.73825	8.57E-04
1170	0.297601	1.17027524	4.8109E-05	0.07639	0.738371	8.49E-04
1180	0.295079	1.17179103	4.8376E-05	0.076899	0.73848	8.42E-04
1190	0.2926	1.1732871	4.8642E-05	0.077408	0.738578	8.35E-04
1200	0.290162	1.17476371	4.8906E-05	0.077916	0.738664	8.28E-04
1210	0.287765	1.17622111	4.9170E-05	0.078424	0.738739	8.21E-04
1220	0.285406	1.17765956	4.9433E-05	0.078931	0.738803	8.15E-04
1230	0.283087	1.17907934	4.9695E-05	0.079438	0.738856	8.08E-04
1240	0.280804	1.1804807	4.9957E-05	0.079944	0.738899	8.02E-04
1250	0.278558	1.18186391	5.0217E-05	0.08045	0.738931	7.95E-04
1260	0.276348	1.18322924	5.0477E-05	0.080955	0.738952	7.89E-04
1270	0.274172	1.18457695	5.0736E-05	0.08146	0.738964	7.83E-04
1280	0.272031	1.18590732	5.0994E-05	0.081964	0.738966	7.76E-04
1290	0.269922	1.18722061	5.1251E-05	0.082468	0.738959	7.70E-04
1300	0.267847	1.18851708	5.1507E-05	0.082971	0.738941	7.65E-04
1310	0.265802	1.189797	5.1763E-05	0.083474	0.738915	7.59E-04
1320	0.263789	1.19106063	5.2018E-05	0.083976	0.73888	7.53E-04
1330	0.261806	1.19230823	5.2272E-05	0.084478	0.738835	7.47E-04
1340	0.259853	1.19354007	5.2525E-05	0.08498	0.738782	7.42E-04
1350	0.257928	1.19475639	5.2778E-05	0.085481	0.738721	7.36E-04
1360	0.256032	1.19595746	5.3030E-05	0.085982	0.738651	7.31E-04
1370	0.254164	1.19714353	5.3281E-05	0.086482	0.738573	7.26E-04
1380	0.252322	1.19831484	5.3531E-05	0.086983	0.738487	7.20E-04
1390	0.250507	1.19947165	5.3781E-05	0.087482	0.738394	7.15E-04
1400	0.248718	1.20061421	5.4030E-05	0.087982	0.738293	7.10E-04
1410	0.246955	1.20174274	5.4278E-05	0.08848	0.738184	7.05E-04
1420	0.245216	1.2028575	5.4526E-05	0.088979	0.738068	7.00E-04
1430	0.243502	1.20395871	5.4773E-05	0.089477	0.737946	6.95E-04
1440	0.241811	1.20504662	5.5019E-05	0.089975	0.737816	6.90E-04
1450	0.240144	1.20612144	5.5264E-05	0.090473	0.737679	6.86E-04
1460	0.238499	1.20718341	5.5509E-05	0.09097	0.737536	6.81E-04
1470	0.236877	1.20823276	5.5754E-05	0.091467	0.737387	6.76E-04
1480	0.235277	1.20926969	5.5997E-05	0.091964	0.737231	6.72E-04
1490	0.233698	1.21029443	5.6240E-05	0.09246	0.737069	6.67E-04
1500	0.23214	1.21130719	5.6482E-05	0.092956	0.736902	6.63E-04
1510	0.230603	1.21230819	5.6724E-05	0.093452	0.736728	6.58E-04
1520	0.229087	1.21329762	5.6965E-05	0.093947	0.736549	6.54E-04
1530	0.22759	1.21427568	5.7206E-05	0.094442	0.736364	6.50E-04
1540	0.226112	1.21524259	5.7446E-05	0.094937	0.736174	6.46E-04
1550	0.224653	1.21619854	5.7685E-05	0.095431	0.735979	6.41E-04
1560	0.223214	1.21714371	5.7924E-05	0.095926	0.735778	6.37E-04
1570	0.221792	1.2180783	5.8162E-05	0.09642	0.735573	6.33E-04
1580	0.220389	1.2190025	5.8399E-05	0.096913	0.735363	6.29E-04

1590	0.219003	1.21991649	5.8636E-05	0.097407	0.735148	6.25E-04
1600	0.217634	1.22082044	5.8872E-05	0.0979	0.734928	6.21E-04
1610	0.216283	1.22171454	5.9108E-05	0.098393	0.734704	6.18E-04
1620	0.214948	1.22259896	5.9343E-05	0.098886	0.734476	6.14E-04
1630	0.21363	1.22347386	5.9578E-05	0.099378	0.734243	6.10E-04
1640	0.212327	1.22433942	5.9812E-05	0.09987	0.734006	6.06E-04
1650	0.211041	1.22519579	6.0046E-05	0.100362	0.733765	6.03E-04
1660	0.20977	1.22604315	6.0279E-05	0.100854	0.733521	5.99E-04
1670	0.208514	1.22688163	6.0511E-05	0.101346	0.733272	5.96E-04
1680	0.207273	1.22771141	6.0743E-05	0.101837	0.73302	5.92E-04
1690	0.206047	1.22853263	6.0975E-05	0.102328	0.732764	5.88E-04
1700	0.204835	1.22934545	6.1205E-05	0.102819	0.732505	5.85E-04
1710	0.203637	1.23014999	6.1436E-05	0.10331	0.732242	5.82E-04
1720	0.202453	1.23094642	6.1666E-05	0.1038	0.731976	5.78E-04
1730	0.201283	1.23173486	6.1895E-05	0.10429	0.731707	5.75E-04
1740	0.200127	1.23251545	6.2124E-05	0.10478	0.731435	5.72E-04
1750	0.198983	1.23328834	6.2352E-05	0.10527	0.731159	5.68E-04
1760	0.197853	1.23405365	6.2580E-05	0.10576	0.730881	5.65E-04
1770	0.196735	1.2348115	6.2808E-05	0.106249	0.7306	5.62E-04
1780	0.19563	1.23556204	6.3035E-05	0.106738	0.730316	5.59E-04
1790	0.194538	1.23630537	6.3261E-05	0.107227	0.730029	5.56E-04
1800	0.193457	1.23704162	6.3487E-05	0.107716	0.72974	5.53E-04
1810	0.192388	1.23777092	6.3712E-05	0.108205	0.729448	5.50E-04
1820	0.191332	1.23849337	6.3937E-05	0.108693	0.729154	5.47E-04
1830	0.190286	1.23920909	6.4162E-05	0.109182	0.728857	5.44E-04
1840	0.189252	1.23991819	6.4386E-05	0.10967	0.728558	5.41E-04
1850	0.188229	1.24062078	6.4610E-05	0.110158	0.728257	5.38E-04
1860	0.187218	1.24131698	6.4833E-05	0.110646	0.727953	5.35E-04
1870	0.186217	1.24200687	6.5056E-05	0.111133	0.727648	5.32E-04
1880	0.185226	1.24269057	6.5278E-05	0.111621	0.72734	5.29E-04
1890	0.184246	1.24336818	6.5500E-05	0.112108	0.72703	5.26E-04
1900	0.183277	1.24403979	6.5721E-05	0.112595	0.726719	5.24E-04
1910	0.182317	1.2447055	6.5942E-05	0.113082	0.726405	5.21E-04
1920	0.181368	1.24536541	6.6163E-05	0.113569	0.72609	5.18E-04
1930	0.180428	1.2460196	6.6383E-05	0.114056	0.725773	5.16E-04
1940	0.179499	1.24666818	6.6602E-05	0.114542	0.725455	5.13E-04
1950	0.178578	1.24731122	6.6822E-05	0.115029	0.725134	5.10E-04
1960	0.177667	1.24794882	6.7041E-05	0.115515	0.724813	5.08E-04
1970	0.176766	1.24858105	6.7259E-05	0.116001	0.724489	5.05E-04
1980	0.175873	1.24920801	6.7477E-05	0.116487	0.724164	5.03E-04
1990	0.174989	1.24982977	6.7695E-05	0.116973	0.723838	5.00E-04
2000	0.174115	1.25044642	6.7912E-05	0.117458	0.72351	4.98E-04
2010	0.173248	1.25105802	6.8129E-05	0.117944	0.723181	4.95E-04
2020	0.172391	1.25166467	6.8345E-05	0.118429	0.722851	4.93E-04
2030	0.171542	1.25226643	6.8561E-05	0.118914	0.722519	4.90E-04
2040	0.170701	1.25286338	6.8777E-05	0.1194	0.722187	4.88E-04
2050	0.169869	1.25345559	6.8992E-05	0.119885	0.721853	4.85E-04
2060	0.169044	1.25404312	6.9207E-05	0.120369	0.721518	4.83E-04
2070	0.168228	1.25462606	6.9421E-05	0.120854	0.721182	4.81E-04
2080	0.167419	1.25520446	6.9635E-05	0.121339	0.720845	4.78E-04
2090	0.166618	1.25577839	6.9849E-05	0.121823	0.720506	4.76E-04
2100	0.165825	1.25634792	7.0062E-05	0.122308	0.720167	4.74E-04
2110	0.165039	1.25691311	7.0275E-05	0.122792	0.719827	4.72E-04
2120	0.164261	1.25747403	7.0488E-05	0.123276	0.719487	4.69E-04
2130	0.16349	1.25803073	7.0700E-05	0.12376	0.719145	4.67E-04
2140	0.162726	1.25858327	7.0912E-05	0.124244	0.718802	4.65E-04
2150	0.161969	1.25913172	7.1123E-05	0.124728	0.718459	4.63E-04
2160	0.161219	1.25967612	7.1334E-05	0.125211	0.718115	4.61E-04
2170	0.160476	1.26021655	7.1545E-05	0.125695	0.71777	4.59E-04
2180	0.15974	1.26075304	7.1756E-05	0.126178	0.717425	4.57E-04
2190	0.159011	1.26128566	7.1966E-05	0.126662	0.717079	4.55E-04
2200	0.158288	1.26181446	7.2175E-05	0.127145	0.716732	4.52E-04

2210	0.157572	1.26233948	7.2385E-05	0.127628	0.716385	4.50E-04
2220	0.156863	1.26286079	7.2594E-05	0.128111	0.716037	4.48E-04
2230	0.156159	1.26337843	7.2802E-05	0.128594	0.715688	4.46E-04
2240	0.155462	1.26389245	7.3011E-05	0.129077	0.715339	4.44E-04
2250	0.154771	1.2644029	7.3219E-05	0.129559	0.71499	4.42E-04
2260	0.154087	1.26490982	7.3426E-05	0.130042	0.71464	4.40E-04
2270	0.153408	1.26541326	7.3634E-05	0.130524	0.71429	4.39E-04
2280	0.152735	1.26591327	7.3841E-05	0.131007	0.713939	4.37E-04
2290	0.152068	1.26640989	7.4047E-05	0.131489	0.713588	4.35E-04
2300	0.151407	1.26690317	7.4254E-05	0.131971	0.713236	4.33E-04
2310	0.150752	1.26739314	7.4460E-05	0.132454	0.712884	4.31E-04
2320	0.150102	1.26787985	7.4665E-05	0.132936	0.712532	4.29E-04
2330	0.149458	1.26836334	7.4871E-05	0.133417	0.71218	4.27E-04
2340	0.14882	1.26884365	7.5076E-05	0.133899	0.711827	4.25E-04
2350	0.148186	1.26932081	7.5281E-05	0.134381	0.711474	4.24E-04
2360	0.147559	1.26979488	7.5485E-05	0.134863	0.711121	4.22E-04
2370	0.146936	1.27026588	7.5689E-05	0.135344	0.710767	4.20E-04
2380	0.146319	1.27073386	7.5893E-05	0.135826	0.710413	4.18E-04
2390	0.145707	1.27119885	7.6097E-05	0.136307	0.71006	4.17E-04
2400	0.1451	1.27166088	7.6300E-05	0.136789	0.709706	4.15E-04
2410	0.144498	1.27212	7.6503E-05	0.13727	0.709351	4.13E-04
2420	0.143901	1.27257623	7.6705E-05	0.137751	0.708997	4.11E-04
2430	0.143309	1.27302962	7.6907E-05	0.138232	0.708643	4.10E-04
2440	0.142721	1.27348018	7.7109E-05	0.138713	0.708288	4.08E-04
2450	0.142139	1.27392797	7.7311E-05	0.139194	0.707934	4.06E-04
2460	0.141561	1.274373	7.7513E-05	0.139675	0.707579	4.05E-04
2470	0.140988	1.27481532	7.7714E-05	0.140156	0.707224	4.03E-04
2480	0.14042	1.27525495	7.7915E-05	0.140636	0.706869	4.02E-04
2490	0.139856	1.27569192	7.8115E-05	0.141117	0.706515	4.00E-04
2500	0.139296	1.27612626	7.8315E-05	0.141597	0.70616	3.98E-04
2510	0.138742	1.27655801	7.8515E-05	0.142078	0.705805	3.97E-04
2520	0.138191	1.27698719	7.8715E-05	0.142558	0.70545	3.95E-04
2530	0.137645	1.27741383	7.8914E-05	0.143039	0.705096	3.94E-04
2540	0.137103	1.27783797	7.9113E-05	0.143519	0.704741	3.92E-04
2550	0.136566	1.27825961	7.9312E-05	0.143999	0.704386	3.91E-04
2560	0.136032	1.2786788	7.9511E-05	0.144479	0.704032	3.89E-04
2570	0.135503	1.27909556	7.9709E-05	0.144959	0.703677	3.88E-04
2580	0.134978	1.27950992	7.9907E-05	0.145439	0.703323	3.86E-04
2590	0.134457	1.27992191	8.0105E-05	0.145919	0.702969	3.85E-04
2600	0.13394	1.28033154	8.0302E-05	0.146399	0.702615	3.83E-04
2610	0.133427	1.28073884	8.0500E-05	0.146879	0.702261	3.82E-04
2620	0.132917	1.28114384	8.0696E-05	0.147358	0.701907	3.80E-04
2630	0.132412	1.28154657	8.0893E-05	0.147838	0.701553	3.79E-04
2640	0.131911	1.28194704	8.1090E-05	0.148318	0.701199	3.77E-04
2650	0.131413	1.28234528	8.1286E-05	0.148797	0.700846	3.76E-04
2660	0.130919	1.28274131	8.1482E-05	0.149277	0.700493	3.74E-04
2670	0.130429	1.28313516	8.1677E-05	0.149756	0.700139	3.73E-04
2680	0.129942	1.28352685	8.1873E-05	0.150235	0.699787	3.72E-04
2690	0.129459	1.2839164	8.2068E-05	0.150714	0.699434	3.70E-04
2700	0.12898	1.28430383	8.2262E-05	0.151194	0.699081	3.69E-04
2710	0.128504	1.28468917	8.2457E-05	0.151673	0.698729	3.68E-04
2720	0.128031	1.28507243	8.2651E-05	0.152152	0.698377	3.66E-04
2730	0.127562	1.28545363	8.2846E-05	0.152631	0.698025	3.65E-04
2740	0.127097	1.28583281	8.3039E-05	0.15311	0.697674	3.64E-04
2750	0.126635	1.28620997	8.3233E-05	0.153589	0.697323	3.62E-04
2760	0.126176	1.28658513	8.3426E-05	0.154068	0.696972	3.61E-04
2770	0.125721	1.28695833	8.3619E-05	0.154546	0.696621	3.60E-04
2780	0.125268	1.28732957	8.3812E-05	0.155025	0.69627	3.58E-04
2790	0.124819	1.28769887	8.4005E-05	0.155504	0.69592	3.57E-04
2800	0.124374	1.28806626	8.4197E-05	0.155982	0.69557	3.56E-04
2810	0.123931	1.28843175	8.4389E-05	0.156461	0.69522	3.55E-04
2820	0.123492	1.28879536	8.4581E-05	0.156939	0.694871	3.53E-04

2830	0.123055	1.28915711	8.4773E-05	0.157418	0.694522	3.52E-04
2840	0.122622	1.28951701	8.4964E-05	0.157896	0.694173	3.51E-04
2850	0.122192	1.2898751	8.5156E-05	0.158375	0.693825	3.50E-04
2860	0.121765	1.29023137	8.5347E-05	0.158853	0.693477	3.48E-04
2870	0.121341	1.29058585	8.5537E-05	0.159331	0.693129	3.47E-04
2880	0.120919	1.29093856	8.5728E-05	0.159809	0.692782	3.46E-04
2890	0.120501	1.2912895	8.5918E-05	0.160288	0.692435	3.45E-04
2900	0.120086	1.29163871	8.6108E-05	0.160766	0.692088	3.44E-04
2910	0.119673	1.2919862	8.6298E-05	0.161244	0.691742	3.42E-04
2920	0.119263	1.29233197	8.6487E-05	0.161722	0.691396	3.41E-04
2930	0.118856	1.29267605	8.6677E-05	0.1622	0.69105	3.40E-04
2940	0.118452	1.29301846	8.6866E-05	0.162678	0.690705	3.39E-04
2950	0.11805	1.2933592	8.7055E-05	0.163155	0.69036	3.38E-04
2960	0.117652	1.2936983	8.7243E-05	0.163633	0.690015	3.37E-04
2970	0.117256	1.29403576	8.7432E-05	0.164111	0.689671	3.36E-04
2980	0.116862	1.29437161	8.7620E-05	0.164589	0.689327	3.34E-04
2990	0.116471	1.29470586	8.7808E-05	0.165066	0.688984	3.33E-04
3000	0.116083	1.29503852	8.7996E-05	0.165544	0.688641	3.32E-04

We can create the same table also for the other common heat transfer fluid: water, In order to create this data International association fro the properties of steam and Water (IAPSW) 97 equation of state[138] is used. Details of the equation of state used for the calculations is as follows:

IAPWS-IF97 & IAPWS 95 STEAM(WATER) EQUATION OF STATES

Steam is one of the most important and most used working fluid in thermodynamic systems. In previous section An equation of state for steam is given. In recent years maximum operating temperatures and pressures of rankine cycle power plants has increased. International Association for the Properties of Water and Steam(IAPWS) is developed a new set of equation of states which are more accurate and covers larger range of data. This new set of equations are developed in 1997[138]. Steam properties are given by 5 sets of equation of states, as shown in the Figure 2.6.1

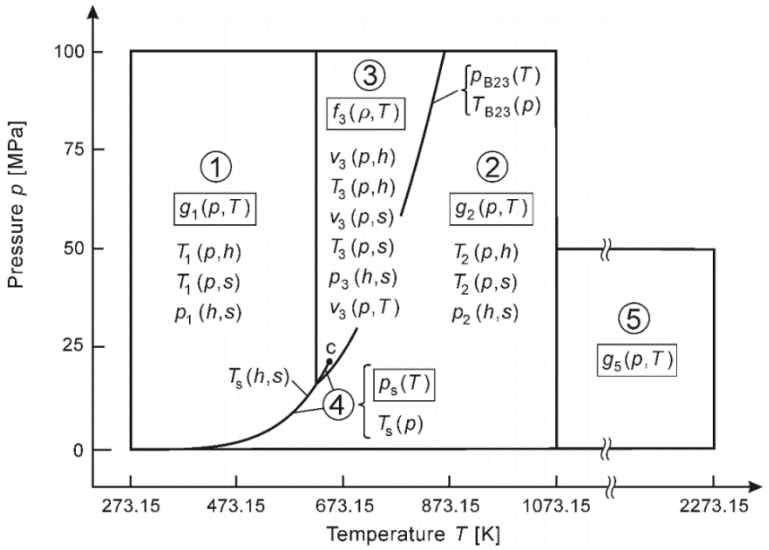


Figure 2.6.1 Equation of state regions for steam

The first equation, which covers basically liquid region has the following gibbs free energy form:

$$\frac{g_1(p,T)}{RT} = \gamma(\pi,\tau) = \sum_{i=1}^{34} n_i (71-\pi)^{I_i} (\tau - 1222)^{J_i}$$

Where $\pi = p / p^*$ $\tau = T^* / T$ p^*=16.62 MPa and T^*=1386 K R=0461526 kJ/(kgK)

Table coefficients of eqn.

i	I_i	J_i	n_i	i	I_i	J_i	n_i
1	0	-2	0.14632971213167	18	2	3	-4.4141845331E-06
2	0	-1	-0.84548187169114	19	2	17	-7.2694996298E-16
3	0	0	-3.75636036720400	20	3	-4	-3.1679644845E-05
4	0	1	3.38551691683850	21	3	0	-2.8270797985E-06
5	0	2	-0.95791963387872	22	3	6	-8.5205128120E-10
6	0	3	0.15772038513228	23	4	-5	-2.2425281908E-06
7	0	4	-0.01661641719950	24	4	-2	-6.5171222896E-07

8	0	5	0.00081214629984	25	4	10	-1.4341729938E-13
9	1	-9	0.00028319080124	26	5	-8	-4.0516996860E-07
10	1	-7	-0.00060706301566	27	8	-11	-1.2734301742E-09
11	1	-1	-0.01899006821842	28	8	-6	-1.7424871231E-10
12	1	0	-0.03252974877051	29	21	-29	-6.8762131296E-19
13	1	1	-0.02184171717541	30	23	-31	1.4478307829E-20
14	1	3	-0.00005283835797	31	29	-38	2.6335781663E-23
15	2	-3	-0.00047184321073	32	30	-39	-1.1947622640E-23
16	2	0	-0.00030001780793	33	31	-40	1.8228094581E-24
17	2	1	0.00004766139391	34	32	-41	-9.3537087292E-26

Thermodynamic relations can be calculated from these thermodynamic relations

Specific volume: $v = \left(\frac{\partial g}{\partial p}\right)_T$

Specific enthalpy: $h = g - T\left(\frac{\partial g}{\partial p}\right)_P$

Specific internal energy: $u = g - T\left(\frac{\partial g}{\partial T}\right)_P - p\left(\frac{\partial g}{\partial p}\right)_T$

Specific entropy: $s = \left(\frac{\partial g}{\partial T}\right)_P$

Specific isobaric heat capacity: $C_p = \left(\frac{\partial h}{\partial T}\right)_P$

Specific isochoric heat capacity: $C_v = \left(\frac{\partial u}{\partial T}\right)_v$

The second equation equation, which covers vapor region has the following gibbs free energy form:

$$\frac{g_2(p,T)}{RT} = \gamma(\pi,\tau) = \gamma^0(\pi,\tau) + \gamma^r(\pi,\tau)$$

Where $\pi = p/p^*$ $\tau = T^*/T$ R=0461526 kJ/(kgK), $\gamma^0(\pi,\tau)$ is the ideal gas part of EOS, and $\gamma^r(\pi,\tau)$ is the real gas part of the EOS. İdeal gas part equation:

$$\gamma^0(\pi,\tau) = \ln\pi + \sum_{i=1}^{9} n_i^0 \tau^{J_i} \quad ($$

Where p^*=1MPa and T^*=540 K

Table coefficients of eqn.

i	J_i	n_i^0	i	J_i	n_i^0
1	0	-9.692768650E+00	6	-2	1.4240819171E+00
2	1	1.008665597E+01	7	-1	-4.3839511319E+00
3	-5	-5.608791128E-03	8	2	-2.8408632461E-01
4	-4	7.145273808E-02	9	3	2.1268463753E-02
5	-3	-4.071049822E-01			

dimensionless residual part of the basic equation *g*2(*p*,*T*) is as follows:

$$\gamma^r(\pi,\tau) = \sum_{i=1}^{43} n_i \pi^{I_i} (\tau - 0.5)^{J_i}$$

Where p^*=1MPa and T^*=540 K

Table coefficients of eqn.

i	I_i	J_i	n_i	i	I_i	J_i	n_i
1	1	0			7	0	-5.9059564324270E-18
2	1	1	-1.7834862292358E-02	24	7	11	-1.2621808899101E-06
3	1	2	-4.5996013696365E-02	25	7	25	-3.8946842435739E-02
4	1	3	-5.7581259083432E-02	26	8	8	1.1256211360459E-11
5	1	6	-5.0325278727930E-02	27	8	36	-8.2311340897998E+00
6	2	1	-3.3032641670203E-05	28	9	13	1.9809712802088E-08
7	2	2	-1.8948987516315E-04	29	10	4	1.0406965210174E-19

8	2	4	-3.9392777243355E-03	30	10	10	-1.0234747095929E-13
9	2	7	-4.3797295650573E-02	31	10	14	-1.0018179379511E-09
10	2	36	-2.6674547914087E-05	32	16	29	-8.0882908646985E-11
11	3	0	2.0481737692309E-08	33	16	50	1.0693031879409E-01
12	3	1	4.3870667284435E-07	34	18	57	-3.3662250574171E-01
13	3	3	-3.2277677238570E-05	35	20	20	8.9185845355421E-25
14	3	6	-1.5033924542148E-03	36	20	35	3.0629316876232E-13
15	3	35	-4.0668253562649E-02	37	20	48	-4.2002467698208E-06
16	4	1	-7.8847309559367E-10	38	21	21	-5.9056029685639E-26
17	4	2	1.2790717852285E-08	39	22	53	3.7826947613457E-06
18	4	3	4.8225372718507E-07	40	23	39	-1.2768608934681E-15
19	5	7	2.2922076337661E-06	41	24	26	7.3087610595061E-29
20	6	3	-1.6714766451061E-11	42	24	40	5.5414715350778E-17
21	6	16	-2.1171472321355E-03	43	24	58	-9.4369707241210E-07
22	6	35	-2.3895741934104E+01				

Region 3 equation is given as Helmholts free energy form:

$$\frac{f_3(\rho,T)}{RT} = \phi(\delta,\tau) = n_1 \ln\delta + \sum_{i=2}^{40} n_i \delta^{I_i} \tau^{J_i}$$

Where $\delta = \rho / \rho^*$ $\tau = T^* / T$, $T^*=T_c=647.096$ and R=0461526 kJ/(kgK)

Table coefficients of eqn.

i	I_i	J_i	n_i	i	I_i	J_i	n_i
1	0	0	1.065807002851E+00	21	3	4	-2.0189915023570E+00
2	0	0	-1.573284529024E+01	22	3	16	-8.2147637173963E-03
3	0	1	2.094439697431E+01	23	3	26	-4.7596035734923E-01
4	0	2	-7.686770787872E+00	24	4	0	4.3984074473500E-02
5	0	7	2.618594778795E+00	25	4	2	-4.4476435428739E-01
6	0	10	-2.808078114862E+00	26	4	4	9.0572070719733E-01
7	0	12	1.205336969652E+00	27	4	26	7.0522450087967E-01
8	0	23	-8.456681281250E-03	28	5	1	1.0770512626332E-01
9	1	2	-1.265431547771E+00	29	5	3	-3.2913623258954E-01
10	1	6	-1.152440780668E+00	30	5	26	-5.0871062041158E-01
11	1	15	8.852104398432E-01	31	6	0	-2.2175400873096E-02
12	1	17	-6.420776518161E-01	32	6	2	9.4260751665092E-02
13	2	0	3.849346018667E-01	33	6	26	1.6436278447961E-01
14	2	2	-8.521470882421E-01	34	7	2	-1.3503372241348E-02
15	2	6	4.897228154188E+00	35	8	26	-1.4834345352472E-02
16	2	7	-3.050261725697E+00	36	9	2	5.7922953628084E-04
17	2	22	3.942053687915E-02	37	9	26	3.2308904703711E-03
18	2	26	1.255840842431E-01	38	10	0	8.0964802996215E-05
19	3	0	-2.799932969871E-01	39	10	1	-1.6557679795037E-04
20	3	2	1.389979956946E+00	40	11	26	-4.4923899061815E-05

It should be noted that this set of equation is function of density and temperature, and basic equation is helmholts equation so, let us list definition of other thermodynamic properties

Pressure: $p = \rho^2 \left(\frac{\partial f}{\partial \rho} \right)_T$

Specific enthalpy: $h = f - T\left(\frac{\partial f}{\partial T} \right)_P + \rho \left(\frac{\partial f}{\partial \rho} \right)_T$

Specific internal energy: $u = f - T\left(\frac{\partial f}{\partial T} \right)_P$

Specific entropy: $s = \left(\frac{\partial f}{\partial T} \right)_\rho$

Specific isobaric heat capacity: $C_p = \left(\frac{\partial h}{\partial T} \right)_P$

Specific isochoric heat capacity: $C_v = \left(\frac{\partial u}{\partial T}\right)_v$

Region 4 of the equation defines saturation region. The basic equation is given as a polynomial

$$\beta^2\vartheta^2 + n_1\beta^2\vartheta + n_2\beta^2 + n_3\beta\vartheta^2 + n_4\beta\vartheta + n_5\beta + n_6\vartheta^2 + n_7\vartheta + n_8 = 0 \quad (2.6.18)$$

Where

$$\beta = \left(p_s / p^*\right)^{0.25}$$

$$\vartheta = \frac{T_s}{T^*} + \frac{n_9}{\left(T_s / T^*\right) - n_{10}}$$

From this equation both saturation pressure and saturation temperature eqaution can be derived.

$$\frac{P_s}{P^*} = \left[\frac{2C}{-B + (B^2 - 4AC)^{0.5}}\right]^4$$

Where p^*=1 MPa

$$A = \vartheta^2 + n_1\vartheta + n_2$$

$$B = n_3\vartheta^2 + n_4\vartheta + n_5$$

$$C = n_6\vartheta^2 + n_7\vartheta + n_8$$

Table coefficients of eqn.

i	n_i	i	n_i
1	1.1670521453E+03	6	1.4915108614E+01
2	-7.2421316703E+05	7	-4.8232657362E+03
3	-1.7073846940E+01	8	4.0511340542E+05
4	1.2020824702E+04	9	-2.3855557568E-01
5	-3.2325550322E+06	10	6.5017534845E+02

It is also possible to drive saturation temperature eqaution from the basic polynomial as:

$$\frac{T_s}{T^*} = \frac{n_{10} + D - \left[(n_{10} + D)^2 - 4(n_9 + n_{10}D\right]^{0.5}}{2}$$

Where T^*=1 K

$$D = \frac{2G}{-F - \left(F^2 - 4EG\right)^{0.5}}$$

$$E = \beta^2 + n_3\beta + n_6$$

$$F = n_1\beta^2 + n_4\beta + n_7$$

$$G = n_2\beta^2 + n_5\beta + n_8$$

And the final region for steam is region 5, again given as gibbs free eqaution type EOS

$$\frac{g_5(p,T)}{RT} = \gamma(\pi,\tau) = \gamma^0(\pi,\tau) + \gamma^r(\pi,\tau)$$

Where $\pi = p / p^*$ $\tau = T^* / T$ R=0461526 kJ/(kgK), $\gamma^0(\pi,\tau)$ is the ideal gas part of EOS, and $\gamma^r(\pi,\tau)$ is the real gas part of the EOS. İdeal gas part equation:

$$\gamma^0(\pi,\tau) = \ln\pi + \sum_{i=1}^{9} n_i^0 \tau^{J_i}$$

Where p^*=1MPa and T^*=1000 K

Table coefficients of eqn.

i	J_i^0	n_i^0	i	J_i^0	n_i^0
1	0	-13.1799836742	4	-2	0.3690153498
2	1	6.8540841634	5	-1	-3.1161318214
3	-3	-0.0248051489	6	2	-0.3296162654

The real gas part of the equation

$$\gamma^r(\pi,\tau)=\sum_{i=1}^{43} n_i \pi^{I_i} \tau^{J_i} \quad (2.6.25)$$

Table coefficients of eqn.

i	I_i	J_i	n_i	i	I_i	J_i	n_i
1	1	1	1.5736404855E-03	4	2	3	2.2440037409E-06
2	1	2	9.0153761674E-04	5	2	9	-4.1163275453E-06
3	1	3	-5.0270077678E-03	6	3	7	3.7919454823E-08

```
public class water_test
{public static void main(String arg[])
 { steamIAPWS_IF97 st=new steamIAPWS_IF97();
   double P=100.0; //kPa
   for(double T=273.15001;T<373.15;T+=1)
   {
   double ro=st.ro_PT(P,T);
   double Cp=st.Cp_PT(P,T);
   double vis=st.viscosity_PT(P,T);
   double k=st.k_PT(P,T);
   double Pr=st.Pr_roT(ro,T);
    double beta=st.kappa1(P,T);
    double sigma=st.sigma(T);
    System.out.println(T+" "+P+" "+ro+" "+Cp+" "+vis+" "+k+" "+Pr+" "+beta+" "+sigma);
   }
  }
}
```

For program code steamIAPWS_IF97 please check www.turhancoban.com (SCO1.jar)

Properties of water at P=100 kPa

T degree K	ρ kg/m^3	Cp kJ/kgK	η Pas	k W/(mK)	Pr	β 1/K	σ N/m
273.15	999.8436	4.219436922	1.791753E-03	0.562031	13.45156	-6.77E-05	0.075648
274.15	999.9023	4.216018206	1.731019E-03	0.564145	12.93641	-4.97E-05	0.075508
275.15	999.9433	4.212885296	1.673514E-03	0.566231	12.45131	-3.24E-05	0.075367
276.15	999.9673	4.210012125	1.619008E-03	0.56829	11.99395	-1.56E-05	0.075226
277.15	999.9748	4.207375111	1.567292E-03	0.570322	11.56221	5.85E-07	0.075084
278.15	999.9663	4.204952928	1.518173E-03	0.572327	11.15419	1.63E-05	0.074942
279.15	999.9423	4.202726295	1.471478E-03	0.574307	10.76814	3.15E-05	0.074799
280.15	999.9034	4.200677782	1.427045E-03	0.576261	10.4025	4.63E-05	0.074655
281.15	999.8499	4.198791641	1.384726E-03	0.57819	10.05582	6.06E-05	0.074511
282.15	999.7823	4.197053644	1.344387E-03	0.580095	9.726799	7.46E-05	0.074366
283.15	999.7009	4.195450946	1.305902E-03	0.581975	9.414234	8.81E-05	0.074221
284.15	999.6062	4.193971948	1.269157E-03	0.583832	9.11703	1.01E-04	0.074075
285.15	999.4984	4.192606185	1.234046E-03	0.585665	8.83418	1.14E-04	0.073929
286.15	999.378	4.191344218	1.200470E-03	0.587475	8.564763	1.27E-04	0.073782
287.15	999.2453	4.190177536	1.168339E-03	0.589262	8.307928	1.39E-04	0.073634
288.15	999.1005	4.189098467	1.137570E-03	0.591027	8.062894	1.51E-04	0.073486
289.15	998.9439	4.188100101	1.108083E-03	0.59277	7.828938	1.63E-04	0.073337
290.15	998.7759	4.187176216	1.079808E-03	0.594492	7.605395	1.74E-04	0.073188
291.15	998.5967	4.186321214	1.052676E-03	0.596191	7.391649	1.85E-04	0.073038
292.15	998.4064	4.185530056	1.026625E-03	0.59787	7.187129	1.96E-04	0.072887
293.15	998.2055	4.184798214	1.001597E-03	0.599528	6.991307	2.07E-04	0.072736
294.15	997.994	4.184121621	9.775378E-04	0.601164	6.80369	2.17E-04	0.072584
295.15	997.7723	4.183496622	9.543966E-04	0.602781	6.623824	2.27E-04	0.072432
296.15	997.5405	4.182919938	9.321260E-04	0.604377	6.451283	2.37E-04	0.072279
297.15	997.2988	4.182388628	9.106817E-04	0.605953	6.285673	2.47E-04	0.072126
298.15	997.0474	4.181900057	8.900223E-04	0.60751	6.126624	2.57E-04	0.071972
299.15	996.7866	4.181451862	8.701091E-04	0.609047	5.973794	2.66E-04	0.071818
300.15	996.5164	4.181041931	8.509055E-04	0.610564	5.826859	2.76E-04	0.071663
301.15	996.237	4.180668371	8.323775E-04	0.612063	5.685521	2.85E-04	0.071507
302.15	995.9487	4.180329493	8.144929E-04	0.613542	5.549496	2.94E-04	0.071351
303.15	995.6515	4.180023786	7.972215E-04	0.615002	5.418523	3.03E-04	0.071194
304.15	995.3456	4.179749902	7.805351E-04	0.616444	5.292353	3.12E-04	0.071037
305.15	995.0312	4.179506639	7.644068E-04	0.617868	5.170754	3.20E-04	0.070879
306.15	994.7083	4.179292922	7.488115E-04	0.619273	5.053509	3.29E-04	0.070721

307.15	994.3772	4.179107798	7.337254E-04	0.62066	4.940412	3.37E-04	0.070562
308.15	994.0379	4.178950414	7.191262E-04	0.622029	4.831271	3.45E-04	0.070402
309.15	993.6907	4.178820015	7.049926E-04	0.623381	4.725903	3.53E-04	0.070242
310.15	993.3355	4.178715926	6.913046E-04	0.624714	4.624138	3.62E-04	0.070081
311.15	992.9725	4.178637549	6.780435E-04	0.626031	4.525814	3.69E-04	0.06992
312.15	992.6019	4.178584352	6.651911E-04	0.627329	4.430779	3.77E-04	0.069759
313.15	992.2237	4.178555861	6.527307E-04	0.628611	4.338887	3.85E-04	0.069596
314.15	991.838	4.178551658	6.406461E-04	0.629875	4.250005	3.93E-04	0.069434
315.15	991.445	4.17857137	6.289221E-04	0.631123	4.164001	4.00E-04	0.06927
316.15	991.0447	4.178614666	6.175443E-04	0.632353	4.080756	4.08E-04	0.069106
317.15	990.6372	4.178681251	6.064989E-04	0.633567	4.000153	4.15E-04	0.068942
318.15	990.2227	4.178770865	5.957730E-04	0.634765	3.922083	4.22E-04	0.068777
319.15	989.8012	4.178883275	5.853541E-04	0.635945	3.846441	4.29E-04	0.068611
320.15	989.3728	4.179018275	5.752304E-04	0.63711	3.773131	4.36E-04	0.068445
321.15	988.9375	4.179175681	5.653908E-04	0.638258	3.702058	4.44E-04	0.068279
322.15	988.4955	4.17935533	5.558246E-04	0.63939	3.633133	4.51E-04	0.068112
323.15	988.0469	4.179557073	5.465216E-04	0.640506	3.566272	4.57E-04	0.067944
324.15	987.5917	4.179780781	5.374722E-04	0.641606	3.501395	4.64E-04	0.067776
325.15	987.13	4.180026336	5.286671E-04	0.64269	3.438425	4.71E-04	0.067607
326.15	986.6618	4.180293633	5.200974E-04	0.643759	3.37729	4.78E-04	0.067438
327.15	986.1873	4.180582575	5.117549E-04	0.644812	3.31792	4.84E-04	0.067268
328.15	985.7064	4.180893078	5.036314E-04	0.645849	3.26025	4.91E-04	0.067098
329.15	985.2194	4.181225064	4.957193E-04	0.646871	3.204216	4.98E-04	0.066927
330.15	984.7261	4.181578462	4.880112E-04	0.647877	3.149758	5.04E-04	0.066755
331.15	984.2268	4.181953207	4.805002E-04	0.648869	3.09682	5.10E-04	0.066584
332.15	983.7214	4.18234924	4.731796E-04	0.649845	3.045345	5.17E-04	0.066411
333.15	983.21	4.182766509	4.660428E-04	0.650806	2.995283	5.23E-04	0.066238
334.15	982.6927	4.183204962	4.590838E-04	0.651752	2.946583	5.29E-04	0.066065
335.15	982.1695	4.183664556	4.522967E-04	0.652683	2.899197	5.36E-04	0.065891
336.15	981.6405	4.184145247	4.456759E-04	0.6536	2.85308	5.42E-04	0.065716
337.15	981.1058	4.184646998	4.392158E-04	0.654502	2.808188	5.48E-04	0.065541
338.15	980.5653	4.185169773	4.329114E-04	0.655389	2.764479	5.54E-04	0.065366
339.15	980.0191	4.185713539	4.267577E-04	0.656261	2.721913	5.60E-04	0.06519
340.15	979.4673	4.186278267	4.207498E-04	0.657119	2.680451	5.66E-04	0.065013
341.15	978.91	4.18686393	4.148832E-04	0.657963	2.640058	5.72E-04	0.064836
342.15	978.3471	4.187470503	4.091535E-04	0.658792	2.600697	5.78E-04	0.064659
343.15	977.7787	4.188097963	4.035564E-04	0.659607	2.562335	5.84E-04	0.064481
344.15	977.2049	4.188746293	3.980879E-04	0.660408	2.524939	5.90E-04	0.064302
345.15	976.6257	4.189415475	3.927441E-04	0.661194	2.488478	5.96E-04	0.064123
346.15	976.0411	4.190105494	3.875212E-04	0.661967	2.452923	6.02E-04	0.063944
347.15	975.4512	4.190816339	3.824155E-04	0.662726	2.418245	6.07E-04	0.063764
348.15	974.8561	4.191548002	3.774236E-04	0.663471	2.384415	6.13E-04	0.063583
349.15	974.2557	4.192300476	3.725422E-04	0.664202	2.351408	6.19E-04	0.063402
350.15	973.65	4.193073759	3.677679E-04	0.664919	2.319197	6.25E-04	0.06322
351.15	973.0393	4.19386785	3.630978E-04	0.665622	2.28776	6.30E-04	0.063038
352.15	972.4233	4.194682752	3.585287E-04	0.666312	2.257071	6.36E-04	0.062856
353.15	971.8023	4.195518472	3.540577E-04	0.666989	2.227108	6.42E-04	0.062673
354.15	971.1762	4.196375019	3.496822E-04	0.667651	2.19785	6.47E-04	0.062489
355.15	970.5451	4.197252407	3.453994E-04	0.668301	2.169276	6.53E-04	0.062305
356.15	969.9089	4.198150653	3.412067E-04	0.668937	2.141364	6.58E-04	0.062121
357.15	969.2678	4.199069776	3.371015E-04	0.669559	2.114096	6.64E-04	0.061936
358.15	968.6217	4.200009802	3.330816E-04	0.670169	2.087453	6.70E-04	0.06175
359.15	967.9707	4.200970759	3.291444E-04	0.670765	2.061417	6.75E-04	0.061565
360.15	967.3148	4.201952679	3.252879E-04	0.671348	2.03597	6.81E-04	0.061378
361.15	966.6541	4.2029556	3.215097E-04	0.671918	2.011095	6.86E-04	0.061191
362.15	965.9885	4.203979561	3.178079E-04	0.672475	1.986777	6.92E-04	0.061004
363.15	965.318	4.20502461	3.141803E-04	0.673019	1.963	6.97E-04	0.060816
364.15	964.6428	4.206090797	3.106249E-04	0.67355	1.939747	7.02E-04	0.060628
365.15	963.9628	4.207178176	3.071400E-04	0.674068	1.917006	7.08E-04	0.060439
366.15	963.2781	4.208286807	3.037236E-04	0.674573	1.894761	7.13E-04	0.06025
367.15	962.5886	4.209416756	3.003739E-04	0.675066	1.873	7.19E-04	0.06006
368.15	961.8944	4.210568092	2.970892E-04	0.675546	1.851709	7.24E-04	0.05987

369.15	961.1956	4.21174089	2.938679E-04	0.676013	1.830875	7.30E-04	0.059679
370.15	960.492	4.212935232	2.907083E-04	0.676468	1.810486	7.35E-04	0.059488
371.15	959.7839	4.214151203	2.876089E-04	0.67691	1.79053	7.40E-04	0.059296
372.15	959.071	4.215388895	2.845682E-04	0.67734	1.770996	7.46E-04	0.059104

Properties of water-steam at saturated state.

T	P	ρf	ρg	hfg	Cpf	Cpg	ηf	ηg	kf	kg	Prf	Prg	σ	β
degree K	kPa	kg/m3	kg/m4	kJ/(kg)	kJ/(kgK)	kJ/(kgK)	Pas	Pas	W/(mK)	W/(mK)			N/m	1/K
273.15	0.6112	4.00E-04	999.79	2500.9	4.2199	1.8882	0.001792	8.95E-06	0.562	0.0165	13.456	1.0241	0.07565	-6.81E-05
275	0.6985	3.99E-04	999.89	2496.5	4.2138	1.8894	0.0016821	9.00E-06	0.5659	0.0166	12.526	1.0227	0.07539	-3.53E-05
276	0.7501	3.99E-04	999.92	2494.2	4.2109	1.8901	0.0016272	9.03E-06	0.5679	0.0167	12.065	1.0219	0.07525	-1.84E-05
277	0.805	3.99E-04	999.93	2491.8	4.2082	1.8908	0.001575	9.06E-06	0.57	0.0168	11.629	1.0212	0.07511	-2.13E-06
278	0.8635	3.98E-04	999.92	2489.4	4.2057	1.8916	0.0015255	9.09E-06	0.572	0.0168	11.217	1.0205	0.07496	1.37E-05
279	0.9257	3.98E-04	999.9	2487	4.2035	1.8923	0.0014785	9.12E-06	0.574	0.0169	10.828	1.0198	0.07482	2.90E-05
280	0.9918	3.98E-04	999.86	2484.7	4.2014	1.8931	0.0014337	9.14E-06	0.5759	0.017	10.459	1.0192	0.07468	4.38E-05
281	1.0621	3.98E-04	999.81	2482.3	4.1995	1.8939	0.0013911	9.17E-06	0.5778	0.0171	10.118	1.0186	0.07453	5.82E-05
282	1.1367	3.97E-04	999.75	2479.9	4.1977	1.8947	0.0013504	9.20E-06	0.5798	0.0171	9.7776	1.018	0.07439	7.23E-05
283	1.2159	3.97E-04	999.67	2477.6	4.1961	1.8956	0.0013117	9.23E-06	0.5816	0.0172	9.4625	1.0175	0.07424	8.59E-05
284	1.2999	3.97E-04	999.57	2475.2	4.1946	1.8965	0.0012746	9.26E-06	0.5835	0.0173	9.1629	1.0169	0.0741	9.91E-05
285	1.389	3.96E-04	999.47	2472.8	4.1932	1.8974	0.0012393	9.29E-06	0.5853	0.0173	8.8779	1.0164	0.07395	1.12E-04
286	1.4834	3.96E-04	999.35	2470.5	4.1919	1.8983	0.0012055	9.32E-06	0.5872	0.0174	8.6064	1.0159	0.0738	1.25E-04
287	1.5834	3.96E-04	999.22	2468.1	4.1907	1.8993	0.0011731	9.35E-06	0.5889	0.0175	8.3476	1.0155	0.07366	1.37E-04
288	1.6893	3.96E-04	999.08	2465.7	4.1896	1.9003	0.0011422	9.39E-06	0.5907	0.0176	8.1007	1.0151	0.07351	1.49E-04
289	1.8014	3.95E-04	998.92	2463.4	4.1886	1.9013	0.0011125	9.42E-06	0.5925	0.0176	7.865	1.0146	0.07336	1.61E-04
290	1.9199	3.95E-04	998.76	2461	4.1876	1.9023	0.001084	9.45E-06	0.5942	0.0177	7.6399	1.0143	0.07321	1.72E-04
291	2.0453	3.95E-04	998.58	2458.6	4.1868	1.9034	0.0010567	9.48E-06	0.5959	0.0178	7.4246	1.0139	0.07306	1.83E-04
292	2.1777	3.94E-04	998.39	2456.3	4.186	1.9045	0.0010305	9.51E-06	0.5976	0.0179	7.2231	1.0136	0.07291	1.94E-04
293	2.3176	3.94E-04	998.19	2453.9	4.1852	1.9056	0.0010053	9.54E-06	0.5992	0.0179	7.0215	1.0132	0.07276	2.05E-04
294	2.4653	3.94E-04	997.98	2451.5	4.1845	1.9067	9.81E-04	9.57E-06	0.6009	0.018	6.8326	1.0129	0.07261	2.15E-04
295	2.6211	3.94E-04	997.76	2449.2	4.1839	1.9079	9.58E-04	9.60E-06	0.6025	0.0181	6.6515	1.0127	0.07246	2.26E-04
296	2.7855	3.93E-04	997.53	2446.8	4.1833	1.909	9.35E-04	9.63E-06	0.6041	0.0182	6.4779	1.0124	0.0723	2.36E-04
297	2.9588	3.93E-04	997.29	2444.4	4.1828	1.9102	9.14E-04	9.66E-06	0.6057	0.0182	6.3112	1.0121	0.07215	2.46E-04
298	3.1415	3.93E-04	997.04	2442.1	4.1823	1.9114	8.93E-04	9.70E-06	0.6072	0.0183	6.1511	1.0119	0.072	2.55E-04
299	3.334	3.92E-04	996.78	2439.7	4.1818	1.9127	8.73E-04	9.73E-06	0.6088	0.0184	5.9973	1.0117	0.07184	2.65E-04
300	3.5366	3.92E-04	996.51	2437.3	4.1814	1.9139	8.54E-04	9.76E-06	0.6103	0.0185	5.8495	1.0115	0.07169	2.74E-04
301	3.7499	3.92E-04	996.24	2434.9	4.181	1.9152	8.35E-04	9.79E-06	0.6118	0.0185	5.7073	1.0113	0.07153	2.83E-04
302	3.9743	3.92E-04	995.95	2432.6	4.1806	1.9165	8.17E-04	9.82E-06	0.6133	0.0186	5.5704	1.0112	0.07137	2.93E-04
303	4.2103	3.91E-04	995.65	2430.2	4.1803	1.9178	8.00E-04	9.86E-06	0.6147	0.0187	5.4387	1.011	0.07122	3.01E-04
304	4.4583	3.91E-04	995.35	2427.8	4.18	1.9192	7.83E-04	9.89E-06	0.6162	0.0188	5.314	1.0109	0.07106	3.10E-04
305	4.719	3.91E-04	995.04	2425.4	4.1798	1.9205	7.67E-04	9.92E-06	0.6176	0.0188	5.1894	1.0108	0.0709	3.19E-04
306	4.9928	3.91E-04	994.72	2423.1	4.1796	1.9219	7.51E-04	9.95E-06	0.619	0.0189	5.0715	1.0107	0.07074	3.27E-04
307	5.2803	3.90E-04	994.39	2420.7	4.1794	1.9233	7.36E-04	9.98E-06	0.6204	0.019	4.9578	1.0106	0.07059	3.36E-04
308	5.5821	3.90E-04	994.05	2418.3	4.1792	1.9247	7.21E-04	1.00E-05	0.6218	0.0191	4.848	1.0105	0.07043	3.44E-04
309	5.8987	3.90E-04	993.7	2415.9	4.1791	1.9261	7.07E-04	1.00E-05	0.6231	0.0192	4.7421	1.0104	0.07027	3.52E-04
310	6.2307	3.89E-04	993.35	2413.5	4.179	1.9276	6.93E-04	1.01E-05	0.6245	0.0192	4.6398	1.0103	0.07011	3.60E-04
311	6.5787	3.89E-04	992.99	2411.1	4.1789	1.929	6.80E-04	1.01E-05	0.6258	0.0193	4.5409	1.0103	0.06994	3.68E-04
312	6.9435	3.89E-04	992.62	2408.8	4.1788	1.9305	6.67E-04	1.01E-05	0.6271	0.0194	4.4454	1.0102	0.06978	3.76E-04
313	7.3256	3.89E-04	992.24	2406.4	4.1788	1.932	6.55E-04	1.02E-05	0.6284	0.0195	4.353	1.0102	0.06962	3.84E-04
314	7.7257	3.88E-04	991.86	2404	4.1788	1.9335	6.42E-04	1.02E-05	0.6296	0.0195	4.2636	1.0102	0.06946	3.91E-04
315	8.1445	3.88E-04	991.46	2401.6	4.1788	1.9351	6.31E-04	1.02E-05	0.6309	0.0196	4.1772	1.0102	0.06929	3.99E-04
316	8.5828	3.88E-04	991.07	2399.2	4.1788	1.9366	6.19E-04	1.03E-05	0.6321	0.0197	4.0935	1.0102	0.06913	4.06E-04
317	9.0412	3.88E-04	990.66	2396.8	4.1789	1.9382	6.08E-04	1.03E-05	0.6333	0.0198	4.0125	1.0102	0.06897	4.14E-04
318	9.5206	3.87E-04	990.25	2394.4	4.179	1.9398	5.97E-04	1.03E-05	0.6345	0.0199	3.934	1.0102	0.0688	4.21E-04
319	10.022	3.87E-04	989.83	2392	4.1791	1.9414	5.87E-04	1.04E-05	0.6357	0.0199	3.858	1.0102	0.06864	4.28E-04
320	10.545	3.87E-04	989.4	2389.6	4.1792	1.943	5.77E-04	1.04E-05	0.6369	0.02	3.7855	1.0102	0.06847	4.35E-04
321	11.092	3.86E-04	988.96	2387.2	4.1794	1.9446	5.67E-04	1.04E-05	0.638	0.0201	3.7129	1.0103	0.0683	4.42E-04
322	11.663	3.86E-04	988.52	2384.7	4.1795	1.9463	5.57E-04	1.05E-05	0.6392	0.0202	3.6447	1.0103	0.06814	4.49E-04
323	12.26	3.86E-04	988.08	2382.3	4.1797	1.948	5.48E-04	1.05E-05	0.6403	0.0203	3.5765	1.0104	0.06797	4.56E-04
324	12.882	3.86E-04	987.62	2379.9	4.1799	1.9497	5.39E-04	1.05E-05	0.6414	0.0203	3.5123	1.0104	0.0678	4.63E-04
325	13.531	3.85E-04	987.16	2377.5	4.1802	1.9514	5.30E-04	1.06E-05	0.6425	0.0204	3.448	1.0105	0.06763	4.70E-04
326	14.208	3.85E-04	986.7	2375.1	4.1804	1.9532	5.21E-04	1.06E-05	0.6436	0.0205	3.3866	1.0105	0.06746	4.77E-04

327	14.913	3.85E-04	986.22	2372.7	4.1807	1.9549	5.13E-04	1.06E-05	0.6446	0.0206	3.327	1.0106	0.06729	4.83E-04
328	15.648	3.85E-04	985.74	2370.2	4.181	1.9567	5.05E-04	1.07E-05	0.6457	0.0207	3.269	1.0107	0.06712	4.90E-04
329	16.415	3.84E-04	985.26	2367.8	4.1814	1.9586	4.97E-04	1.07E-05	0.6467	0.0208	3.2128	1.0108	0.06695	4.97E-04
330	17.212	3.84E-04	984.76	2365.4	4.1817	1.9604	4.89E-04	1.07E-05	0.6477	0.0208	3.1581	1.0109	0.06678	5.03E-04
331	18.043	3.84E-04	984.27	2362.9	4.1821	1.9623	4.82E-04	1.08E-05	0.6487	0.0209	3.1049	1.011	0.06661	5.09E-04
332	18.908	3.84E-04	983.76	2360.5	4.1825	1.9642	4.74E-04	1.08E-05	0.6497	0.021	3.0532	1.0111	0.06644	5.16E-04
333	19.808	3.83E-04	983.25	2358.1	4.1829	1.9661	4.67E-04	1.08E-05	0.6506	0.0211	3.0029	1.0113	0.06626	5.22E-04
334	20.744	3.83E-04	982.74	2355.6	4.1833	1.968	4.60E-04	1.09E-05	0.6516	0.0212	2.954	1.0114	0.06609	5.29E-04
335	21.717	3.83E-04	982.21	2353.2	4.1838	1.97	4.53E-04	1.09E-05	0.6525	0.0213	2.9064	1.0115	0.06592	5.35E-04
336	22.729	3.83E-04	981.69	2350.7	4.1842	1.972	4.47E-04	1.10E-05	0.6534	0.0213	2.8601	1.0117	0.06574	5.41E-04
337	23.781	3.82E-04	981.15	2348.3	4.1847	1.974	4.40E-04	1.10E-05	0.6543	0.0214	2.815	1.0118	0.06557	5.47E-04
338	24.874	3.82E-04	980.61	2345.8	4.1853	1.9761	4.34E-04	1.10E-05	0.6552	0.0215	2.7711	1.012	0.06539	5.53E-04
339	26.009	3.82E-04	980.07	2343.3	4.1858	1.9782	4.28E-04	1.11E-05	0.6561	0.0216	2.7284	1.0122	0.06522	5.59E-04
340	27.187	3.82E-04	979.52	2340.9	4.1864	1.9803	4.22E-04	1.11E-05	0.657	0.0217	2.6867	1.0123	0.06504	5.65E-04
341	28.411	3.81E-04	978.96	2338.4	4.1869	1.9825	4.16E-04	1.11E-05	0.6578	0.0218	2.6462	1.0125	0.06486	5.71E-04
342	29.681	3.81E-04	978.4	2335.9	4.1875	1.9847	4.10E-04	1.12E-05	0.6586	0.0219	2.6067	1.0127	0.06469	5.77E-04
343	30.999	3.81E-04	977.83	2333.5	4.1882	1.9869	4.04E-04	1.12E-05	0.6594	0.0219	2.5681	1.0129	0.06451	5.83E-04
344	32.366	3.81E-04	977.26	2331	4.1888	1.9892	3.99E-04	1.12E-05	0.6603	0.022	2.5306	1.0131	0.06433	5.89E-04
345	33.783	3.80E-04	976.68	2328.5	4.1895	1.9915	3.94E-04	1.13E-05	0.661	0.0221	2.494	1.0134	0.06415	5.95E-04
346	35.253	3.80E-04	976.1	2326	4.1901	1.9939	3.88E-04	1.13E-05	0.6618	0.0222	2.4583	1.0136	0.06397	6.01E-04
347	36.776	3.80E-04	975.51	2323.5	4.1908	1.9962	3.83E-04	1.13E-05	0.6626	0.0223	2.4235	1.0138	0.06379	6.07E-04
348	38.354	3.80E-04	974.92	2321	4.1916	1.9987	3.78E-04	1.14E-05	0.6633	0.0224	2.3895	1.0141	0.06361	6.12E-04
349	39.989	3.79E-04	974.32	2318.5	4.1923	2.0012	3.73E-04	1.14E-05	0.6641	0.0225	2.3564	1.0144	0.06343	6.18E-04
350	41.682	3.79E-04	973.72	2316	4.1931	2.0037	3.68E-04	1.14E-05	0.6648	0.0226	2.3241	1.0146	0.06325	6.24E-04
351	43.435	3.79E-04	973.11	2313.5	4.1939	2.0062	3.64E-04	1.15E-05	0.6655	0.0227	2.2925	1.0149	0.06307	6.30E-04
352	45.25	3.79E-04	972.49	2311	4.1947	2.0089	3.59E-04	1.15E-05	0.6662	0.0228	2.2617	1.0152	0.06288	6.35E-04
353	47.127	3.78E-04	971.87	2308.4	4.1955	2.0115	3.55E-04	1.15E-05	0.6669	0.0228	2.2316	1.0155	0.0627	6.41E-04
354	49.07	3.78E-04	971.25	2305.9	4.1964	2.0142	3.50E-04	1.16E-05	0.6675	0.0229	2.2023	1.0159	0.06252	6.46E-04
355	51.08	3.78E-04	970.62	2303.4	4.1972	2.017	3.46E-04	1.16E-05	0.6682	0.023	2.1736	1.0162	0.06233	6.52E-04
356	53.158	3.78E-04	969.98	2300.9	4.1981	2.0198	3.42E-04	1.16E-05	0.6688	0.0231	2.1456	1.0166	0.06215	6.58E-04
357	55.307	3.77E-04	969.34	2298.3	4.199	2.0227	3.38E-04	1.17E-05	0.6694	0.0232	2.1182	1.0169	0.06196	6.63E-04
358	57.528	3.77E-04	968.7	2295.8	4.2	2.0256	3.34E-04	1.17E-05	0.6701	0.0233	2.0915	1.0173	0.06178	6.69E-04
359	59.823	3.77E-04	968.05	2293.2	4.2009	2.0286	3.30E-04	1.17E-05	0.6707	0.0234	2.0653	1.0177	0.06159	6.74E-04
360	62.194	3.77E-04	967.4	2290.7	4.2019	2.0316	3.26E-04	1.18E-05	0.6712	0.0235	2.0398	1.0181	0.06141	6.80E-04
361	64.643	3.77E-04	966.74	2288.1	4.2029	2.0347	3.22E-04	1.18E-05	0.6718	0.0236	2.0148	1.0185	0.06122	6.85E-04
362	67.172	3.76E-04	966.07	2285.5	4.2039	2.0378	3.18E-04	1.18E-05	0.6724	0.0237	1.9904	1.0189	0.06103	6.91E-04
363	69.783	3.76E-04	965.41	2282.9	4.2049	2.0411	3.15E-04	1.19E-05	0.6729	0.0238	1.9666	1.0194	0.06084	6.96E-04
364	72.479	3.76E-04	964.73	2280.4	4.206	2.0443	3.11E-04	1.19E-05	0.6735	0.0239	1.9432	1.0199	0.06066	7.02E-04
365	75.26	3.76E-04	964.05	2277.8	4.2071	2.0477	3.08E-04	1.19E-05	0.674	0.024	1.9204	1.0203	0.06047	7.07E-04
366	78.13	3.75E-04	963.37	2275.2	4.2082	2.0511	3.04E-04	1.20E-05	0.6745	0.0241	1.8981	1.0208	0.06028	7.13E-04
367	81.09	3.75E-04	962.68	2272.6	4.2093	2.0546	3.01E-04	1.20E-05	0.675	0.0242	1.8763	1.0214	0.06009	7.18E-04
368	84.143	3.75E-04	961.99	2270	4.2104	2.0581	2.98E-04	1.21E-05	0.6755	0.0243	1.8549	1.0219	0.0599	7.23E-04
369	87.291	3.75E-04	961.29	2267.4	4.2116	2.0617	2.94E-04	1.21E-05	0.6759	0.0244	1.834	1.0224	0.05971	7.29E-04
370	90.536	3.74E-04	960.59	2264.8	4.2128	2.0654	2.91E-04	1.21E-05	0.6764	0.0245	1.8135	1.023	0.05952	7.34E-04
371	93.88	3.74E-04	959.89	2262.1	4.214	2.0692	2.88E-04	1.22E-05	0.6768	0.0246	1.7935	1.0236	0.05933	7.39E-04
372	97.326	3.74E-04	959.18	2259.5	4.2152	2.073	2.85E-04	1.22E-05	0.6773	0.0247	1.7739	1.0242	0.05913	7.45E-04
373	100.88	3.74E-04	958.46	2256.9	4.2165	2.0769	2.82E-04	1.22E-05	0.6777	0.0248	1.7547	1.0248	0.05894	7.50E-04
374	104.53	3.74E-04	957.74	2254.2	4.2177	2.0809	2.79E-04	1.23E-05	0.6781	0.0249	1.7359	1.0255	0.05875	7.56E-04
375	108.3	3.73E-04	957.02	2251.6	4.219	2.0849	2.76E-04	1.23E-05	0.6785	0.025	1.7175	1.0262	0.05855	7.61E-04
376	112.18	3.73E-04	956.29	2248.9	4.2203	2.0891	2.73E-04	1.23E-05	0.6789	0.0251	1.6995	1.0269	0.05836	7.66E-04
377	116.17	3.73E-04	955.56	2246.3	4.2217	2.0933	2.71E-04	1.24E-05	0.6793	0.0252	1.6819	1.0276	0.05817	7.72E-04
378	120.28	3.73E-04	954.82	2243.6	4.223	2.0976	2.68E-04	1.24E-05	0.6796	0.0253	1.6646	1.0283	0.05797	7.77E-04
379	124.5	3.72E-04	954.08	2240.9	4.2244	2.102	2.65E-04	1.24E-05	0.68	0.0254	1.6477	1.0291	0.05778	7.82E-04
380	128.85	3.72E-04	953.33	2238.2	4.2258	2.1065	2.63E-04	1.25E-05	0.6803	0.0255	1.6311	1.0298	0.05758	7.88E-04
381	133.32	3.72E-04	952.58	2235.5	4.2272	2.1111	2.60E-04	1.25E-05	0.6806	0.0256	1.6148	1.0306	0.05738	7.93E-04
382	137.92	3.72E-04	951.82	2232.8	4.2287	2.1157	2.57E-04	1.25E-05	0.6809	0.0257	1.5989	1.0315	0.05719	7.98E-04
383	142.66	3.72E-04	951.06	2230.1	4.2301	2.1205	2.55E-04	1.26E-05	0.6812	0.0258	1.5833	1.0323	0.05699	8.04E-04
384	147.52	3.71E-04	950.3	2227.4	4.2316	2.1253	2.53E-04	1.26E-05	0.6815	0.0259	1.568	1.0332	0.05679	8.09E-04
385	152.52	3.71E-04	949.53	2224.7	4.2331	2.1302	2.50E-04	1.26E-05	0.6818	0.026	1.553	1.0341	0.0566	8.14E-04
386	157.65	3.71E-04	948.76	2221.9	4.2347	2.1353	2.48E-04	1.27E-05	0.6821	0.0262	1.5383	1.035	0.0564	8.20E-04
387	162.93	3.71E-04	947.98	2219.2	4.2362	2.1404	2.45E-04	1.27E-05	0.6823	0.0263	1.5239	1.0359	0.0562	8.25E-04
388	168.35	3.71E-04	947.2	2216.4	4.2378	2.1456	2.43E-04	1.27E-05	0.6826	0.0264	1.5098	1.0369	0.056	8.30E-04

389	173.92	3.70E-04	946.41	2213.7	4.2394	2.1509	2.41E-04	1.28E-05	0.6828	0.0265	1.496	1.0379	0.0558	8.36E-04
390	179.64	3.70E-04	945.62	2210.9	4.2411	2.1563	2.39E-04	1.28E-05	0.683	0.0266	1.4824	1.0389	0.0556	8.41E-04
391	185.51	3.70E-04	944.83	2208.1	4.2427	2.1618	2.37E-04	1.29E-05	0.6832	0.0267	1.4691	1.0399	0.0554	8.46E-04
392	191.54	3.70E-04	944.03	2205.4	4.2444	2.1674	2.34E-04	1.29E-05	0.6834	0.0268	1.456	1.041	0.0552	8.52E-04
393	197.72	3.70E-04	943.23	2202.6	4.2461	2.1731	2.32E-04	1.29E-05	0.6836	0.0269	1.4432	1.0421	0.055	8.57E-04
394	204.07	3.69E-04	942.42	2199.8	4.2478	2.1789	2.30E-04	1.30E-05	0.6838	0.0271	1.4307	1.0432	0.0548	8.63E-04
395	210.59	3.69E-04	941.61	2197	4.2496	2.1849	2.28E-04	1.30E-05	0.6839	0.0272	1.4183	1.0444	0.05459	8.68E-04
396	217.27	3.69E-04	940.79	2194.1	4.2514	2.1909	2.26E-04	1.30E-05	0.6841	0.0273	1.4062	1.0456	0.05439	8.73E-04
397	224.12	3.69E-04	939.97	2191.3	4.2532	2.197	2.24E-04	1.31E-05	0.6842	0.0274	1.3944	1.0468	0.05419	8.79E-04
398	231.15	3.69E-04	939.15	2188.5	4.255	2.2032	2.22E-04	1.31E-05	0.6843	0.0275	1.3827	1.048	0.05399	8.84E-04
399	238.36	3.68E-04	938.32	2185.6	4.2569	2.2096	2.20E-04	1.31E-05	0.6844	0.0276	1.3713	1.0492	0.05378	8.90E-04
400	245.75	3.68E-04	937.48	2182.8	4.2587	2.216	2.19E-04	1.32E-05	0.6845	0.0278	1.3601	1.0505	0.05358	8.95E-04
401	253.33	3.68E-04	936.65	2179.9	4.2607	2.2226	2.17E-04	1.32E-05	0.6846	0.0279	1.3491	1.0518	0.05337	9.01E-04
402	261.09	3.68E-04	935.8	2177	4.2626	2.2292	2.15E-04	1.32E-05	0.6847	0.028	1.3383	1.0531	0.05317	9.06E-04
403	269.05	3.68E-04	934.96	2174.1	4.2646	2.236	2.13E-04	1.33E-05	0.6848	0.0281	1.3277	1.0545	0.05296	9.12E-04
404	277.2	3.67E-04	934.11	2171.2	4.2665	2.2429	2.11E-04	1.33E-05	0.6848	0.0283	1.3173	1.0559	0.05276	9.17E-04
405	285.55	3.67E-04	933.25	2168.3	4.2686	2.2499	2.10E-04	1.33E-05	0.6849	0.0284	1.3071	1.0573	0.05255	9.23E-04
406	294.11	3.67E-04	932.4	2165.4	4.2706	2.257	2.08E-04	1.34E-05	0.6849	0.0285	1.2971	1.0587	0.05234	9.28E-04
407	302.87	3.67E-04	931.53	2162.5	4.2727	2.2642	2.06E-04	1.34E-05	0.685	0.0286	1.2873	1.0602	0.05214	9.34E-04
408	311.84	3.67E-04	930.67	2159.5	4.2748	2.2715	2.05E-04	1.34E-05	0.685	0.0288	1.2776	1.0617	0.05193	9.39E-04
409	321.02	3.67E-04	929.79	2156.6	4.2769	2.2789	2.03E-04	1.35E-05	0.685	0.0289	1.2682	1.0632	0.05172	9.45E-04
410	330.42	3.66E-04	928.92	2153.6	4.2791	2.2865	2.02E-04	1.35E-05	0.685	0.029	1.2589	1.0647	0.05151	9.50E-04
411	340.05	3.66E-04	928.04	2150.7	4.2812	2.2941	2.00E-04	1.35E-05	0.6849	0.0291	1.2497	1.0663	0.05131	9.56E-04
412	349.89	3.66E-04	927.15	2147.7	4.2835	2.3019	1.98E-04	1.36E-05	0.6849	0.0293	1.2407	1.0678	0.0511	9.62E-04
413	359.97	3.66E-04	926.27	2144.7	4.2857	2.3097	1.97E-04	1.36E-05	0.6849	0.0294	1.2319	1.0694	0.05089	9.67E-04
414	370.28	3.66E-04	925.37	2141.7	4.288	2.3177	1.95E-04	1.36E-05	0.6848	0.0295	1.2233	1.0711	0.05068	9.73E-04
415	380.82	3.66E-04	924.48	2138.7	4.2903	2.3258	1.94E-04	1.37E-05	0.6848	0.0297	1.2148	1.0727	0.05047	9.79E-04
416	391.61	3.65E-04	923.57	2135.6	4.2926	2.334	1.92E-04	1.37E-05	0.6847	0.0298	1.2064	1.0744	0.05026	9.85E-04
417	402.64	3.65E-04	922.67	2132.6	4.295	2.3423	1.91E-04	1.38E-05	0.6846	0.0299	1.1982	1.0761	0.05005	9.90E-04
418	413.92	3.65E-04	921.76	2129.6	4.2974	2.3507	1.90E-04	1.38E-05	0.6845	0.0301	1.1902	1.0779	0.04983	9.96E-04
419	425.45	3.65E-04	920.85	2126.5	4.2998	2.3593	1.88E-04	1.38E-05	0.6844	0.0302	1.1823	1.0796	0.04962	0.001002
420	437.24	3.65E-04	919.93	2123.4	4.3023	2.3679	1.87E-04	1.39E-05	0.6843	0.0303	1.1745	1.0813	0.04941	0.001008
421	449.29	3.65E-04	919	2120.3	4.3048	2.3767	1.85E-04	1.39E-05	0.6842	0.0305	1.1669	0	0.0492	0.001014
422	461.61	3.64E-04	918.08	2117.2	4.3073	2.3855	1.84E-04	1.39E-05	0.684	0.0306	1.1594	1.0849	0.04899	0.00102
423	474.19	3.64E-04	917.15	2114.1	4.3099	2.3945	1.83E-04	1.40E-05	0.6839	0.0308	1.1521	1.0867	0.04877	0.001026
424	487.05	3.64E-04	916.21	2111	4.3125	2.4036	1.82E-04	1.40E-05	0.6837	0.0309	1.1448	1.0886	0.04856	0.001031
425	500.18	3.64E-04	915.27	2107.9	4.3151	2.4128	1.80E-04	1.40E-05	0.6836	0.031	1.1377	1.0904	0.04835	0.001037
426	513.6	3.64E-04	914.33	2104.7	4.3178	2.4221	1.79E-04	1.41E-05	0.6834	0.0312	1.1308	1.0923	0.04813	0.001043
427	527.3	3.64E-04	913.38	2101.6	4.3205	2.4315	1.78E-04	1.41E-05	0.6832	0.0313	1.1239	1.0942	0.04792	0.00105
428	541.3	3.63E-04	912.43	2098.4	4.3232	2.441	1.76E-04	1.41E-05	0.683	0.0315	1.1172	1.0961	0.0477	0.001056
429	555.59	3.63E-04	911.47	2095.2	4.326	2.4507	1.75E-04	1.42E-05	0.6828	0.0316	1.1106	1.0981	0.04749	0.001062
430	570.18	3.63E-04	910.51	2092	4.3288	2.4604	1.74E-04	1.42E-05	0.6826	0.0318	1.1041	1.1	0.04727	0.001068
431	585.07	3.63E-04	909.54	2088.8	4.3317	2.4702	1.73E-04	1.42E-05	0.6823	0.0319	1.0977	1.102	0.04706	0.001074
432	600.27	3.63E-04	908.57	2085.6	4.3346	2.4802	1.72E-04	1.43E-05	0.6821	0.032	1.0914	1.104	0.04684	0.00108
433	615.78	3.63E-04	907.6	2082.3	4.3375	2.4903	1.71E-04	1.43E-05	0.6818	0.0322	1.0853	1.106	0.04662	0.001087
434	631.62	3.63E-04	906.62	2079.1	4.3405	2.5004	1.69E-04	1.43E-05	0.6816	0.0323	1.0792	1.108	0.04641	0.001093
435	647.77	3.62E-04	905.63	2075.8	4.3435	2.5107	1.68E-04	1.44E-05	0.6813	0.0325	1.0733	1.11	0.04619	0.001099
436	664.25	3.62E-04	904.65	2072.5	4.3465	2.5211	1.67E-04	1.44E-05	0.681	0.0326	1.0674	1.1121	0.04597	0.001106
437	681.07	3.62E-04	903.65	2069.3	4.3496	2.5316	1.66E-04	1.44E-05	0.6807	0.0328	1.0617	1.1141	0.04575	0.001112
438	698.22	3.62E-04	902.66	2065.9	4.3527	2.5422	1.65E-04	1.45E-05	0.6804	0.033	1.0561	1.1162	0.04554	0.001118
439	715.71	3.62E-04	901.66	2062.6	4.3559	2.5529	1.64E-04	1.45E-05	0.6801	0.0331	1.0506	1.1183	0.04532	0.001125
440	733.55	3.62E-04	900.65	2059.3	4.3591	2.5637	1.63E-04	1.45E-05	0.6798	0.0333	1.0451	1.1204	0.0451	0.001131
441	751.74	3.62E-04	899.64	2055.9	4.3624	2.5746	1.62E-04	1.46E-05	0.6795	0.0334	1.0398	1.1225	0.04488	0.001138
442	770.28	3.61E-04	898.63	2052.6	4.3657	2.5857	1.61E-04	1.46E-05	0.6791	0.0336	1.0345	1.1249	0.04466	0.001145
443	789.19	3.61E-04	897.61	2049.2	4.369	2.5968	1.60E-04	1.46E-05	0.6788	0.0337	1.0294	1.1268	0.04444	0.001151
444	808.46	3.61E-04	896.58	2045.8	4.3724	2.6081	1.59E-04	1.47E-05	0.6784	0.0339	1.0243	1.129	0.04422	0.001158
445	828.1	3.61E-04	895.56	2042.4	4.3758	2.6195	1.58E-04	1.47E-05	0.678	0.0341	1.0194	1.1312	0.044	0.001165
446	848.12	3.61E-04	894.52	2039	4.3793	2.6309	1.57E-04	1.47E-05	0.6776	0.0342	1.0145	1.1334	0.04378	0.001172
447	868.51	3.61E-04	893.49	2035.5	4.3828	2.6425	1.56E-04	1.48E-05	0.6772	0.0344	1.0097	1.1356	0.04356	0.001178
448	889.3	3.61E-04	892.44	2032.1	4.3864	2.6543	1.55E-04	1.48E-05	0.6768	0.0345	1.005	1.1378	0.04334	0.001185
449	910.47	3.61E-04	891.4	2028.6	4.39	2.6661	1.54E-04	1.48E-05	0.6764	0.0347	1.0004	1.14	0.04311	0.001192
450	932.04	3.60E-04	890.35	2025.1	4.3937	2.678	1.53E-04	1.49E-05	0.676	0.0349	0.9958	1.1423	0.04289	0.001199

451	954.01	3.60E-04	889.29	2021.6	4.3974	2.6901	1.52E-04	1.49E-05	0.6756	0.035	0.9914	1.1445	0.04267	0.001206
452	976.39	3.60E-04	888.23	2018.1	4.4012	2.7023	1.51E-04	1.49E-05	0.6751	0.0352	0.987	1.1468	0.04245	0.001213
453	999.18	3.60E-04	887.17	2014.6	4.405	2.7145	1.51E-04	1.50E-05	0.6747	0.0354	0.9827	0	0.04222	0.001221
454	1022.4	3.60E-04	886.1	2011	4.4088	2.727	1.50E-04	1.50E-05	0.6742	0.0356	0.9785	1.1514	0.042	0.001228
455	1046	3.60E-04	885.02	2007.4	4.4128	2.7395	1.49E-04	1.50E-05	0.6737	0.0357	0.9744	1.1537	0.04178	0.001235
456	1070.1	3.60E-04	883.94	2003.9	4.4167	2.7521	1.48E-04	1.51E-05	0.6732	0.0359	0.9703	1.1561	0.04155	0.001242
457	1094.6	3.60E-04	882.86	2000.3	4.4208	2.7649	1.47E-04	1.51E-05	0.6727	0.0361	0.9664	1.1584	0.04133	0.00125
458	1119.5	3.60E-04	881.77	1996.6	4.4249	2.7778	1.46E-04	1.52E-05	0.6722	0.0363	0.9624	1.1608	0.0411	0.001257
459	1144.9	3.59E-04	880.68	1993	4.429	2.7909	1.45E-04	1.52E-05	0.6717	0.0364	0.9586	1.1632	0.04088	0.001265
460	1170.7	3.59E-04	879.58	1989.4	4.4332	2.804	1.45E-04	1.52E-05	0.6712	0.0366	0.9549	1.1656	0.04066	0.001272
461	1197	3.59E-04	878.47	1985.7	4.4374	2.8173	1.44E-04	1.53E-05	0.6707	0.0368	0.9512	1.168	0.04043	0.00128
462	1223.7	3.59E-04	877.36	1982	4.4418	2.8308	1.43E-04	1.53E-05	0.6701	0.037	0.9475	1.1704	0.0402	0.001288
463	1250.9	3.59E-04	876.25	1978.3	4.4461	2.8443	1.42E-04	1.53E-05	0.6696	0.0372	0.944	1.1729	0.03998	0.001296
464	1278.6	3.59E-04	875.13	1974.6	4.4505	2.858	1.41E-04	1.54E-05	0.669	0.0373	0.9405	1.1754	0.03975	0.001303
465	1306.7	3.59E-04	874.01	1970.8	4.455	2.8718	1.41E-04	1.54E-05	0.6684	0.0375	0.9371	1.1778	0.03953	0.001311
466	1335.4	3.59E-04	872.88	1967.1	4.4596	2.8858	1.40E-04	1.54E-05	0.6678	0.0377	0.9338	1.1804	0.0393	0.001319
467	1364.5	3.59E-04	871.75	1963.3	4.4642	2.8999	1.39E-04	1.55E-05	0.6673	0.0379	0.9305	1.1829	0.03907	0.001327
468	1394.1	3.59E-04	870.61	1959.5	4.4689	2.9142	1.38E-04	1.55E-05	0.6666	0.0381	0.9273	1.1854	0.03885	0.001336
469	1424.2	3.58E-04	869.47	1955.7	4.4736	2.9286	1.38E-04	1.55E-05	0.666	0.0383	0.9241	1.188	0.03862	0.001344
470	1454.8	3.58E-04	868.32	1951.9	4.4784	2.9432	1.37E-04	1.56E-05	0.6654	0.0385	0.921	1.1906	0.03839	0.001352
471	1486	3.58E-04	867.17	1948	4.4833	2.9579	1.36E-04	1.56E-05	0.6648	0.0387	0.918	1.1932	0.03816	0.001361
472	1517.6	3.58E-04	866.01	1944.1	4.4882	2.9727	1.35E-04	1.56E-05	0.6641	0.0388	0.9151	1.1958	0.03794	0.001369
473	1549.8	3.58E-04	864.84	1940.3	4.4933	2.9878	1.35E-04	1.57E-05	0.6635	0.039	0.9122	1.1985	0.03771	0.001378
474	1582.5	3.58E-04	863.67	1936.3	4.4983	3.0029	1.34E-04	1.57E-05	0.6628	0.0392	0.9093	1.2012	0.03748	0.001386
475	1615.7	3.58E-04	862.5	1932.4	4.5035	3.0183	1.33E-04	1.57E-05	0.6621	0.0394	0.9066	1.2039	0.03725	0.001395
476	1649.5	3.58E-04	861.32	1928.5	4.5087	3.0338	1.33E-04	1.58E-05	0.6615	0.0396	0.9039	1.2066	0.03702	0.001404
477	1683.9	3.58E-04	860.13	1924.5	4.514	3.0494	1.32E-04	1.58E-05	0.6608	0.0398	0.9012	1.2093	0.03679	0.001413
478	1718.7	3.58E-04	858.94	1920.5	4.5194	3.0653	1.31E-04	1.58E-05	0.6601	0.04	0.8986	1.2125	0.03656	0.001421
479	1754.2	3.58E-04	857.75	1916.5	4.5248	3.0813	1.31E-04	1.59E-05	0.6593	0.0402	0.8961	1.2149	0.03634	0.001431
480	1790.2	3.58E-04	856.55	1912.5	4.5303	3.0975	1.30E-04	1.59E-05	0.6586	0.0404	0.8936	1.2177	0.03611	0.00144
481	1826.8	3.58E-04	855.34	1908.4	4.5359	3.1139	1.29E-04	1.59E-05	0.6579	0.0407	0.8912	1.2206	0.03588	0.001449
482	1863.9	3.58E-04	854.13	1904.3	4.5416	3.1304	1.29E-04	1.60E-05	0.6571	0.0409	0.8888	1.2238	0.03565	0.001458
483	1901.7	3.57E-04	852.91	1900.2	4.5474	3.1471	1.28E-04	1.60E-05	0.6564	0.0411	0.8865	1.2263	0.03542	0.001468
484	1940	3.57E-04	851.69	1896.1	4.5532	3.164	1.27E-04	1.60E-05	0.6556	0.0413	0.8843	1.2293	0.03518	0.001477
485	1978.9	3.57E-04	850.46	1892	4.5591	3.1811	1.27E-04	1.61E-05	0.6549	0.0415	0.8821	1.2322	0.03495	0.001487
486	2018.5	3.57E-04	849.22	1887.8	4.5651	3.1984	1.26E-04	1.61E-05	0.6541	0.0417	0.8799	1.2352	0.03472	0.001497
487	2058.6	3.57E-04	847.98	1883.6	4.5712	3.2159	1.25E-04	1.61E-05	0.6533	0.0419	0.8778	1.2382	0.03449	0.001507
488	2099.4	3.57E-04	846.74	1879.4	4.5774	3.2336	1.25E-04	1.62E-05	0.6525	0.0421	0.8758	1.2413	0.03426	0.001517
489	2140.8	3.57E-04	845.48	1875.2	4.5837	3.2515	1.24E-04	1.62E-05	0.6517	0.0424	0.8738	1.2443	0.03403	0.001527
490	2182.8	3.57E-04	844.23	1871	4.5901	3.2696	1.24E-04	1.62E-05	0.6508	0.0426	0.8719	1.2475	0.0338	0.001537
491	2225.4	3.57E-04	842.96	1866.7	4.5965	3.2879	1.23E-04	1.63E-05	0.65	0.0428	0.87	1.2506	0.03357	0.001547
492	2268.7	3.57E-04	841.69	1862.4	4.6031	3.3064	1.22E-04	1.63E-05	0.6492	0.043	0.8682	1.2538	0.03333	0.001558
493	2312.6	3.57E-04	840.42	1858.1	4.6097	3.3252	1.22E-04	1.63E-05	0.6483	0.0432	0.8664	1.257	0.0331	0.001568
494	2357.2	3.57E-04	839.14	1853.7	4.6164	3.3441	1.21E-04	1.64E-05	0.6474	0.0435	0.8647	1.2602	0.03287	0.001579
495	2402.5	3.57E-04	837.85	1849.3	4.6233	3.3633	1.21E-04	1.64E-05	0.6466	0.0437	0.8631	1.2635	0.03264	0.00159
496	2448.4	3.57E-04	836.55	1845	4.6302	3.3827	1.20E-04	1.65E-05	0.6457	0.0439	0.8615	1.2668	0.0324	0.0016
497	2495	3.57E-04	835.25	1840.5	4.6373	3.4023	1.20E-04	1.65E-05	0.6448	0.0442	0.8599	1.2701	0.03217	0.001612
498	2542.3	3.57E-04	833.95	1836.1	4.6444	3.4222	1.19E-04	1.65E-05	0.6439	0.0444	0.8584	1.2735	0.03194	0.001623
499	2590.2	3.57E-04	832.64	1831.6	4.6517	3.4423	1.18E-04	1.66E-05	0.643	0.0446	0.8569	1.2769	0.03171	0.001634
500	2638.9	3.57E-04	831.32	1827.1	4.659	3.4626	1.18E-04	1.66E-05	0.6421	0.0449	0.8555	1.2804	0.03147	0.001646
501	2688.3	3.57E-04	829.99	1822.6	4.6665	3.4832	1.17E-04	1.66E-05	0.6411	0.0451	0.8542	1.2838	0.03124	0.001657
502	2738.3	3.57E-04	828.66	1818.1	4.6741	3.504	1.17E-04	1.67E-05	0.6402	0.0454	0.8529	1.2874	0.03101	0.001669
503	2789.1	3.57E-04	827.32	1813.5	4.6818	3.5251	1.16E-04	1.67E-05	0.6392	0.0456	0.8516	1.2909	0.03077	0.001681
504	2840.6	3.57E-04	825.98	1808.9	4.6896	3.5465	1.16E-04	1.67E-05	0.6383	0.0458	0.8504	1.2945	0.03054	0.001693
505	2892.9	3.57E-04	824.63	1804.3	4.6975	3.5681	1.15E-04	1.68E-05	0.6373	0.0461	0.8492	1.2982	0.0303	0.001705
506	2945.8	3.57E-04	823.27	1799.6	4.7056	3.5899	1.15E-04	1.68E-05	0.6363	0.0463	0.8481	1.3018	0.03007	0.001717
507	2999.5	3.57E-04	821.91	1794.9	4.7137	3.6121	1.14E-04	1.68E-05	0.6353	0.0466	0.8471	1.3056	0.02984	0.00173
508	3054	3.57E-04	820.54	1790.2	4.722	3.6345	1.14E-04	1.69E-05	0.6343	0.0468	0.8461	1.3093	0.0296	0.001743
509	3109.2	3.57E-04	819.16	1785.5	4.7305	3.6572	1.13E-04	1.69E-05	0.6333	0.0471	0.8451	1.3131	0.02937	0.001756
510	3165.2	3.57E-04	817.77	1780.7	4.739	3.6802	1.13E-04	1.69E-05	0.6323	0.0474	0.8442	1.317	0.02913	0.001769
511	3222	3.57E-04	816.38	1775.9	4.7477	3.7035	1.12E-04	1.70E-05	0.6312	0.0476	0.8434	1.3209	0.0289	0.001782
512	3279.5	3.57E-04	814.98	1771.1	4.7565	3.727	1.12E-04	1.70E-05	0.6302	0.0479	0.8425	1.3248	0.02866	0.001795

513	3337.8	3.57E-04	813.58	1766.3	4.7655	3.7509	1.11E-04	1.71E-05	0.6291	0.0481	0.8418	1.3288	0.02843	0.001809
514	3397	3.57E-04	812.16	1761.4	4.7746	3.7751	1.11E-04	1.71E-05	0.6281	0.0484	0.8411	1.3328	0.02819	0.001823
515	3456.9	3.57E-04	810.74	1756.5	4.7838	3.7996	1.10E-04	1.71E-05	0.627	0.0487	0.8404	1.3369	0.02796	0.001837
516	3517.6	3.57E-04	809.32	1751.5	4.7932	3.8244	1.10E-04	1.72E-05	0.6259	0.049	0.8398	1.3409	0.02772	0.001851
517	3579.1	3.57E-04	807.88	1746.6	4.8028	3.8496	1.09E-04	1.72E-05	0.6248	0.0492	0.8393	1.3452	0.02749	0.001865
518	3641.5	3.57E-04	806.44	1741.6	4.8125	3.8751	1.09E-04	1.72E-05	0.6237	0.0495	0.8387	1.3494	0.02725	0.00188
519	3704.7	3.57E-04	804.99	1736.5	4.8223	3.9009	1.08E-04	1.73E-05	0.6226	0.0498	0.8383	1.3537	0.02702	0.001894
520	3768.7	3.57E-04	803.53	1731.5	4.8323	3.9271	1.08E-04	1.73E-05	0.6214	0.0501	0.8379	1.358	0.02678	0.001909
521	3833.6	3.57E-04	802.07	1726.4	4.8425	3.9536	1.07E-04	1.73E-05	0.6203	0.0503	0.8375	1.3624	0.02655	0.001925
522	3899.3	3.57E-04	800.59	1721.3	4.8529	3.9805	1.07E-04	1.74E-05	0.6191	0.0506	0.8372	1.3668	0.02631	0.00194
523	3965.9	3.57E-04	799.11	1716.1	4.8634	4.0078	1.06E-04	1.74E-05	0.618	0.0509	0.837	1.3711	0.02608	0.001956
524	4033.3	3.57E-04	797.62	1710.9	4.8741	4.0354	1.06E-04	1.75E-05	0.6168	0.0512	0.8368	1.3759	0.02584	0.001972
525	4101.6	3.57E-04	796.13	1705.7	4.8849	4.0635	1.05E-04	1.75E-05	0.6156	0.0515	0.8366	1.3805	0.02561	0.001988
526	4170.9	3.57E-04	794.62	1700.4	4.896	4.0919	1.05E-04	1.75E-05	0.6144	0.0518	0.8365	1.3852	0.02537	0.002004
527	4241	3.57E-04	793.11	1695.1	4.9072	4.1208	1.05E-04	1.76E-05	0.6132	0.0521	0.8365	1.3899	0.02514	0.002021
528	4311.9	3.57E-04	791.59	1689.8	4.9187	4.1501	1.04E-04	1.76E-05	0.612	0.0524	0.8365	1.3947	0.0249	0.002038
529	4383.8	3.57E-04	790.06	1684.4	4.9303	4.1798	1.04E-04	1.76E-05	0.6108	0.0527	0.8365	1.3996	0.02467	0.002055
530	4456.7	3.57E-04	788.52	1679.1	4.9421	4.2099	1.03E-04	1.77E-05	0.6096	0.053	0.8366	1.4045	0.02443	0.002073
531	4530.4	3.57E-04	786.97	1673.6	4.9541	4.2406	1.03E-04	1.77E-05	0.6083	0.0533	0.8368	1.4095	0.0242	0.002091
532	4605.1	3.57E-04	785.42	1668.2	4.9664	4.2717	1.02E-04	1.78E-05	0.607	0.0536	0.837	1.4146	0.02396	0.002109
533	4680.6	3.58E-04	783.86	1662.6	4.9788	4.3032	1.02E-04	1.78E-05	0.6058	0.054	0.8373	1.4198	0.02372	0.002127
534	4757.2	3.58E-04	782.28	1657.1	4.9915	4.3353	1.01E-04	1.78E-05	0.6045	0.0543	0.8377	1.425	0.02349	0.002146
535	4834.7	3.58E-04	780.7	1651.5	5.0044	4.3679	1.01E-04	1.79E-05	0.6032	0.0546	0.838	1.4303	0.02325	0.002165
536	4913.1	3.58E-04	779.11	1645.9	5.0176	4.401	1.01E-04	1.79E-05	0.6019	0.0549	0.8385	1.4357	0.02302	0.002184
537	4992.6	3.58E-04	777.51	1640.3	5.0309	4.4347	1.00E-04	1.80E-05	0.6006	0.0553	0.839	1.4412	0.02278	0.002204
538	5073	3.58E-04	775.9	1634.6	5.0445	4.4689	9.97E-05	1.80E-05	0.5993	0.0556	0.8396	1.4468	0.02255	0.002224
539	5154.4	3.58E-04	774.28	1628.8	5.0584	4.5037	9.93E-05	1.80E-05	0.5979	0.0559	0.8402	1.4525	0.02231	0.002245
540	5236.7	3.58E-04	772.65	1623.1	5.0725	4.5391	9.89E-05	1.81E-05	0.5966	0.0563	0.8409	1.4583	0.02208	0.002266
541	5320.1	3.58E-04	771.01	1617.2	5.0869	4.5751	9.85E-05	1.81E-05	0.5952	0.0566	0.8416	1.4641	0.02184	0.002287
542	5404.5	3.58E-04	769.36	1611.4	5.1016	4.6118	9.81E-05	1.82E-05	0.5938	0.057	0.8424	1.4701	0.02161	0.002308
543	5489.9	3.58E-04	767.71	1605.5	5.1165	4.6491	9.76E-05	1.82E-05	0.5925	0.0573	0.8433	1.4762	0.02137	0.00233
544	5576.4	3.59E-04	766.04	1599.5	5.1317	4.6871	9.72E-05	1.82E-05	0.5911	0.0577	0.8442	1.4824	0.02114	0.002353
545	5663.9	3.59E-04	764.36	1593.6	5.1473	4.7257	9.68E-05	1.83E-05	0.5896	0.058	0.8452	1.4887	0.0209	0.002376
546	5752.4	3.59E-04	762.67	1587.5	5.1631	4.7651	9.64E-05	1.83E-05	0.5882	0.0584	0.8463	1.4951	0.02067	0.002399
547	5841.9	3.59E-04	760.97	1581.5	5.1792	4.8053	9.60E-05	1.84E-05	0.5868	0.0588	0.8474	1.5017	0.02043	0.002423
548	5932.6	3.59E-04	759.26	1575.4	5.1957	4.8462	9.56E-05	1.84E-05	0.5854	0.0591	0.8486	1.5084	0.0202	0.002447
549	6024.3	3.59E-04	757.54	1569.2	5.2125	4.8879	9.52E-05	1.85E-05	0.5839	0.0595	0.8499	1.5152	0.01996	0.002472
550	6117.1	3.59E-04	755.81	1563	5.2296	4.9304	9.48E-05	1.85E-05	0.5824	0.0599	0.8512	1.5221	0.01973	0.002497
551	6210.9	3.59E-04	754.06	1556.8	5.2471	4.9738	9.44E-05	1.85E-05	0.581	0.0603	0.8526	1.5292	0.0195	0.002523
552	6305.9	3.60E-04	752.31	1550.5	5.2649	5.0181	9.40E-05	1.86E-05	0.5795	0.0607	0.8541	1.5365	0.01926	0.002549
553	6402	3.60E-04	750.54	1544.1	5.2831	5.0632	9.36E-05	1.86E-05	0.578	0.0611	0.8557	1.5438	0.01903	0.002576
554	6499.1	3.60E-04	748.76	1537.7	5.3017	5.1093	9.32E-05	1.87E-05	0.5764	0.0615	0.8573	1.5502	0.01879	0.002604
555	6597.4	3.60E-04	746.97	1531.3	5.3207	5.1564	9.28E-05	1.87E-05	0.5749	0.0619	0.859	1.5591	0.01856	0.002632
556	6696.9	3.60E-04	745.17	1524.8	5.3401	5.2045	9.24E-05	1.88E-05	0.5734	0.0623	0.8608	1.5657	0.01833	0.00266
557	6797.5	3.60E-04	743.35	1518.2	5.3599	5.2536	9.20E-05	1.88E-05	0.5718	0.0627	0.8627	1.575	0.01809	0.00269
558	6899.2	3.61E-04	741.53	1511.6	5.3801	5.3038	9.16E-05	1.88E-05	0.5703	0.0631	0.8646	1.5833	0.01786	0.00272
559	7002.1	3.61E-04	739.69	1505	5.4008	5.3551	9.13E-05	1.89E-05	0.5687	0.0636	0.8667	1.5917	0.01763	0.00275
560	7106.1	3.61E-04	737.83	1498.3	5.422	5.4075	9.09E-05	1.89E-05	0.5671	0.064	0.8688	1.6003	0.0174	0.002782
561	7211.4	3.61E-04	735.97	1491.5	5.4436	5.4611	9.05E-05	1.90E-05	0.5655	0.0644	0.871	1.6091	0.01716	0.002814
562	7317.8	3.61E-04	734.09	1484.7	5.4657	5.5159	9.01E-05	1.90E-05	0.5639	0.0649	0.8734	1.6164	0.01693	0.002847
563	7425.4	3.61E-04	732.19	1477.9	5.4884	5.572	8.97E-05	1.91E-05	0.5622	0.0653	0.8758	1.6266	0.0167	0.002881
564	7534.2	3.62E-04	730.29	1471	5.5115	5.6294	8.93E-05	1.91E-05	0.5606	0.0658	0.8783	1.6367	0.01647	0.002915
565	7644.3	3.62E-04	728.36	1464	5.5352	5.6882	8.89E-05	1.92E-05	0.5589	0.0662	0.8809	1.6463	0.01624	0.00295
566	7755.5	3.62E-04	726.43	1456.9	5.5595	5.7483	8.86E-05	1.92E-05	0.5573	0.0667	0.8836	1.6562	0.016	0.002987
567	7868	3.62E-04	724.48	1449.8	5.5844	5.8098	8.82E-05	1.93E-05	0.5556	0.0672	0.8864	1.6663	0.01577	0.003024
568	7981.8	3.62E-04	722.51	1442.7	5.6099	5.8729	8.78E-05	1.93E-05	0.5539	0.0677	0.8893	1.6766	0.01554	0.003062
569	8096.8	3.63E-04	720.53	1435.5	5.636	5.9375	8.74E-05	1.94E-05	0.5522	0.0682	0.8923	1.6872	0.01531	0.003101
570	8213.1	3.63E-04	718.53	1428.2	5.6627	6.0037	8.70E-05	1.94E-05	0.5504	0.0687	0.8955	1.698	0.01508	0.003141
571	8330.6	3.63E-04	716.52	1420.8	5.6901	6.0715	8.67E-05	1.95E-05	0.5487	0.0692	0.8988	1.7091	0.01485	0.003182
572	8449.4	3.63E-04	714.49	1413.4	5.7183	6.141	8.63E-05	1.95E-05	0.547	0.0697	0.9021	1.7205	0.01462	0.003224
573	8569.6	3.64E-04	712.45	1405.9	5.7471	6.2122	8.59E-05	1.96E-05	0.5452	0.0702	0.9057	1.7322	0.01439	0.003268
574	8691	3.64E-04	710.38	1398.4	5.7767	6.2853	8.55E-05	1.96E-05	0.5434	0.0707	0.9093	1.7441	0.01417	0.003312

575	8813.8	3.64E-04	708.3	1390.8	5.8071	6.3603	8.52E-05	1.97E-05	0.5416	0.0713	0.9131	1.7563	0.01394	0.003358
576	8937.9	3.64E-04	706.21	1383.1	5.8383	6.4372	8.48E-05	1.97E-05	0.5398	0.0718	0.917	1.7689	0.01371	0.003405
577	9063.3	3.65E-04	704.09	1375.3	5.8704	6.5162	8.44E-05	1.98E-05	0.538	0.0724	0.9211	1.7817	0.01348	0.003454
578	9190.1	3.65E-04	701.96	1367.5	5.9033	6.5972	8.40E-05	1.98E-05	0.5361	0.0729	0.9253	1.7949	0.01325	0.003504
579	9318.2	3.65E-04	699.81	1359.6	5.9372	6.6805	8.37E-05	1.99E-05	0.5343	0.0735	0.9296	1.8084	0.01303	0.003555
580	9447.7	3.66E-04	697.63	1351.6	5.972	6.766	8.33E-05	2.00E-05	0.5324	0.0741	0.9341	1.8223	0.0128	0.003608
581	9578.6	3.66E-04	695.44	1343.5	6.0078	6.8539	8.29E-05	2.00E-05	0.5305	0.0747	0.9388	1.8365	0.01257	0.003662
582	9710.9	3.66E-04	693.23	1335.4	6.0446	6.9442	8.25E-05	2.01E-05	0.5286	0.0753	0.9437	1.8511	0.01235	0.003718
583	9844.6	3.67E-04	691	1327.2	6.0825	7.0371	8.21E-05	2.01E-05	0.5267	0.0759	0.9487	1.8661	0.01212	0.003776
584	9979.7	3.67E-04	688.75	1318.9	6.1216	7.1327	8.18E-05	2.02E-05	0.5247	0.0765	0.9539	1.8815	0.0119	0.003836
585	10116	3.67E-04	686.48	1310.5	6.1618	7.2311	8.14E-05	2.02E-05	0.5228	0.0772	0.9594	1.8973	0.01167	0.003897
586	10254	3.68E-04	684.18	1302	6.2032	7.3325	8.10E-05	2.03E-05	0.5208	0.0778	0.965	1.9136	0.01145	0.003961
587	10394	3.68E-04	681.86	1293.4	6.246	7.4369	8.06E-05	2.04E-05	0.5188	0.0785	0.9708	1.9304	0.01123	0.004026
588	10535	3.68E-04	679.52	1284.8	6.2901	7.5446	8.03E-05	2.04E-05	0.5168	0.0791	0.9768	1.9476	0.011	0.004094
589	10677	3.69E-04	677.16	1276	6.3356	7.6557	7.99E-05	2.05E-05	0.5148	0.0798	0.9831	1.9654	0.01078	0.004164
590	10821	3.69E-04	674.77	1267.1	6.3826	7.7703	7.95E-05	2.06E-05	0.5128	0.0805	0.9896	1.9837	0.01056	0.004237
591	10966	3.69E-04	672.36	1258.2	6.4312	7.8888	7.91E-05	2.06E-05	0.5107	0.0813	0.9964	2.0026	0.01034	0.004312
592	11113	3.70E-04	669.92	1249.2	6.4815	8.0113	7.87E-05	2.07E-05	0.5087	0.082	1.0034	2.0221	0.01012	0.00439
593	11261	3.70E-04	667.46	1240	6.5335	8.1381	7.84E-05	2.08E-05	0.5066	0.0827	1.0107	2.0422	0.0099	0.004471
594	11411	3.71E-04	664.96	1230.7	6.5873	8.2694	7.80E-05	2.08E-05	0.5045	0.0835	1.0183	2.0631	0.00968	0.004554
595	11563	3.71E-04	662.45	1221.4	6.6431	8.4054	7.76E-05	2.09E-05	0.5024	0.0843	1.0262	2.0798	0.00946	0.004641
596	11716	3.72E-04	659.9	1211.9	6.701	8.5466	7.72E-05	2.10E-05	0.5002	0.0851	1.0344	2.1071	0.00924	0.004732
597	11871	3.72E-04	657.32	1202.3	6.7611	8.6931	7.68E-05	2.10E-05	0.4981	0.0859	1.0429	2.1303	0.00902	0.004826
598	12027	3.72E-04	654.72	1192.6	6.8235	8.8455	7.64E-05	2.11E-05	0.4959	0.0867	1.0519	2.1544	0.00881	0.004924
599	12185	3.73E-04	652.08	1182.7	6.8885	9.004	7.61E-05	2.12E-05	0.4937	0.0876	1.0612	2.1738	0.00859	0.005026
600	12344	3.73E-04	649.41	1172.8	6.9561	9.1692	7.57E-05	2.13E-05	0.4915	0.0885	1.0709	2.2055	0.00838	0.005133
601	12505	3.74E-04	646.71	1162.7	7.0266	9.3414	7.53E-05	2.14E-05	0.4893	0.0893	1.0811	2.2327	0.00816	0.005245
602	12668	3.74E-04	643.97	1152.4	7.1001	9.5212	7.49E-05	2.14E-05	0.487	0.0903	1.0917	2.2611	0.00795	0.005362
603	12833	3.75E-04	641.2	1142.1	7.177	9.7091	7.45E-05	2.15E-05	0.4847	0.0912	1.1029	2.2907	0.00773	0.005485
604	12999	3.76E-04	638.39	1131.6	7.2574	9.9058	7.41E-05	2.16E-05	0.4824	0.0922	1.1146	2.3217	0.00752	0.005614
605	13167	3.76E-04	635.54	1120.9	7.3416	10.112	7.37E-05	2.17E-05	0.4801	0.0932	1.1268	2.3542	0.00731	0.00575
606	13336	3.77E-04	632.66	1110.1	7.4299	10.328	7.33E-05	2.18E-05	0.4778	0.0942	1.1397	2.3882	0.0071	0.005892
607	13508	3.77E-04	629.73	1099.1	7.5227	10.555	7.29E-05	2.19E-05	0.4754	0.0952	1.1533	2.4239	0.00689	0.006043
608	13681	3.78E-04	626.75	1087.9	7.6203	10.793	7.25E-05	2.20E-05	0.473	0.0963	1.1676	2.4614	0.00668	0.006202
609	13855	3.79E-04	623.74	1076.6	7.7232	11.044	7.21E-05	2.21E-05	0.4706	0.0974	1.1827	2.5008	0.00648	0.00637
610	14032	3.79E-04	620.67	1065.1	7.8317	11.309	7.17E-05	2.22E-05	0.4682	0.0985	1.1986	2.5422	0.00627	0.006548
611	14210	3.80E-04	617.56	1053.4	7.9464	11.587	7.12E-05	2.23E-05	0.4658	0.0997	1.2155	2.5859	0.00606	0.006737
612	14391	3.81E-04	614.39	1041.5	8.0679	11.882	7.08E-05	2.24E-05	0.4633	0.1009	1.2333	2.632	0.00586	0.006938
613	14573	3.81E-04	611.17	1029.5	8.1966	12.193	7.04E-05	2.25E-05	0.4608	0.1022	1.2523	2.6805	0.00566	0.007152
614	14757	3.82E-04	607.89	1017.1	8.3333	12.522	7.00E-05	2.26E-05	0.4582	0.1035	1.2724	2.7318	0.00545	0.00738
615	14942	3.83E-04	604.55	1004.6	8.4786	12.871	6.95E-05	2.27E-05	0.4557	0.1048	1.2938	2.7859	0.00525	0.007623
616	15130	3.84E-04	601.15	991.85	8.6334	13.24	6.91E-05	2.28E-05	0.4531	0.1062	1.3167	2.8431	0.00505	0.007883
617	15320	3.84E-04	597.69	978.85	8.7985	13.632	6.87E-05	2.29E-05	0.4505	0.1076	1.341	2.9036	0.00485	0.008161
618	15511	3.85E-04	594.15	965.58	8.9748	14.047	6.82E-05	2.30E-05	0.4478	0.109	1.367	2.9675	0.00466	0.008458
619	15705	3.86E-04	590.54	952.05	9.1633	14.489	6.78E-05	2.32E-05	0.4451	0.1106	1.3948	3.0351	0.00446	0.008777
620	15900	3.87E-04	586.85	938.24	9.3652	14.958	6.73E-05	2.33E-05	0.4424	0.1122	1.4245	3.1067	0.00427	0.009118
621	16098	3.88E-04	583.08	924.13	9.5815	15.457	6.68E-05	2.34E-05	0.4396	0.1138	1.4564	3.1824	0.00407	0.009484
622	16297	3.89E-04	579.23	909.72	9.8137	15.988	6.64E-05	2.36E-05	0.4368	0.1155	1.4906	3.2625	0.00388	0.009877
623	16499	3.90E-04	575.29	894.97	10.063	16.554	6.59E-05	2.37E-05	0.434	0.1173	1.5272	3.3472	0.00369	0.010299
624	16702	3.91E-04	571.22	879.87	10.363	17.254	6.54E-05	2.39E-05	0.4311	0.1191	1.5715	3.4569	0.00351	0.010814
625	16908	3.92E-04	567.06	864.38	10.672	17.971	6.49E-05	2.40E-05	0.4282	0.1211	1.6168	3.5663	0.00332	0.011345
626	17116	3.93E-04	562.79	848.47	11.009	18.754	6.44E-05	2.42E-05	0.4253	0.1231	1.6664	3.6857	0.00313	0.01193
627	17326	3.94E-04	558.39	832.13	11.381	19.614	6.38E-05	2.44E-05	0.4222	0.1252	1.7208	3.8166	0.00295	0.012577
628	17538	3.96E-04	553.85	815.31	11.792	20.563	6.33E-05	2.45E-05	0.4192	0.1274	1.781	3.9606	0.00277	0.013298
629	17752	3.97E-04	549.17	797.98	12.25	21.615	6.28E-05	2.47E-05	0.4161	0.1298	1.8479	4.1198	0.00259	0.014105
630	17969	3.98E-04	544.33	780.09	12.763	22.789	6.22E-05	2.49E-05	0.4129	0.1322	1.9227	4.297	0.00242	0.015014
631	18188	4.00E-04	539.31	761.6	13.34	24.106	6.16E-05	2.51E-05	0.4096	0.1348	2.0069	4.4954	0.00224	0.016046
632	18409	4.01E-04	534.1	742.44	13.998	25.595	6.10E-05	2.54E-05	0.4063	0.1376	2.1024	4.7192	0.00207	0.017225
633	18633	4.03E-04	528.67	722.55	14.751	27.294	6.04E-05	2.56E-05	0.4029	0.1405	2.2118	4.9735	0.0019	0.018586
634	18859	4.05E-04	523.01	701.83	15.625	29.25	5.98E-05	2.59E-05	0.3994	0.1436	2.3383	5.2656	0.00174	0.020173
635	19087	4.06E-04	517.07	680.2	16.65	31.527	5.91E-05	2.61E-05	0.3958	0.147	2.4863	5.6044	0.00157	0.022046
636	19318	4.08E-04	510.83	657.52	17.87	34.212	5.84E-05	2.64E-05	0.3921	0.1505	2.6621	6.0028	0.00141	0.024288

637	19551	4.10E-04	504.23	633.63	19.345	37.427	5.77E-05	2.67E-05	0.3883	0.1544	2.8741	6.4781	0.00126	0.027015
638	19787	4.13E-04	497.22	608.34	21.165	41.349	5.69E-05	2.71E-05	0.3843	0.1586	3.135	7.0559	0.0011	0.0304
639	20025	4.15E-04	489.71	581.4	23.467	46.242	5.61E-05	2.74E-05	0.3801	0.1632	3.4639	7.7738	9.54E-04	0.034705
640	20266	4.18E-04	481.61	552.43	26.468	52.521	5.52E-05	2.79E-05	0.3757	0.1683	3.8913	8.6912	8.09E-04	0.04035
641	20510	4.21E-04	472.76	520.96	30.538	60.876	5.43E-05	2.83E-05	0.371	0.1741	4.469	9.9064	6.69E-04	0.048048
642	20756	4.24E-04	462.95	486.27	36.357	72.548	5.33E-05	2.89E-05	0.3659	0.1806	5.2917	11.595	5.35E-04	0.059115
643	21006	4.28E-04	451.84	447.2	45.32	90.007	5.21E-05	2.95E-05	0.3602	0.1882	6.5535	14.107	4.07E-04	0.076249
644	21258	4.33E-04	438.82	401.77	60.776	118.98	5.07E-05	3.03E-05	0.3537	0.1973	8.7188	18.249	2.86E-04	0.105929
645	21514	4.39E-04	422.7	345.92	93.093	176.45	4.91E-05	3.13E-05	0.3456	0.209	13.221	26.398	1.76E-04	0.168199
646	21774	4.47E-04	400.26	268.67	195.3	344.02	4.68E-05	3.28E-05	0.3343	0.226	27.356	49.915	7.79E-05	0.365237
647	22038	4.68E-04	349.56	93.648	3291.9	4401.3	4.19E-05	3.68E-05	0.3056	0.2664	451.22	608.73	3.66E-06	6.237477

```
/*
      a[0]=P;
      a[1]=T;
      a[2]=v;
      a[3]=h;
      a[4]=u;
      a[5]=s;
      a[6]=x;
      a[7]=Cp;
      a[8]=Cv;
      a[9]=alpha_Tx(T,x);
      a[10]=kappa_Tx(T,x);
      a[11]=viscosity_Tx(T,x);
      a[12]=k_Tx(T,x);
      a[13]=sigma(T);
      a[14]=Pr_Tx(T,x);
      a[15]=1.0/a[2];
      a[16]=sound_Tx(T,x); */
public class water_test1
{public static void main(String arg[])
 { steamIAPWS_IF97 st=new steamIAPWS_IF97();
   double T=273.15;
   double a1[]=st.property("tx",T,0.0);
   double a2[]=st.property("tx",T,1.0);
    double P=a1[0];
    double rof=1.0/a1[2];
    double rog=1.0/a2[3];
    double hf=a1[3];
    double hg=a2[3];
    double hfg=hg-hf;
    double Cpl=a1[7];
     double Cpg=a2[7];
    double etaf=a1[11];
    double etag=a2[11];
     double kf=a1[12];
    double kg=a2[12];
    double Prf=a1[14];
    double Prg=a2[14];
    double sigma=a1[13];
    double beta=a1[9];
    System.out.println(T+" "+P+" "+rog+" "+rof+" "+hfg+" "+Cpl+" "+Cpg+" "+etaf+" "+etag+" "+kf+" "+kg+" "+Prf+" "+Prg+"
"+sigma+" "+beta);
   for(T=275;T<647.3;T+=1)
   {
a1=st.property("tx",T,0.0);
a2=st.property("tx",T,1.0);
     P=a1[0];
      rof=1.0/a1[2];
    rog=1.0/a2[3];
      hf=a1[3];
      hg=a2[3];
      hfg=hg-hf;
      Cpl=a1[7];
       Cpg=a2[7];
      etaf=a1[11];
      etag=a2[11];
       kf=a1[12];
      kg=a2[12];
      Prf=a1[14];
      Prg=a2[14];
      sigma=a1[13];
      beta=a1[9];
```

```
    System.out.println(T+" "+P+" "+rog+" "+rof+" "+hfg+" "+Cpl+" "+Cpg+" "+etaf+" "+etag+" "+kf+" "+kg+" "+Prf+" "+Prg+"
"+sigma+" "+beta);
    }
  }
}
```

THERMODYNAMIC PROPERTIES OF SECONDARY COOLANTS & HEAT TRANSFER FLUIDS

As a practical heat transfer fluid a secondary coolant or heat transfer fluid is utilized. In some of the refrigeration applications, it is required that transfer medium should performed in low temperature ranges (sub zero temperatures). The most common heat transfer fluid is water and air. For low temperature applications most common heat transfer fluids are brines (mixture of sodiumchloride NaCl and water or potasiumchloride and water), ethylene gylicol-water and propylene gylicol-water. Since they are binary mixtures, their properties depends on the mixture rate. In addition to that several synthetic oils are used as heat transfer fluids both on heat applications and cold applications. Curve fitting functions are used to express properties of binary mixtures. For example freezing points of a binary mixture is expressed as:

$$T_f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7 + a_8 x^8 + a_9 x^9 +$$

$$+ \frac{a_{10}}{x+0.01} + \frac{a_{11}}{x^2+0.01} + \frac{a_{12}}{x^3+0.01} + \frac{a_{13}}{x^4+0.01} + a_{15}\log(x+0.01)$$

In such a binary mixture, properties of density will be given with a curve fitting function again as:

$$\rho(T,x) = a_0 + a_1 x + a_2 T + a_3 x^2 + a_4 T^2 + a_5 xT + a_6 T^3 + a_7 xT^2 + a_8 T^4 + a_9 xT^3 +$$

$$+ a_{10} T^5 + a_{11} xT^4 + a_{12} T^6 + a_{13} xT^5 + a_{14} T^7$$

$$C_p(T,x) = a_0 + a_1 x + a_2 T + a_3 x^2 + a_4 T + a_5 xT + a_6 T^3 + a_7 xT^2 + a_8 T^4 + a_9 xT^3 +$$

$$+ a_{10} T^5 + a_{11} xT^4 + a_{12} T^6 + a_{13} xT^5 + a_{14} T^7$$

For the enthalpy and entropy general thermodynamic relation is used

$$h(T,x) = \int_{T_{ref}}^{T} C_p(T,x)dT \text{ where } T_{ref}=0 \text{ degree C}$$

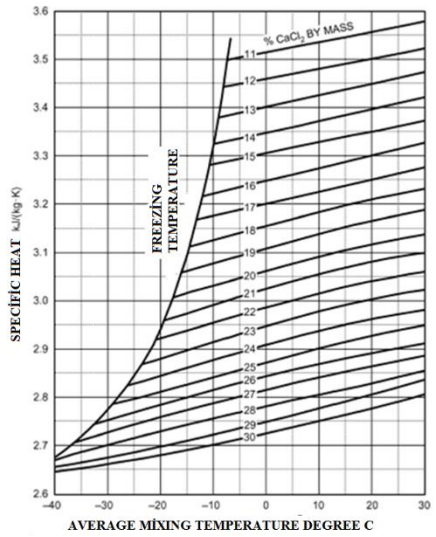
$$s(T,x) = \int_{T_{ref}}^{T} \frac{C_p(T,x)}{T} dT$$

Some of the basic properties for such binary mixtures are given below as table and graphic forms

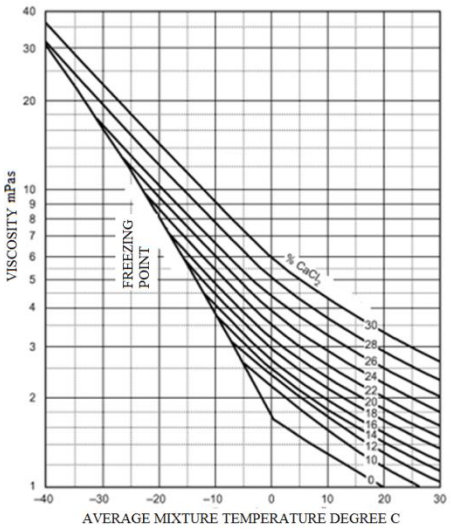
Table 2.14.1 Salt -water mixture

CaCl %	Cp 15 C J/kgK	Freezing temperature degree C	Density at 16 C NaCl	Density at 16 C water	Densities at different temperatures -20 C	-10	0 C	10 C
0	4184	0	0	999			,	
5	3866	-2.4	52.2	1044			1042	1041
6	3824	-2.9	63	1049			1051	1050
7	3757	-3.4	74.2	1059			1060	1059
8	3699	-4.1	85.5	1068			1070	1068
9	3636	-4.7	96.9	1078			1079	1077
10	3577	-5.4	108.6	1087			1088	1086
11	3523	-6.2	120.5	1095			1097	1095
12	3464	-7.1	132.5	1104			1107	1104
13	3414	-8	144.8	1113			1116	1114
14	3364	-9.2	157.1	1123			1126	1123
15	3318	-10.3	169.8	1132		1140	1136	1133
16	3259	-11.6	182.6	1141		1150	1145	1142
17	3209	-13	195.7	1152		1160	1155	1152
18	3163	-14.5	209	1161		1170	1165	1162
19	3121	-16.2	222.7	1171		1179	1175	1172

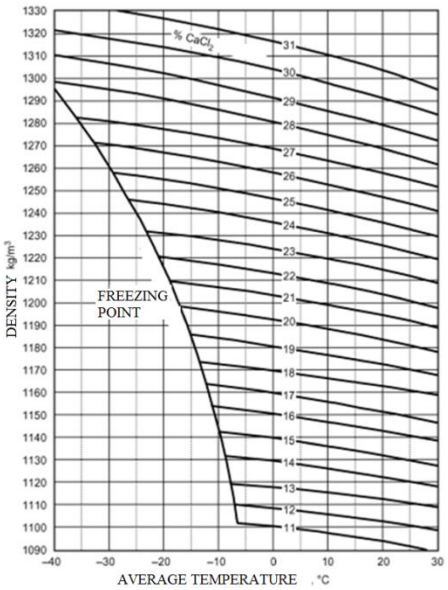
20	3084	-18	236	1180		1189	1185	1182
21	3050	-19.9	249.6	1189				
22	2996	-22.1	264.3	1201	1214	1210	1206	1202
23	2958	-24.4	278.7	1211				
24	2916	-26.8	293.5	1223	1235	1231	1227	1223
25	2882	-29.4	308.2	1232				
26	2853	-32.1	323.1	1242				
27	2816	-35.1	338.5	1253				
28	2782	-38.8	354	1264				
29	2753	-45.2	369.9	1275				
29.87	2741	-55	378.8	1289				
30	2732	-46	358.4	1294				
32	2678	-28.6	418.1	1316				
34	2636	-15.4	452	1339				



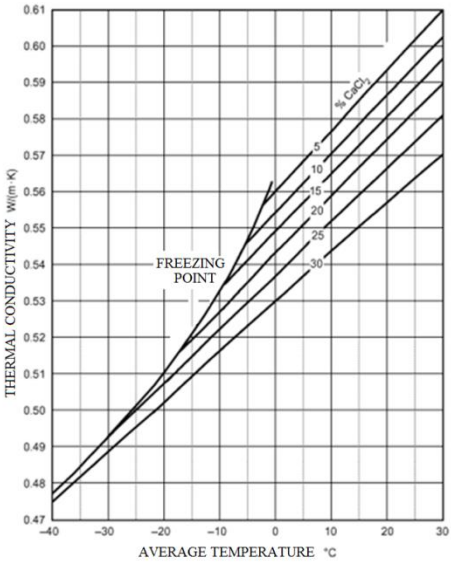
SPECIFIC HEAT OF CALCIUM CHLORIDE



VISCOSITY OF CALCIUM CHLORIDE



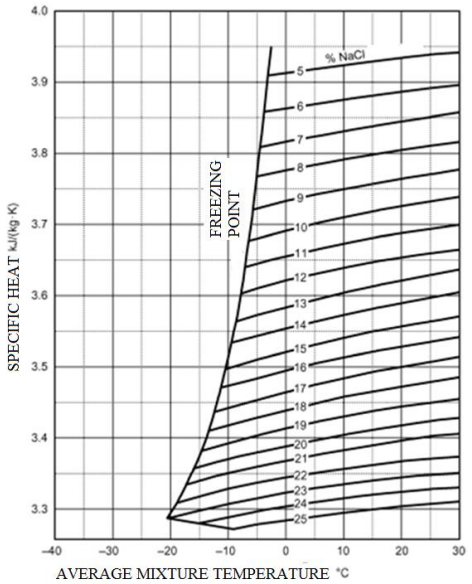
DENSITY OF CALCIUM CHLORIDE



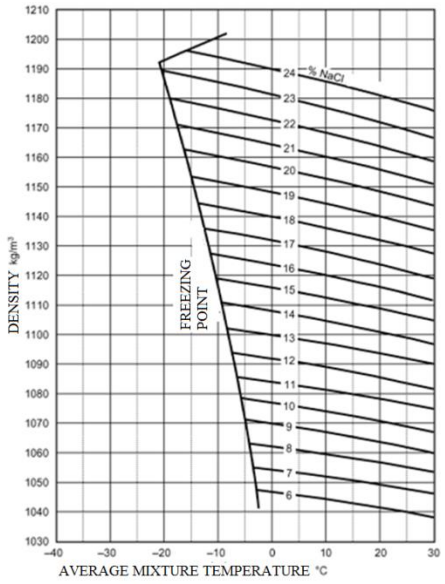
THERMAL CONDUCTIVITY OF CALCIUM CHLORIDE

Table Salt (NaCl) –water mixture

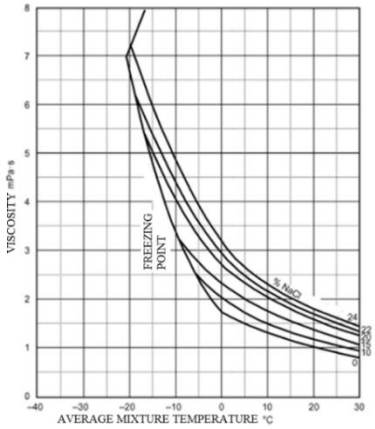
NaCl	Spec. heat 15 C	Freezing point	Density at 16 C		Density in various temperatures			
% by	J/kgK	degree C	NaCl	water	-10 C	0 C	10 C	20 C
0	4184	0	0	1000				
5	3925	-2.9	51.7	1035		1038.1	1036.5	1034
6	3879	-3.6	62.5	1043		1045.8	1043.9	1041.2
7	3836	-4.3	73.4	1049		1053.7	1051.4	1048.5
8	3795	-5	84.6	1057		1061.2	1058.9	1055.8
9	3753	-5.8	95.9	1065		1069	1066.4	1063.2
10	3715	-6.6	107.2	1072		1076.8	1074	1070.6
11	3678	-7.3	118.8	1080		1084.8	1081.6	1078.1
12	3640	-8.2	130.3	1086		1092.4	1089.6	1085.6
13	3607	-9.1	142.2	1094		1100.3	1097	1093.2
14	3573	-10.1	154.3	1102		1108.2	1104.7	1100.8
15	3544	-10.9	166.5	1110	1119.4	1116.2	1112.5	1108.5
16	3515	-11.9	178.9	1118	1127.6	1124.2	1120.4	1116.2
17	3485	-13	191.4	1126	1135.8	1132.2	1128.3	1124
18	3456	-14.1	204.1	1134	1144.1	1140.3	1136.2	1131.8
19	3427	-15.3	217	1142	1153.4	1148.5	1144.3	1139.7
20	3402	-16.5	230	1150	1160.7	1156.7	1154.1	1147.7
21	3376	-17.8	243.2	1158	1169.1	1165	1160.5	1155.8
22	3356	-19.1	256.6	1166	1177.6	1173.3	1168.7	1163.9
23	3330	-20.6	270	1174	1186.1	1181.7	1177	1172
24	3310	-15.7	283.7	1182	1194.7	1190.1	1185.3	1180.3
25	3289	-8.8	297.5	1190				
25.2		0						



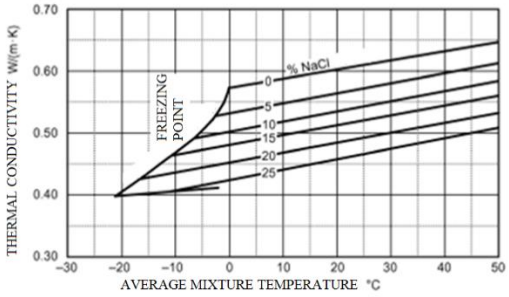
SODIUM CHLORIDE(SALT)-WATER MIXTURE SPECIFIC HEAT



SODIUM CHLORIDE(SALT)-WATER MIXTURE DENSITY



SODIUM CHLORIDE(SALT)-WATER MIXTURE VISCOSITY

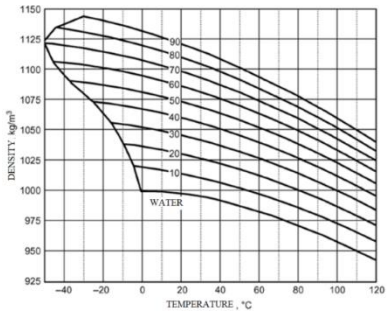


SODIUM CHLORIDE(SALT)-WATER MIXTURE THERMAL CONDUCTIVITY

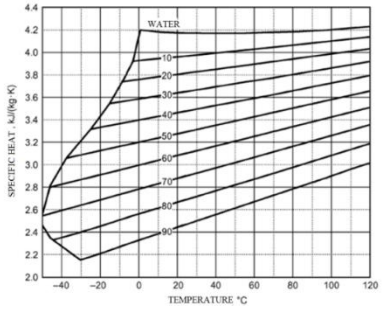
Table 2.14.3 Ethylene Gylicol -water mixture properties

ETHYLENE GYLICOL FREEZING AND BOILING POINTS				ETHYLENE GYLICOL FREEZING AND BOILING POINTS			
% by mass	% by mass	% by mass	% by mass	% by mass	volumetric	T freezing	Tboiling
0	0	0	100	0	0	0	100
5	4.4	-1.4	100.6	5	4.8	-1.6	100
10	8.9	-3.2	101.1	10	9.6	-3.3	100
15	13.6	-5.4	101.7	15	14.5	-5.1	100
20	18.1	-7.8	102.2	20	19.4	-7.1	100.6
21	19.2	-8.4	102.2	21	20.4	-7.6	100.6
22	20.1	-8.9	102.2	22	21.4	-8	100.6
23	21	-9.5	102.8	23	22.4	-8.6	100.6
24	22	-10.2	102.8	24	23.4	-9.1	100.6
25	22.9	-10.7	103.3	25	24.4	-9.6	101.1
26	23.9	-11.4	103.3	26	25.3	-10.2	101.1
27	24.8	-12	103.3	27	26.4	-10.8	101.1
28	25.8	-12.7	103.9	28	27.4	-11.4	101.7
29	26.7	-13.3	103.9	29	28.4	-12	101.7
30	27.7	-14.1	104.4	30	29.4	-12.7	102.2
31	28.7	-14.8	104.4	31	30.4	-13.4	102.2
32	29.6	-15.4	104.4	32	31.4	-14.1	102.2
33	30.6	-16.2	104.4	33	32.4	-14.8	102.2
34	31.6	-17	104.4	34	33.5	-15.6	102.2
35	32.6	-17.9	105	35	34.4	-16.4	102.8
36	33.5	-18.6	105	36	35.5	-17.3	102.8
37	34.5	-19.4	105	37	36.5	-18.2	102.8
38	35.5	-20.3	105	38	37.5	-19.1	103.3
39	36.5	-21.3	105	39	38.5	-20.1	103.3
40	37.5	-22.3	105.6	40	39.6	-21.1	103.9
41	38.5	-23.2	105.6	41	40.6	-22.1	103.9
42	39.5	-24.3	105.6	42	41.6	-23.2	103.9
43	40.5	-25.3	106.1	43	42.6	-24.3	103.9
44	41.5	-26.4	106.1	44	43.7	-25.5	103.9
45	42.5	-27.5	106.7	45	44.7	-26.7	104.4
46	43.5	-28.8	106.7	46	45.7	-27.9	104.4
47	44.5	-29.8	106.7	47	46.8	-29.3	104.4
48	45.5	-31.1	106.7	48	47.8	-30.6	105
49	46.6	-32.6	106.7	49	48.9	-32.1	105
50	47.6	-33.8	107.2	50	49.9	-33.5	105.6
51	48.6	-35.1	107.2	51	50.9	-35	105.6
52	49.6	-36.4	107.2	52	51.9	-36.6	105.6
53	50.6	-37.9	107.8	53	53	-38.2	106.1
54	51.6	-39.3	107.8	54	54	-39.8	106.1
55	52.7	-41.1	108.3	55	55	-41.6	106.1
56	53.7	-42.6	108.3	56	56	-43.3	106.1
57	54.7	-44.2	108.9	57	57	-45.2	106.7
58	55.7	-45.6	108.9	58	58	-47.1	106.7
59	56.8	-47.1	109.4	59	59	-49	106.7
60	57.8	-48.3	110	60	60	-51.1	107.2
65	62.8	*	112.8	65	65	*	108.3
70	68.3	*	116.7	70	70	*	110
75	73.6	*	120	75	75	*	113.9

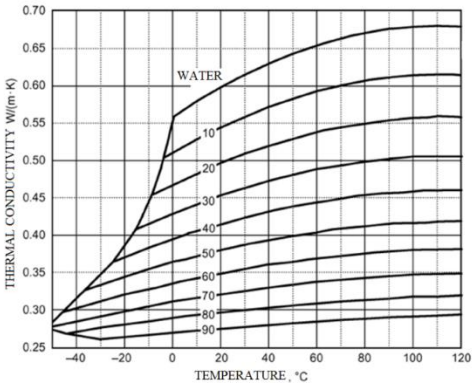
80	78.9	-46.8	123.9				
85	84.3	-36.9	133.9				
90	89.7	-29.8	140.6				
95	95	-19.4	158.3				



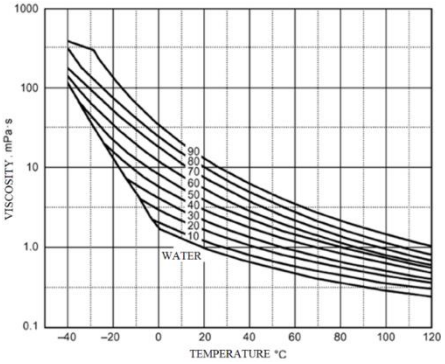
WATER ETHYLENE GYLICOL MIXTURE(% VOLUME) DENSITY



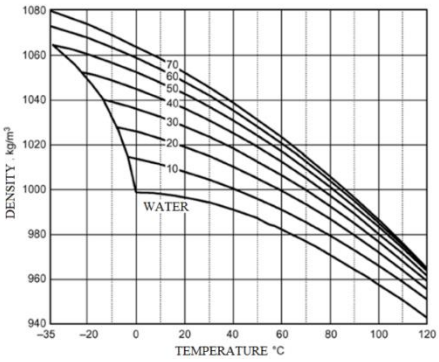
WATER ETHYLENE GYLICOL MIXTURE(% VOLUME)) SPECIFIC HEAT



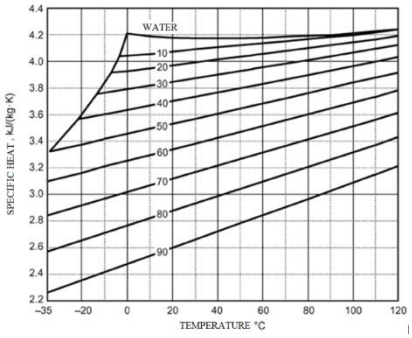
WATER ETHYLENE GYLICOL MIXTURE(% VOLUME) THERMAL CONDUCTIVITY



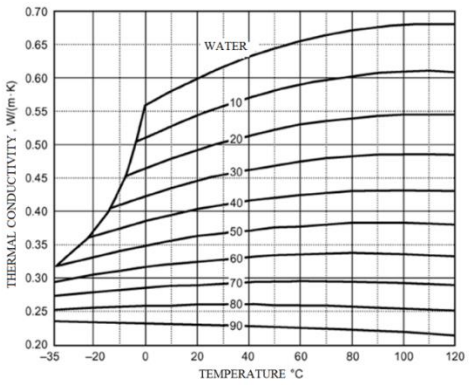
WATER ETHYLENE GYLICOL MIXTURE(% VOLUME) VISCOSITY



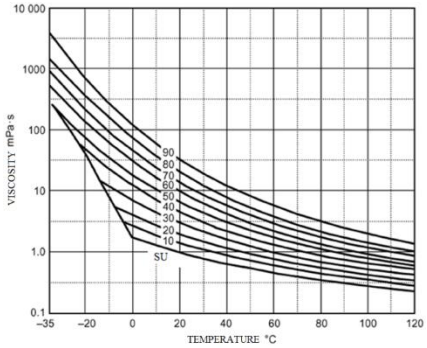
WATER PROPYLENE GYLICOL MIXTURE(% VOLUME) DENSITY



WATER PROPYLENE GYLICOL MIXTURE(% VOLUME) SPECIFIC HEAT



WATER PROPYLENE GYLICOL MIXTURE(% VOLUME) THERMAL CONDUCTIVITY



WATER PROPYLENE GYLICOL MIXTURE(% VOLUME) VISCOSITY

Table polydimetilsiloaksan heat transfer fluid properties

Flame temperature: 46.7 C

Boiling point : 175 C

Freezing point : -111.1 C

Temperature range : -73.3 to 260 degree C

Temperature	Vapor pressure		density	Specific heat	Thermal conductivity
°C	kPa	mPa·s	kg/m3	kJ/(kg·K)	W/(m·K)
-73	0	12.4	924.6	1.41	0.1294
-70	0	11.2	922.1	1.418	0.1288
-60	0	8.26	913.5	1.443	0.1269
-50	0	6.24	905	1.469	0.1251
-40	0	4.83	896.4	1.495	0.1231
-30	0	3.81	887.9	1.52	0.1212
-20	0	3.07	879.3	1.546	0.1192
-10	0.01	2.51	870.7	1.572	0.1171
0	0.03	2.09	862	1.597	0.115
10	0.08	1.76	853.3	1.623	0.1129
20	0.16	1.49	844.5	1.649	0.1108
30	0.32	1.29	835.5	1.674	0.1086
40	0.61	1.12	826.5	1.7	0.1064
50	1.09	0.98	817.3	1.726	0.1042
60	1.85	0.86	807.9	1.751	0.1019
70	3.02	0.77	798.4	1.777	0.0996
80	4.76	0.69	788.7	1.803	0.0973
90	7.25	0.62	778.8	1.828	0.0949
100	10.73	0.56	768.7	1.854	0.0925
110	15.45	0.51	758.3	1.88	0.0901
120	21.75	0.47	747.7	1.905	0.0877
130	29.95	0.43	736.8	1.931	0.0852
140	40.45	0.4	725.6	1.957	0.0827
150	53.67	0.37	714.1	1.982	0.0802
160	70.06	0.34	702.3	2.008	0.0777
170	90.1	0.32	690.2	2.033	0.0751
180	114.29	0.3	677.7	2.059	0.0725
190	143.17	0.28	664.8	2.085	0.0699
200	177.27	0.26	651.6	2.11	0.0673
210	217.14	0.25	638	2.136	0.0646
220	263.36	0.24	623.9	2.162	0.062
230	316.47	0.22	609.5	2.187	0.0593
240	377.03	0.21	594.5	2.213	0.0566
250	445.61	0.2	579.1	2.239	0.0538
260	522.74	0.19	563.3	2.264	0.0511

Properties of d-limonen heat transfer fluid

Flame temperature : 46.1 C

Boiling point : 154.4 C

Freezing point : -96.7 C

Temperature degree C	Specific heat kJ/(kg K)	Viscosity mPas	Density kg/m^3	Thermal conductivity W/(m K)

-73	1.27	3.8	914.3	0.137
-50	1.39	3	897.1	0.133
-25	1.51	2.3	878.3	0.128
0	1.65	1.8	859.2	0.124
25	1.78	1.4	839.8	0.119
50	1.91	1.1	820.1	0.114
75	2.04	0.8	800	0.11
100	2.17	0.7	779.5	0.105
125	2.3	0.5	758.4	0.1
150	2.41	0.4	736.6	0.096

Table 2.14.5 Properties of Duratherm XLT-50 heat transfer oil

Temperature range : -46 to 177 C

Temperature degree C	Density (kg/m3)	Viscosity (Centipoise)	Specific heat (W/m.K)	Thermal (kJ/kgK)	Pressure (kPa)
-46	871	87.42	0.14	1.926	0
-40	868	59.49	0.138	1.926	0
-34	865	32.2	0.138	1.938	0
-29	863	13.91	0.138	1.95	0
-23	860	4.46	0.137	1.962	0
-18	857	3.87	0.137	1.974	0
-12	854	3.41	0.137	1.986	0
-7	852	3.03	0.136	1.998	0
-1	849	2.72	0.136	2.01	0
4	846	2.45	0.135	2.022	0
10	844	2.24	0.135	2.034	0
16	841	2.04	0.135	2.046	0
21	838	1.88	0.134	2.058	0
27	835	1.74	0.134	2.069	0
32	833	1.62	0.134	2.081	0
38	830	1.51	0.133	2.093	0.14
43	827	1.41	0.133	2.11	0.67
49	824	1.33	0.132	2.127	1.21
54	820	1.25	0.132	2.144	1.74
60	817	1.18	0.131	2.16	2.28
66	814	1.12	0.131	2.177	2.41
71	811	1.05	0.13	2.194	3.61
77	808	1	0.13	2.211	4.81
82	805	0.96	0.129	2.227	6.01
88	801	0.91	0.129	2.244	7.21
93	798	0.87	0.128	2.261	8.41
99	795	0.83	0.128	2.278	11.35
104	784	0.79	0.127	2.294	14.29
110	773	0.75	0.127	2.311	17.22
116	761	0.72	0.126	2.328	20.16
121	750	0.68	0.125	2.345	23.1
127	739	0.65	0.125	2.361	26.41
132	728	0.62	0.124	2.378	29.72
138	716	0.59	0.124	2.395	33.03
143	705	0.56	0.123	2.412	36.34
149	694	0.53	0.123	2.428	39.64
154	683	0.51	0.123	2.445	44.48
160	672	0.49	0.119	2.462	49.33
166	661	0.48	0.119	2.474	54.17
171	651	0.45	0.119	2.487	59.01
176	641	0.43	0.116	2.5	63.85

In order to convert the data to computer programs general least square curve fitting programs are used

Program Surface least square curve fitting program

```
import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;

// surface curve fitting
//f(x,y)=a0+a1*x+a2*T+a3*x^2+a4*T^2+a5*xT+a6*T^3+a7*xT^2+a8*T^2
//      +a9*xT^3+a10*T^5+a11*xT^4+a12*T^6+a13xT^5+a14*T^7
abstract class f_xir
{
  abstract double func(double x[],int equation_ref);
}

class fa extends f_xir
{
            double func(double x[],int i)
```

```
                {//a general surface curve fitting model
    double xx=0.0;
    //density function
    if(i==0)       {xx=1.0;          }
    else if (i==1) {xx=x[0];         }
    else if (i==2) {xx=x[1];         }
    else if (i==3) {xx=x[0]*x[0];    }
    else if (i==4) {xx=x[1]*x[1];    }
    else if (i==5) {xx=x[0]*x[1];    }
    else if (i==6) {xx=x[1]*x[1]*x[1];}
    else if (i==7) {xx=x[0]*x[1]*x[1];}
    else if (i==8) {xx=x[1]*x[1]*x[1]*x[1];}
    else if (i==9) {xx=x[0]*x[1]*x[1]*x[1];}
    else if (i==10){xx=x[1]*x[1]*x[1]*x[1]*x[1];}
    else if (i==11){xx=x[0]*x[1]*x[1]*x[1]*x[1];}
    else if (i==12){xx=x[1]*x[1]*x[1]*x[1]*x[1]*x[1];}
    else if (i==13){xx=x[0]*x[1]*x[1]*x[1]*x[1]*x[1];}
    else if (i==14){xx=x[1]*x[1]*x[1]*x[1]*x[1]*x[1]*x[1];}
    return xx;
                }
}

class NA46
{
//  General least square curve fitting with multivariable
                public static double[][] Transpose(double [][] left)
    { //transpose matrix (if A=a(i,j) Transpose(A)=a(j,i)
    int i,j;
    int n=left.length;
    int m=left[0].length;
    double b[][];
    b=new double[m][n];
    for(i=0;i<n;i++)
      {for(j=0;j<m;j++)  {b[j][i]=left[i][j];} }
    return b;
    }

  public static double[] gausswithpartialpivot(double a[][],double b[])
  { //Gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //partial pivoting
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                   if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
                  b[k]=dummy;
                }
                //
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
                }
  }
  //backward substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
```

```
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}


public static double[] GeneralLeastSquare(double c[][],int n)
{
int n1=c.length;
  int n2=c[0].length-1;
  System.out.println("n1="+n1+"n2="+n2);
  double xi[][]=new double[n1][n2];
  double yi[]=new double[n1];
  for(int i=0;i<n1;i++){for(int j=0;j<n2;j++){xi[i][j]=c[i][j];};yi[i]=c[i][n2];}
  return GeneralLeastSquare(xi,yi,n);
}

public static double[] GeneralLeastSquare(double xi[][],double yi[],int n)
{ // n dimensional surface general least square curve fitting
fa f=new fa();
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  B[i]=0;
              for(j=0;j<np1;j++)
    {
              A[i][j]=0.0;
    for(k=0;k<l;k++) A[i][j]+=f.func(xi[k],i)*f.func(xi[k],j);
    }
    for(k=0;k<l;k++) B[i]+= f.func(xi[k],i)*yi[k];
  }
X=gausswithpartialpivot(A,B);
//X=B/A;
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
Text.printT(X);
return X;
}

public static double funcGeneralLeastSquare(double e[],double x[])
{
// multidimensional function calculation
fa f=new fa();
int n=e.length;
double ff=0;
if(n!=0.0)
  {
  for(int i=n-1;i>=0;i--)
   {ff+=e[i]*f.func(x,i);}
  }
return ff;
}

public static double[] funcGeneralLeastSquare(double e[],double xi[][])
{
// multidimensional function calculation
fa f=new fa();
int n=e.length;
double ff[]=new double[xi.length];
for(int k=0;k<xi.length;k++)
{
if(n!=0.0)
  {
```

```
  for(int i=n-1;i>=0;i--)
   {ff[k]+=e[i]*f.func(xi[k],i);}
  }
}
return ff;
}

public static double[][] cikti(double c[][],int polinomkatsayisi,int aradegersayisi)
{ int n1=c.length;
  int n2=c[0].length-1;
  System.out.println("n1="+n1+"n2="+n2);
  double xi[][]=new double[n1][n2];
  double yi[]=new double[n1];
  for(int i=0;i<n1;i++){for(int j=0;j<n2;j++){xi[i][j]=c[i][j];};yi[i]=c[i][n2];
  }
  return cikti(xi,yi,polinomkatsayisi,aradegersayisi);
}


public static double[][] cikti(double xi[][],double yi[],int polinomkatsayisi,int aradegersayisi)
{
                int n=xi.length;
                int nk=xi[0].length;
                int nn=(n-1)*(aradegersayisi+1)+1;
   double E[]=GeneralLeastSquare(xi,yi,polinomkatsayisi);
   double x[][]=new double[nn][nk];
   double c[][]=new double[nn][nk+1];
   double yy[]=new double[nn];
   double dx[]=new double[nk];
   int k=0;
   int i,j,w;
   for(i=0;i<(n-1);i++)
     {   for(w=0;w<nk;w++){x[k][w]=xi[i][w];dx[w]=(xi[i+1][w]-xi[i][w])/((double)aradegersayisi+1.0);};k++;
         for(j=0;j<aradegersayisi;j++)
         { for(w=0;w<nk;w++){x[k][w]=x[k-1][w]+dx[w];};k++;}
     }
   for(w=0;w<nk;w++){x[k][w]=xi[i][w];}
   yy=funcGeneralLeastSquare(E,x);
   for(i=0;i<x.length;i++)
    {for(w=0;w<nk;w++) c[i][w]=x[i][w];
     c[i][nk]=yy[i];
    }
   return c;
}

public static double funcGeneralLeastSquare(double e[],double x,double y)
{
// multidimensionaş surface fitting
double xx[]=new double[2];
xx[0]=x;
xx[1]=y;
return funcGeneralLeastSquare(e,xx);
}

public static double hata(double c[][],double e[])
{int n1=c.length;
  int n2=c[0].length-1;
  System.out.println("n1="+n1+"n2="+n2);
  double xi[][]=new double[n1][n2];
  double yi[]=new double[n1];
  for(int i=0;i<n1;i++){for(int j=0;j<n2;j++){xi[i][j]=c[i][j];};yi[i]=c[i][n2];
  }
return hata(xi,yi,e);
}

public static double hata(double x[][],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcGeneralLeastSquare(e,x[k]))*(y[k]-funcGeneralLeastSquare(e,x[k]));
 }
total=Math.sqrt(total);
return total;
```

```
}

  public static void main(String[] args)
  {
  String s1=JOptionPane.showInputDialog("file name : ");
  int n=Integer.parseInt(JOptionPane.showInputDialog("polinom derecesi (n)"));
  double c[][]=Text.readDouble(s1);
  Text.print(c,"a");
  double b[]=GeneralLeastSquare(c,n);
  Text.print(b,"b");
  double xi[][]=cikti(c,n,1);
  String s="";
  System.out.println("n_xi = "+xi.length+"n_xi[0] = "+xi[0].length);
  for(int i=0;i<xi.length;i++)
  { for(int j=0;j<xi[0].length;j++)
   {s+=xi[i][j]+" ";}
   s+="\n";
  }
  System.out.println(s);
  //System.out.println("output \n"+Matrix.toString(xi));
  //System.out.println("error \n"+Matrix.toString(hata(c,b)));

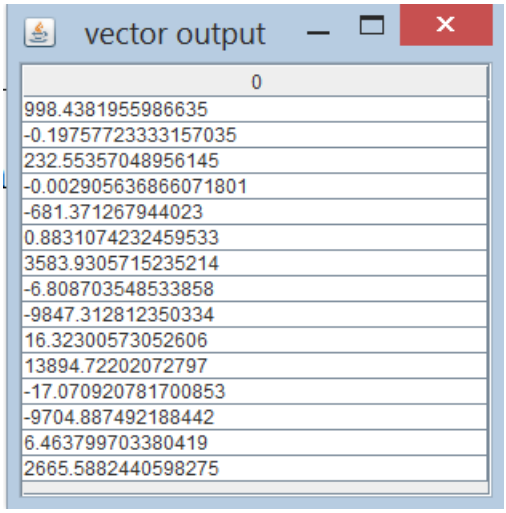
  }
}
```

A sample data for the curve fitting program:

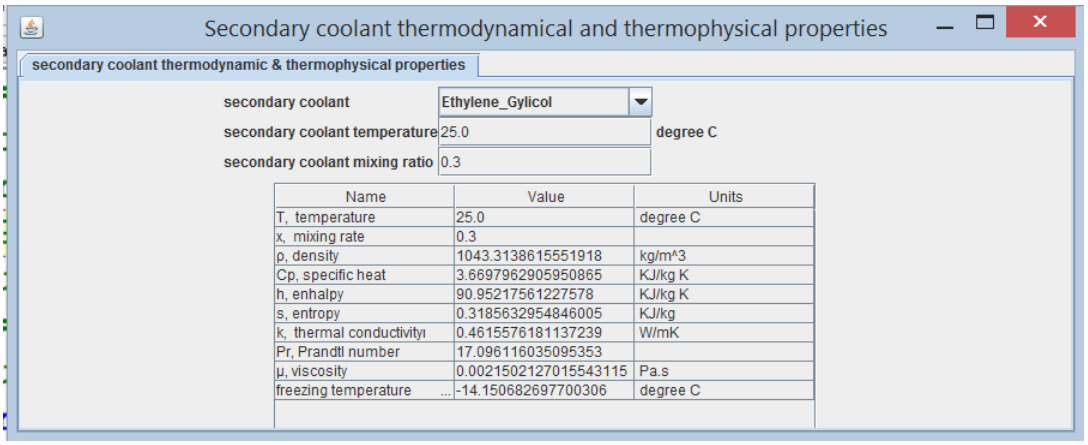
Temp %volume ethylene gylicol density

Degree	10%	20%	30%	40%	50%	60%	70%	80%	90%
-35						1089.9	1104.6	1118.6	1132.1
-30						1089	1103.5	1117.4	1130.7
-25					1088	1102.4	1116	1129.2	1141.9
-20				1072	1086.9	1101.1	1114.6	1127.6	1140.1
-15				1070.9	1085.6	1099.6	1113	1125.8	1138.1
-10			1054.3	1069.6	1084.2	1098.1	1111.3	1123.9	1136.1
-5		1036.9	1053.1	1068.3	1082.7	1096.4	1109.5	1121.9	1133.9
0	1018.7	1035.7	1051.8	1066.8	1081.1	1094.6	1107.5	1119.8	1131.6
5	1017.6	1034.4	1050.3	1065.2	1079.3	1092.7	1105.4	1117.6	1129.2
10	1016.3	1032.9	1048.8	1063.5	1077.5	1090.7	1103.2	1115.2	1126.7
15	1014.9	1031.4	1047.1	1061.7	1075.5	1088.5	1100.9	1112.7	1124
20	1013.3	1029.7	1045.3	1059.7	1073.4	1086.3	1098.5	1110.1	1121.2
25	1011.7	1027.9	1043.3	1057.6	1071.1	1083.9	1095.9	1107.4	1118.3
30	1009.9	1026	1041.3	1055.4	1068.8	1081.4	1093.2	1104.6	1115.3
35	1008	1024	1039.1	1053.1	1066.3	1078.7	1090.4	1101.6	1112.2
40	1006	1021.8	1036.8	1050.6	1063.7	1076	1087.5	1098.5	1108.9
45	1003.9	1019.6	1034.4	1048.1	1060.9	1073.1	1084.5	1095.3	1105.5
50	1001.6	1017.2	1031.8	1045.4	1058.1	1070.1	1081.3	1091.9	1102
55	999.23	1014.6	1029.2	1042.5	1055.1	1066.9	1078	1088.5	1098.4
60	996.72	1012	1026.4	1039.6	1052	1063.7	1074.6	1084.9	1094.6
65	994.1	1009.2	1023.5	1036.6	1048.8	1060.3	1071.1	1081.2	1090.7
70	991.35	1006.4	1020.4	1033.4	1045.5	1056.8	1067.4	1077.4	1086.7
75	988.49	1003.3	1017.3	1030.1	1042	1053.2	1063.6	1073.4	1082.6
80	985.5	1000.2	1014	1026.7	1038.5	1049.5	1059.7	1069.4	1078.4
85	982.39	996.96	1010.6	1023.1	1034.8	1045.6	1055.7	1065.2	1074
90	979.15	993.59	1007.1	1019.4	1031	1041.7	1051.6	1060.9	1069.5
95	975.8	990.1	1003.5	1015.7	1027	1037.6	1047.3	1056.4	1064.9
100	972.32	986.48	999.69	1011.7	1023	1033.3	1042.9	1051.9	1060.2
105	968.73	982.75	995.81	1007.7	1018.8	1029	1038.4	1047.2	1055.4
110	965.01	978.89	991.81	1003.6	1014.5	1024.5	1033.8	1042.4	1050.4
115	961.17	974.91	987.68	999.29	1010	1019.9	1029.1	1037.5	1045.3
120	957.21	970.81	983.43	994.9	1005.5	1015.2	1024.2	1032.5	1040.1
125	953.12	966.59	979.07	990.38	1000.8	1010.4	1019.2	1027.3	1034.7

When program runs the resulting curve fitting coefficients are:

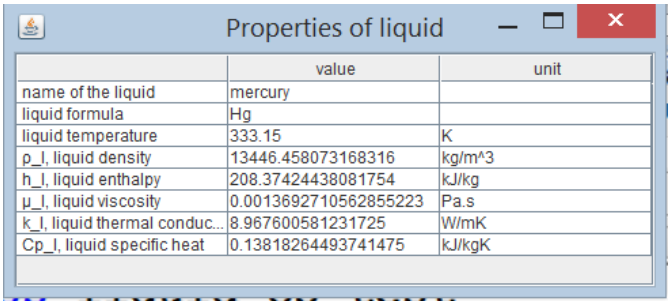


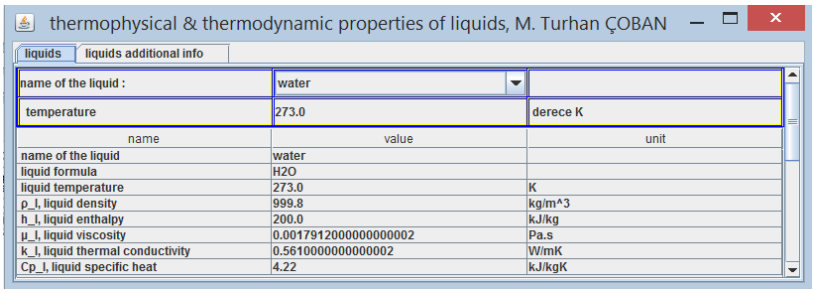
By using this curve fittings a program for the fluid properties, secondarycoolantEN.java and user interface program secondarycoolantTableEN are prepared. Program codes are given in Appendix 6.4



Another liquid properties pogram is developed by using cubic splines. This set is only one dimensional curve fitting (temperature dependent only). Program lists liquid_CS_Data, liquid_CS_EN, liquid_CS_Model_EN and a GUI program liquid_CS_Table_EN. A sample program and a GUI output is shown below:

```
public class liquid_CS_test
{
public static void main(String arg[])
{ liquid_CS_EN l=new liquid_CS_EN("mercury");
  String s1[]={"","value","unit"};
  double T=60.0+273.15;//degree K
  Text.print(l.toString(T),s1,"Properties of liquid");
}}
```





Program List of: secondatyCoolantEN

secondarycoolantEN	Basic Equation of state(based on curve fitting of data)
secondarycoolantTableEN	Graphical user interface for secondarycoolantEN
secondarycoolantModelEN	GUI support program(creates table format of output data)

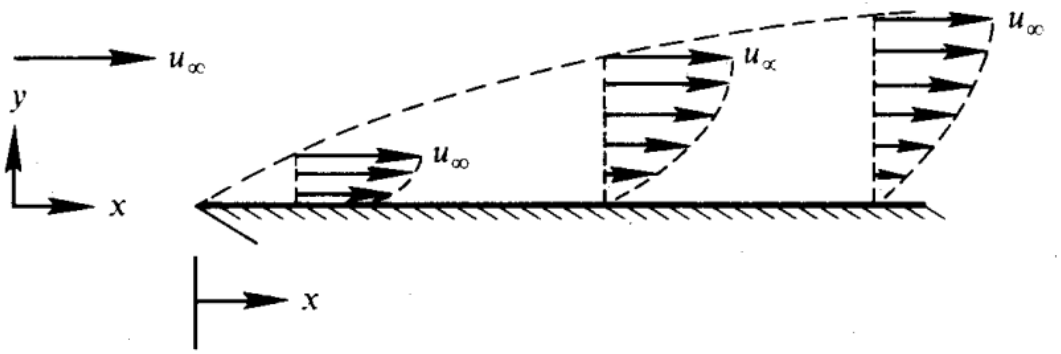
Fluids defined in secondaryCoolantEN

Water	
NaCl brine	Salt+water
CaCl brine	CaCl salt+water
Ethylene Gylicol-water	Antifreeze
Propylene Gylicol-water	Antifreeze
polydimethylsilohexane	

12. CONVECTIVE HEAT TRANSFER OVER A BODY

12.1 BLASSIUS FLAT PLATE PROBLEM AND WEDGE FLOW PROBLEM SOLUTIONS

Blassius flat plate problem is one of the basic problem that shapes heat convection as we know today. Problem is given in all heat transfer text book, but details of the problem is mostly avoided. We would like to investigate problem here including necasary numeric methods to solve it.



Conservation of mass:

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0$$

Conservation of momentum

$\nu\frac{\partial^2 u}{\partial y^2}=u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}$ where ν is kinematic viscosity, u and v x and y direction velocities

The boundary conditions are

$u=0\ v=0\ \ at\ y=0$

$u\to u_\infty\ \ at\ y\to\infty$

$u=u_\infty\ \ at\ x=0$

The shape of velocity profile suggest that it might be similar , different only with a stretching factor on the y coordinate so Let us assume a velocity profile such as

$u=g(y.w(x))$ where

$\eta=y.w(x)$ so

$u=g(\eta)$

$\frac{\partial u}{\partial x}=\frac{\partial g(\eta)}{\partial x}=\frac{\partial g(\eta)}{\partial \eta}\frac{\partial \eta}{\partial x}=g'(\eta)yw'(x)$

$\frac{\partial u}{\partial y}=\frac{\partial g(\eta)}{\partial y}=\frac{\partial g(\eta)}{\partial \eta}\frac{\partial \eta}{\partial y}=g'(\eta)w(x)$

$\frac{\partial^2 u}{\partial y^2}=\frac{\partial^2 g(\eta)}{\partial y^2}=\frac{\partial}{\partial y}\left(\frac{\partial g(\eta)}{\partial y}\right)=\frac{\partial}{\partial \eta}\left(\frac{\partial g(\eta)}{\partial y}\right)\frac{\partial \eta}{\partial y}=g"(\eta)w^2(x)$

Substituting these into the momentum equation

$\nu g"(\eta)w^2(x)\ =g(\eta)g'(\eta)yw'(x)+\nu g'(\eta)w(x)$

Conservation of mass

$g'(\eta)yw'(x)+\frac{\partial v}{\partial y}=0$

Now, by substituting these equations into each other and solving them:

$$v=\frac{\nu g"(\eta)w^2(x)-g(\eta)g'(\eta)yw'(x)}{g'(\eta)w(x)}$$

$$\frac{\partial v}{\partial y} = -g'(\eta) y w'(x)$$

$$\frac{\partial}{\partial y}\left(\frac{\nu g"(\eta) w^2(x) - g(\eta) g'(\eta) y w'(x)}{g'(\eta) w(x)}\right) = -g'(\eta) y w'(x)$$

$$\nu w(x) \frac{\partial}{\partial y}\left(\frac{g"(\eta)}{g'(\eta)}\right) - \frac{w'(x)}{w(x)} \frac{\partial}{\partial y}(g(\eta) y) = -g'(\eta) y w'(x)$$

$$\nu w(x) \frac{\partial}{\partial y}\left(\frac{g"(\eta)}{g'(\eta)}\right) - \frac{w'(x)}{w(x)}\left(\frac{\partial g(\eta)}{\partial \eta}\frac{\partial \eta}{\partial y} y + g(\eta)\right) = -g'(\eta) y w'(x)$$

$$\nu w(x) \frac{\partial}{\partial y}\left(\frac{g"(\eta)}{g'(\eta)}\right) - \frac{w'(x)}{w(x)}(g'(\eta) w(x) y + g(\eta)) = -g'(\eta) y w'(x)$$

$$\nu w(x) \frac{\partial}{\partial y}\left(\frac{g"(\eta)}{g'(\eta)}\right) - \frac{w'(x) g'(\eta) w(x) y}{w(x)} - \frac{w'(x)}{w(x)} g(\eta) + g'(\eta) y w'(x) = 0$$

$$\nu w(x) \frac{\partial}{\partial y}\left(\frac{g"(\eta)}{g'(\eta)}\right) - \frac{w'(x)}{w(x)} g(\eta) = 0$$

$$\frac{\partial}{\partial y}\left(\frac{g"(\eta)}{g'(\eta)}\right) = \frac{\partial}{\partial \eta}\left(\frac{g"(\eta)}{g'(\eta)}\right)\frac{\partial \eta}{\partial y} = \frac{\partial}{\partial \eta}\left(\frac{g"(\eta)}{g'(\eta)}\right) w(x)$$

$$\nu w^2(x) \frac{\partial}{\partial \eta}\left(\frac{g"(\eta)}{g'(\eta)}\right) - \frac{w'(x)}{w(x)} g(\eta) = 0$$

$$\frac{1}{g(\eta)} \frac{\partial}{\partial \eta}\left(\frac{g"(\eta)}{g'(\eta)}\right) = \frac{w'(x)}{\nu w^3(x)}$$

In this form, the differential equation is separated so it can be solved as two different set of equation with the common arbitrary constant (eigenvalue).

$$\frac{1}{g(\eta)} \frac{\partial}{\partial \eta}\left(\frac{g"(\eta)}{g'(\eta)}\right) = \frac{w'(x)}{\nu w^3(x)} = -k$$

$$\frac{dw(x)}{w^3(x)} = -k\nu dx$$

$$-\frac{1}{2w^2(x)} = -k\nu x + C$$

At $y \to \infty \;\; u \to u_\infty$; but x=0 a boundary condition is that $u = u_\infty$ even at y=0. Thus g(0) must be infinite. It follows that C=0. Then

$w(x) = \frac{1}{\sqrt{2k\nu x}} \;\; \eta = y.w(x) = \frac{y}{\sqrt{2k\nu x}}$ So from this results, it is clear that velocity profile

$$u = f\left(\frac{y}{\sqrt{x}}\right)$$

The left hand side of the equation:

$$\frac{1}{g(\eta)} \frac{\partial}{\partial \eta}\left(\frac{g"(\eta)}{g'(\eta)}\right) = -k$$

$$\partial\left(\frac{g"(\eta)}{g'(\eta)}\right) = -k g(\eta) \partial\eta$$

$$\left(\frac{g"(\eta)}{g'(\eta)}\right) = -k \int g(\eta) \partial\eta + C$$

Boundary conditions $\eta = 0 \;\; u = 0 \;\; v = 0 \;\; g = 0 \;\; at\; y = 0 \;\; \frac{\partial^2 u}{\partial y^2} = 0 = \frac{\partial^2 g}{\partial \eta^2} = g"$ and thus C=0

$$\frac{g"(\eta)}{g'(\eta)} = -k \int g(\eta) \partial\eta$$

Let us now define a non-dimensional velocity in terms of a derivative of a function of η, derivative being for the purpose of eliminating the integral of the above equation. Let

$$f'(\eta) = \frac{u}{u_\infty} = \frac{g(\eta)}{u_\infty}$$

Then

$$g(\eta) = u_\infty f'(\eta)$$
$$g'(\eta) = u_\infty f"(\eta)$$
$$g"(\eta) = u_\infty f'''(\eta)$$

and

$$\frac{g"(\eta)}{g'(\eta)} = \frac{f'''(\eta)}{f"(\eta)}$$

Substituting,

$$\frac{f'''(\eta)}{f''(\eta)} = -k\int u_\infty f'(\eta)\partial\eta = -ku_\infty f(\eta)$$

k is an arbitrary constant but ku_∞must be nondimensional, let $ku_\infty = \frac{1}{2}$ Thus,

$$f''' (\eta) + \frac{1}{2}f(\eta) - f''(\eta) = 0$$

Where $\eta = \frac{y}{\sqrt{\nu x/u_\infty}}$ and $f'(\eta) = \frac{u}{u_\infty} = \frac{g(\eta)}{u_\infty}$

Boundary conditions:

u=0 at y=0

v=0 at y=0

$y \to \infty \quad u \to u_\infty$

Then $f'(0) = 0 \quad f'(\infty) = 1$

Since $f''(0) = 0 \quad f'''(0) = 0$ (provided that v(0)=0)

Then from the differential equation $.f(0)f''(0) = 0$ It is not possible that all the derivatives are zero, in fact $f''(0) = 0$ would correspond to zero shear stress at the wall, so we conclude that

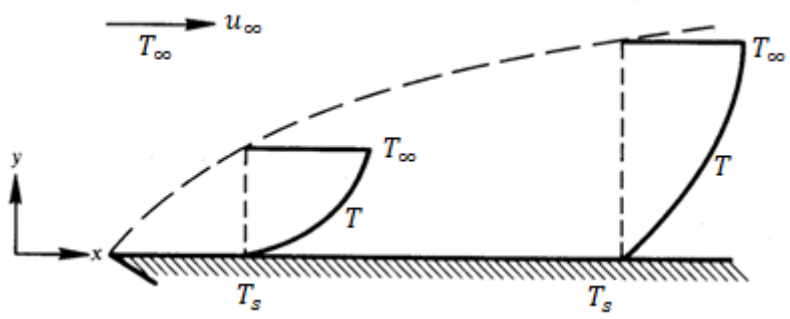
$f(0) = 0$

The same set of equations can also be developed by using stream function $\Psi(x, y)$. Stream function is basically a single function to replace velocities u and v. It is defined as:

$u = \frac{\partial\Psi(x,y)}{\partial y} \quad v = -\frac{\partial\Psi(x,y)}{\partial x}$

Due to the definition, conservation of mass is always satisfied. The relation between $\Psi(x, y)$and f is as follows: $\Psi(\eta) = \sqrt{\nu x u_\infty} f(\eta)$

Now that we have a single variable differential similarity equation for the momentum equation, a similarity solution for energy equation should also be developed.



For $\theta = \frac{T_s - T}{T_s - T_\infty}$ The energy equation for the boundary layer:

$$\alpha\frac{\partial^2\theta}{\partial y^2} = u\frac{\partial\theta}{\partial x} + v\frac{\partial\theta}{\partial y}$$

With the boundary conditions:

$\theta = 0\ at\ y = 0$

$\theta = 1\ at\ y \to \infty$

$\theta = 1\ at\ x = 0$

Assume $\theta = \theta(\eta)$ where $\eta = \frac{y}{\sqrt{\nu x/u_\infty}}$

Energy equation becomes:

$$\frac{\partial^2\theta}{\partial\eta^2} + \frac{Pr}{2} f \frac{\partial\theta}{\partial\eta} = 0$$

Where Pr, Prantl number $Pr = \frac{C_p\mu}{k} = \frac{C_p\rho\nu}{k}$

Boundary conditions:

$\theta(0) = 0$

$\theta(\infty) = 1$

Equation can be solved directly as a set of differential equations. Equation set change as follows:

$$f'''(\eta) + \frac{1}{2} f(\eta) - f''^{(\eta)} = 0$$

$$\theta''(\eta) + \frac{Pr}{2} f(\eta)\theta'(\eta) = 0$$

Conversion of the equation:

$x[0] = \eta \quad x[1] = f(\eta) \quad x[2] = f'(\eta) \quad x[3] = f''(\eta) \quad x[4] = \theta(\eta) \quad x[5] = \theta'(\eta)$

Differential equation set becomes:

$\frac{dx[1]}{dx[0]} = x[2]$

$\frac{dx[2]}{dx[0]} = x[3]$

$\frac{dx[3]}{dx[0]} = -0.5x[1]x[3]$

$\frac{dx[4]}{dx[0]} = x[5]$

$\frac{dx[5]}{dx[0]} = -\frac{Pr}{2} x[1]x[5]$

Boundary conditions:

$x[4] = 0\ at\ x[0] = 0$

$x[4] = 1\ at\ x[0] = \infty$

$x[2] = 0\ at\ x[0] = 0$

$x[2] = 1\ at\ x[0] = \infty$

$x[1] = 0\ at\ x[0] = 0$

As it is seen from boundary conditions, two given conditions are not initial conditions. One way of solving boundary conditions is using non-linear system of equation solving methods. In order to solve differential equation set 6th order Runge-Kutta Method is used. In order to solve boundary value nonlinear system of equation Nelder & Mead optimization method is used.

Runge-Kutta method as in most general form can be given as: For differential form $\frac{dy}{dx} = f(x, y)$

$$y_{n+1} = y_n + h\phi(x_i, y_i, h)$$

$$\phi(x_i, y_i, h) = \sum_{i=1}^{n} b_i k_i$$

In this equation a is constant values depends on polynamial degree of Runge-Kutta equation and coefficients k, can be defined as:

$k_1 = f(x_i, y_i)$

$k_2 = f(x_i + c_2 h, y_i + a_{21} k_1 h)$

$k_3 = f(x_i + c_3 h, y_i + a_{31} k_1 h + a_{32} k_2 h)$

…

$k_3 = f\left(x_i + c_n h, y_i + a_{n,1} k_1 h + a_{n,2} k_2 h + \cdots + a_{n,n-1} k_{n-1} h\right)$

In more generalized form, these equations can be written as:

$x_i = x_n + c_i h \quad i = 1 \ldots s$

$y_i = y_n + h \sum_{j=1}^{i-1} a_{ij} k_k$

$k_i = f(x_i, y_i)$

$$y_{n+1} = y_n + h \sum_{j=1}^{s} b_i k_i$$

where s is the degree of coefficients of equations. In numerical analysis books, coefficients are usually shown as Buthcher tableu which puts the coefficients of the method in a table as follows:

$$\begin{array}{c|ccccc} c_1 & a_{11} & a_{12} & a_{13} & \ldots & a_{1s} \\ c_2 & a_{21} & a_{22} & a_{23} & \ldots & a_{2s} \\ c_3 & a_{31} & a_{32} & a_{33} & \ldots & a_{3s} \\ \vdots & \vdots & \vdots & \vdots & \ldots & \vdots \\ c_s & a_{s1} & a_{s2} & a_{s3} & \ldots & a_{s4} \\ \hline & b_1 & b_2 & b_3 & \ldots & b_s \end{array}$$

As it is seen from the equation the definition is very general. Depends on polynamial degree of Runge-Kutta equation accuracy will improve. **Runge-Kutta with sixth degree polynomial solution RK6** is given as:

$y_{i+1} = y_i + (1/90)*(7k_1 + 32k_3 + 12k_4 + 32k_5 + 7k_6)h$

$k_1 = f(x_i, y_i)$

$k_2 = f(x_i + 0.25h , y_i + 0.25k_1 h)$

$k_3 = f(x_i + 0.25h , y_i + 0.125k_1 h + 0.125k_2 h)$

$k_4 = f(x_i + 0.5h , y_i - 0.5k_2 h + k_3 h)$

$k_5 = f(x_i + 0.75h , y_i + (3/16)k_1 h + (9/16)k_4 h)$

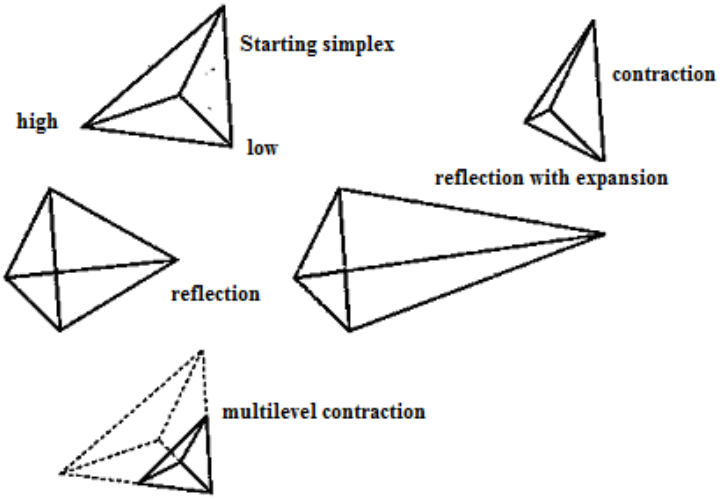
$k_6 = f(x_i + h , y_i - (3/7)k_1 h + (2/7)k_2 h + (12/7)k_3 h - (12/7)k_4 h + (8/7)k_5 h)$

This equation can be given as Buthcher tableu as:

$$\begin{array}{c|ccccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 1/4 & 1/4 & 0 & 0 & 0 & 0 \\ 1/4 & 1/8 & 1/8 & 0 & 0 & 0 \\ 2/4 & 1/4 & -1/4 & 1 & 0 & 0 \\ 3/4 & 3/16 & 0 & 0 & 9/16 & 0 \\ 1 & -3/7 & 2/7 & 12/7 & -12/7 & 8/7 \\ \hline & 1/90 & 7/90 & 32/90 & 12/90 & 7/90 \end{array}$$

Nelder & Mead optimization method is as follows:

The simplex method developed by Nelder and Mead method is a multidimensional search method . Simplex is an n dimensional geometric entity. It contains (n+1) points in n dimensional space. The method uses the concept of a simplex, which is a special polytop of $N + 1$ vertices in N dimensions. Examples of simplices include a line segment on a line, a triangle on a plane, a tetrahedron in three-dimensional space and so forth. Required processes for the simplex amobea's movements are shown in the figure



Process can be given as follows

$$f(x_1) \leq f(x_r) < f(x_n)$$
$$x_r = x_0 + \alpha(x_0 - x_{n+1})$$

- **1. Order** according to the values at the vertices:
$$f(x_1) \leq f(x_2) \ldots \leq f(x_{n+1})$$
- **2.** Calculate x_o, the center of gravity of all points except x_{n+1}.
- **3. Reflection**
  Compute reflected point $x_r = x_0 + \alpha(x_0 - x_{n+1})$
  If the reflected point is better than the second worst, but not better than the best, i.e.:
  $f(x_1) \leq f(x_r) < f(x_n)$,
  then obtain a new simplex by replacing the worst point x_{n+1} with the reflected point x_r, and go to step 1.
- **4. Expansion**
  If the reflected point is the best point so far, $f(x_r) < f(x_1)$

then compute the expanded point

$$x_e = x_0 + \gamma(x_0 - x_{n+1})$$

  If the expanded point is better than the reflected point, $f(x_e) < f(x_r)$
  then obtain a new simplex by replacing the worst point x_{n+1} with the expanded point x_e, and go to step 1.
  Else obtain a new simplex by replacing the worst point x_{n+1} with the reflected point x_r, and go to step 1.
  Else (i.e. reflected point is worse than second worst) continue at step 5.
- **5. Contraction**
  Here, it is certain that $f(x_r) \geq f(x_n)$
  Compute contracted point $x_c = x_{n+1} + \rho(x_0 - x_{n+1})$
  If the contracted point is better than the worst point, i.e. $f(x_c) \leq f(x_{n+1})$
  then obtain a new simplex by replacing the worst point x_{n+1} with the contracted point x_c, and go to step 1. Else go to step 6.
- **6. Reduction**
  For all but the best point, replace the point with $x_i = x_1 + \sigma(x_i - x_1)\ for\ all\ i\{2, \ldots, n+1\}$
  go to step 1.

Note: $\alpha, \gamma, \rho\ and\ \sigma$ are respectively the reflection, the expansion, the contraction and the shrink coefficient. Standard values are $\alpha = 1, \gamma = 2, \rho = 0.5, and\ \sigma = 0.5$

For the **reflection**, since x_{n+1} is the vertex with the higher associated value among the vertices, we can expect to find a lower value at the reflection of x_{n+1}in the opposite face formed by all vertices point x_i except x_{n+1}.

For the **expansion**, if the reflection point x_r is the new minimum along the vertices we can expect to find interesting values along the direction from x_o to x_r.

Concerning the **contraction**: If $f(x_r) > f(x_n)$ we can expect that a better value will be inside the simplex formed by all the vertices x_i.
The initial simplex is important, indeed, a too small initial simplex can lead to a local search, consequently the NM can get more easily stuck. So this simplex should depend on the nature of the problem. Instead of given n+1 point same process can be done by giving one point and a change vector by defining
$x_{i-1} = P_i + \lambda dx_j$ where if i=j $\lambda = 1$ $else\ if\ i \neq j\ \lambda = 0$
For example:
If $P = \begin{Bmatrix} a \\ b \\ c \end{Bmatrix}$ and $dx = \begin{Bmatrix} da \\ db \\ dc \end{Bmatrix}$

$$x_0 = \begin{Bmatrix} a \\ b \\ c \end{Bmatrix} \quad x_1 = \begin{Bmatrix} a + da \\ b \\ c \end{Bmatrix} x_2 = \begin{Bmatrix} a \\ b + db \\ c \end{Bmatrix} \quad x_3 = \begin{Bmatrix} a \\ b \\ c + dc \end{Bmatrix}$$

Non-Linear system of equations can also be solved by using optimization methods through an adaptation function.
To solve function $f_i(x_j)=0$

$$g(x_j) = \sum_{i=0}^{n_equation} f_i(x_j) * f_i(x_j)$$

The derivative of this equation is the root of non-linear system of equation again became $f_i(x_j)=0$, therefore optimum of $g(x_j)$ is the same problem as solution of $f_i(x_j)=0$.
Another program utilized in this set is generalized **least square linear curve fitting**. Assuming having a linear function $f(a_j,x)$ where a_j are m linear coefficient and x are independent variable and $\phi_j(x)$ are m sub-funtions linearly multiplied with the coefficients

$$f(a_j, x) = \sum_{j=0}^{m} a_j^{(m)} \phi_j(x)$$

It should be noted that linearity is only for the coefficients, $\phi_j(x)$ functions does not have to be linear. It is desired to fit data x_i, f_i, i=0..n into the equation so that the difference between data and fitted function dependent values for all points will be minimum. In order to establish this, the following function H will be minimized with respect to a_j

$$H(a_0^{(m)}, ..., a_m^{(m)}) = \sum_{i=1}^{n} w(x_i) \left[f_i - \sum_{j=0}^{m} a_j^{(m)} \phi_j(x_i) \right]^2$$

Where $w(x_i)$ values in the equation are called weight function. In order to minimiz the function root of the derivative of the function can be calculated.

$$\frac{\partial H(a_0^{(m)}, ..., a_m^{(m)})}{\partial a_k^{(m)}} = 2 \sum_{i=1}^{n} w(x_i) \left[f_i - \sum_{j=0}^{m} a_j^{(m)} \phi_j(x_i) \right] \phi_k(x_i) = 0 \quad k = 0, ..., m$$

$$\left\{ \sum_{j=0}^{m} w(x_i) \phi_j(x_i) \phi_k(x_i) \right\} \left[a_j^{(m)} \right] = \left[\sum_{i=1}^{n} w(x_i) \phi_k(x_i) f_i \right] \quad k = 0, ..., m$$

For weight function to be taken equal to unity , $w(x_i)$=1, equation in the open form can be written in the following form.

$$\begin{Bmatrix} \sum_{i=1}^{n}\phi_0^2(x_i) & \sum_{i=1}^{n}\phi_0(x_i)\phi_1(x_i) & \sum_{i=1}^{n}\phi_0(x_i)\phi_2(x_i) & .... & \sum_{i=1}^{n}\phi_0(x_i)\phi_m(x_i) \\ \sum_{i=1}^{n}\phi_0(x_i)\phi_1(x_i) & \sum_{i=1}^{n}\phi_1^2(x_i) & \sum_{i=1}^{n}\phi_1(x_i)\phi_2(x_i) & .... & \sum_{i=1}^{n}\phi_1(x_i)\phi_m(x_i) \\ \sum_{i=1}^{n}\phi_0(x_i)\phi_2(x_i) & \sum_{i=1}^{n}\phi_1(x_i)\phi_2(x_i) & \sum_{i=1}^{n}\phi_2^2(x_i) & .... & \sum_{i=1}^{n}\phi_2(x_i)\phi_m(x_i) \\ .... & .... & .... & .... & .... \\ \sum_{i=1}^{n}\phi_0(x_i)\phi_m(x_i) & \sum_{i=1}^{n}\phi_1(x_i)\phi_m(x_i) & \sum_{i=1}^{n}\phi_2(x_i)\phi_m(x_i) & .... & \sum_{i=1}^{n}\phi_m^2(x_i) \end{Bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ .... \\ a_m \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n}\phi_0(x_i)f(x_i) \\ \sum_{i=1}^{n}\phi_1(x_i)f(x_i) \\ \sum_{i=1}^{n}\phi_2(x_i)f(x_i) \\ .... \\ \sum_{i=1}^{n}\phi_m(x_i)f(x_i) \end{bmatrix}$$

This equation is an m+1 linear system of equation. It can easily be solved by using a system of equation solving method.

Now that basic mathematical calculation methods are established, The equation can be solved by using differential equation boundary value solving method.

```
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
class fb extends f_xr
{
              public double func(double x,int i)
              {
    double xx=1.0;
     if(i==0) xx=1.0;
     else if(i==1) xx=x;
      else if(i==2) xx=x*x;
       else if(i==3) xx=x*x*x;
        else if(i==4) xx=x*x*x*x;
         else if(i==5) xx=x*x*x*x*x;
          else if(i==6) xx=x*x*x*x*x*x;
           else if(i==7) xx=x*x*x*x*x*x*x;
            else if(i==8) xx=x*x*x*x*x*x*x*x;
             else if(i==9) xx=x*x*x*x*x*x*x*x*x;
              else if(i==10) xx=x*x*x*x*x*x*x*x*x*x*x;
  return xx;
              }
}
class f2 extends f_xj
{//adaptation function
fi_xi ff1;
public f2(fi_xi ffi) {ff1=ffi;}

public double func(double x[])
{
double ff=0.0;
double fa[]=ff1.func(x);
for(int i=0;i<fa.length;i++)  ff+=fa[i]*fa[i];
return ff;
}
}

class fa extends fi_xi
{ // Nonlinear system of equation
double y[];
fm2 b3;
double w;
double a[][];
public fa(double Pr,double yi[],double wi)
{ b3=new fm2(Pr);
  y=new double[yi.length];
  for(int i=0;i<y.length;i++) y[i]=yi[i];
  w=wi;
}

public double[] func(double x[])
{
// shooting method
y[2]=x[0];
y[4]=x[1];
```

```
a=RK6(b3,0.0,w,y,10000);
double ff[]=new double[2];
ff[0]=a[2][10000]-1.0;
ff[1]=a[4][10000]-1.0;
return ff;
}
 //differenetial equation solution
public static double[][] RK6(f_xi fp,double x0,double xn,double f0[],int N)
{
//6th order Runge Kutta Method
//fp  : given set of derivative functions dfj/dxi(fj,x)
// xo : initial value of the independent variable
// xn : final value of the independent variable
// f0 : initial value of the dependent variable
// N  : number of dependent variable to be calculated
// fi : dependent variable
double h=(xn-x0)/N;
int M=f0.length;
double fi[][];
fi=new double[M][N+1];
double xi[]=new double[M+1];
double k[]=new double[6];
int i,j;
double x;
for(j=0;j<M;j++)
  {
  fi[j][0]=f0[j];
  xi[j+1]=f0[j];
  }
for(x=x0,i=0;i<N;x+=h,i++)
  {
  for(j=1;j<=M;j++)
    {
    xi[0]=x;
    xi[j]=fi[j-1][i];
    k[0]=h*fp.func(xi,j-1);
    xi[0]=x+h/2.0;
    xi[j]=fi[j-1][i]+k[0]/2;
    k[1]=h*fp.func(xi,j-1);
    xi[0]=x+h/2.0;
    xi[j]=fi[j-1][i]+k[0]/4.0+k[1]/4.0;
    k[2]=h*fp.func(xi,j-1);
    xi[0]=x+h;
    xi[j]=fi[j-1][i]-k[1]+2.0*k[2];
    k[3]=h*fp.func(xi,j-1);
    xi[0]=x+2.0/3.0*h;
    xi[j]=fi[j-1][i]+7.0/27.0*k[0]+10.0/27.0*k[1]+1.0/27.0*k[3];
    k[4]=h*fp.func(xi,j-1);
    xi[0]=x+1.0/5.0*h;
    xi[j]=fi[j-1][i]+28.0/625.0*k[0]-1.0/5.0*k[1]+546.0/625.0*k[2]+54.0/625.0*k[3]-378/625.0*k[4];
    k[5]=h*fp.func(xi,j-1);
    fi[j-1][i+1]=fi[j-1][i]+k[0]/24.0+5.0*k[3]/48.0+27.0*k[4]/56.0+125.0*k[5]/336.0;
    xi[j]=fi[j-1][i];
    }
  }
double a[][]=new double[M+1][N+1];
for(x=x0,i=0;i<=N;x+=h,i++)
{
  a[0][i]=x;
  for(j=1;j<=M;j++)
  {
  a[j][i]=fi[j-1][i];
  }
}

return a;
}
}

//Blassius flat plate differential equation
class fm2 extends f_xi
{ double Pr;
  fm2(double Pri) {Pr=Pri;}
// multivariable function
 double func(double x[],int x_ref)
 {
   //Blasius flat plate diferential equation
```

```
  //f"'+0.5*ff'' = 0
  //f"'=-0.5*x[1]*x[3] f"'=x[3] f'=x[2] f=x[1]]
  double a=0;
  if(x_ref==4) a= -0.5*Pr*x[1]*x[5];
  if(x_ref==3) a= x[5];
  if(x_ref==2) a= -0.5*x[1]*x[3];
  if(x_ref==1) a= x[3];
  if(x_ref==0) a= x[2];;
  return a;
 }
}

public class blassius_flat_plate1
{
                public static double[] inputdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }
public static String toString(double x[])
{ String s="";
int n=x.length;
  for(int k=0;k<n;k++)
  {s+=x[k]+" ";}
  return s;
}
public static double[] nelder(f_xj fnelder,double a[],double da[],int maxiteration,double tolerance,int printlist)
{
               int i,j;
double x[][]=new double[a.length+1][a.length];
double p[][]=new double[a.length+1][a.length+1];
for(i=0;i<x.length;i++)
   {for(j=0;j<x[0].length;j++)
    {if(i==j){x[i][j]=a[j]+da[j];p[i][j]=x[i][j];}
     else    {x[i][j]=a[j];p[i][j]=x[i][j]; }
    }
    p[i][j] = fnelder.func(p[i]);
   }


   // Inlet variable definitions
   // fnelder : abstract multivariable function f(x)
   // x : independent variable set of n+1 simplex elements
   // maxiteration : maximum iteration number
   // tolerance :
       int NDIMS = x.length-1;
       int NPTS = x.length;
       int FUNC = NDIMS;
       int ncalls = 0;
         ////// construct the starting simplex //////////////////
         //double p[][]=new double[NPTS][NPTS]; // [row][col] = [whichvx][coord,FUNC]
         double z[]=new double[NDIMS];
         double best = 1E99;
         ///////////////// calculate the first function values for the simplex ////////////////

         int iter=0;

         for (iter=1; iter<maxiteration; iter++)
         {
                        ////////////  define lo, nhi, hi (low high next_to_high //////////////
            int  ilo=0, ihi=0, inhi = -1; // -1 means missing
            double flo = p[0][FUNC];
            double fhi = flo;
            double pavg,sterr;
            for (i=1; i<NPTS; i++)
            {
              if (p[i][FUNC] < flo)
```

```
      {flo=p[i][FUNC]; ilo=i;}
    if (p[i][FUNC] > fhi)
      {fhi=p[i][FUNC]; ihi=i;}
  }
  double fnhi = flo;
  inhi = ilo;
  for (i=0; i<NPTS; i++)
    if ((i != ihi) && (p[i][FUNC] > fnhi))
      {fnhi=p[i][FUNC]; inhi=i;}
  ////////// exit criteria //////////////
  if ((iter % 4*NDIMS) == 0)
  {
     // calculate the avarage (including maximum value)
     pavg=0;
     for(i=0;i<NPTS;i++)
       pavg+=p[i][FUNC];
     pavg/=NPTS;
     double tot=0;
     if(printlist!=0)
     {  System.out.print(iter);
        for (j=0; j<=NDIMS; j++)
          { System.out.print(p[ilo][j]+" ");}
        System.out.println("");
     }
     for(i=0;i<NPTS;i++)
     { tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg);}
     sterr=Math.sqrt(tot/NPTS);
     //if(sterr < tolerance)
       { for (j=0; j<NDIMS; j++)
           { z[j]=p[ilo][j];}
         //break;
       }
     best = p[ilo][FUNC];
  }

  ///// calculate avarage without maximum value //////

  double ave[] = new double[NDIMS];
  for (j=0; j<NDIMS; j++)
    ave[j] = 0;
  for (i=0; i<NPTS; i++)
    if (i != ihi)
      for (j=0; j<NDIMS; j++)
        ave[j] += p[i][j];
  for (j=0; j<NDIMS; j++)
    ave[j] /= (NPTS-1);


  ///////// reflect ////////////////

  double r[] = new double[NDIMS];
  for (j=0; j<NDIMS; j++)
    r[j] = 2*ave[j] - p[ihi][j];
  double fr = fnelder.func(r);

  if ((flo <= fr) && (fr < fnhi))  // in zone: accept
  {
     for (j=0; j<NDIMS; j++)
       p[ihi][j] = r[j];
     p[ihi][FUNC] = fr;
     continue;
  }

  if (fr < flo)  //// expand
  {
     double e[] = new double[NDIMS];
     for (j=0; j<NDIMS; j++)
       e[j] = 3*ave[j] - 2*p[ihi][j];
     double fe = fnelder.func(e);
     if (fe < fr)
     {
        for (j=0; j<NDIMS; j++)
          p[ihi][j] = e[j];
        p[ihi][FUNC] = fe;
        continue;
     }
     else
```

```
                {
                  for (j=0; j<NDIMS; j++)
                    p[ihi][j] = r[j];
                  p[ihi][FUNC] = fr;
                  continue;
                }
              }

              ////////////// shrink:

              if (fr < fhi)
              {
                 double c[] = new double[NDIMS];
                 for (j=0; j<NDIMS; j++)
                   c[j] = 1.5*ave[j] - 0.5*p[ihi][j];
                 double fc = fnelder.func(c);
                 if (fc <= fr)
                 {
                    for (j=0; j<NDIMS; j++)
                      p[ihi][j] = c[j];
                    p[ihi][FUNC] = fc;
                    continue;
                 }
                 else   /////// daralt
                 {
                    for (i=0; i<NPTS; i++)
                      if (i != ilo)
                      {
                         for (j=0; j<NDIMS; j++)
                           p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                         p[i][FUNC] = fnelder.func(p[i]);
                      }
                    continue;
                 }
              }

              if (fr >= fhi)   ///
              {
                 double cc[] = new double[NDIMS];
                 for (j=0; j<NDIMS; j++)
                   cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
                 double fcc = fnelder.func(cc);
                 if (fcc < fhi)
                 {
                    for (j=0; j<NDIMS; j++)
                      p[ihi][j] = cc[j];
                    p[ihi][FUNC] = fcc;
                    continue;
                 }
                 else    /////////
                 {
                    for (i=0; i<NPTS; i++)
                      if (i != ilo)
                      {
                         for (j=0; j<NDIMS; j++)
                           p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                         p[i][FUNC] = fnelder.func(p[i]);
                      }
                 }
              }
           }
       return z;
}
    public static double[] nelder(f_xj fnelder,double a[],double da[],double tolerance)
    {return nelder(fnelder,a,da,500,tolerance,0);}

    public static double[] nelder(f_xj fnelder,double a[],double da[])
    {return nelder(fnelder,a,da,500,1.0e-10,0);}

    public static double[] nelder(f_xj fnelder,double a[])
    {
                    double [] da=new double[a.length];
          for(int i=0;i<a.length;i++) da[i]=0.1*a[i];
          return nelder(fnelder,a,da);
    }
// Polynomial least square
  public static double[] gausswithpartialpivot(double a[][],double b[])
```

```
  { //Gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //partial pivoting
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                  if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
                  b[k]=dummy;
                }
                //
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
                }
  }
  //backward substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}

public static double[] PolynomialLSQ(double xi[],double yi[],int n)
{ //Polynomial least square
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  for(j=0;j<n+1;j++)
     {if(i==0 && j==0) A[i][j]=l;
     else  for(k=0;k<l;k++) A[i][j] += Math.pow(xi[k],(i+j));
     }
     for(k=0;k<l;k++) { if(i==0)  B[i]+= yi[k];
                   else      B[i] += Math.pow(xi[k],i)*yi[k];}
  }
X=gausswithpartialpivot(A,B);
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
return X;
}

public static double funcPolynomialLSQ(double e[],double x)
```

```
{
// this function calculates the value of
// least square curve fitting function
int n=e.length;
double ff;
if(n!=0.0)
  { ff=e[n-1];
  for(int i=n-2;i>=0;i--)
   { ff=ff*x+e[i]; }
  }
else
  ff=0;
return ff;
}

public static double error(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcPolynomialLSQ(e,x[k]))*(y[k]-funcPolynomialLSQ(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

public static double[][]  funcPolynomialLSQ(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
 {
              //aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
              // ara değer sayisi 2 dir
               int n=xi.length;
              int nn=(n-1)*(aradegersayisi+1)+1;
              double z[][]=new double[2][nn];
    double E[]=PolynomialLSQ(xi,yi,polinomkatsayisi);
    System.out.println("katsayilar :\n"+Matrix.toStringT(E));
    double dx=0;
    int k=0;
    int i;
    for(i=0;i<(n-1);i++)
      {z[0][k]=xi[i];z[1][k]=funcPolynomialLSQ(E,z[0][k]);k++;
      for(int j=0;j<aradegersayisi;j++)
      {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
      z[0][k]=z[0][k-1]+dx;z[1][k]=funcPolynomialLSQ(E,z[0][k]);k++;}
      }
      z[0][k]=xi[i];z[1][k]=funcPolynomialLSQ(E,z[0][k]);
    return z;
 }
//general least square curve fitting

public static double[] GeneralLeastSquare(double xi[],double yi[],int n)
{
fb f=new fb();
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  B[i]=0;
              for(j=0;j<np1;j++)
    {
              A[i][j]=0.0;
    for(k=0;k<l;k++) A[i][j]+=f.func(xi[k],i)*f.func(xi[k],j);
    }
    for(k=0;k<l;k++) B[i]+= f.func(xi[k],i)*yi[k];
  }
//System.out.println("A = \n"+Matrix.toString(A));
//System.out.println("B = \n"+Matrix.toStringT(B));
X=gausswithpartialpivot(A,B);
//X=B/A;
```

```
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
Text.printT(X);
return X;
}

public static double funcGeneralLeastSquare(double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
fb f=new fb();
int n=e.length;
double ff=0;
if(n!=0.0)
  {
  for(int i=n-1;i>=0;i--)
   {ff+=e[i]*f.func(x,i);}
  }
return ff;
}

public static double hata(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcGeneralLeastSquare(e,x[k]))*(y[k]-funcGeneralLeastSquare(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}
 public static double[][]  funcGeneralLeastSquare(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
 {  // aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
                // ara değer sayisi 2 dir
                 int n=xi.length;
                int nn=(n-1)*(aradegersayisi+1)+1;
                double z[][]=new double[2][nn];
    double E[]=GeneralLeastSquare(xi,yi,polinomkatsayisi);
    double dx=0;
    int k=0;
    int i;
    for(i=0;i<(n-1);i++)
       {z[0][k]=xi[i];z[1][k]=funcGeneralLeastSquare(E,z[0][k]);k++;
       for(int j=0;j<aradegersayisi;j++)
       {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
       z[0][k]=z[0][k-1]+dx;z[1][k]=funcGeneralLeastSquare(E,z[0][k]);k++;}
       }
       z[0][k]=xi[i];z[1][k]=funcGeneralLeastSquare(E,z[0][k]);
    return z;
 }
public static void main(String arg[]) throws IOException
{
  double y[]={0,0,0.1,0,0.1};
 double x[]={0.33,0.0173};//inputdata(s);
 fa ff;
 f2 f;
 double r1[];
 double
Pr[]={0.001,0.002,0.003,0.004,0.005,0.006,0.007,0.008,0.009,0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.
9,1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0,11,12,13,14,15.0,30.0,40.0,50,60,70,80,90,100,110.0,120.0,130.0
,140,150,160,170,180,190,200,300,400,500,600,700,800,900,1000};
 double
w[]={200,200,200,200,200,200,200,200,200,100.0,100,100,100,100,100,100,100,100,50,50,50,50,50,50,50,50,50,25,25,25,25,25,25,25,25,
25,10,10,10,10,10,10,10,10,10,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3};
 double xi[]=new double[Pr.length];
 double yi[]=new double[Pr.length];
 double Xi[]=new double[Pr.length];
 double Yi[]=new double[Pr.length];
 //Plot pp=new Plot();
 //Plot pp1=new Plot();
 for(int i=0;i<xi.length;i++)
```

```
 {ff=new fa(Pr[i],y,w[i]);
  f=new f2(ff);
  r1=nelder(f,x);
  xi[i]=Pr[i];
  yi[i]=r1[1];
  Xi[i]=Math.log(Pr[i]);
  Yi[i]=Math.log(r1[1]);
  System.out.println("i="+i+"Pr="+xi[i]+"w="+w[i]+"logPr="+Xi[i]+"Nux/Rex^0.5="+Yi[i]);
}
  double a[][]={Xi,Yi};
  Text.printT(a);
  double b[]=GeneralLeastSquare(Xi,Yi,6);
  double gPr1[]=new double[Pr.length];
  double egPr1[]=new double[Pr.length];
  for(int i=0;i<Pr.length;i++)

{gPr1[i]=b[0]*Math.pow(Pr[i],0.25)+b[1]*Math.pow(Pr[i],0.5)+b[2]*Math.pow(Pr[i],0.75)+b[3]*Pr[i]+b[4]*Math.pow(Pr[i],1.25)+b[5]*
Math.pow(Pr[i],1.5)+b[6]*Math.pow(Pr[i],1.75);
   egPr1[i]=(gPr1[i]-yi[i])/yi[i]*100;
  }

  Text.printT(b);
  double d[][]={xi,yi};
  Text.printT(d,"d");
  //pp.plot();
  //pp1.plot();
  Plot pp5=new Plot(xi,yi);
  pp5.addData(xi,gPr1);
  pp5.setColor(1,0,0,255);
  pp5.setPlotType(1,23);
  pp5.setColor(2,255,0,0);
  pp5.setPlotType(2,28);
  pp5.plot();
  Plot pp6=new Plot(xi,egPr1);
  pp6.setColor(1,0,0,255);
  pp6.setColor(2,255,0,0);
  pp6.plot();
 }
 }
```

Program investigates a wide range of Prandtl numbers and as a results:

$$q''_s = h_x(T_s - T_\infty) = -k\left.\frac{\partial T}{\partial y}\right|_{y=0}$$

$$\theta = \frac{T_s - T}{T_s - T_\infty}$$

$$\left.\frac{\partial \theta}{\partial y}\right|_{y=0} = -\frac{1}{T_s - T_\infty}\left.\frac{\partial T}{\partial y}\right|_{y=0}$$

$$\left.\frac{\partial T}{\partial y}\right|_{y=0} = -(T_s - T_\infty)\left.\frac{\partial \theta}{\partial y}\right|_{y=0}$$

$$h_x = \frac{q''_s}{T_s - T_\infty} = k\frac{(T_s - T_\infty)\left.\frac{\partial \theta}{\partial y}\right|_{y=0}}{(T_s - T_\infty)} = k\left.\frac{\partial \theta}{\partial y}\right|_{y=0} = k\left.\frac{\partial \theta(\eta)}{\partial y}\right|_{\eta=0}\frac{\partial \eta}{\partial y}$$

$$\eta = \frac{y}{\sqrt{\nu x/u_\infty}}$$

$$\frac{\partial \eta}{\partial y} = \frac{1}{\sqrt{\nu x/u_\infty}} = \frac{1}{\sqrt{\nu x/u_\infty}} = \frac{1}{x\sqrt{\nu/(u_\infty x)}} = \frac{\sqrt{u_\infty x/\nu}}{x} = \frac{\sqrt{Re_x}}{x}$$

$$h_x = k\frac{\sqrt{Re_x}}{x}\left.\frac{\partial \theta(\eta)}{\partial y}\right|_{\eta=0}$$

$$\boldsymbol{Nu_x} = \frac{\boldsymbol{h_x x}}{\boldsymbol{k}} = \sqrt{Re_x}\left.\frac{\partial \theta(\eta)}{\partial y}\right|_{\eta=0}$$

Nu is called Nusselt's number



Wilhelm Nusselt

Partial continuous curve fitting values are as follows:

$$Nu_x = \frac{h_x x}{k} = 0.44544717 Re_x^{0.5} Pr^{0.4691455} \quad 10^{-3} \leq Pr \leq 10^{-2}$$

$$Nu_x = \frac{h_x x}{k} = 0.3466675 Re_x^{0.5} Pr^{0.4015967} \quad 10^{-2} \leq Pr \leq 0.5$$

$$Nu_x = \frac{h_x x}{k} = 0.33206065 Re_x^{0.5} Pr^{0.3367932} \quad 0.5 \leq Pr \leq 1000$$

Energy equation can also be directly integrated as follows :

$$\frac{\partial^2 \theta}{\partial \eta^2} + \frac{Pr}{2} f \frac{\partial \theta}{\partial \eta} = 0$$

$$\frac{\partial \theta'}{\partial \eta} + \frac{Pr}{2} f \theta' = 0$$

$$\frac{\partial \theta'}{\theta'} = -\frac{Pr}{2} f d\eta$$

$$\ln(\theta') = -\frac{Pr}{2} \int_0^{\eta} f(\eta) d\eta + C$$

$$\theta'(\eta) = C_1 exp\left(-\frac{Pr}{2} \int_0^{\eta} f(\eta) d\eta \right)$$

$$\theta(\eta) = C_1 \int_0^{\eta} \left[exp\left(-\frac{Pr}{2} \int_0^{\eta} f(\eta) d\eta \right) \right] d\eta + C_2$$

By applying the boundary condition $\theta(\eta) = 0 \; at \; \eta = 0 \;\; C_2 = 0$

For the boundary condition $\theta(\eta) = 1 \; at \; \eta \to \infty$

$$C_1 = \frac{1}{\int_0^{\infty} \left[exp\left(-\frac{Pr}{2} \int_0^{\eta} f(\eta) d\eta \right) \right] d\eta}$$

Therefore

$$\theta(\eta) = \frac{\int_0^{\eta} \left[exp\left(-\frac{Pr}{2} \int_0^{\eta} f(\eta) d\eta \right) \right] d\eta}{\int_0^{\infty} \left[exp\left(-\frac{Pr}{2} \int_0^{\eta} f(\eta) d\eta \right) \right] d\eta}$$

Remember that

$$Nu_x = \frac{h_x x}{k} = \sqrt{Re_x}\frac{\partial\theta(\eta)}{\partial y}\Big|_{\eta=0}$$

and from the above equation

$$\theta'(\eta) = \frac{exp\left(-\frac{Pr}{2}\int_0^\eta f(\eta)d\eta\right)}{\int_0^\infty\left[exp\left(-\frac{Pr}{2}\int_0^\eta f(\eta)d\eta\right)\right]d\eta}$$

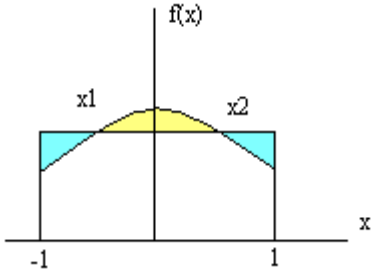
$$\theta'(0) = \frac{1}{\int_0^\infty\left[exp\left(-\frac{Pr}{2}\int_0^\eta f(\eta)d\eta\right)\right]d\eta} = C_1$$

The integration can still be solved by this equation an approximation of curve fit polynomial substitution will be used to find the integral. Of course momentum boundary layer still should be solved by using numerical differential equation solution methods.In order to solve integration the following polynomial equation is applied

$$f(\eta) = a_0 + a_1\eta + a_2\eta^2 + a_3\eta^3 + a_4\eta^4 + a_5\eta^5 + a_6\eta^6 + a_7\eta^7 + a_8\eta^8 + a_9\eta^9 + a_{10}\eta^{10}$$

$$\int_0^\eta f(\eta)d\eta = a_0\eta + \frac{a_1}{2}\eta^2 + \frac{a_2}{3}\eta^3 + \frac{a_3}{4}\eta^4 + \frac{a_4}{5}\eta^5 + \frac{a_5}{6}\eta^6 + \frac{a_6}{7}\eta^7 + \frac{a_7}{8}\eta^8 + \frac{a_8}{9}\eta^9 + \frac{a_9}{10}\eta^{10} + \frac{a_{10}}{11}\eta^{11}$$

Outside integral is solved by using Gauss-Legendre Numerical integration method. A problem known from antique times, is to create a rectangle with the same area with a polynomial. It is called a quadrature problem.



a rectangle with the same area with a polynomial

If point x1 and x2 intersecting the polynomial and rectangle is known, area calculation of the polynomial can be interchange with the area calculation of the rectangle. Or in more general means instead of integration process of summation can be substituted. As a general definition:

$$I_w(-1,1) \cong \int_{-1}^{1} f(x)w(x)dx \cong \sum_{k=1}^{n} c_k f(x_k)$$

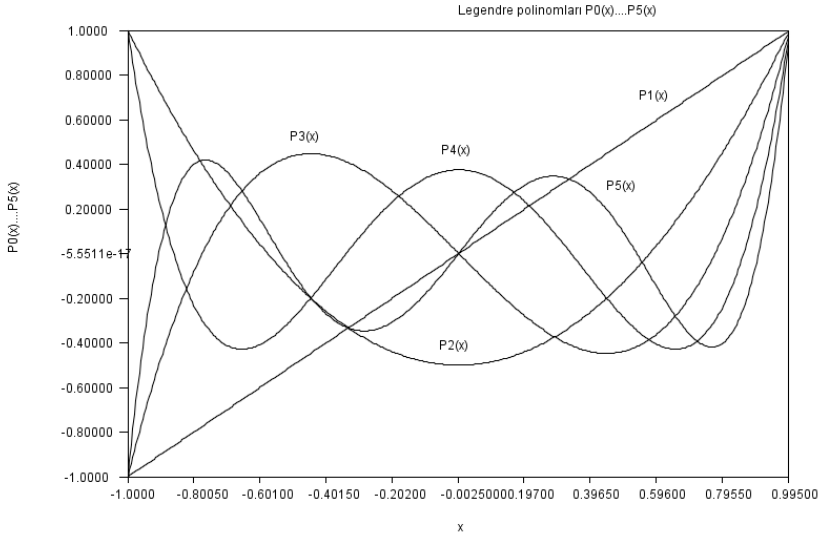
Can be written. In this equation w(x) is the weight factor. In the first of a such formulation, Gauss-Legendre integral formulation, weight factor can be taken as 1.

$$I_w(-1,1) \cong \int_{-1}^{1} f(x)dx \cong \sum_{k=1}^{n} c_k f(x_k)$$

The general solution of this problem can be defined with the Legendre polynomials. Legendre polynomials has a general definition as:

$(k+1)P_{k+1}(x) = (2k+1)xP_k(x) - kP_{k-1}(x), \quad k \geq 1$ and $P_0(x)=1$,$P_1(x)=x$

The first 6 Legendre polynomials are giving in graphic form.



The roots of Legendre Polynomials are the roots of the Gauss-Legendre integration Formula. For example if $P_2(x)$ value is calculated from the above general form:

$P_2(x)=(3x^2-1)/2$. The root of this is equal to $x_{1,2}=\pm 1/\sqrt{3}$. After finding the roots coeficients can be calculated as

$$c_1 x_1^{2j} + c_2 x_2^{2j} + ... + c_n x_n^{2j} = \frac{2}{2j+1}, \quad 0 \le j \le n \text{ or}$$

$$c_k = \frac{2(1-x_k^2)}{[nP_{n-1}(x_k)]^2}$$

Consider that Gauss-Legendre integration limits are -1 and 1

Region x=[-1,1] can be converted to z=[a,b] by changing the variables

$$z = \left(\frac{b-a}{2}\right)x + \left(\frac{b+a}{2}\right) = \alpha x + \beta$$

$$\alpha = \left(\frac{b-a}{2}\right) \quad \beta = \left(\frac{b+a}{2}\right)$$

In this case, integration Formula will became :

$$I_w(a,b) \cong \int_a^b f(z)dx \cong \alpha \sum_{k=1}^{n} c_k f(\alpha x_k + \beta)$$

Now the program using this approach to calculate Blassius flat plate problem is given below:

```
//=======================================================
// Blassius flat plate problem
// Dr. Turhan Coban
// =======================================================
//Blasius flat plate problem

import java.io.*;

class fm1 extends f_xi
{
//multivariable function
 double func(double x[],int x_ref)
 {double a=0;
   if(x_ref==2) a= -0.5*x[1]*x[3];
   if(x_ref==1) a= x[3];
   if(x_ref==0) a= x[2];
   return a;
 }
}

class f1x extends f_x
```

```
{  double Pr;
   double a[];
   double a1[][];
public f1x(double Pri,double w)
{ Pr=Pri;
  fm1 b2=new fm1();
  double y1[]=new double[3];
  y1[0]=0.0; //y0 first guess for f"
  y1[1]=0.0;
  y1[2]=0.3;
  //solution of the differential system :
  double xi1[]=new double[40];
  double fi1[]=new double[40];
  //shooting method for y[2]
  for(int i=1;i<40;i++)
  {
  a1=RK6(b2,0.0,w,y1,10000);
  xi1[i]=a1[2][10000];
  fi1[i]=y1[2];
  y1[2]+=0.02;  //change the boundary value
  }
  //curve fit to find actual boundary value
  double S1[][]=cubic_spline(xi1,fi1,0,0);
  y1[2]=funcSpline(S1,1.0);
  //System.out.println("y1[2]= "+y1[2]);
  a1=RK6(b2,0.0,w,y1,10000);
  // energy equation as integral equation solution
  double xx[]=a1[0];
  double yy[]=a1[1];
  a=PolynomialLSQ(xx,yy,10);
}
public double func(double x)
{   double x2=x*x,x3=x2*x,x4=x2*x2,x5=x4*x,x6=x5*x,x7=x6*x,x8=x7*x,x9=x8*x,x10=x9*x,x11=x10*x;
    double
y1=a[0]*x+a[1]*x2/2.0+a[2]*x3/3.0+a[3]*x4/4.0+a[4]*x5/5.0+a[5]*x6/6.0+a[6]*x7/7.0+a[7]*x8/8.0+a[8]*x9/9.0+a[9]*x10/10.0+a[10]*x1
1/11.0;
    double y=Math.exp(-Pr/2.0*y1);
//return funcPolynomialLSQ(a,x);
return y;
}
public static double[][] RK6(f_xi fp,double x0,double xn,double f0[],int N)
{
//6th order Runge Kutta Method
//fp  : given set of derivative functions dfj/dxi(fj,x)
// xo : initial value of the independent variable
// xn : final value of the independent variable
// f0 : initial value of the dependent variable
// N  : number of dependent variable to be calculated
// fi : dependent variable
double h=(xn-x0)/N;
int M=f0.length;
double fi[][];
fi=new double[M][N+1];
double xi[]=new double[M+1];
double k[]=new double[6];
int i,j;
double x;
for(j=0;j<M;j++)
  {
  fi[j][0]=f0[j];
  xi[j+1]=f0[j];
  }
for(x=x0,i=0;i<N;x+=h,i++)
  {
  for(j=1;j<=M;j++)
    {
    xi[0]=x;
    xi[j]=fi[j-1][i];
    k[0]=h*fp.func(xi,j-1);
    xi[0]=x+h/2.0;
    xi[j]=fi[j-1][i]+k[0]/2;
    k[1]=h*fp.func(xi,j-1);
    xi[0]=x+h/2.0;
    xi[j]=fi[j-1][i]+k[0]/4.0+k[1]/4.0;
    k[2]=h*fp.func(xi,j-1);
    xi[0]=x+h;
    xi[j]=fi[j-1][i]-k[1]+2.0*k[2];
```

```
    k[3]=h*fp.func(xi,j-1);
    xi[0]=x+2.0/3.0*h;
    xi[j]=fi[j-1][i]+7.0/27.0*k[0]+10.0/27.0*k[1]+1.0/27.0*k[3];
    k[4]=h*fp.func(xi,j-1);
    xi[0]=x+1.0/5.0*h;
    xi[j]=fi[j-1][i]+28.0/625.0*k[0]-1.0/5.0*k[1]+546.0/625.0*k[2]+54.0/625.0*k[3]-378/625.0*k[4];
    k[5]=h*fp.func(xi,j-1);
    fi[j-1][i+1]=fi[j-1][i]+k[0]/24.0+5.0*k[3]/48.0+27.0*k[4]/56.0+125.0*k[5]/336.0;
    xi[j]=fi[j-1][i];
    }
  }
double a[][]=new double[M+1][N+1];
for(x=x0,i=0;i<=N;x+=h,i++)
{
  a[0][i]=x;
  for(j=1;j<=M;j++)
  {
  a[j][i]=fi[j-1][i];
  }
}

return a;
}
  public static double [] thomas(double a[][],double r[])
  {
  //
  int n=a.length;
  double f[]=new double[n];
  double e[]=new double[n];
  double g[]=new double[n];
  double x[]=new double[n];
  for(int i=0;i<n;i++)     {f[i]=a[i][i];}
  for(int i=0;i<(n-1);i++) {g[i]=a[i][i+1];}
  for(int i=0;i<(n-1);i++) {e[i+1]=a[i+1][i];}
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
     }
  for(int k=1;k<n;k++)
     {r[k]=r[k]-e[k]*r[k-1];
     }
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
     {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
  return x;
  }

  public static double [] thomas(double f[],double e[],double g[],double r[])
  {
  int n=f.length;
  double x[]=new double[n];
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
     }
  for(int k=1;k<n;k++)
     {r[k]=r[k]-e[k]*r[k-1];
     }
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
     {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
  return x;
  }

  public static double [][] cubic_spline(double xi[],double yi[],double c0,double cn)
  {
  int n=xi.length;
  double h[]=new double[n];
  double w[]=new double[n];
  double f[]=new double[n];
  double e[]=new double[n];
  double g[]=new double[n];
  double d[]=new double[n];
  double x[]=new double[n];
  double S[][]=new double[4][n];
  int k;
  for(k=0;k<(n-1);k++)
```

```
    {h[k]=xi[k+1]-xi[k];
     w[k]=(yi[k+1]-yi[k])/h[k];
          }
  d[0]=c0;
  d[n-1]=cn;
  for(k=1;k<(n-1);k++)
    {d[k]=6.0*(w[k]-w[k-1]);}
  f[0]=1.0;
  f[n-1]=1.0;
  g[0]=0.0;
  g[n-1]=0.0;
  e[0]=0.0;
  e[n-1]=0.0;
  for(k=1;k<(n-1);k++)
  {f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];}
  S[2]=thomas(f,e,g,d);
  S[3]=xi;
  for(k=0;k<(n-1);k++)
    {S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
     S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
    }
  return S;
  }

  public static double funcSpline(double S[][],double x)
  {
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y=0;
  double hk=0;
  for(int k=0;k<(n-1);k++)
     {if(S[3][k]<=x && x<=S[3][k+1])
      {hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
       break;
       }
     }
       if(y==0 && S[3][n-2]<=x )
       {
                int k=n-2;
                hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
       }
  return y;
  }

    public static double[][]  funcSpline(double xi[],double yi[],int aradegersayisi)
    {
                //aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
                // ara değer sayısı 2 dir
                int n=xi.length;
                int nn=(n-1)*(aradegersayisi+1)+1;
                double z[][]=new double[2][nn];
    double S[][]=cubic_spline(xi,yi,0,0);
    double dx=0;
    int k=0;
    int i;
    for(i=0;i<(n-1);i++)
      {  z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);k++;
         for(int j=0;j<aradegersayisi;j++)
           {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
         z[0][k]=z[0][k-1]+dx;z[1][k]=funcSpline(S,z[0][k]);k++;}
      }
      z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);
    return z;
    }
// Polynomial least square
  public static double[] gausswithpartialpivot(double a[][],double b[])
  { //Gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
```

```
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //partial pivoting
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                   if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
                  b[k]=dummy;
                }
                //
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
                }
  }
  //backward substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}

public static double[] PolynomialLSQ(double xi[],double yi[],int n)
{ //Polynomial least square
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  for(j=0;j<n+1;j++)
     {if(i==0 && j==0) A[i][j]=l;
     else  for(k=0;k<l;k++) A[i][j] += Math.pow(xi[k],(i+j));
     }
     for(k=0;k<l;k++) { if(i==0)  B[i]+= yi[k];
                   else      B[i] += Math.pow(xi[k],i)*yi[k];}
  }
X=gausswithpartialpivot(A,B);
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
return X;
}

public static double funcPolynomialLSQ(double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=e.length;
```

```
double ff;
if(n!=0.0)
  { ff=e[n-1];
  for(int i=n-2;i>=0;i--)
   { ff=ff*x+e[i]; }
  }
else
  ff=0;
return ff;
}

public static double error(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcPolynomialLSQ(e,x[k]))*(y[k]-funcPolynomialLSQ(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

public static double[][]  funcPolynomialLSQ(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
 {
              //aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
              // ara değer sayisi 2 dir
               int n=xi.length;
              int nn=(n-1)*(aradegersayisi+1)+1;
              double z[][]=new double[2][nn];
    double E[]=PolynomialLSQ(xi,yi,polinomkatsayisi);
    System.out.println("katsayilar :\n"+Matrix.toStringT(E));
    double dx=0;
    int k=0;
    int i;
    for(i=0;i<(n-1);i++)
       {z[0][k]=xi[i];z[1][k]=funcPolynomialLSQ(E,z[0][k]);k++;
       for(int j=0;j<aradegersayisi;j++)
       {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
       z[0][k]=z[0][k-1]+dx;z[1][k]=funcPolynomialLSQ(E,z[0][k]);k++;}
       }
       z[0][k]=xi[i];z[1][k]=funcPolynomialLSQ(E,z[0][k]);
    return z;
 }
}
class fm2 extends f_xi
{ double Pr;
  fm2(double Pri) {Pr=Pri;}
// multivariable function
 double func(double x[],int x_ref)
 {
   //Blasius flat plate diferential equation
   //f"'+0.5*ff" = 0
   //f"'=-0.5*x[1]*x[3] f"=x[3] f'=x[2] f=x[1]]
   double a=0;
   if(x_ref==4) a= -0.5*Pr*x[1]*x[5];
   if(x_ref==3) a= x[5];
   if(x_ref==2) a= -0.5*x[1]*x[3];
   if(x_ref==1) a= x[3];
   if(x_ref==0) a= x[2];;
   return a;
 }
}

class diferansiyel1B3
{
  public static double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{
              //calculates legendre gauss-coefficients as coefficients of the integral
              //for n terms
              double EPS=3.0e-15;
              int m,j,i;
              double z1,z,xm,xl,pp,p3,p2,p1;
    //double x[]=new double[n];
    //double w[]=new double[n];
```

```
    double a[][]=new double[2][n];//a[0][i]=x[i]   a[1][i]=w[i]
            m=(n+1)/2;
            xm=0.5*(x2+x1);
            xl=0.5*(x2-x1);
            for (i=1;i<=m;i++)  {
                    z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                    do {
                            p1=1.0;
                            p2=0.0;
                            for (j=1;j<=n;j++) {
                                    p3=p2;
                                    p2=p1;
                                    p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                            }
                            pp=n*(z*p1-p2)/(z*z-1.0);
                            z1=z;
                            z=z1-p1/pp;
                    } while (Math.abs(z-z1) > EPS);
                    a[0][i-1]=xm-xl*z;
                    a[0][n-i]=xm+xl*z;
                    a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                    a[1][n-i]=a[1][i-1];
                            }
return a;
}

public static double integral(f_x f_xnt,double x1,double x2,int n)
{
//n : number of integral coefficients
// this routine first generates gauss legendre coefficients
// for [x1,x2] band
// then calculates gauss legendre integral
double a[][]=new double[2][n];
a=gauss_legendre_coefficients(x1,x2,n);
double z=0;
for(int i=0;i<n;i++)
{ z+=a[1][i]*f_xnt.func(a[0][i]);}
return z;
}
  public static void main(String args[]) throws IOException
  {  double
Pr[]={0.001,0.002,0.003,0.004,0.005,0.006,0.007,0.008,0.009,0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.
9,1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0,11,12,13,14,15.0,30.0,40.0,50,60,70,80,90,100,110.0,120.0,130.0
,140,150,160,170,180,190,200,300,400,500,600,700,800,900,1000};
    double
w[]={200,200,200,200,200,200,200,200,200,100.0,100,100,100,100,100,100,100,100,50,50,50,50,50,50,50,50,50,25,25,25,25,25,25,25,25,
25,10,10,10,10,10,10,10,10,10,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3};
    double Xi[]=new double[Pr.length];
    double Yi[]=new double[Pr.length];
    for(int i=0;i<Pr.length;i++)
    {
        f1x ff=new f1x(Pr[i],w[i]);
        Xi[i]=Pr[i];
        Yi[i]=1.0/integral(ff,0,w[i],100);
        System.out.println("Pr = "+Xi[i]+"teta'="+Yi[i]);
    }
    double z[][]={Xi,Yi};
    Text.printT(z,"Blassius flat palete NuxRex^-0.5 integral method");
    double Pr1=1.0;
    f1x ff=new f1x(Pr1,6.8);
    Text.printT(ff.a,"coefficient of curve fitting f(eta)");
  }}
```

Curve fitting coefficients $f(\eta)$ for as follows

a_0	-1.4214607426133100E-05
a_1	-2.7490470731684500E-05
a_2	1.6425932953124300E-01
a_3	3.6067764550437600E-03
a_4	-4.1973033042777800E-03
a_5	2.3935684285465600E-03
a_6	-1.1730013965284500E-03
a_7	2.9023032420905100E-04
a_8	-3.7647193571235300E-05
a_9	2.5045430633530600E-06
a_{10}	-6.8014637715603100E-08

Resulting $\theta'(\eta)$values are giving in the following table

Pr	Nux/Rex^0.5
0.001	1.729413262306600E-02
0.002	2.415544166595620E-02
0.003	2.930890400324250E-02
0.004	3.358120336476310E-02
0.005	3.729221792552660E-02
0.006	4.060584965389220E-02
0.007	4.361937115572090E-02
0.008	4.639608880595110E-02
0.009	4.897991598433690E-02
0.01	5.149458266428290E-02
0.02	7.040630431060980E-02
0.03	8.414370225346290E-02
0.04	9.526381880762620E-02
0.05	1.047452799486890E-01
0.06	1.130837306523290E-01
0.07	1.205699518112150E-01
0.08	1.273911396645470E-01
0.09	1.336758511642230E-01
0.1	1.397049456770640E-01
0.2	1.835650337268280E-01
0.3	2.140695063892500E-01
0.4	2.381456439762600E-01
0.5	2.583263894319400E-01
0.6	2.758548026899100E-01
0.7	2.914431591789320E-01
0.8	3.055415336430820E-01
0.9	3.184543073705920E-01
1	3.312954754476690E-01
1.1	3.425243531610520E-01
1.2	3.530677925048750E-01
1.3	3.630204591015030E-01
1.4	3.724578178515520E-01
1.5	3.814410953788210E-01
1.6	3.900207199841780E-01
1.7	3.982387708167000E-01
1.8	4.061307623149510E-01
1.9	4.144877482111790E-01
2	4.218486966103280E-01
3	4.844673987151180E-01
4	5.341031722208660E-01
5	5.759124157275020E-01
6	6.123969275242920E-01
7	6.449824545098310E-01
8	6.745675885350030E-01
9	7.017594347981730E-01
10	7.273604360913650E-01
11	7.509684948102970E-01
12	7.731770279049560E-01
13	7.941765123191470E-01
14	8.141190622552570E-01
15	8.331283108236750E-01
30	1.050198952039010E+00
40	1.155975784381680E+00
50	1.245256993110980E+00
60	1.323271776036590E+00
70	1.393014856049930E+00
80	1.456380317117830E+00
90	1.514651201226540E+00
100	1.568741602664960E+00
110	1.619328478420450E+00
120	1.666928800405430E+00
130	1.716730579095950E+00
140	1.759627853164530E+00
150	1.800524439991490E+00
160	1.839638248832950E+00
170	1.877151533095050E+00
180	1.913218451133240E+00
190	1.947970697495480E+00
200	1.981521771720000E+00
300	2.267844938698240E+00
400	2.495681758564390E+00
500	2.688011500291470E+00
600	2.856078449312840E+00
700	3.006326418245540E+00
800	3.142832760682760E+00
900	3.268360331085450E+00
1000	3.384877975113260E+00

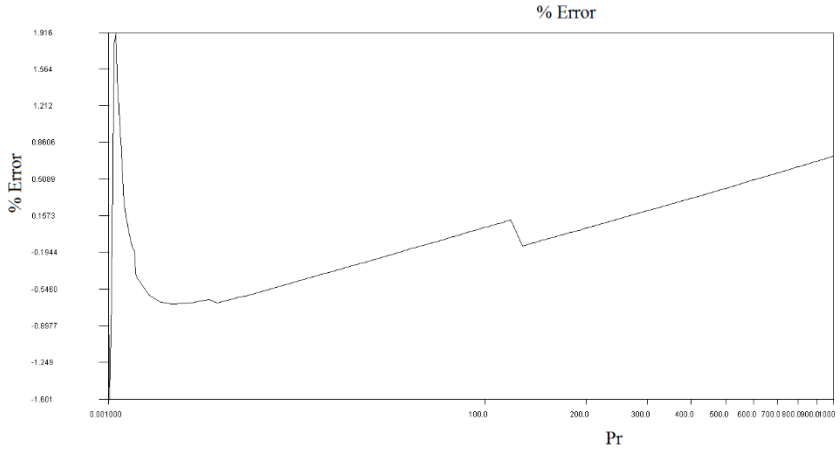
Partial continuous curve fitting values are as follows:

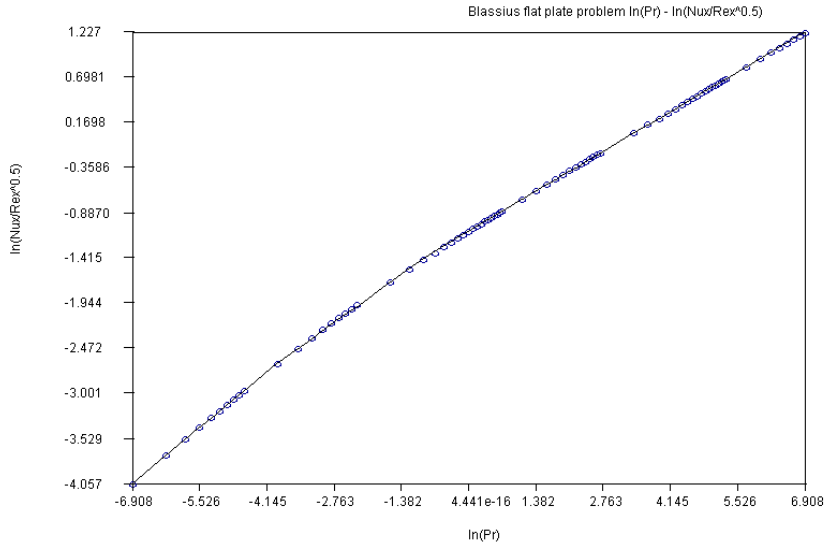
$$Nu_x = \frac{h_x x}{k} = 0.445584275 Re_x^{0.5} Pr^{0.47294717} \quad 10^{-3} \le Pr \le 10^{-2}$$

$$Nu_x = \frac{h_x x}{k} = 0.35271867 Re_x^{0.5} Pr^{0.40922589} \quad 10^{-2} \le Pr \le 0.5$$

$$Nu_x = \frac{h_x x}{k} = 0.33253715 Re_x^{0.5} Pr^{0.33694685} \quad 0.5 \le Pr \le 1000$$

As it is seen from the results equations are slightly differs from the previous set, but not that much. Let us look at equation errors and fitted lines as plots as well.



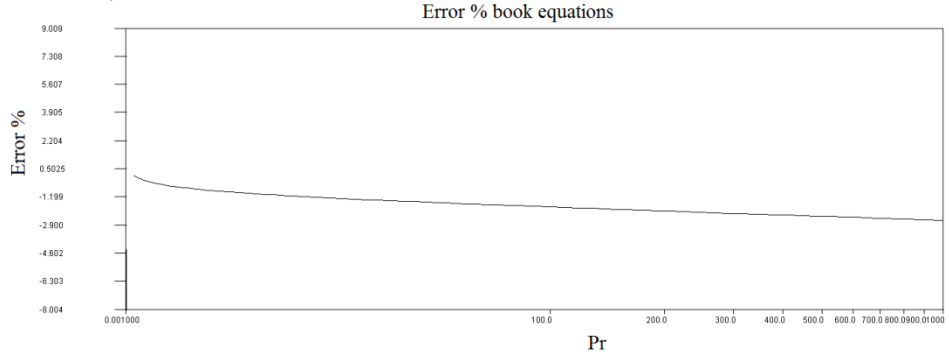


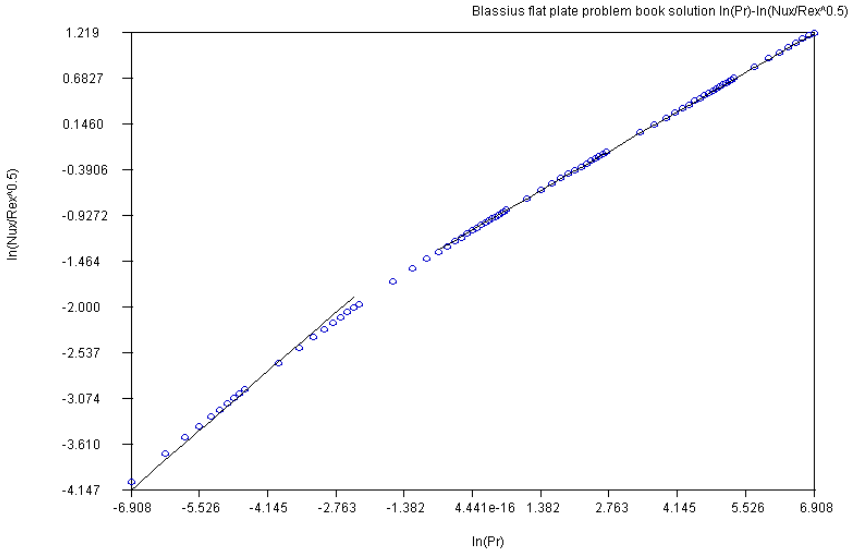
In most of the region plot errors are less than 1 %

As comparison solutions listed in books (a heat and mass transfer by Eckert and Drake) listed the following equations

$$Nu_x = \frac{h_x x}{k} = 0.5 Re_x^{0.5} Pr^{0.5} \quad 0.005 \le Pr \le 0.05$$

$$Nu_x = \frac{h_x x}{k} = 0.332 Re_x^{0.5} Pr^{0.3333333} \quad 0.6 \le Pr$$





This form of equation has an error less than 5% in most of the regions. As the last thing let us print out tables for η,f,f',f'',θ and θ' for **Pr=1.0**. In programs total of 10000 data points are used for accurate differential equation solution. In order to keep following table in manageable size one in every 50 data is printed out.

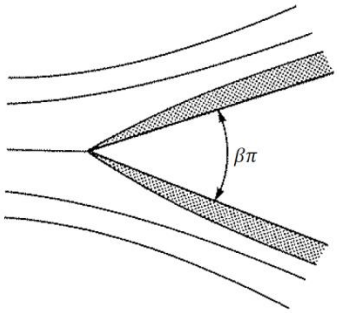
i	**η**	**f**	**f'**	**f''**	**θ**	**θ'**
0	0	0	0	0.3317675	0	0.33176751
50	0.034	1.80E-04	0.01128009	0.3317672	0.0112801	0.331767191
100	0.068	7.44E-04	0.02256015	0.3317648	0.0225601	0.331764796
150	0.102	0.0016915	0.03384006	0.3317582	0.0338401	0.331758161
200	0.136	0.0030223	0.04511965	0.3317451	0.0451197	0.331745124
250	0.17	0.0047366	0.05639867	0.3317235	0.0563987	0.331723523
300	0.204	0.0068344	0.06767678	0.3316912	0.0676768	0.331691196
350	0.238	0.0093156	0.0789536	0.331646	0.0789536	0.331645987
400	0.272	0.0121802	0.09022865	0.3315857	0.0902286	0.331585737
450	0.306	0.0154281	0.10150138	0.3315083	0.1015014	0.331508296
500	0.34	0.0190593	0.11277117	0.3314115	0.1127712	0.331411515
550	0.374	0.0230735	0.12403733	0.3312933	0.1240373	0.331293254
600	0.408	0.0274708	0.13529911	0.3311514	0.1352991	0.331151379
650	0.442	0.0322509	0.14655565	0.3309838	0.1465557	0.330983766
700	0.476	0.0374135	0.15780606	0.3307883	0.1578061	0.330788303
750	0.51	0.0429586	0.16904934	0.3305629	0.1690493	0.330562888
800	0.544	0.0488859	0.18028446	0.3303054	0.1802845	0.330305437
850	0.578	0.055195	0.19151029	0.3300139	0.1915103	0.330013881
900	0.612	0.0618856	0.20272563	0.3296862	0.2027256	0.329686173
950	0.646	0.0689573	0.21392922	0.3293203	0.2139292	0.329320285
1000	0.68	0.0764098	0.22511974	0.3289142	0.2251197	0.328914216
1050	0.714	0.0842425	0.23629578	0.328466	0.2362958	0.32846599
1100	0.748	0.0924549	0.24745588	0.3279737	0.2474559	0.327973662
1150	0.782	0.1010465	0.25859852	0.3274353	0.2585985	0.32743532
1200	0.816	0.1100167	0.26972209	0.3268491	0.2697221	0.326849089
1250	0.85	0.1193647	0.28082494	0.3262131	0.2808249	0.326213132
1300	0.884	0.1290899	0.29190534	0.3255257	0.2919053	0.325525652
1350	0.918	0.1391914	0.30296153	0.3247849	0.3029615	0.324784903
1400	0.952	0.1496685	0.31399166	0.3239892	0.3139917	0.323989183
1450	0.986	0.1605201	0.32499383	0.3231368	0.3249938	0.323136845
1500	1.02	0.1717453	0.3359661	0.3222263	0.3359661	0.322226296
1550	1.054	0.1833431	0.34690646	0.321256	0.3469065	0.321256005
1600	1.088	0.1953123	0.35781286	0.3202245	0.3578129	0.320224502
1650	1.122	0.2076517	0.36868318	0.3191304	0.3686832	0.319130384
1700	1.156	0.2203601	0.37951529	0.3179723	0.3795153	0.317972318
1750	1.19	0.2334362	0.39030699	0.316749	0.390307	0.316749044
1800	1.224	0.2468786	0.40105604	0.3154594	0.401056	0.315459381
1850	1.258	0.2606856	0.41176016	0.3141022	0.4117602	0.314102227
1900	1.292	0.2748559	0.42241705	0.3126766	0.422417	0.312676564
1950	1.326	0.2893878	0.43302436	0.3111815	0.4330244	0.311181463
2000	1.36	0.3042794	0.44357971	0.3096161	0.4435797	0.309616085
2050	1.394	0.3195291	0.45408071	0.3079797	0.4540807	0.307979684
2100	1.428	0.3351349	0.46452493	0.3062716	0.4645249	0.306271613
2150	1.462	0.3510949	0.47490992	0.3044913	0.4749099	0.304491323
2200	1.496	0.3674069	0.48523322	0.3026384	0.4852332	0.302638371
2250	1.53	0.384069	0.49549235	0.3007124	0.4954924	0.300712416

2300	1.564	0.4010788	0.50568483	0.2987132	0.5056848	0.298713226
2350	1.598	0.4184341	0.51580817	0.2966407	0.5158082	0.296640682
2400	1.632	0.4361324	0.52585987	0.2944948	0.5258599	0.294494775
2450	1.666	0.4541713	0.53583744	0.2922756	0.5358374	0.29227561
2500	1.7	0.4725482	0.54573838	0.2899834	0.5457384	0.28998341
2550	1.734	0.4912604	0.55556023	0.2876185	0.5555602	0.287618513
2600	1.768	0.5103054	0.56530052	0.2851814	0.5653005	0.285181378
2650	1.802	0.5296801	0.5749568	0.2826726	0.5749568	0.282672582
2700	1.836	0.5493818	0.58452664	0.2800928	0.5845266	0.280092822
2750	1.87	0.5694075	0.59400764	0.2774429	0.5940076	0.277442917
2800	1.904	0.5897541	0.60339745	0.2747238	0.6033974	0.274723804
2850	1.938	0.6104184	0.61269371	0.2719365	0.6126937	0.271936543
2900	1.972	0.6313973	0.62189412	0.2690823	0.6218941	0.269082309
2950	2.006	0.6526874	0.63099645	0.2661624	0.6309964	0.266162401
3000	2.04	0.6742855	0.63999846	0.2631782	0.6399985	0.263178232
3050	2.074	0.6961879	0.648898	0.2601313	0.648898	0.26013133
3100	2.108	0.7183913	0.65769297	0.2570233	0.657693	0.257023339
3150	2.142	0.740892	0.66638132	0.253856	0.6663813	0.253856013
3200	2.176	0.7636864	0.67496105	0.2506312	0.674961	0.250631216
3250	2.21	0.7867707	0.68343025	0.2473509	0.6834302	0.247350918
3300	2.244	0.8101412	0.69178705	0.2440172	0.6917871	0.24401719
3350	2.278	0.833794	0.70002968	0.2406322	0.7000297	0.240632204
3400	2.312	0.8577252	0.70815644	0.2371982	0.7081564	0.237198226
3450	2.346	0.8819308	0.71616569	0.2337176	0.7161657	0.233717615
3500	2.38	0.9064069	0.72405588	0.2301928	0.7240559	0.230192815
3550	2.414	0.9311493	0.73182556	0.2266264	0.7318256	0.226626353
3600	2.448	0.9561539	0.73947334	0.2230208	0.7394733	0.223020831
3650	2.482	0.9814166	0.74699795	0.2193789	0.746998	0.219378925
3700	2.516	1.0069331	0.75439819	0.2157034	0.7543982	0.215703377
3750	2.55	1.0326993	0.76167296	0.211997	0.761673	0.211996987
3800	2.584	1.0587107	0.76882126	0.2082626	0.7688213	0.208262613
3850	2.618	1.0849632	0.77584218	0.2045032	0.7758422	0.204503158
3900	2.652	1.1114524	0.78273491	0.2007216	0.7827349	0.200721569
3950	2.686	1.1381738	0.78949876	0.1969208	0.7894988	0.196920829
4000	2.72	1.1651231	0.79613312	0.1931039	0.7961331	0.19310395
4050	2.754	1.192296	0.80263748	0.189274	0.8026375	0.189273965
4100	2.788	1.2196879	0.80901146	0.1854339	0.8090115	0.185433924
4150	2.822	1.2472944	0.81525475	0.1815869	0.8152547	0.181586888
4200	2.856	1.2751111	0.82136717	0.1777359	0.8213672	0.177735917
4250	2.89	1.3031335	0.82734864	0.1738841	0.8273486	0.173884069
4300	2.924	1.3313573	0.83319917	0.1700344	0.8331992	0.170034391
4350	2.958	1.3597778	0.83891888	0.1661899	0.8389189	0.166189911
4400	2.992	1.3883907	0.84450801	0.1623536	0.844508	0.162353633
4450	3.026	1.4171916	0.84996687	0.1585285	0.8499669	0.158528532
4500	3.06	1.446176	0.8552959	0.1547175	0.8552959	0.154717542
4550	3.094	1.4753395	0.86049562	0.1509236	0.8604956	0.150923558
4600	3.128	1.5046778	0.86556665	0.1471494	0.8655666	0.147149424
4650	3.162	1.5341864	0.87050971	0.1433979	0.8705097	0.143397926
4700	3.196	1.563861	0.87532562	0.1396718	0.8753256	0.139671794
4750	3.23	1.5936974	0.88001528	0.1359737	0.8800153	0.135973688
4800	3.264	1.6236912	0.88457969	0.1323062	0.8845797	0.132306197
4850	3.298	1.6538382	0.88901993	0.1286718	0.8890199	0.128671835
4900	3.332	1.6841342	0.89333716	0.125073	0.8933372	0.125073034
4950	3.366	1.7145751	0.89753263	0.1215121	0.8975326	0.121512142
5000	3.4	1.7451566	0.90160766	0.1179914	0.9016077	0.117991415
5050	3.434	1.7758748	0.90556367	0.114513	0.9055637	0.114513018
5100	3.468	1.8067257	0.90940211	0.111079	0.9094021	0.11107902
5150	3.502	1.8377052	0.91312452	0.1076914	0.9131245	0.107691388
5200	3.536	1.8688094	0.91673253	0.104352	0.9167325	0.10435199
5250	3.57	1.9000344	0.92022779	0.1010626	0.9202278	0.101062588
5300	3.604	1.9313766	0.92361203	0.0978248	0.923612	0.097824838
5350	3.638	1.962832	0.92688703	0.0946403	0.926887	0.094640286
5400	3.672	1.9943971	0.93005463	0.0915104	0.9300546	0.091510371
5450	3.706	2.0260682	0.9331167	0.0884364	0.9331167	0.088436421
5500	3.74	2.0578417	0.93607517	0.0854197	0.9360752	0.085419653
5550	3.774	2.0897142	0.938932	0.0824612	0.938932	0.082461171
5600	3.808	2.1216822	0.94168918	0.079562	0.9416892	0.079561971
5650	3.842	2.1537424	0.94434875	0.0767229	0.9443487	0.076722936
5700	3.876	2.1858915	0.94691276	0.0739448	0.9469128	0.073944839
5750	3.91	2.2181263	0.9493833	0.0712283	0.9493833	0.071228345
5800	3.944	2.2504436	0.95176247	0.068574	0.9517625	0.068574011
5850	3.978	2.2828404	0.9540524	0.0659823	0.9540524	0.065982286
5900	4.012	2.3153136	0.95625521	0.0634535	0.9562552	0.063453518
5950	4.046	2.3478604	0.95837305	0.0609879	0.958373	0.060987949
6000	4.08	2.3804778	0.96040807	0.0585857	0.9604081	0.058585724
6050	4.114	2.4131631	0.96236243	0.0562469	0.9623624	0.05624689
6100	4.148	2.4459136	0.96423829	0.0539714	0.9642383	0.053971399
6150	4.182	2.4787267	0.96603778	0.0517591	0.9660378	0.051759112
6200	4.216	2.5115998	0.96776307	0.0496098	0.9677631	0.049609799
6250	4.25	2.5445303	0.96941628	0.0475231	0.9694163	0.04752315
6300	4.284	2.577516	0.97099954	0.0454988	0.9709995	0.045498768
6350	4.318	2.6105544	0.97251497	0.0435362	0.972515	0.043536181
6400	4.352	2.6436432	0.97396464	0.0416348	0.9739646	0.041634841

6450	4.386	2.6767804	0.97535064	0.0397941	0.9753506	0.039794131
6500	4.42	2.7099636	0.97667502	0.0380134	0.976675	0.038013366
6550	4.454	2.7431909	0.9779398	0.0362918	0.9779398	0.036291797
6600	4.488	2.7764603	0.97914698	0.0346286	0.979147	0.034628616
6650	4.522	2.8097699	0.98029853	0.033023	0.9802985	0.033022963
6700	4.556	2.8431177	0.9813964	0.0314739	0.9813964	0.031473922
6750	4.59	2.8765021	0.98244249	0.0299805	0.9824425	0.029980534
6800	4.624	2.9099212	0.98343869	0.0285418	0.9834387	0.028541793
6850	4.658	2.9433734	0.98438682	0.0271567	0.9843868	0.027156656
6900	4.692	2.9768571	0.98528871	0.025824	0.9852887	0.025824043
6950	4.726	3.0103707	0.98614611	0.0245428	0.9861461	0.024542842
7000	4.76	3.0439128	0.98696076	0.0233119	0.9869608	0.023311914
7050	4.794	3.077482	0.98773434	0.0221301	0.9877343	0.022130095
7100	4.828	3.1110768	0.98846851	0.0209962	0.9884685	0.0209962
7150	4.862	3.1446959	0.98916488	0.019909	0.9891649	0.019909025
7200	4.896	3.1783382	0.98982501	0.0188674	0.989825	0.018867354
7250	4.93	3.2120023	0.99045044	0.01787	0.9904504	0.017869958
7300	4.964	3.2456872	0.99104265	0.0169156	0.9910426	0.016915601
7350	4.998	3.2793917	0.99160308	0.016003	0.9916031	0.016003044
7400	5.032	3.3131147	0.99213314	0.015131	0.9921331	0.015131042
7450	5.066	3.3468554	0.99263418	0.0142984	0.9926342	0.014298354
7500	5.1	3.3806126	0.99310752	0.0135037	0.9931075	0.01350374
7550	5.134	3.4143854	0.99355443	0.012746	0.9935544	0.012745968
7600	5.168	3.4481731	0.99397616	0.0120238	0.9939762	0.012023812
7650	5.202	3.4819747	0.99437388	0.0113361	0.9943739	0.011336057
7700	5.236	3.5157895	0.99474875	0.0106815	0.9947488	0.0106815
7750	5.27	3.5496166	0.99510189	0.010059	0.9951019	0.010058953
7800	5.304	3.5834555	0.99543435	0.0094672	0.9954344	0.009467243
7850	5.338	3.6173053	0.99574718	0.0089052	0.9957472	0.008905215
7900	5.372	3.6511654	0.99604135	0.0083717	0.9960413	0.008371732
7950	5.406	3.6850353	0.99631782	0.0078657	0.9963178	0.007865679
8000	5.44	3.7189143	0.99657752	0.007386	0.9965775	0.007385961
8050	5.474	3.7528019	0.99682131	0.0069315	0.9968213	0.006931507
8100	5.508	3.7866975	0.99705004	0.0065013	0.99705	0.006501268
8150	5.542	3.8206007	0.99726451	0.0060942	0.9972645	0.006094222
8200	5.576	3.8545109	0.99746551	0.0057094	0.9974655	0.005709368
8250	5.61	3.8884278	0.99765376	0.0053457	0.9976538	0.005345736
8300	5.644	3.9223509	0.99782997	0.0050024	0.99783	0.005002378
8350	5.678	3.9562797	0.99799482	0.0046784	0.9979948	0.004678375
8400	5.712	3.9902141	0.99814895	0.0043728	0.9981489	0.004372834
8450	5.746	4.0241535	0.99829298	0.0040849	0.998293	0.004084891
8500	5.78	4.0580976	0.99842748	0.0038137	0.9984275	0.003813707
8550	5.814	4.0920462	0.99855303	0.0035585	0.998553	0.003558472
8600	5.848	4.1259989	0.99867014	0.0033184	0.9986701	0.003318403
8650	5.882	4.1599554	0.99877932	0.0030927	0.9987793	0.003092744
8700	5.916	4.1939155	0.99888104	0.0028808	0.998881	0.002880767
8750	5.95	4.227879	0.99897578	0.0026818	0.9989758	0.00268177
8800	5.984	4.2618456	0.99906394	0.0024951	0.9990639	0.002495079
8850	6.018	4.2958151	0.99914594	0.00232	0.9991459	0.002320043
8900	6.052	4.3297873	0.99922217	0.002156	0.9992222	0.002156042
8950	6.086	4.363762	0.999293	0.0020025	0.999293	0.002002477
9000	6.12	4.3977391	0.99935876	0.0018588	0.9993588	0.001858775
9050	6.154	4.4317182	0.99941978	0.0017244	0.9994198	0.00172439
9100	6.188	4.4656994	0.99947638	0.0015988	0.9994764	0.001598796
9150	6.222	4.4996825	0.99952884	0.0014815	0.9995288	0.001481494
9200	6.256	4.5336673	0.99957744	0.001372	0.9995774	0.001372005
9250	6.29	4.5676536	0.99962244	0.0012699	0.9996224	0.001269874
9300	6.324	4.6016414	0.99966407	0.0011747	0.9996641	0.001174667
9350	6.358	4.6356307	0.99970258	0.001086	0.9997026	0.00108597
9400	6.392	4.6696211	0.99973816	0.0010034	0.9997382	0.00100339
9450	6.426	4.7036127	0.99977103	9.27E-04	0.999771	9.27E-04
9500	6.46	4.7376055	0.99980138	8.55E-04	0.9998014	8.55E-04
9550	6.494	4.7715992	0.99982938	7.89E-04	0.9998294	7.89E-04
9600	6.528	4.8055938	0.9998552	7.27E-04	0.9998552	7.27E-04
9650	6.562	4.8395892	0.99987899	6.70E-04	0.999879	6.70E-04
9700	6.596	4.8735855	0.99990091	6.17E-04	0.9999009	6.17E-04
9750	6.63	4.9075824	0.99992108	5.68E-04	0.9999211	5.68E-04
9800	6.664	4.9415801	0.99993964	5.22E-04	0.9999396	5.22E-04
9850	6.698	4.9755783	0.9999567	4.80E-04	0.9999567	4.80E-04
9900	6.732	5.0095771	0.99997238	4.41E-04	0.9999724	4.41E-04
9950	6.766	5.0435764	0.99998678	4.05E-04	0.9999868	4.05E-04
10000	6.8	5.0775761	1	3.71E-04	1	3.71E-04

Wedge flow and Falkner and Skan similarity solution

In 1930 Falkner & Skan [91] discovered a whole family of situations for which the same transformation is possible, and in 1939 Golstein[92] investigated in detail the actual transformations and solutions.



If a potential flow of constant property fluid at a surface of a wedge is given as:
$u_s = Cx^m$ In this equation exponent m is connected with the opening angle $\beta\pi$ of the wedge by relation
$\beta = \frac{2m}{m+1}$
Continuity equation can be eliminated by introducing of stream function φ satisfying the conditions
$u = \frac{\partial \varphi}{\partial y} \quad v = -\frac{\partial \varphi}{\partial x}$
A similarity parameter $\eta(x, y)$ is introduced as a new variable.
$$\eta(x,y) = \sqrt{\frac{m+1}{2}}\frac{y}{x}\sqrt{Re_x} = \sqrt{\frac{m+1}{2}}y\sqrt{\frac{u_s}{\nu x}} = \sqrt{\frac{m+1}{2}}\sqrt{\frac{C}{\nu}}x^{(m-1)/2}y$$
And the new similarity function f which is function of this new η only
$$f(\eta) = \sqrt{\frac{m+1}{2}}\frac{\varphi}{u_s x}\sqrt{Re_x} = \sqrt{\frac{m+1}{2}}\frac{\varphi}{\sqrt{\nu\, u_s x}} = \sqrt{\frac{m+1}{2}}\frac{1}{\sqrt{\nu C}}x^{(m+1)/2}\varphi$$
the velocity component u can now be written as
$$u = \frac{\partial \varphi}{\partial y} = \sqrt{\frac{2C\nu}{m+1}}x^{(m+1)/2}\frac{df}{d\eta}\sqrt{\frac{m+1}{2}}\sqrt{\frac{C}{\nu}}x^{(m+1)/2} = Cx^m\frac{df}{d\eta} = u_s\frac{df}{d\eta}$$
the velocity component v can now be written as
$$u = -\frac{\partial \varphi}{\partial x} = -\sqrt{\frac{2C\nu}{m+1}}x^{(m-1)/2}\left(\frac{m+1}{2}f + \frac{m-1}{2}\frac{df}{d\eta}\eta\right)$$
Bernoulli equation
$$\frac{1}{\rho}\frac{\partial P}{\partial x} = -u_s\frac{du_s}{dx} = -mC^2x^{2m-1}$$
Equation converts to
$$\frac{d^3f}{d\eta^3} + f\frac{d^2f}{d\eta^2} - \beta\left[\left(\frac{df}{d\eta}\right)^2 - 1\right] = 0$$
Bounday conditions
$\eta=0 \ \frac{df}{d\eta} = 0 \ \ f = 0$
$\eta=\infty \ \frac{df}{d\eta} = 1$
For heat transfer constant surface temperature
$$u\frac{dT}{dx} + v\frac{dT}{dy} = \alpha\frac{d^2T}{dy^2}$$
$T_w - T_s = C_1x^\gamma$ where T_w is Wall surface temperature and T_s is the stream temperature. $\gamma = 0$ means constant wall temperature.
$\theta = \frac{T-T_s}{T_w-T_s}$ assuming that θ is function of η only
$$\frac{d^2\theta}{d\eta^2} + Prf\frac{d\theta}{d\eta} - (2-\beta)Pr\gamma\frac{df}{d\eta}\theta = 0$$
Boundary conditions
$\eta=0 \ \ \theta = 1$
$\eta=\infty \ \theta = 0$
Convertion of the equations:

$X[0]=\eta \quad X[1]=f \quad X[2]=f'=\frac{df}{d\eta} \quad X[3]=f''=\frac{d^2f}{d\eta^2} \quad X[4]=\theta \quad X[5]=\theta'=\frac{d\theta}{d\eta}$

Differential equation set becomes:

$\frac{dX[1]}{dX[0]}=X[2] \quad \frac{dX[2]}{dX[0]}=X[3] \quad \frac{dX[3]}{dX[0]}=-X[1]*X[3]+\beta*(X[2]*X[2]-1) \quad \frac{dX[4]}{dX[0]}=X[5]$

$\frac{dX[5]}{dX[0]}=-Pr*X[1]*X[5]+(2-\beta)*Pr*\gamma*X[2]*X[4]$

```
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
class fb extends f_xr
{
              public double func(double x,int i)
              { double y=Math.pow(x,0.25);
    double xx=1.0;
     if(i==0) xx=1;
     else if(i==1) xx=x;
      else if(i==2) xx=x*x;
       else if(i==3) xx=x*x*x;
        else if(i==4) xx=x*x*x*x;
         else if(i==5) xx=x*x*x*x*x;
          else if(i==6) xx=x*x*x*x*x*x;
           else if(i==7) xx=x*x*x*x*x*x*x;
            else if(i==8) xx=x*x*x*x*x*x*x*x;
             else if(i==9) xx=x*x*x*x*x*x*x*x*x;
              else if(i==10) xx=x*x*x*x*x*x*x*x*x*x;
  return xx;
              }
}
class f2 extends f_xj
{//adaptation function
fi_xi ff1;
public f2(fi_xi ffi) {ff1=ffi;}

public double func(double x[])
{
double ff=0.0;
double fa[]=ff1.func(x);
for(int i=0;i<fa.length;i++)  ff+=fa[i]*fa[i];
return ff;
}}

class fa extends fi_xi
{ // Nonlinear system of equation
double y[];
fm2 b3;
double w;
double a[][];
public fa(double Pr,double beta,double gamma,double yi[],double wi)
{ b3=new fm2(Pr,beta,gamma);
  y=new double[yi.length];
  for(int i=0;i<y.length;i++) y[i]=yi[i];
  w=wi;
}

public double[] func(double x[])
{
// shooting method
y[2]=x[0];
y[4]=x[1];
a=RK6(b3,0.0,w,y,10000);
double ff[]=new double[2];
ff[0]=a[2][10000]-1.0;
ff[1]=a[4][10000]-1.0;
return ff;
}
 //differenetial equation solution
public static double[][] RK6(f_xi fp,double x0,double xn,double f0[],int N)
{
//6th order Runge Kutta Method
//fp  : given set of derivative functions dfj/dxi(fj,x)
// xo : initial value of the independent variable
```

```
// xn : final value of the independent variable
// f0 : initial value of the dependent variable
// N  : number of dependent variable to be calculated
// fi : dependent variable
double h=(xn-x0)/N;
int M=f0.length;
double fi[][];
fi=new double[M][N+1];
double xi[]=new double[M+1];
double k[]=new double[6];
int i,j;
double x;
for(j=0;j<M;j++)
  {
  fi[j][0]=f0[j];
  xi[j+1]=f0[j];
  }
for(x=x0,i=0;i<N;x+=h,i++)
  {
  for(j=1;j<=M;j++)
    {
    xi[0]=x;
    xi[j]=fi[j-1][i];
    k[0]=h*fp.func(xi,j-1);
    xi[0]=x+h/2.0;
    xi[j]=fi[j-1][i]+k[0]/2;
    k[1]=h*fp.func(xi,j-1);
    xi[0]=x+h/2.0;
    xi[j]=fi[j-1][i]+k[0]/4.0+k[1]/4.0;
    k[2]=h*fp.func(xi,j-1);
    xi[0]=x+h;
    xi[j]=fi[j-1][i]-k[1]+2.0*k[2];
    k[3]=h*fp.func(xi,j-1);
    xi[0]=x+2.0/3.0*h;
    xi[j]=fi[j-1][i]+7.0/27.0*k[0]+10.0/27.0*k[1]+1.0/27.0*k[3];
    k[4]=h*fp.func(xi,j-1);
    xi[0]=x+1.0/5.0*h;
    xi[j]=fi[j-1][i]+28.0/625.0*k[0]-1.0/5.0*k[1]+546.0/625.0*k[2]+54.0/625.0*k[3]-378/625.0*k[4];
    k[5]=h*fp.func(xi,j-1);
    fi[j-1][i+1]=fi[j-1][i]+k[0]/24.0+5.0*k[3]/48.0+27.0*k[4]/56.0+125.0*k[5]/336.0;
    xi[j]=fi[j-1][i];
    }
  }
double a[][]=new double[M+1][N+1];
for(x=x0,i=0;i<=N;x+=h,i++)
{
  a[0][i]=x;
  for(j=1;j<=M;j++)
  {
  a[j][i]=fi[j-1][i];
  }
}
return a;
}
}

//Blassius flat plate differential equation
class fm2 extends f_xi
{ double Pr;
  double beta,gamma;
  fm2(double Pri,double betai,double gammai) {Pr=Pri;beta=betai;gamma=gammai;}
// multivariable function
 double func(double x[],int x_ref)
 {
   //Falkner - Skan diferential equation
   //f'''+0.5*ff''-beta*(f'^2-1) = 0
   //teta''+Pr*f*teta'-(2-beta)Pr*gamma*teta=0
   double a=0;
   if(x_ref==0) a= x[2];
   else if(x_ref==1) a= x[3];
   else if(x_ref==2) a= -x[1]*x[3]+beta*(x[2]*x[2]-1.0);
   else if(x_ref==3) a= x[5];
   else if(x_ref==4) a= -Pr*x[1]*x[5]+(2.0-beta)*Pr*gamma*x[2]*x[4];
   return a;
 }
}
```

```
public class falkner_skan2
{
                public static double[] inputdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }
public static String toString(double x[])
{ String s="";
int n=x.length;
  for(int k=0;k<n;k++)
  {s+=x[k]+" ";}
  return s;
}
public static double[] nelder(f_xj fnelder,double a[],double da[],int maxiteration,double tolerance,int printlist)
{
              int i,j;
double x[][]=new double[a.length+1][a.length];
double p[][]=new double[a.length+1][a.length+1];
for(i=0;i<x.length;i++)
   {for(j=0;j<x[0].length;j++)
    {if(i==j){x[i][j]=a[j]+da[j];p[i][j]=x[i][j];}
     else    {x[i][j]=a[j];p[i][j]=x[i][j]; }
    }
    p[i][j] = fnelder.func(p[i]);
   }


   // Inlet variable definitions
   // fnelder : abstract multivariable function f(x)
   // x : independent variable set of n+1 simplex elements
   // maxiteration : maximum iteration number
   // tolerance :
       int NDIMS = x.length-1;
       int NPTS = x.length;
       int FUNC = NDIMS;
       int ncalls = 0;
         ////// construct the starting simplex //////////////////
         //double p[][]=new double[NPTS][NPTS]; // [row][col] = [whichvx][coord,FUNC]
         double z[]=new double[NDIMS];
         double best = 1E99;
         ///////////////// calculate the first function values for the simplex /////////////////

         int iter=0;

         for (iter=1; iter<maxiteration; iter++)
         {
                    ////////////  define lo, nhi, hi (low high next_to_high //////////////
           int  ilo=0, ihi=0, inhi = -1; // -1 means missing
           double flo = p[0][FUNC];
           double fhi = flo;
           double pavg,sterr;
           for (i=1; i<NPTS; i++)
           {
             if (p[i][FUNC] < flo)
               {flo=p[i][FUNC]; ilo=i;}
             if (p[i][FUNC] > fhi)
               {fhi=p[i][FUNC]; ihi=i;}
           }
           double fnhi = flo;
           inhi = ilo;
           for (i=0; i<NPTS; i++)
            if ((i != ihi) && (p[i][FUNC] > fnhi))
              {fnhi=p[i][FUNC]; inhi=i;}
           ////////// exit criteria //////////////
           if ((iter % 4*NDIMS) == 0)
           {
```

```
    // calculate the avarage (including maximum value)
    pavg=0;
    for(i=0;i<NPTS;i++)
      pavg+=p[i][FUNC];
    pavg/=NPTS;
    double tot=0;
    if(printlist!=0)
    {  System.out.print(iter);
       for (j=0; j<=NDIMS; j++)
          { System.out.print(p[ilo][j]+" ");}
       System.out.println("");
    }
    for(i=0;i<NPTS;i++)
    { tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg);}
    sterr=Math.sqrt(tot/NPTS);
    //if(sterr < tolerance)
      { for (j=0; j<NDIMS; j++)
          { z[j]=p[ilo][j];}
        //break;
      }
    best = p[ilo][FUNC];
}

///// calculate avarage without maximum value //////

double ave[] = new double[NDIMS];
for (j=0; j<NDIMS; j++)
  ave[j] = 0;
for (i=0; i<NPTS; i++)
  if (i != ihi)
    for (j=0; j<NDIMS; j++)
      ave[j] += p[i][j];
for (j=0; j<NDIMS; j++)
   ave[j] /= (NPTS-1);


///////// reflect ////////////////

double r[] = new double[NDIMS];
for (j=0; j<NDIMS; j++)
  r[j] = 2*ave[j] - p[ihi][j];
double fr = fnelder.func(r);

if ((flo <= fr) && (fr < fnhi))  // in zone: accept
{
    for (j=0; j<NDIMS; j++)
      p[ihi][j] = r[j];
    p[ihi][FUNC] = fr;
    continue;
}

if (fr < flo)  //// expand
{
    double e[] = new double[NDIMS];
    for (j=0; j<NDIMS; j++)
      e[j] = 3*ave[j] - 2*p[ihi][j];
    double fe = fnelder.func(e);
    if (fe < fr)
    {
       for (j=0; j<NDIMS; j++)
         p[ihi][j] = e[j];
       p[ihi][FUNC] = fe;
       continue;
    }
    else
    {
       for (j=0; j<NDIMS; j++)
         p[ihi][j] = r[j];
       p[ihi][FUNC] = fr;
       continue;
    }
}

////////////// shrink:

if (fr < fhi)
{
```

```
                double c[] = new double[NDIMS];
                for (j=0; j<NDIMS; j++)
                  c[j] = 1.5*ave[j] - 0.5*p[ihi][j];
                double fc = fnelder.func(c);
                if (fc <= fr)
                {
                   for (j=0; j<NDIMS; j++)
                     p[ihi][j] = c[j];
                   p[ihi][FUNC] = fc;
                   continue;
                }
                else   /////// daralt
                {
                   for (i=0; i<NPTS; i++)
                     if (i != ilo)
                     {
                        for (j=0; j<NDIMS; j++)
                          p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                        p[i][FUNC] = fnelder.func(p[i]);
                     }
                   continue;
                }
             }

             if (fr >= fhi)   ///
             {
                double cc[] = new double[NDIMS];
                for (j=0; j<NDIMS; j++)
                  cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
                double fcc = fnelder.func(cc);
                if (fcc < fhi)
                {
                   for (j=0; j<NDIMS; j++)
                     p[ihi][j] = cc[j];
                   p[ihi][FUNC] = fcc;
                   continue;
                }
                else    /////////
                {
                   for (i=0; i<NPTS; i++)
                     if (i != ilo)
                     {
                        for (j=0; j<NDIMS; j++)
                          p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                        p[i][FUNC] = fnelder.func(p[i]);
                     }
                }
             }
          }
       return z;
}
    public static double[] nelder(f_xj fnelder,double a[],double da[],double tolerance)
    {return nelder(fnelder,a,da,500,tolerance,0);}

    public static double[] nelder(f_xj fnelder,double a[],double da[])
    {return nelder(fnelder,a,da,500,1.0e-10,0);}

    public static double[] nelder(f_xj fnelder,double a[])
    {
                    double [] da=new double[a.length];
          for(int i=0;i<a.length;i++) da[i]=0.1*a[i];
          return nelder(fnelder,a,da);
    }
// Polynomial least square
  public static double[] gausswithpartialpivot(double a[][],double b[])
  { //Gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //partial pivoting
              p=k;
```

```
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                   if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
                  b[k]=dummy;
                }
                //
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
                }
  }
  //backward substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}

public static double[] PolynomialLSQ(double xi[],double yi[],int n)
{ //Polynomial least square
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  for(j=0;j<n+1;j++)
     {if(i==0 && j==0) A[i][j]=l;
     else  for(k=0;k<l;k++) A[i][j] += Math.pow(xi[k],(i+j));
     }
     for(k=0;k<l;k++) { if(i==0)  B[i]+= yi[k];
                   else      B[i] += Math.pow(xi[k],i)*yi[k];}
  }
X=gausswithpartialpivot(A,B);
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
return X;
}

public static double funcPolynomialLSQ(double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=e.length;
double ff;
if(n!=0.0)
  { ff=e[n-1];
  for(int i=n-2;i>=0;i--)
   { ff=ff*x+e[i]; }
  }
else
  ff=0;
```

```
return ff;
}

public static double error(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcPolynomialLSQ(e,x[k]))*(y[k]-funcPolynomialLSQ(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

public static double[][]  funcPolynomialLSQ(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
 {
              //aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
              // ara değer sayisi 2 dir
               int n=xi.length;
              int nn=(n-1)*(aradegersayisi+1)+1;
              double z[][]=new double[2][nn];
    double E[]=PolynomialLSQ(xi,yi,polinomkatsayisi);
    System.out.println("katsayilar :\n"+Matrix.toStringT(E));
    double dx=0;
    int k=0;
    int i;
    for(i=0;i<(n-1);i++)
      {z[0][k]=xi[i];z[1][k]=funcPolynomialLSQ(E,z[0][k]);k++;
      for(int j=0;j<aradegersayisi;j++)
      {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
      z[0][k]=z[0][k-1]+dx;z[1][k]=funcPolynomialLSQ(E,z[0][k]);k++;}
      }
      z[0][k]=xi[i];z[1][k]=funcPolynomialLSQ(E,z[0][k]);
    return z;
 }
//general least square curve fitting

public static double[] GeneralLeastSquare(double xi[],double yi[],int n)
{
fb f=new fb();
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  B[i]=0;
               for(j=0;j<np1;j++)
    {
               A[i][j]=0.0;
    for(k=0;k<l;k++) A[i][j]+=f.func(xi[k],i)*f.func(xi[k],j);
    }
    for(k=0;k<l;k++) B[i]+= f.func(xi[k],i)*yi[k];
  }
//System.out.println("A = \n"+Matrix.toString(A));
//System.out.println("B = \n"+Matrix.toStringT(B));
X=gausswithpartialpivot(A,B);
//X=B/A;
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
Text.printT(X);
return X;
}

public static double funcGeneralLeastSquare(double e[],double x)
{
// this function calculates the value of
```

```
// least square curve fitting function
fb f=new fb();
int n=e.length;
double ff=0;
if(n!=0.0)
  {
  for(int i=n-1;i>=0;i--)
   {ff+=e[i]*f.func(x,i);}
  }
return ff;
}

public static double hata(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcGeneralLeastSquare(e,x[k]))*(y[k]-funcGeneralLeastSquare(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}
 public static double[][]  funcGeneralLeastSquare(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
 {  // aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
                // ara değer sayisi 2 dir
                 int n=xi.length;
                int nn=(n-1)*(aradegersayisi+1)+1;
                double z[][]=new double[2][nn];
    double E[]=GeneralLeastSquare(xi,yi,polinomkatsayisi);
    double dx=0;
    int k=0;
    int i;
    for(i=0;i<(n-1);i++)
       {z[0][k]=xi[i];z[1][k]=funcGeneralLeastSquare(E,z[0][k]);k++;
       for(int j=0;j<aradegersayisi;j++)
       {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
       z[0][k]=z[0][k-1]+dx;z[1][k]=funcGeneralLeastSquare(E,z[0][k]);k++;}
       }
       z[0][k]=xi[i];z[1][k]=funcGeneralLeastSquare(E,z[0][k]);
    return z;
 }
public static void main(String arg[]) throws IOException
{
  double y[]={0,0,0.1,0,0.1};
 double x[]={0.33,0.0173};//inputdata(s);
 fa ff;
 f2 f;
 double r1[];
 double beta=0.1;
 double gamma=0.0;
 double Pr1[]={0.001,0.002,0.003,0.004,0.005,0.006,0.007,0.008,0.009,0.01};
 double w1[]={200,200,200,200,200,200,200,200,200,100.0};
 double Pr2[]={0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.1,0.2,0.3,0.4,0.5};
 double w2[]={100.0,100.0,100,100,100,100,100,100,100,100,50,50,50,50,50};
 double Pr3[]={0.5,0.6,0.7,0.8,0.9,1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0,11,12,13,14,15.0,30.0,40.0,50};
 double w3[]={50,50,50,50,25,25,25,25,25,25,25,25,25,10,10,10,10,10,10,10,10,10,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7};
 double Pr4[]={50,60,70,80,90,100,110.0,120.0,130.0,140,150,160,170,180,190,200,300,400,500,600,700,800,900,1000};
 double w4[]={5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3,5.3};
 double xi1[]=new double[Pr1.length];
 double yi1[]=new double[Pr1.length];
 double Xi1[]=new double[Pr1.length];
 double Yi1[]=new double[Pr1.length];
 double xi2[]=new double[Pr2.length];
 double yi2[]=new double[Pr2.length];
 double Xi2[]=new double[Pr2.length];
 double Yi2[]=new double[Pr2.length];
 double xi3[]=new double[Pr3.length];
 double yi3[]=new double[Pr3.length];
 double Xi3[]=new double[Pr3.length];
 double Yi3[]=new double[Pr3.length];
 double xi4[]=new double[Pr4.length];
 double yi4[]=new double[Pr4.length];
 double Xi4[]=new double[Pr4.length];
 double Yi4[]=new double[Pr4.length];
```

```
 double b1[];
 double b2[];
 double b3[];
 double b4[];
 for(int i=0;i<Pr1.length;i++)
 {ff=new fa(Pr1[i],beta,gamma,y,w1[i]*beta);
  f=new f2(ff);
  r1=nelder(f,x);
  xi1[i]=Pr1[i];
  yi1[i]=r1[1];
  Xi1[i]=Math.log(Pr1[i]);
  Yi1[i]=Math.log(r1[1]);
  System.out.println("i="+i+"Pr="+xi1[i]+" Nux/Rex^0.5="+yi1[i]);
  }
  for(int i=0;i<Pr2.length;i++)
  {ff=new fa(Pr2[i],beta,gamma,y,w2[i]*beta);
  f=new f2(ff);
  r1=nelder(f,x);
  xi2[i]=Pr2[i];
  yi2[i]=r1[1];
  Xi2[i]=Math.log(Pr2[i]);
  Yi2[i]=Math.log(r1[1]);
  System.out.println("i = "+i+" Pr= "+xi2[i]+" Nux/Rex^0.5="+yi2[i]);
  }
  for(int i=0;i<Pr3.length;i++)
  {ff=new fa(Pr3[i],beta,gamma,y,w3[i]*beta);
  f=new f2(ff);
  r1=nelder(f,x);
  xi3[i]=Pr3[i];
  yi3[i]=r1[1];
  Xi3[i]=Math.log(Pr3[i]);
  Yi3[i]=Math.log(r1[1]);
  System.out.println("i = "+i+"Pr = "+xi3[i]+" Nux/Rex^0.5="+yi3[i]);
  }
  for(int i=0;i<Pr4.length;i++)
  {ff=new fa(Pr4[i],beta,gamma,y,w4[i]*beta);
  f=new f2(ff);
  r1=nelder(f,x);
  xi4[i]=Pr4[i];
  yi4[i]=r1[1];
  Xi4[i]=Math.log(Pr4[i]);
  Yi4[i]=Math.log(r1[1]);
  System.out.println("i = "+i+" Pr="+xi4[i]+" Nux/Rex^0.5="+yi4[i]);
  }
  b1=GeneralLeastSquare(Xi1,Yi1,1);
  System.out.println("0.001<Pr<0.01 a="+Math.exp(b1[0])+" b="+b1[1]);
  b2=GeneralLeastSquare(Xi2,Yi2,1);
  System.out.println("0.01<Pr<0.5 a="+Math.exp(b2[0])+" b="+b2[1]);
  b3=GeneralLeastSquare(Xi3,Yi3,1);
  System.out.println("0.5<Pr<50 a="+Math.exp(b3[0])+" b="+b3[1]);
  b4=GeneralLeastSquare(Xi4,Yi4,1);
  System.out.println("50<Pr<1000 a="+Math.exp(b4[0])+" b="+b4[1]);
}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" falkner_skan2
i=0Pr=0.001 Nux/Rex^0.5=0.052838585958506955
i=1Pr=0.002 Nux/Rex^0.5=0.05568439004805982
i=2Pr=0.003 Nux/Rex^0.5=0.058528451086160366
i=3Pr=0.004 Nux/Rex^0.5=0.06136255984188095
i=4Pr=0.005 Nux/Rex^0.5=0.06417932292947877
i=5Pr=0.006 Nux/Rex^0.5=0.0669721969228364
i=6Pr=0.007 Nux/Rex^0.5=0.0697354959029941
i=7Pr=0.008 Nux/Rex^0.5=0.07246437637025133
i=8Pr=0.009 Nux/Rex^0.5=0.07515480379431305
i=9Pr=0.01 Nux/Rex^0.5=0.11205588920743956
i = 0 Pr= 0.01 Nux/Rex^0.5=0.11205588920743956
i = 1 Pr= 0.02 Nux/Rex^0.5=0.12395151866250631
i = 2 Pr= 0.03 Nux/Rex^0.5=0.13553863140044367
i = 3 Pr= 0.04 Nux/Rex^0.5=0.14671316479801308
i = 4 Pr= 0.05 Nux/Rex^0.5=0.15741124821744934
i = 5 Pr= 0.06 Nux/Rex^0.5=0.16760215238090806
i = 6 Pr= 0.07 Nux/Rex^0.5=0.1772802936278064
i = 7 Pr= 0.08 Nux/Rex^0.5=0.18645770511575543
i = 8 Pr= 0.09 Nux/Rex^0.5=0.19515771000451182
```

i = 9 Pr= 0.1 Nux/Rex^0.5=0.20341002695454757
i = 10 Pr= 0.2 Nux/Rex^0.5=0.2841350963165023
i = 11 Pr= 0.3 Nux/Rex^0.5=0.32059954900699383
i = 12 Pr= 0.4 Nux/Rex^0.5=0.3530598178408308
i = 13 Pr= 0.5 Nux/Rex^0.5=0.3819538367853976
i = 0Pr = 0.5 Nux/Rex^0.5=0.3819538367853976
i = 1Pr = 0.6 Nux/Rex^0.5=0.4078354022748689
i = 2Pr = 0.7 Nux/Rex^0.5=0.43122731650764307
i = 3Pr = 0.8 Nux/Rex^0.5=0.45257066060822737
i = 4Pr = 0.9 Nux/Rex^0.5=0.5226234080781739
i = 5Pr = 1.0 Nux/Rex^0.5=0.5351747910351455
i = 6Pr = 1.1 Nux/Rex^0.5=0.5474862124668214
i = 7Pr = 1.2 Nux/Rex^0.5=0.5595561046378349
i = 8Pr = 1.3 Nux/Rex^0.5=0.5713842346231784
i = 9Pr = 1.4 Nux/Rex^0.5=0.5829715406219594
i = 10Pr = 1.5 Nux/Rex^0.5=0.5943199765440061
i = 11Pr = 1.6 Nux/Rex^0.5=0.605432366337465
i = 12Pr = 1.7 Nux/Rex^0.5=0.6163122690177371
i = 13Pr = 1.8 Nux/Rex^0.5=1.07843526057971
i = 14Pr = 1.9 Nux/Rex^0.5=1.0827301489808536
i = 15Pr = 2.0 Nux/Rex^0.5=1.087017850234281
i = 16Pr = 3.0 Nux/Rex^0.5=1.1294826213403983
i = 17Pr = 4.0 Nux/Rex^0.5=1.1711499568140122
i = 18Pr = 5.0 Nux/Rex^0.5=1.211963030834299
i = 19Pr = 6.0 Nux/Rex^0.5=1.2518790389776722
i = 20Pr = 7.0 Nux/Rex^0.5=1.2908679003183552
i = 21Pr = 8.0 Nux/Rex^0.5=1.3289108735798996
i = 22Pr = 9.0 Nux/Rex^0.5=1.6894698550922782
i = 23Pr = 10.0 Nux/Rex^0.5=1.7171634489663514
i = 24Pr = 11.0 Nux/Rex^0.5=1.744566745589914
i = 25Pr = 12.0 Nux/Rex^0.5=1.7716743069707244
i = 26Pr = 13.0 Nux/Rex^0.5=1.7984816070930691
i = 27Pr = 14.0 Nux/Rex^0.5=1.8249849823816064
i = 28Pr = 15.0 Nux/Rex^0.5=1.851181581603489
i = 29Pr = 30.0 Nux/Rex^0.5=2.2073023003762247
i = 30Pr = 40.0 Nux/Rex^0.5=2.409227460829409
i = 31Pr = 50.0 Nux/Rex^0.5=2.5876756067443543
i = 0 Pr=50.0 Nux/Rex^0.5=2.866196403058497
i = 1 Pr=60.0 Nux/Rex^0.5=3.0245877351786525
i = 2 Pr=70.0 Nux/Rex^0.5=3.1718665682913283
i = 3 Pr=80.0 Nux/Rex^0.5=3.3091378766342614
i = 4 Pr=90.0 Nux/Rex^0.5=3.437487478797009
i = 5 Pr=100.0 Nux/Rex^0.5=3.55792449115799
i = 6 Pr=110.0 Nux/Rex^0.5=3.6713556427982894
i = 7 Pr=120.0 Nux/Rex^0.5=3.7785789246436887
i = 8 Pr=130.0 Nux/Rex^0.5=3.8802879432061186
i = 9 Pr=140.0 Nux/Rex^0.5=3.977081466292545
i = 10 Pr=150.0 Nux/Rex^0.5=4.06947486723898
i = 11 Pr=160.0 Nux/Rex^0.5=4.157911638100899
i = 12 Pr=170.0 Nux/Rex^0.5=4.242774052422481
i = 13 Pr=180.0 Nux/Rex^0.5=4.324392594357212
i = 14 Pr=190.0 Nux/Rex^0.5=4.403054068457771
i = 15 Pr=200.0 Nux/Rex^0.5=4.479008456728227
i = 16 Pr=300.0 Nux/Rex^0.5=5.127398066520598
i = 17 Pr=400.0 Nux/Rex^0.5=5.643467206121851
i = 18 Pr=500.0 Nux/Rex^0.5=6.079180893347311
i = 19 Pr=600.0 Nux/Rex^0.5=6.459981724270481
i = 20 Pr=700.0 Nux/Rex^0.5=6.800451316074847
i = 21 Pr=800.0 Nux/Rex^0.5=7.109816763950922
i = 22 Pr=900.0 Nux/Rex^0.5=7.394330723521469
i = 23 Pr=1000.0 Nux/Rex^0.5=7.658449093385441

0.001<Pr<0.01 a=0.23533274210000105 b=0.23167892227279793
0.01<Pr<0.5 a=0.46410108309973847 b=0.34259990608566376
0.5<Pr<50 a=0.5649974547131759 b=0.43945107376090803
50<Pr<1000 a=0.7763291971873808 b=0.3310579556701941

For $\beta = 0.1$ opening angle =0.314159 radian = 18 degree $\gamma = 0$ (isothermal plate)

$Nu = \mathbf{0.23533274210000105}\ \mathrm{Re}^{0.5} Pr^{\mathbf{0.23167892227279793}} \quad 0.001 \le Pr \le 0.01$

$Nu = 0.46410108309973847\ \mathrm{Re}^{0.5} Pr^{0.34259990608566376} \quad 0.01 \le Pr \le 0.5$

$Nu = 0.5649974547131759\ \mathrm{Re}^{0.5} Pr^{0.43945107376090803} \quad 0.5 \le Pr \le 50$

$Nu = 0.7763291971873808\ \mathrm{Re}^{0.5} Pr^{0.3310579556701941} \quad 50 \le Pr \le 1000$

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" falkner_skan2
i=0Pr=0.001 Nux/Rex^0.5=0.3335480441892363
i=1Pr=0.002 Nux/Rex^0.5=0.333762732229488
i=2Pr=0.003 Nux/Rex^0.5=0.3339773973373252
i=3Pr=0.004 Nux/Rex^0.5=0.3341920393961866
i=4Pr=0.005 Nux/Rex^0.5=0.3344066582896734
i=5Pr=0.006 Nux/Rex^0.5=0.33462125390162084
i=6Pr=0.007 Nux/Rex^0.5=0.3348358261160087
i=7Pr=0.008 Nux/Rex^0.5=0.33505037481704425
i=8Pr=0.009 Nux/Rex^0.5=0.3352648998890954
i=9Pr=0.01 Nux/Rex^0.5=0.33547940121673925
i = 0 Pr= 0.01 Nux/Rex^0.5=0.253421061894859
i = 1 Pr= 0.02 Nux/Rex^0.5=0.2568358547768611
i = 2 Pr= 0.03 Nux/Rex^0.5=0.2602435827426917
i = 3 Pr= 0.04 Nux/Rex^0.5=0.26364347566404794
i = 4 Pr= 0.05 Nux/Rex^0.5=0.2670347895158677
i = 5 Pr= 0.06 Nux/Rex^0.5=0.27041680663249856
i = 6 Pr= 0.07 Nux/Rex^0.5=0.27378883589400793
i = 7 Pr= 0.08 Nux/Rex^0.5=0.2771502128452698
i = 8 Pr= 0.09 Nux/Rex^0.5=0.28050029975043916
i = 9 Pr= 0.1 Nux/Rex^0.5=0.28383848558575353
i = 10 Pr= 0.2 Nux/Rex^0.5=0.316450169918763
i = 11 Pr= 0.3 Nux/Rex^0.5=0.34738929942134955
i = 12 Pr= 0.4 Nux/Rex^0.5=0.376435621274995
i = 13 Pr= 0.5 Nux/Rex^0.5=0.40353923080931065
i = 0Pr = 0.5 Nux/Rex^0.5=0.3967777602996435
i = 1Pr = 0.6 Nux/Rex^0.5=0.4245252151748051
i = 2Pr = 0.7 Nux/Rex^0.5=0.4495786853586725
i = 3Pr = 0.8 Nux/Rex^0.5=0.4724305505302953
i = 4Pr = 0.9 Nux/Rex^0.5=0.5357643115864267
i = 5Pr = 1.0 Nux/Rex^0.5=0.5495806291641702
i = 6Pr = 1.1 Nux/Rex^0.5=0.5631114536497109
i = 7Pr = 1.2 Nux/Rex^0.5=0.5763553971849682
i = 8Pr = 1.3 Nux/Rex^0.5=0.5893128347762455
i = 9Pr = 1.4 Nux/Rex^0.5=0.6019856624409455
i = 10Pr = 1.5 Nux/Rex^0.5=0.6143770703546839
i = 11Pr = 1.6 Nux/Rex^0.5=0.6264913330571373
i = 12Pr = 1.7 Nux/Rex^0.5=0.6383336178903841
i = 13Pr = 1.8 Nux/Rex^0.5=1.080860506314767
i = 14Pr = 1.9 Nux/Rex^0.5=1.085287745456006
i = 15Pr = 2.0 Nux/Rex^0.5=1.089707490793113
i = 16Pr = 3.0 Nux/Rex^0.5=1.1334742878878064
i = 17Pr = 4.0 Nux/Rex^0.5=1.1764058811902651
i = 18Pr = 5.0 Nux/Rex^0.5=1.2184405505728213
i = 19Pr = 6.0 Nux/Rex^0.5=1.2595320622144668
i = 20Pr = 7.0 Nux/Rex^0.5=1.2996482023212421
i = 21Pr = 8.0 Nux/Rex^0.5=1.338769212589115
i = 22Pr = 9.0 Nux/Rex^0.5=1.6935839761605147
i = 23Pr = 10.0 Nux/Rex^0.5=1.7217037719289932
i = 24Pr = 11.0 Nux/Rex^0.5=1.749525714515282
i = 25Pr = 12.0 Nux/Rex^0.5=1.777044154610579
i = 26Pr = 13.0 Nux/Rex^0.5=1.804254401475994
i = 27Pr = 14.0 Nux/Rex^0.5=1.8311526701643182
i = 28Pr = 15.0 Nux/Rex^0.5=1.8577360281534534
i = 29Pr = 30.0 Nux/Rex^0.5=2.218713821164285
i = 30Pr = 40.0 Nux/Rex^0.5=2.4230524425125015
i = 31Pr = 50.0 Nux/Rex^0.5=2.603460669031582
i = 0 Pr=50.0 Nux/Rex^0.5=2.8746451804241273
i = 1 Pr=60.0 Nux/Rex^0.5=3.0341867308060273
i = 2 Pr=70.0 Nux/Rex^0.5=3.182477947059634
i = 3 Pr=80.0 Nux/Rex^0.5=3.3206487477818905
i = 4 Pr=90.0 Nux/Rex^0.5=3.4498065393990984
i = 5 Pr=100.0 Nux/Rex^0.5=3.5709781453825187
i = 6 Pr=110.0 Nux/Rex^0.5=3.6850843511054094
i = 7 Pr=120.0 Nux/Rex^0.5=3.7929340944342878
i = 8 Pr=130.0 Nux/Rex^0.5=3.89522942779183
i = 9 Pr=140.0 Nux/Rex^0.5=3.9925756225433635
i = 10 Pr=150.0 Nux/Rex^0.5=4.0854930792072555
i = 11 Pr=160.0 Nux/Rex^0.5=4.174429208393844
i = 12 Pr=170.0 Nux/Rex^0.5=4.259769374018717
i = 13 Pr=180.0 Nux/Rex^0.5=4.341846531713771
i = 14 Pr=190.0 Nux/Rex^0.5=4.420949492777605
i = 15 Pr=200.0 Nux/Rex^0.5=4.497329894013379
i = 16 Pr=300.0 Nux/Rex^0.5=5.14935646315043
```

i = 17 Pr=400.0 Nux/Rex^0.5=5.668325369270466
i = 18 Pr=500.0 Nux/Rex^0.5=6.1064900529916315
i = 19 Pr=600.0 Nux/Rex^0.5=6.489434384478669
i = 20 Pr=700.0 Nux/Rex^0.5=6.831821245536183
i = 21 Pr=800.0 Nux/Rex^0.5=7.1429292829709565
i = 22 Pr=900.0 Nux/Rex^0.5=7.429046147303017
i = 23 Pr=1000.0 Nux/Rex^0.5=7.694652708450487
0.001<Pr<0.01 a=0.33909842208904095 b=0.002522323431681117
0.01<Pr<0.5 a=0.3970088423802754 b=0.12179317628513694
0.5<Pr<50 a=0.5808370023452061 b=0.4303534940489292
50<Pr<1000 a=0.7772410241736414 b=0.3315896030668632

For $\beta = 0.25$ opening angle =0785398 radian = 45 degree $\gamma = 0$ (isothermal plate)

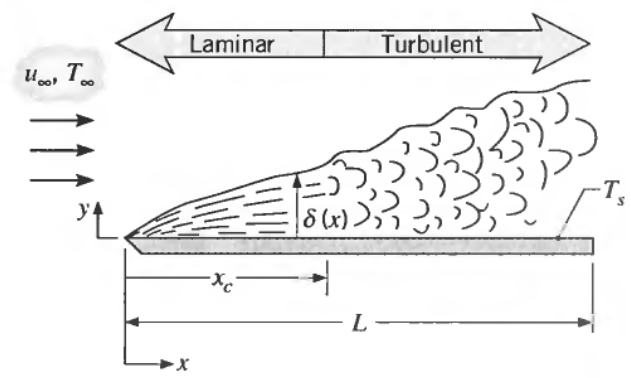
$Nu = 0.33909842208904095\ \mathrm{Re}^{0.5} Pr^{0.002522323431681117}$ $0.001 \leq Pr \leq 0.01$

$Nu = 0.3970088423802754 \mathrm{Re}^{0.5} Pr^{0.12179317628513694}$ $0.01 \leq Pr \leq 0.5$

$Nu = 0.5808370023452061\ \mathrm{Re}^{0.5} Pr^{0.4303534940489292}$ $0.5 \leq Pr \leq 50$

$Nu = 0.7772410241736414\ \mathrm{Re}^{0.5} Pr^{0.3315896030668632}$ $50 \leq Pr \leq 1000$

12.2 CONVECTIVE HEAT TRANSFER EQUATIONS AND EXAMPLES FOR OVER A FLAT PLATE



$Pr = \frac{C_p \mu}{k}$ Prandtl number

Local values

$Re_x = \frac{\rho u_\infty x}{\mu}$ Reynols number $Nu_x = \frac{hx}{k}$ Nusselt number

Avarege values

$Re_L = \frac{\rho u_\infty L}{\mu}$ $Nu_L = \frac{hL}{k}$

Where ρ is density, μ is dynamic viscosity, k is thermal conductivity and C_p is specific heat at constant temperature. It should be note that viscosity and themal conductivity is function of temperature and pressure. Re_x is given as Re in the following equations if local conditions exist and similarly instead of . Re_L Re is given to simplify equations.

Laminar flow (Re<5x10^5) and constant plate temperature (isothermal plate)

Coban-1 eqn (similarity solution by Runge Kutta method and Nelder-Mead optimisation)

$Nu_x = 0.44547101 Re^{0.5} Pr^{0.46914552}$ $10^{-3} \leq Pr \leq 10^{-2}$

$Nu_x = 0.346667574 Re^{0.5} Pr^{0.40159677}$ $10^{-2} \leq Pr \leq 0.5$

$Nu_x = 0.332060655 Re^{0.5} Pr^{0.33679327}$ $0.5 \leq Pr \leq 1000$

Coban-2 eqn (similarity solution by Runge Kutta method and and least square curve fitting and Gauss integration)

$Nu_x = 0.45584275 Re^{0.5} Pr^{0.47294717}$ $10^{-3} \leq Pr \leq 10^{-2}$

$Nu_x = 0.35271867 Re^{0.5} Pr^{0.40922589}$ $10^{-2} \le Pr \le 0.5$

$Nu_x = 0.33253715 Re^{0.5} Pr^{0.33694685}$ $0.5 \le Pr \le 1000$

a heat and mass transfer by **Eckert and Drake** (similarity solution)

$Nu_x = 0.5 Re^{0.5} Pr^{0.5}$ $0.005 \le Pr \le 0.05$

$Nu_x = 0.332 Re^{0.5} Pr^{0.3333333}$ $0.6 \le Pr$

Local Friction coefficient

$C_{fx} = \frac{\tau_x}{\rho\, u_\infty^2/2} = 0.664 Re^{-0.5}$

Boundary layer thickness

$\delta = 5x Re^{-1/2}$

Churchill and Ozoe experimental correlation:

$Nu_x = \frac{0.332 Re^{0.5} Pr^{0.3333333}}{\left[1+\left(\frac{0.0468}{\mathrm{Pr}}\right)^{2/3}\right]^{1/4}}$ $Pe_x = RePr \ge 100$

Average values for the plate :

$Nu_L = 2Nu_x$

Turbulent flow (Re>5x10^5) and constant plate temperature (isothermal plate)

Local Friction coefficient

$C_{fx} = \frac{\tau_x}{\rho\, u_\infty^2/2} = 0.0592 Re^{-1/5}$ $2300 \le Re \le 10^8$

Boundary layer thickness

$\delta = 0.37 x Re^{-1/5}$ $2300 \le Re \le 10^8$

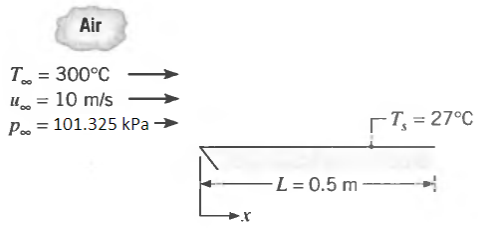
$Nu_x = 0.0296 Re^{4/5} Pr^{1/3}$ $0.6 \le Pr \le 60$

Average values for the plate (mix boundary layer laminar+turbulent):

$Nu_L = \left(0.037 Re^{4/5} - 871\right) Pr^{1/3}$

Example problems:

PROBLEM: Air at a pressure of 101.325 kPa and a temperature of 300 C flows with a velocity of 10 m/s over a flat plate 0.5 m long. Estimate the cooling rate per unit width of the plate needed to maintain it at a surface temperature of 27 C.



In excel format thermophysical properties calculated as polynomial of degree 10

$$y = a_0 + a_1 T + a_2 T^2 + a_3 T^3 + a_4 T^4 + a_5 T^5 + a_6 T^6 + a_7 T^7 + a_8 T^8 + a_9 T^9 + a_{10} T^{10}$$

	Heat transfer over a flat plate		**working fluid : air**		
M	**2.8965000E+01**	**kg/kmol**			
R	**2.8705334E+02**	**J/kgK**			
P	**1.0132500E+02**	**kPa**			
	ρ	Cp	u	k	Pr
	kg/m^3				
a0		1.89668613E+00	-7.00796942E-01	-5.96297846E-01	1.47352515E+00
a1		-2.23937197E-02	8.85836561E-02	1.00247973E-01	-1.73708682E-02
a2		2.40885259E-04	-1.20909367E-04	7.32722776E-05	1.80473009E-04
a3		-1.45254290E-06	1.73074497E-07	-1.02111800E-06	-1.07436904E-06
a4		5.43552393E-09	-1.45011707E-10	4.21691040E-09	3.99304707E-09
a5		-1.32310554E-11	-4.33936641E-14	-1.04115589E-11	-9.67956666E-12
a6		2.13239465E-14	3.14868466E-16	1.67088379E-14	1.55567247E-14
a7		-2.25744420E-17	-4.41548734E-19	-1.75371889E-17	-1.64359568E-17
a8		1.50864395E-20	3.25615211E-22	1.16134919E-20	1.09674418E-20
a9		-5.76865768E-24	-1.28963912E-25	-4.40290627E-24	-4.18869082E-24
a10		9.61540784E-28	2.16661936E-29	7.28284943E-28	6.97523663E-28
result:for T	8.0838928E-01	1.0195815E+00	2.4582335E-05	3.5890189E-02	6.9836485E-01
result: for Tw	1.1760226E+00	1.0059478E+00	1.8544579E-05	2.6395395E-02	7.0673940E-01

	degree C	degree K	
T	1.6350000E+02	4.3665000E+02	air temperature
Ts	2.7000000E+01	3.0015000E+02	surface temperature
Tinf	3.0000000E+02	5.7315000E+02	
P	1.0132500E+00	bar	
ro	8.0838928E-01		
U	1.0000000E+01		air velocity
L	5.0000000E-01		plate length
Re=ρUL/μ	1.6442484E+05		Reynolds' number
Nu	2.3887970E+02		
k	3.5890189E-02	W/mK	
h	1.7146875E+01	W/m^2K	thermal convectivity
q	-2.3405484E+03	W/m	heat transfer

The problem is also solved in java format, in here in order to calculate thermophysical properties cubic spline surface fitting y=(T,P) is used. Small differences due to curve fitting method should be noted.

```
//Cubic spline curve fitting
import java.io.*;
import javax.swing.*;

class CS1
{  public double S[][]; // cubic spline coefficients
   public double c[][]; // data matrix
   public int nx,ny;    // x and y data column numbers
   public int kk;       // equation number that given x located
   public CS1(String name,int nxi,int nyi,double c0,double cn)
   {nx=nxi;ny=nyi;
    c=Text.readDoubleT(name);
    double x[]=c[nx];double y[]=c[ny];
    S=cubic_spline(x,y,c0,cn);
   }
   public CS1(double a[][],int nxi,int nyi,double c0,double cn)
   {c=a;nx=nxi;ny=nyi;
    double x[]=c[nx];double y[]=c[ny];
    S=cubic_spline(x,y,c0,cn);
   }
   public CS1(double xi[],double yi[],double c0,double cn)
   {double c1[][]={xi,yi};c=c1;S=cubic_spline(xi,yi,c0,cn);}
  public static double [] thomas(double f[],double e[],double g[],double r[])
  {
  // Thomas algorithm
  // 3 band matrix system of equation solving algorithm
  int n=f.length;
  double x[]=new double[n];
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
     }
  for(int k=1;k<n;k++)
     {r[k]=r[k]-e[k]*r[k-1];
     }
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
     {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
  return x;
  }

  public static double [][] cubic_spline(double xi[],double yi[],double c0,double cn)
  {
  int n=xi.length;
  double h[]=new double[n];
  double w[]=new double[n];
  double f[]=new double[n];
  double e[]=new double[n];
  double g[]=new double[n];
  double d[]=new double[n];
  double x[]=new double[n];
  double S[][]=new double[4][n];
  int k;
  for(k=0;k<(n-1);k++)
     {h[k]=xi[k+1]-xi[k];
      w[k]=(yi[k+1]-yi[k])/h[k];
                }
  d[0]=c0;
  d[n-1]=cn;
```

```
  for(k=1;k<(n-1);k++)
     {d[k]=6.0*(w[k]-w[k-1]);}
  f[0]=1.0;
  f[n-1]=1.0;
  g[0]=0.0;
  g[n-1]=0.0;
  e[0]=0.0;
  e[n-1]=0.0;
  for(k=1;k<(n-1);k++)
  {f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];}
  S[2]=thomas(f,e,g,d);
  S[3]=xi;
  for(k=0;k<(n-1);k++)
    {S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
     S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
    }
  return S;
  }

  public double func(double x)
  {
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y=0;
  double hk=0;
  int k=0;
  for(k=0;k<(n-1);k++)
     {if(S[3][k]<=x && x<=S[3][k+1])
      {hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
       break;
       }
     }

       if(y==0 && S[3][n-2]<=x )
       {
                k=n-2;
                hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
       }
     kk=k+1;
  return y;
  }

  // Türev formülleri
// ==================


  public double dfunc(double x)
  {
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y=0;
  double hk=0;
  for(int k=0;k<(n-1);k++)
     {if(S[3][k]<=x && x<=S[3][k+1])
      {hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]-S[1][k]+(xx1*xx1*S[2][k+1]-xx2*xx2*S[2][k])/(2.0*hk);
       break;
       }
     }
       if(y==0 && S[3][n-2]<=x )
       {
                int k=n-2;
                hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
       }
```

```
  return y;
  }

  public double integral(double a,double b)
  {
    //kübik şerit integral[S(x)]dx  formülü
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y1,y2;
  double hk=0;
  double toplam=0;
  y1=0;y2=0;
  for(int k=0;k<(n-1);k++)
     { hk=(S[3][k+1]-S[3][k]);
      if(a>S[3][k+1])
      {toplam=0;
      }
                else if(S[3][k]<=a && a<=S[3][k+1] && b>S[3][k+1])//şart 2
       {
       xx1=(a-S[3][k]);
       xx2=(S[3][k+1]-a);
       y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       xx1=hk;
       xx2=0;
       y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       toplam+=(y2-y1);
       }
       else if(S[3][k]<=a && a<=S[3][k+1] && S[3][k]<=b && b<=S[3][k+1])//şart 3
       {
       xx1=(a-S[3][k]);
       xx2=(S[3][k+1]-a);
       y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       xx1=(b-S[3][k]);
       xx2=(S[3][k+1]-b);
       y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       toplam+=(y2-y1);
       }
       else if(a<S[3][k] && b>=S[3][k+1]) //şart 4
       {
       xx1=0;
       xx2=hk;
       y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       xx1=hk;
       xx2=0;
       y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       toplam+=(y2-y1);
       }
       else if(S[3][k]<=b && b<=S[3][k+1])//şart 5
       {xx1=0;
        xx2=hk;
        y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
        xx1=(b-S[3][k]);
        xx2=(S[3][k+1]-b);
        y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
        toplam+=(y2-y1);
        }
        else break;
     }
  return toplam;
  }


public double golden(double a,double b)
{
//  find the minimum of the function
//  note  maximum f(x) = minimum (-f(x))
    double epsilon=1.0e-10;
    double delta=1.0e-8;
    int print=0;
    double r1 = (Math.sqrt(5.0)-1.0)/2.0; // golden ratio
    double r2 = r1*r1;
    double h = b - a;
    double ya = -func(a);
    double yb = -func(b);
    double c = a + r2*h;
    double d = a + r1*h;
```

```
    double yc = -func(c);
    double yd = -func(d);
    int k = 1;
    double dp,dy,p,yp;
    while ((Math.abs(yb-ya)>epsilon) || (h>delta))
    {
     k++;
     if (yc<yd)
       {
       b = d;
       yb = yd;
       d = c;
       yd = yc;
       h = b - a;
       c = a + r2 * h;
       yc = -func(c);
       }
     else
      {
       a = c;
       ya = yc;
       c = d;
       yc = yd;
       h = b - a;
       d = a + r1 * h;
       yd = -func(d);
      }//end of if
   }//end of while
   dp = Math.abs(b-a);
   dy = Math.abs(yb-ya);
   p = a;
   yp = ya;
   if (yb<ya)
   {
     p = b;
     yp = yb;
   }
   if(print==1)
   {System.out.println("x min = "+p+"ymin = "+yp+"error of x ="+dp+"error of y"+dy); }
   return p;
}
public static void  printData(double c[][])
  {       String s3[]={"x[i]","y[i]"};
       Text.print(Text.T(c),s3,"Input data");
  }

  public static void main(String args[]) throws IOException
  { CS1 cs=new CS1("a100.txt",0,2,0.0,0.0);
    double T=300.0;
    double ro=cs.func(T);
    System.out.println("T="+T+"ro="+ro);
  }
}
```

```
//3D Cubic spline surface fitting
import java.io.*;
import javax.swing.*;

class CS2
{  public double S[][];
   String name[];
   public CS1 cs[];
   public int n;
   double xx[],yy[];
   public double c0,cn;
   public int nyy;
   public int nx,ny;
   public double d[][][];
   //use set of files each with a constant property
   // namei: name of datafile to read data
   // use one file with a repeated column nselect
   // c: data matrix with a repeated one column
   // nyi y selection column
   // nxi  x selection column
   // nzi  z selection column
   // c0x and cnx x axis cubic spline end conditions (second derivatives)
```

```
 // c0y and cny y axis cubic spline end conditions (second derivatives)
 public CS2(String namei,int nxi,int nyi,int nzi,double c0x,double cnx,double c0y,double cny)
 { double c[][]=Text.readDouble(namei);
   set(c,nxi,nyi,nzi,c0x,cnx,c0y,cny);
 }
 public CS2(double c[][],int nxi,int nyi,int nzi,double c0x,double cnx,double c0y,double cny)
 {  set(c,nxi,nyi,nzi,c0x,cnx,c0y,cny);}
 // use one file with a repeated column nyi

 public void set(double c[][],int nxi,int nyi,int nzi,double c0x,double cnx,double c0y,double cny)
 { nyy=nyi;
   nx=nxi;
   ny=nzi;
   c0=c0y;
   cn=cny;
   d=readDouble(c,nyy);
              n=d.length;
   cs=new CS1[n];
   xx=new double[n];
   yy=new double[n];
   for(int i=0;i<n;i++)
   {   cs[i]=new CS1(Text.T(d[i]),nx,ny,c0x,cnx);}
 }

public static double[][][] readDouble(String name,int nc)
{double c[][]=Text.readDouble(name);
 int n=c.length;
 int m=c[0].length;
 double dd[][][]=new double[1000][n][m];
 int k=0;
 int i=0;
 int j=0;
 int nj[]=new int[100];
 for(i=0;i<n-1;i++)
 {if(c[i][nc]==c[i+1][nc]) {dd[k][j]=c[i];j++;}
  else {dd[k][j]=c[i];nj[k]=j;j=0;
        k++;
       }
 }
 double ee[][][]=new double[k][][];
 for(i=0;i<k;i++)
 {   ee[i]=new double[nj[i]+1][m];
     for(j=0;j<nj[i]+1;j++) ee[i][j]=dd[i][j];
 }
 return ee;
}
public static double[][][] readDouble(double c[][],int nc)
{int n=c.length;
 int m=c[0].length;
 double dd[][][]=new double[1000][n][m];
 int k=0;
 int i=0;
 int j=0;
 int nj[]=new int[1000];
 for(i=0;i<n-1;i++)
 {if(c[i][nc]==c[i+1][nc]) {dd[k][j]=c[i];j++;}
  else {dd[k][j]=c[i];nj[k]=j;j=0;
        k++;
       }
 }
 double ee[][][]=new double[k][][];
 for(i=0;i<k;i++)
 {
                ee[i]=new double[nj[i]+1][m];
     for(j=0;j<nj[i]+1;j++) ee[i][j]=dd[i][j];
 }
 return ee;
}
public static double [] thomas(double f[],double e[],double g[],double r[])
{
// 3 band matrix system of equation solving algorithm
int n=f.length;
double x[]=new double[n];
for(int k=1;k<n;k++)
   {e[k]=e[k]/f[k-1];
    f[k]=f[k]-e[k]*g[k-1];
   }
```

```
for(int k=1;k<n;k++)
   {r[k]=r[k]-e[k]*r[k-1];
   }
x[n-1]=r[n-1]/f[n-1];
for(int k=(n-2);k>=0;k--)
   {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
return x;
}

public static double [][] cubic_spline(double xi[],double yi[],double c0,double cn)
{
int n=xi.length;
double h[]=new double[n];
double w[]=new double[n];
double f[]=new double[n];
double e[]=new double[n];
double g[]=new double[n];
double d[]=new double[n];
double x[]=new double[n];
double S[][]=new double[4][n];
int k;
for(k=0;k<(n-1);k++)
   {h[k]=xi[k+1]-xi[k];
    w[k]=(yi[k+1]-yi[k])/h[k];
                }
d[0]=c0;
d[n-1]=cn;
for(k=1;k<(n-1);k++)
   {d[k]=6.0*(w[k]-w[k-1]);}
f[0]=1.0;
f[n-1]=1.0;
g[0]=0.0;
g[n-1]=0.0;
e[0]=0.0;
e[n-1]=0.0;
for(k=1;k<(n-1);k++)
{f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];}
S[2]=thomas(f,e,g,d);
S[3]=xi;
for(k=0;k<(n-1);k++)
  {S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
   S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
  }
return S;
}
public double func(double x1,double y1)
{ double xx1[]={x1,y1};
  return func(xx1);
}
public double func(double x[])
{ //create data set

              for(int i=0;i<n;i++)
              {  yy[i]=cs[i].func(x[0]);
                 xx[i]=cs[i].c[nyy][0];
                 //xx[i]=cs[i].c[nyy][cs[i].kk];
              }
              S=cubic_spline(xx,yy,c0,cn);
 int n=S[0].length;
double xx1=0;
double xx2=0;
double y=0;
double hk=0;
int k=0;
for(k=0;k<n-1;k++)
   {if(S[3][k]<=x[1] && x[1]<=S[3][k+1])
    {hk=(S[3][k+1]-S[3][k]);
     xx1=(x[1]-S[3][k]);
     xx2=(S[3][k+1]-x[1]);
     y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
     break;
     }
   }
     if(y==0 && S[3][n-2]<=x[1] )
     {
                k=n-2;
                hk=(S[3][k+1]-S[3][k]);
```

```
      xx1=(x[1]-S[3][k]);
      xx2=(S[3][k+1]-x[1]);
      y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
      }

  return y;
  }

  // Türev formülleri
// =================

  public double dfunc(double x1,double y1)
  {double xx1[]={x1,y1};
   return dfunc(xx1) ;
  }
  public double dfunc(double x[])
  {
  double xx1=0;
  double xx2=0;
  double y=0;
  double hk=0;
  double yy[]=new double[n];
               for(int i=0;i<=n;i++)
               {yy[i]=cs[i].func(x[0]);}
                S=cubic_spline(xx,yy,c0,cn);
  for(int k=0;k<(n-1);k++)
    {if(S[3][k]<=x[1] && x[1]<=S[3][k+1])
     {hk=(S[3][k+1]-S[3][k]);
      xx1=(x[1]-S[3][k]);
      xx2=(S[3][k+1]-x[1]);
      y=S[0][k]-S[1][k]+(xx1*xx1*S[2][k+1]-xx2*xx2*S[2][k])/(2.0*hk);
      break;
      }
    }
      if(y==0 && S[3][n-2]<=x[1] )
      {
                int k=n-2;
                hk=(S[3][k+1]-S[3][k]);
      xx1=(x[1]-S[3][k]);
      xx2=(S[3][k+1]-x[1]);
      y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
      }
  return y;
  }

  public static void main(String args[]) throws IOException
  { //String
name[]={"a10.txt","a50.txt","a90.txt","a95.txt","a100.txt","a101.txt","a110.txt","a150.txt","a200.txt","a500.txt","a1000.txt","a2000.txt","a3
000.txt"};
               //CS2 c2=new CS2(name,2,0,1,0.0,0.0,0.0,0.0);
               //column of P = 1
               //column of T 0 column of ro 2
               double c[][]=Text.readDouble("axy.txt");
               Text.print(c);
               CS2 c2=new CS2(c,1,0,2,0.0,0.0,0.0,0.0);
    double x=-1.0;
    double y=-1.0;
    double z=c2.func(x,y);
    System.out.println("x="+x+"y="+y+"z="+z);
  }
}
```

```
import javax.swing.*;
public class air
{ public CS2 cs;
  public double c[][][];
  public static double M=2.8965000E+01;
  public static double R=8314.5/M;
  public double ro(double T,double P)
  {return P*1e3/R/T;}
  public double[] property_TP(double T,double P)
  { // T degree K P kPa
    // 0 T 1 P 2 ro 3 h 4 s 5 Cp 6 k 7 mu 8 alpha 9 Pr
    double a[]=new double[10];
    a[0]=T;
    a[1]=P;
```

```
    double multiply[]={1,1,1,1,1,1,1e-3,1e-6,1e-4,1};
    for(int i=2;i<10;i++)
      {cs=new CS2("air.txt",0,1,i,0.0,0.0,0.0,0.0);
       a[i]=cs.func(T,P)*multiply[i];
      }
    return a;
  }

  public static void main(String arg[])
  {air a=new air();
   double T=Double.parseDouble(JOptionPane.showInputDialog("T degree K = "));
   double P=Double.parseDouble(JOptionPane.showInputDialog("P kPa = "));
   double b[]=a.property_TP(T,P);
   String s[]={"T degree K","P kPa",""+'\u03C1'+" kg/m^3","h kJ/kg","s kJ/kgK","Cp kJ/kgK","k W/mK",""+'\u03BC'+"
Pas",""+'\u03B1'+" m^2/s","Pr"};
   Text.print(b,s,"Thermophysical properties of dry air");
  }
}
```

```
   //0="P, pressure                ";
   //1="T, temperature             ";
   //2="v, specific volume         ";
   //3="h, enthalpy                ";
   //4="u, internal energy         ";
   //5="s, entropy                 ";
   //6="g, qibbs free energy       ";
   //7="ht,chemical entropy        ";
   //8="gt,chemical gibbs f.e.     ";
   //9="Cp, specific heat at const P ";
   //10="Cv, specific heat at const v";
   //11][0]="Cp/Cv, adiabatic constant   ";
   //12][0]="c, speed of sound           ";
   //13][0]="viscosity                   ";
   //14][0]="thermal conductivity        ";
   //15][0]="M, molecular weight         ";
   //16][0]="Prandtl number              ";
   //17][0]="Pr, reduced pressure        ";
   //18][0]="vr, reduced volume          ";
public class HT_flat_plate
{ public static void main(String arg[])
  {air g=new air();
   double Tinf=300+273.15;  //air temp. degree K;
   double Tw=27+273.15;       //wall temp. degree K
   double T=(Tinf+Tw)/2.0;    //average temperature degree K
   double P=101.325; //kPa
   double a[]=new double[10];

   a=g.property_TP(T,P);
   double Rec=5e5;
   double ro=g.ro(T,P); //kg/m^3
   double mu=a[7];
   double k=a[6]; //W/mK
   double Pr=a[9];
   double L=0.5; //m
   double U=10.0; //m/s
   double Nu=0.0;
   double Re=U*L*ro/mu;
   double Nu1=0.664*Math.sqrt(Re)*Math.pow(Pr,(1.0/3.0));
   double Nu2=0.036*(Math.pow(Re,0.8)-9200.0)*Math.pow(Pr,0.43);
   if(Re<Rec) Nu=Nu1;
   else       Nu=Nu2;
   double h=Nu*k/L;
   double q=h*L*(Tw-Tinf);
   System.out.println("ro="+ro+"kg/m^3");
   System.out.println("mu="+mu+"Ns/m^2");
   System.out.println("k="+k+"W/mK");
   System.out.println("Pr="+Pr+" ");
   System.out.println("Re="+Re+" ");
   System.out.println("Nu="+Nu+" ");
   System.out.println("h="+h+" ");
    System.out.println("q="+q+" ");
 }}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_flat_plate
```

ro=0.8083892751173132kg/m^3
mu=2.4582336829766457E-5Ns/m^2
k=0.035889877242974 98W/mK
Pr=0.6979957267070102
Re=164424.8227325655
Nu=238.8375963348956
h=17.14370402693323
q=-2340.1155996763855

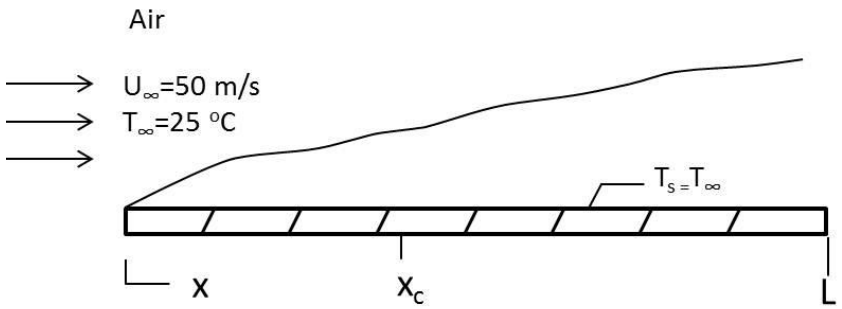
> Terminated with exit code 0.

Heat transfer over a flat plate **working fluid : air P=1.01325 bar** 85 K<=T<=100K

	ρ	Cp	μ	k	Pr
	kg/m^3	kJ/kgK	Pas	W/(mK)	
a0	1.53970719E+01	1.85645219E+00	-6.96977489E-01	-5.94260000E-01	1.44585054E+00
a1	-2.64146711E-01	-2.16797564E-02	8.84646695E-02	9.98680478E-02	-1.68940685E-02
a2	2.43939374E-03	2.38367524E-04	-1.19377693E-04	8.20898664E-05	1.79103033E-04
a3	-1.37152197E-05	-1.47868744E-06	1.62277415E-07	-1.10918682E-06	-1.09607903E-06
a4	4.98959966E-08	5.72130928E-09	-9.85161050E-11	4.69871331E-09	4.21033463E-09
a5	-1.21018968E-10	-1.44575109E-11	-1.72146547E-13	-1.20040535E-11	-1.05922995E-11
a6	1.97292216E-13	2.42642600E-14	5.48663333E-16	2.00250979E-14	1.77240903E-14
a7	-2.13349280E-16	-2.68138976E-17	-7.18151386E-19	-2.19145061E-17	-1.95441958E-17
a8	1.46625301E-19	1.87402720E-20	5.30795939E-22	1.51648507E-20	1.36371672E-20
a9	-5.79352041E-23	-7.50469834E-24	-2.15550615E-25	-6.01785056E-24	-5.45416504E-24
a10	1.00148738E-26	1.31155075E-27	3.75178265E-29	1.04324926E-27	9.52222881E-28
result:for T	8.1127953E-01	1.0195485E+00	2.4582341E-05	3.5890204E-02	6.9834255E-01
result: for	1.1738617E+00	1.0059238E+00	1.8544572E-05	2.6395335E-02	7.0672053E-01

	degree C	**degree K**	
T	3.0007500E+02	4.3665000E+02	air temperature
Ts	2.7000000E+01	3.0015000E+02	surface temperature
Tinf	3.0000000E+02	5.7315000E+02	
P	1.0132500E+00	bar	
U	1.0000000E+01		air velocity
L	5.0000000E-01		plate length
Re=ρUD/μ	1.6501267E+05		Reynolds' number
Pr	6.9834255E-01		
Nu	2.3930378E+02		
h	1.7177323E+01	W/m^2K	thermal convectivity
q	-2.3447046E+03	W/m	heat transfer

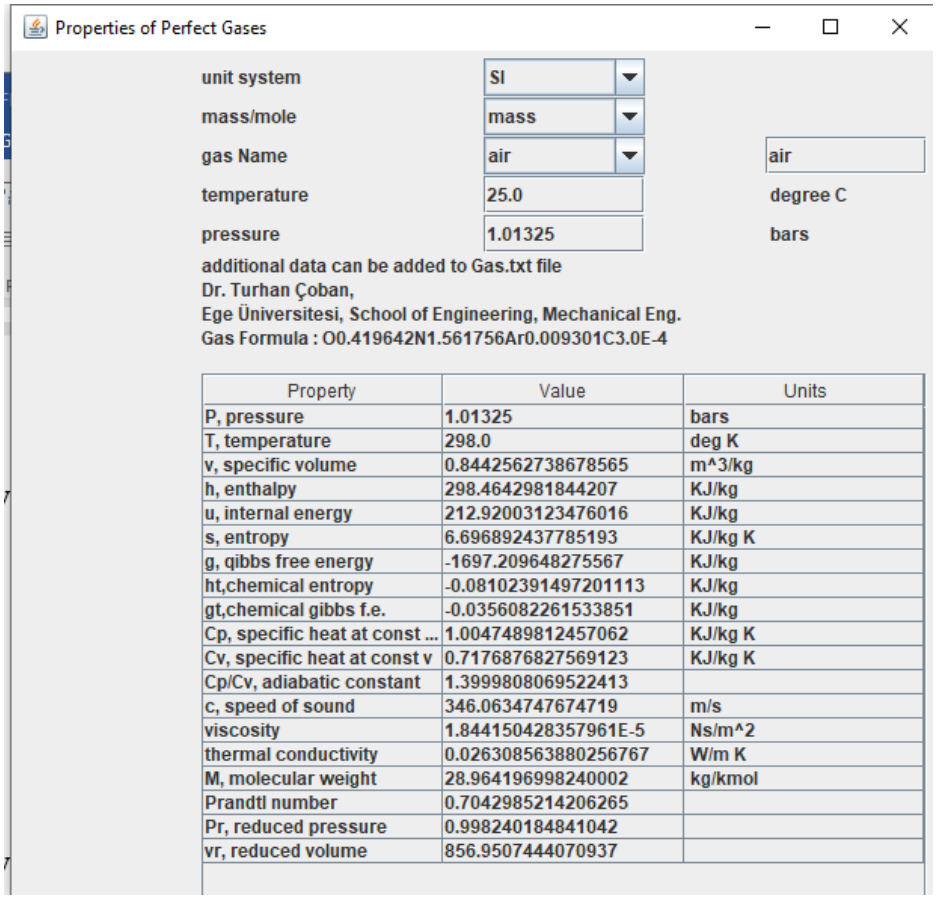
PROBLEM:



Atmospheric air at 25°C and with a speed of 50 m/s is pumped by a fan over a flat plate.

a)What is the minimum plate length in order to achieve a Reynolds number of 10^8 ?

b)At what distance from the leading edge would transition occur if critical Re #= $5x10^5$?

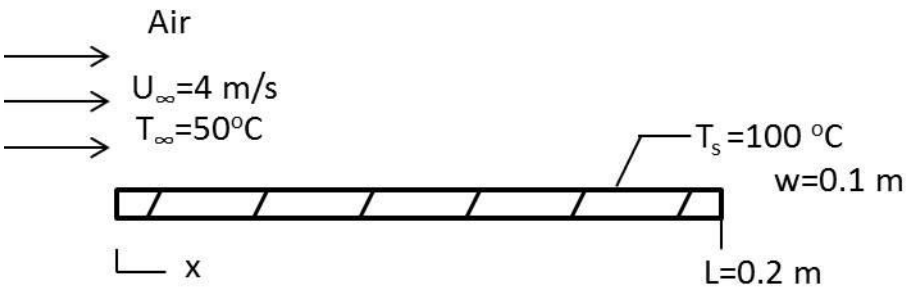


Property	Value	Units
P, pressure	1.01325	bars
T, temperature	298.0	deg K
v, specific volume	0.8442562738678565	m^3/kg
h, enthalpy	298.4642981844207	KJ/kg
u, internal energy	212.92003123476016	KJ/kg
s, entropy	6.696892437785193	KJ/kg K
g, qibbs free energy	-1697.209648275567	KJ/kg
ht,chemical entropy	-0.08102391497201113	KJ/kg
gt,chemical gibbs f.e.	-0.0356082261533851	KJ/kg
Cp, specific heat at const ...	1.0047489812457062	KJ/kg K
Cv, specific heat at const v	0.7176876827569123	KJ/kg K
Cp/Cv, adiabatic constant	1.3999808069522413	
c, speed of sound	346.0634747674719	m/s
viscosity	1.844150428357961E-5	Ns/m^2
thermal conductivity	0.026308563880256767	W/m K
M, molecular weight	28.964196998240002	kg/kmol
Prandtl number	0.7042985214206265	
Pr, reduced pressure	0.998240184841042	
vr, reduced volume	856.9507444070937	

$$Re = \frac{\rho U L}{\mu}$$

ρ	1.184474467	kg/m^3		
μ	1.84415E-05	Ns/m^2		
U	50	m/s		
Re	1.00E+08	L=	31.13871	m
Re	5.00E+05	L=	0.155694	m

PROBLEM:



Air at 101.325 kPa with a velocity of U_∞ =4 m/s and a temperature of T_∞ =50 °C flows over a flat plate that is at a uniform temperature of T_s =100°C. The plate has a length of 0.20 m and a width of 0.1 m.

a) What is the average heat transfer coefficient?

b) What is the heat transfer rate from plate to the air?

	Heat transfer over a flat plate		working fluid : air		
M	**2.8965000E+01**	**kg/kmol**			
R	**2.8705334E+02**	**J/kgK**			
P	**1.0132500E+02**	**kPa**			
	ρ	Cp	μ	k	Pr
	kg/m^3				
a0		1.89668613E	-7.00796942E-01	-	1.47352515E
a1		-	8.85836561E-02	1.00247973E-	-
a2		2.40885259E-	-1.20909367E-04	7.32722776E-	1.80473009E-
a3		-	1.73074497E-07	-	-
a4		5.43552393E-	-1.45011707E-10	4.21691040E-	3.99304707E-
a5		-	-4.33936641E-14	-	-
a6		2.13239465E-	3.14868466E-16	1.67088379E-	1.55567247E-
a7		-	-4.41548734E-19	-	-
a8		1.50864395E-	3.25615211E-22	1.16134919E-	1.09674418E-
a9		-	-1.28963912E-25	-	-
a10		9.61540784E-	2.16661936E-29	7.28284943E-	6.97523663E-
result:for T	1.0138825E+00	1.0082283E+	2.0783611E-05	2.9872015E-	7.0144974E-
result: for	9.4595518E-01	1.0106767E+	2.1896460E-05	3.1619486E-	6.9986853E-

	degree C	**degree K**	
T	7.5000000E+01	3.4815000E+	air temperature
Ts	1.0000000E+02	3.7315000E+	surface temperature
Tinf	5.0000000E+01	3.2315000E+	
P	1.0132500E+00	bar	
ro	1.0138825E+00		
U	4.0000000E+00		air velocity
L	2.0000000E-01		plate length
Re=ρUL/μ	3.9026229E+04		Reynolds' number
Nu	1.1654990E+02		
k	2.9872015E-02	W/mK	
h	1.7407901E+01	W/m^2K	thermal convectivity
q	1.7407901E+02	W/m	heat transfer

Java version:

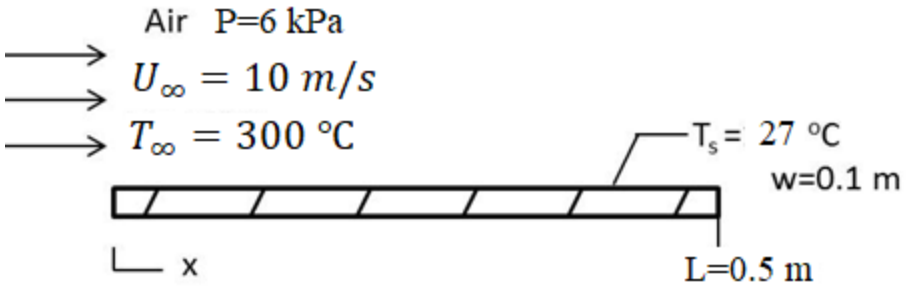
```
   //0="P, pressure                ";
   //1="T, temperature             ";
   //2="v, specific volume         ";
   //3="h, enthalpy                ";
   //4="u, internal energy         ";
   //5="s, entropy                 ";
   //6="g, qibbs free energy       ";
   //7="ht,chemical entropy        ";
   //8="gt,chemical gibbs f.e.     ";
   //9="Cp, specific heat at const P ";
   //10="Cv, specific heat at const v";
   //11][0]="Cp/Cv, adiabatic constant   ";
   //12][0]="c, speed of sound           ";
   //13][0]="viscosity                   ";
   //14][0]="thermal conductivity        ";
   //15][0]="M, molecular weight         ";
   //16][0]="Prandtl number              ";
   //17][0]="Pr, reduced pressure        ";
   //18][0]="vr, reduced volume          ";
public class HT_flat_plate
{ public static void main(String arg[])
  {Gas g=new Gas("air");
    g.mole=false;
   double Tinf=50+273.15;  //air temp. degree K;
   double Tw=100+273.15;       //wall temp. degree K
   double T=(Tinf+Tw)/2.0;    //average temperature degree K
   double P=1.01325; //bar
   double a[]=new double[10];
```

```
    a=g.property(T,P);
    double Rec=5e5;
    double ro=1.0/a[2]; //kg/m^3
    double mu=a[13];
    double k=a[14]; //W/mK
    double Pr=a[16];
    double L=0.2; //m
    double U=4.0; //m/s
    double Nu=0.0;
    double Re=U*L*ro/mu;
    double Nu1=0.664*Math.sqrt(Re)*Math.pow(Pr,(1.0/3.0));
    double Nu2=0.036*(Math.pow(Re,0.8)-9200.0)*Math.pow(Pr,0.43);
    if(Re<Rec) Nu=Nu1;
    else       Nu=Nu2;
    double h=Nu*k/L;
    double q=h*L*(Tw-Tinf);
    System.out.println("ro="+ro+"kg/m^3");
    System.out.println("mu="+mu+"Ns/m^2");
    System.out.println("k="+k+"W/mK");
    System.out.println("Pr="+Pr+" ");
    System.out.println("Re="+Re+" ");
    System.out.println("Nu="+Nu+" ");
    System.out.println("h="+h+" ");
     System.out.println("q="+q+" ");
}}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_flat_plate
ro=1.0138543477398665kg/m^3
mu=2.078413954854959E-5Ns/m^2
k=0.0299436386431054W/mK
Pr=0.6994343290825921
Re=39024.15475498933
Nu=116.43507127820062
h=17.4324484986933
q=174.32448498693302

> Terminated with exit code 0.
```

PROBLEM: Air at a pressure of **6 kPa** and a temperature of 300 C flows with a velocity of 10 m/s over a flat plate 0.5 m long. Estimate the cooling rate per unit width of the plate needed to maintain it at a surface temperature of 27 C.




```
    //0="P, pressure                ";
    //1="T, temperature             ";
    //2="v, specific volume         ";
    //3="h, enthalpy                ";
    //4="u, internal energy         ";
    //5="s, entropy                 ";
    //6="g, qibbs free energy       ";
    //7="ht,chemical entropy        ";
    //8="gt,chemical gibbs f.e.     ";
    //9="Cp, specific heat at const P ";
    //10="Cv, specific heat at const v";
    //11][0]="Cp/Cv, adiabatic constant  ";
    //12][0]="c, speed of sound         ";
    //13][0]="viscosity                 ";
```

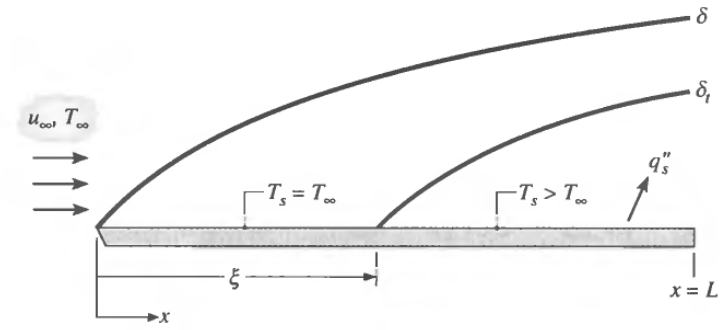
```
   //14][0]="thermal conductivity        ";
   //15][0]="M, molecular weight         ";
   //16][0]="Prandtl number             ";
   //17][0]="Pr, reduced pressure        ";
   //18][0]="vr, reduced volume          ";
public class HT_flat_plate
{ public static void main(String arg[])
  {Gas g=new Gas("air");
    g.mole=false;
   double Tinf=300+273.15;  //air temp. degree K;
   double Tw=27+273.15;       //wall temp. degree K
   double T=(Tinf+Tw)/2.0;    //average temperature degree K
   double P=0.06; //bar
   double a[]=new double[10];

   a=g.property(T,P);
   double Rec=5e5;
   double ro=1.0/a[2]; //kg/m^3
   double mu=a[13];
   double k=a[14]; //W/mK
   double Pr=a[16];
   double L=0.5; //m
   double U=10.0; //m/s
   double Nu=0.0;
   double Re=U*L*ro/mu;
   double Nu1=0.664*Math.sqrt(Re)*Math.pow(Pr,(1.0/3.0));
   double Nu2=0.036*(Math.pow(Re,0.8)-9200.0)*Math.pow(Pr,0.43);
   if(Re<Rec) Nu=Nu1;
   else       Nu=Nu2;
   double h=Nu*k/L;
   double q=h*L*(Tw-Tinf);
   System.out.println("ro="+ro+"kg/m^3");
   System.out.println("mu="+mu+"Ns/m^2");
   System.out.println("k="+k+"W/mK");
   System.out.println("Pr="+Pr+" ");
   System.out.println("Re="+Re+" ");
   System.out.println("Nu="+Nu+" ");
   System.out.println("h="+h+" ");
    System.out.println("q="+q+" ");
}}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_flat_plate
ro=0.04786776396743621kg/m^3
mu=2.458207128943916E-5Ns/m^2
k=0.03595715331030523W/mK
Pr=0.6963693569205517
Re=9736.316237110772
Nu=58.07358442684175
h=4.176321557029806
q=-570.0678925345685

> Terminated with exit code 0.
```

Unheated starting region (0<=x<=ξ) of plate and then (x>ξ) constant temperature heating



$$Nu_x = \frac{Nu_x|_{\xi=0}}{\left[1-(\xi/x)^{9/10}\right]^{1/9}}$$

$Nu_L = Nu_L|_{\xi=0}\frac{L}{L-\xi}\left[1-(\xi/x)^{(p+1)/(p+2)}\right]^{p/(p+1)}$ where P=2 for laminar flow and p=8 for turbulent flow and $Nu_L|_{\xi=0}$ Nussel number calculated from starting of the plate as it is given in previous cases.

Laminar flow (Re<5e5) and constant heat flux

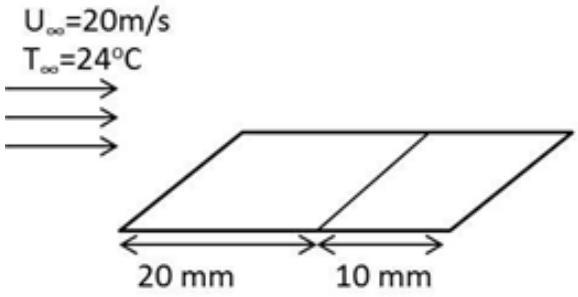
$Nu_x = 0.453Re^{0.5}Pr^{0.5} \quad 0.6 \leq Pr$

Turbulent flow (R>5e5) and constant heat flux

$Nu_x = 0.0308Re^{4/5}Pr^{1/3} \quad 0.6 \leq Pr \leq 60$

PROBLEM Rectangular plate has an unheated starting length of 20 mm. Air is flowing through at $U_\infty = 20\ m/s$ and $T_\infty = 24$ ℃. If the surface temperature of the plate is 80 ℃. The width of the plate is W=1 m

a) What is the convection coefficients at x=L=30 mm?
b) What is the heat transfer rate at x=L=30 mm?



(Properties are calculated by Gas.java

	Ts	8.0000000E+01	3.5315000E+02
	Tinf	2.4000000E+01	2.9715000E+02
0	P, pressure	1.0000000E+00	bars
1	T, temperature	3.2500000E+02	deg K
2	v, specific volume	9.3294922E-01	m^3/kg
3	h, enthalpy	3.2560649E+02	KJ/kg
4	u, internal energy	2.3231157E+02	KJ/kg
5	s, entropy	6.7878587E+00	KJ/kg K
6	g, qibbs free energy	-1.8804476E+03	KJ/kg
7	ht,chemical entropy	8.5607058E-01	KJ/kg
8	gt,chemical gibbs f.e.	-1.1510861E-01	KJ/kg
9	Cp, specific heat at const P	1.0059462E+00	KJ/kg K
10	Cv, specific heat at const v	7.1888485E-01	KJ/kg K
11	Cp/Cv, adiabatic constant	1.3993147E+00	
12	c, speed of sound	3.6131498E+02	m/s
13	μ dynamic viscosity	1.9722159E-05	Ns/m^2
	ν kinematic viscosity	1.8399773E-05	
14	k thermal conductivity	2.8288677E-02	W/m K
15	M, molecular weight	2.8964197E+01	kg/kmol
16	Prandtl number	7.0132054E-01	
17	Pr, reduced pressure	1.3525121E+00	
18	vr, reduced volume	6.8978994E+02	
19	ρ	1.0718697E+00	kg/m^3
	ξ	2.0000000E-02	m
	L	3.0000000E-02	m
	L - ξ	1.0000000E-02	m
	U velocity	2.0000000E+01	m/s
	Re	3.2609098E+04	
	Nu_0	7.2678582E+01	
	Nu	1.1355046E+02	
	h	1.0707307E+02	W/(mK)
	q	5.9960922E+03	W/m^2
	W	1.0000000E+00	m

	A=W*(L-ξ)	1.0000000E-02	m^2
	Q	5.9960922E+01	W

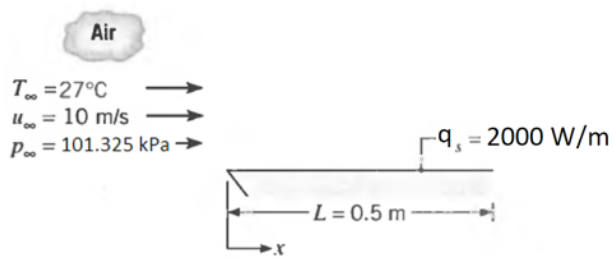
```
public class HT_unheated_starting_plate
{
 public static void main(String arg[])
 { Gas g=new Gas("air");
   g.mole=false;
   double Tinf=24+273.0;  //degree K
   double Ts=80+273.0;    //degree K
            double Tf=(Tinf+Ts)/2.0;  //degree K
            System.out.println("Tf="+Tf+" degree K");
            double P=1.01325; //bar
            double a[]=g.property(Tf,P);
            double mu=a[13];  //Pas
            double k=a[14]; //W/(mK)
            double Pr=a[16];
            double ro=1.0/a[2];

            double nu=mu/ro;
                    System.out.println("mu="+mu+"nu="+nu+"ro="+ro+"k="+k);
            double U=20.0;  //m/s
            double xsi=20e-3; //m
            double L=30e-3;  //m
            double Re=U*L/nu;
            System.out.println("Re="+Re);
            double Nu0=0.453*Math.sqrt(Re)*Math.pow(Pr,(1.0/3.0));
            System.out.println("Nu0="+Nu0);
   double Nu=Nu0/Math.pow((1-Math.pow((xsi/L),(3.0/4.0))),(1.0/3.0));
   System.out.println("Nu="+Nu);
   double h=Nu*k/L;
   System.out.println("h="+h);
   double W=10.0;//m
   double A=(L-xsi)*W;  //m^2
   double q=h*(Ts-Tinf);
   System.out.println("q="+q+" W/m^2");
   double Q=q*A;
   System.out.println("Q="+Q+" W");

}}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_unheated_starting_plate
Tf=325.0 degree K
mu=1.9722159212665443E-5nu=1.8159164131180882E-5ro=1.086071972817337k=0.028288676714311668
Re=33041.168396608475
Nu0=73.15849352801939
Nu=114.30025607297654
h=107.78009974704906
q=6035.685585834748 W/m^2
Q=603.5685585834747 W
> Terminated with exit code 0.
```

PROBLEM :



Air at a pressure of 101.325 kPa and a temperature of 27 C flows with a velocity of 10 m/s over a flat plate 0.5 m long. Estimate the temperature and convective heat transfer if plate is heated with a constant flux of qs=2000 W/m

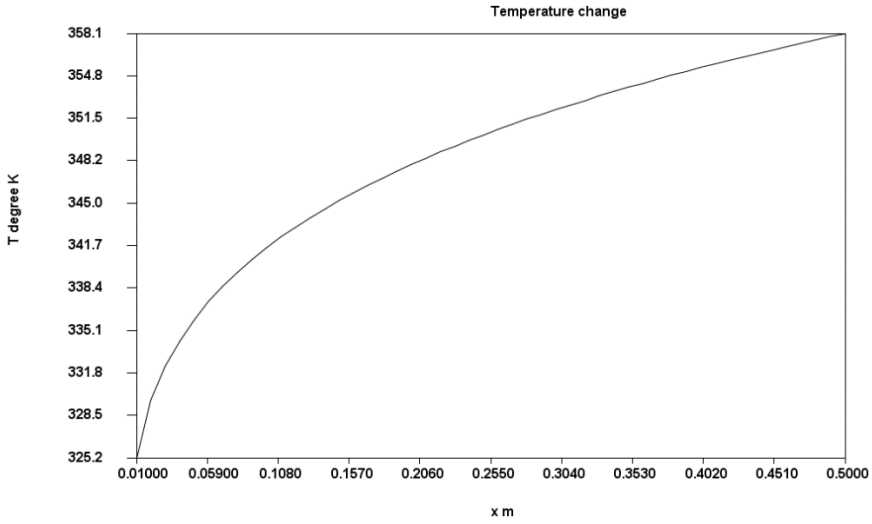
```
   public class HT_flat_plate_constant_flux
{  double TT[];
   double hh[];
   double xx[];
   double kavg,roavg,muavg;
```

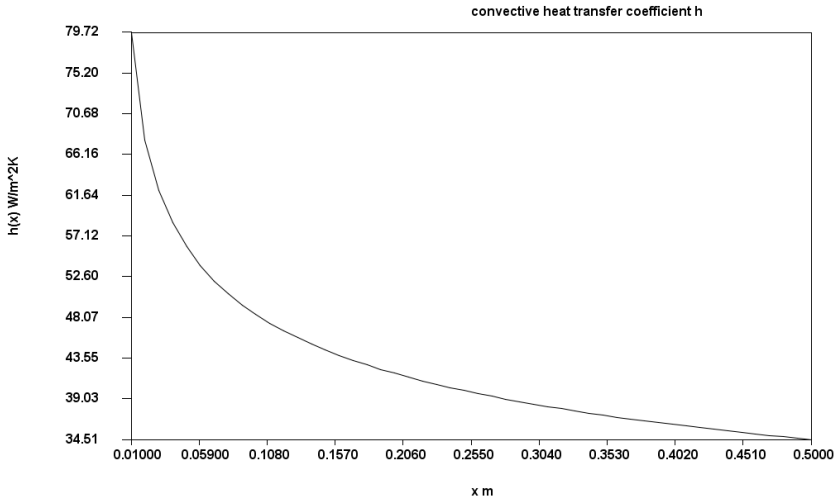
```
            public static double mu(double T)
  {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double k(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pr(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;

return y;
}
  public static double ro(double T,double P)
  {   double M=2.8965000E+01;
      double R=8314.5/M;
                return P*1e3/R/T;
  }
  public static double Nu(double U,double T,double P,double x)
  {//Local nusselt number
   double Re=Re(U,T,P,x);
   double Rec=2300.0;
   double Pr=Pr(T);
   double Nu=0.0;
   double Nu1=0.453*Math.sqrt(Re)*Math.pow(Pr,0.5);
   double Nu2=0.0308*Math.pow(Re,0.8)*Math.pow(Pr,(1.0/3.0));

   if(Re<Rec) Nu=Nu1;
   else       Nu=Nu2;
   return Nu;
  }
  public  double hL(double Uinf,double Tinf,double qs,double P,double L)
  {//overall heat transfer coefficient number
   double nul=0;
   double T=Tinf+0.0001;
   double Ts=Tinf;
   double U=Uinf;
   double dx=0.01;
   double x=0;
```

```
    double Nu=0;
    double h=0;
    double Re=0;
    int n=(int)(L/dx+0.0000001);
    TT=new double[n];
    hh=new double[n];
    xx=new double[n];
    TT[0]=Ts;
    hh[0]=0;
    xx[0]=0;
    kavg=0;
    roavg=0;
    muavg=0;
    int i;
    for(i=0;i<n;i++)
    {T=(Ts+Tinf)/2.0;
     x=i*dx;
     Nu=Nu(U,T,P,x+dx);
     kavg+=k(T);
     h=Nu*k(T)/(x+dx);
     Ts=Tinf+qs/h;
     TT[i]=Ts;
     hh[i]=h;
     xx[i]=x+dx;
     nul+=h*dx;
    }
    kavg/=n;
    nul/=L;
    return nul;
    }
    public static double Re(double U,double T,double P,double x)
    {   double mu=mu(T);
        double ro=ro(T,P);
        double Re=ro*U*x/mu;
                return Re;
    }
    public static void main(String arg[])
    { double Tinf=27+273.15;  //air temp. degree K;
     double U=10.0; //m/s
     double qs=2000.0; //W/m
     double L=0.5; //m
     double P=101.325; //kPa
     HT_flat_plate_constant_flux fp=new HT_flat_plate_constant_flux();
     double hL=fp.hL(U,Tinf,qs,P,L);
     Plot p=new Plot(fp.xx,fp.TT);
     p.setPlabel("Temperature change");
     p.setXlabel("x m");
     p.setYlabel("T degree K");
     p.plot();
     Plot p1=new Plot(fp.xx,fp.hh);
     p1.setPlabel("convective heat transfer coefficient h ");
     p1.setXlabel("x m");
     p1.setYlabel("h(x) W/m^2K");
     p1.plot();
     System.out.println("hL="+hL);
   }}
```

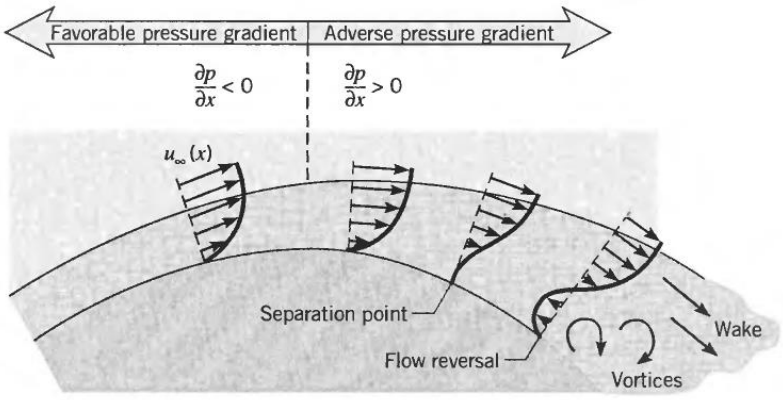


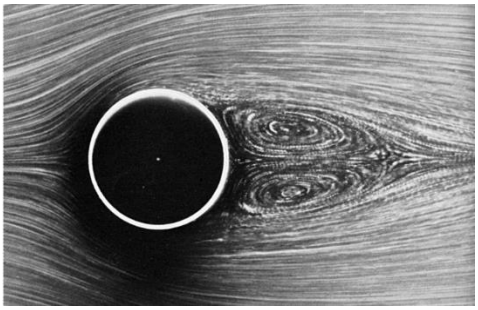



```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_flat_plate_constant_flux
hL=42.87425649829565
```

12.3. CONVECTIVE HEAT TRANSFER FOR THE CYLINDER IN CROSS FLOW

When flow passes through a cylinder curvature of the surface causes seperatio of the boundary layer from the surface, inverse direction flows flow vortices. Therefore flow can not be seperated as turbulent and laminar flow (being turbulent or laminar is depends on angle from the entry





Visualisation of flow field across the cylinder

Drag Coefficient in flow across the cylinder:

$C_D = \frac{F_D}{A_f(\frac{\rho U^2}{2})}$ where F_D is the drag force, A_f is the cylinder frontal area. Drag coefficient for a smooth cylinder with different Re numbers are listed in the table below.

Curve fitting coefficients $z=\log_{10}(Re)$

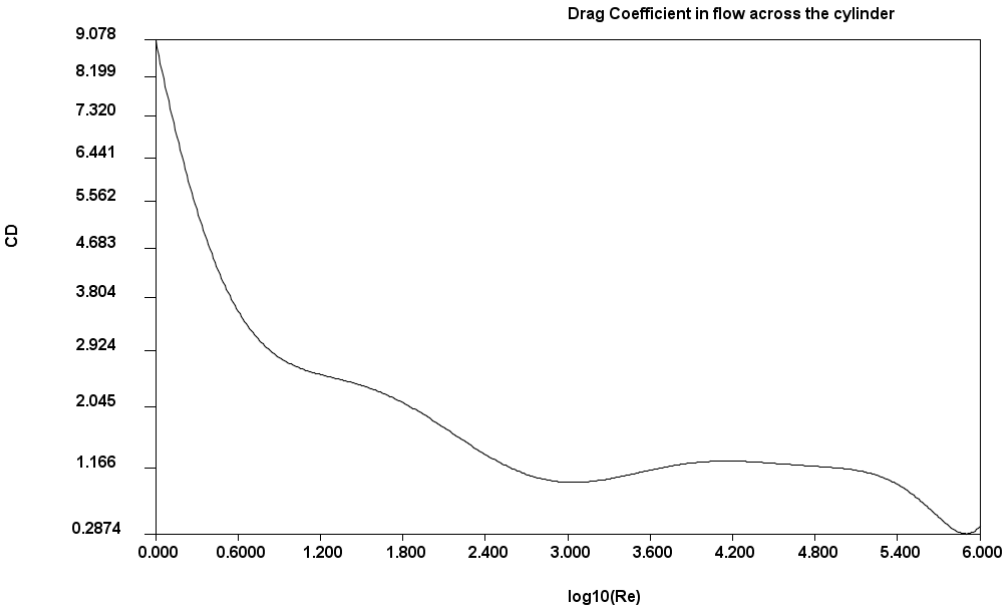
$C_D = a_o + a_1 z + a_2 z^2 + a_3 z^3 + .. + a_{10} z^{10}$

a0	9.07761707258
a1	-16.33173325480
a2	15.95794360346
a3	-8.53288480165
a4	4.48774899067
a5	-3.25082959759
a6	1.72646242637
a7	-0.53044708645
a8	0.09163176753
a9	-0.00833081014
a10	0.00031109691

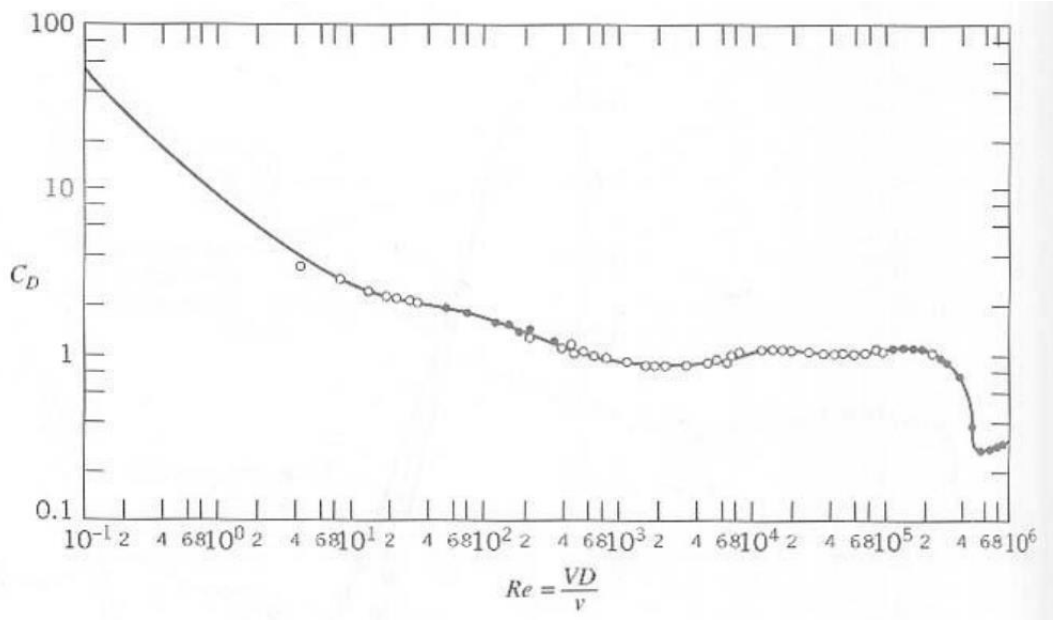
Let us plot the curve fitting function to observe the change of the drag coefficient

```
public class if_poly implements if_x
{ double a[];
  public if_poly(double ai[])
  {a=ai;}
   public double func(double x)
   {// this function calculates the value of
    // a polynomial
    int n=a.length;
    double ff;
    if(n!=0.0)
    { ff=a[n-1];
      for(int i=n-2;i>=0;i--)
      { ff=ff*x+a[i]; }
    }
    else
      ff=0;
    return ff;
    }
}
```

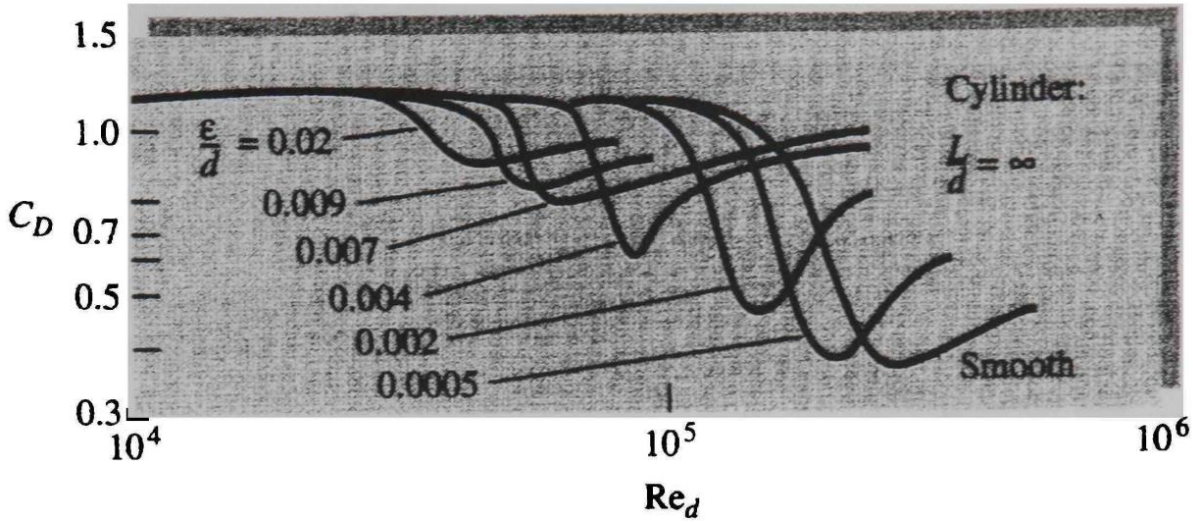
```
public class HT_drag_cylinder
{ public static void main(String arg[])
  {double a[]={9.07761707258,-16.33173325480,15.95794360346,-8.53288480165,
  4.48774899067,-3.25082959759,1.72646242637,-0.53044708645,0.09163176753,
  -0.00833081014,0.00031109691};
  if_poly f=new if_poly(a);
  Plot p=new Plot(f,0.0,6.0);
  p.plot();
}}
```



In the following figure drag coefficient is shown with actual data points



In the following figure effect of surface rougness on variation of the drag coefficient for across the cylinder is shown.



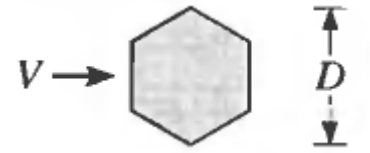
Heat transferr across a cylinder

Hilpert equation[31]:

$Nu_D = \frac{hD}{k} = CRe^m Pr^{1/3}$ $0.7 \leq Pr$ Properties are evaluated at T_∞

Re	C	m
0.4 - 4	0.989	0.330
4-40	0.911	0.385
40-4000	0.683	0.466
4000 - 40000	0.193	0.618
40000 - 400000	0.027	0.805

Hilpert equation can correlate different shapes as well

Geometry		Re	C	m
Square		$5x10^3 – 10^5$	0.246	0.588
Square		$5x10^3 – 10^5$	0.102	0.675
Hexagon		$5x10^3 – 10^5$	0.153	0.638
Hexagon		$5x10^3 – 1.95x10^4$ $1.95x10^4 – 10^5$	0.160 0.0385	0.638 0.782
Vertical Plate		$5x10^3 – 10^5$	0.153	0.638

Churchill & Bernstein[30] equation:

$$Nu_D = \frac{hD}{k} = 0.3 + \frac{0.62Re^{1/2}Pr^{1/3}}{\left[1+(0.4/Pr)^{2/3}\right]^{1/4}}\left[1+\left(\frac{Re}{282000}\right)^{5/8}\right]^{4/5} \qquad RePr \geq 0.2$$

Properties are evaluated at $T_f = (T_\infty + T_s)/2.0$ film temperature

Zukauskas[29] equation:

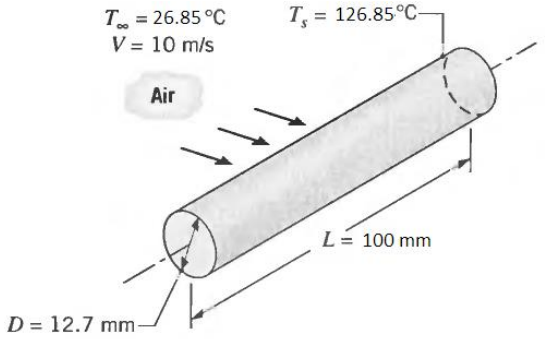
$Nu_D = \frac{hD}{k} = CRe^m Pr^n \left(\frac{Pr}{Pr_s}\right)^{1/4}$ $0.7 \leq Pr \leq 500$ $1 \leq Re \leq 10^6$ Properties are evaluated at T_∞ except Pr_s is evaluated at surface temperature. If $Pr \leq 10$ n=0.37, if $Pr \geq 10$ n=0.36

Coefficients of the equation are given as

Re	C	m
1 - 40	0.75	0.4
40-1000	0.51	0.5
$10^3 – 2x10^5$	0.26	0.6
$2x10^5 – 10^6$	0.076	0.7

In this equation Pr_s is calculated at surface temperature while remaining properties are at film temperature

Problem: Air at T_∞=26.85 C and 101.325 kPa pressure flowing through a cylinder with a velocity of 10 m/s. Cylinder has a diameter of D=12.7x10^{-3} m and has a length of 100 mm. Its surface temperature is measured as T_s=126.85 C. Calculate overall heat transfer coefficients and heat transfer by using all three equations given above.




```
public class HT_crossflow_cylinder_air
{
  public static double mu(double T)
  {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double k(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pr(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
```

```
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;

return y;
}
  public static double ro(double T,double P)
  {   double M=2.8965000E+01;
      double R=8314.5/M;
                return P*1e3/R/T;
  }
  public static double Nu1(double U,double Tinf,double Ts,double P,double D)
  {//Nusselt number Hilpert equation
   double Re=Re(U,Tinf,P,D);
   double Pr=Pr(Tinf);
   double Nu=0.0;
   double C=0;
   double m=0;
   if(Re>0.4 && Re<=4) {C=0.989;m=0.330;}
   else if(Re>4.0 && Re<=40) {C=0.911;m=0.385;}
   else if(Re>40.0 && Re<=4000) {C=0.683;m=0.466;}
   else if(Re>4000.0 && Re<=40000) {C=0.193;m=0.618;}
   else if(Re>40000.0 && Re<=400000) {C=0.027;m=0.805;}
   Nu=C*Math.pow(Re,m)*Math.pow(Pr,(1.0/3.0));
   return Nu;
   }
   public static double Nu2(double U,double Tinf,double Ts,double P,double D)
  {//Nusselt number Churchill & Bernstein equation
   double Tf=(Tinf+Ts)/2.0;
   double Re=Re(U,Tf,P,D);
   double Pr=Pr(Tf);
   double Nu=0.3+0.62*Math.pow(Re,0.5)*Math.pow(Pr,(1.0/3.0))/Math.pow((1+Math.pow((0.4/Pr),(2.0/3.0))),0.25);
   Nu*=Math.pow((1+Math.pow((Re/282000),(5.0/8.0))),(4.0/5.0));
   return Nu;
   }
  public static double Nu3(double U,double Tinf,double Ts,double P,double D)
  {//Nusselt number Zukauskas equation
   double Re=Re(U,Tinf,P,D);
   double Pr=Pr(Tinf);
   double Prs=Pr(Ts);
   double Nu=0.0;
   double C=0;
   double m=0;
   double n=0;
   if(Re>1 && Re<=40) {C=0.75;m=0.4;}
   else if(Re>40.0 && Re<=1000) {C=0.51;m=0.5;}
   else if(Re>40.0 && Re<=4000) {C=0.683;m=0.466;}
   else if(Re>1000.0 && Re<=2e5) {C=0.26;m=0.6;}
   else if(Re>2e5 && Re<=1e6) {C=0.076;m=0.7;}
   if(Pr<=10) n=0.37;
   else n=0.36;
  Nu=C*Math.pow(Re,m)*Math.pow(Pr,n)*Math.pow((Pr/Prs),0.25);
   return Nu;
   }
  public static double Re(double U,double T,double P,double D)
  {   double mu=mu(T);
      double ro=ro(T,P);
      double Re=ro*U*D/mu;
                return Re;
  }
  public static void main(String arg[])
  { double Tinf=26.85+273.15;  //air temp. degree K;
    double Ts=126.85+273.15;
    double U=10.0; //m/s
    double L=0.1; // m
    double P=101.325; //kPa
    double D=12.7e-3; //m
    double Nu1=Nu1(U,Tinf,Ts,P,D);
    double Nu2=Nu2(U,Tinf,Ts,P,D);
    double Nu3=Nu3(U,Tinf,Ts,P,D);
    System.out.println("Nu1="+Nu1+" Nu2="+Nu2+" Nu3="+Nu3);
    double k1=k(Tinf);
```

```
    double h1=Nu1*k1/D;
    double T=(Tinf+Ts)/2.0;
    double k2=k(T);
    double h2=Nu2*k2/D;
    double h3=Nu3*k1/D;
    System.out.println("h1="+h1+" h2="+h2+" h3="+h3);
    double A=Math.PI*D*L;
    double Q1=h1*A*(Ts-Tinf);
    double Q2=h2*A*(Ts-Tinf);
    double Q3=h3*A*(Ts-Tinf);
    System.out.println("Q1="+Q1+" Q2="+Q2+" Q3="+Q3);
}}
```

```
> "D:\co\java\bin\java.exe" HT_crossflow_cylinder_air
Nu1=44.613169311900535 Nu2=40.92978883398657 Nu3=50.61218317504857
h1=92.68388740754186 h2=96.69333508003521 h3=105.14684249511012
Q1=36.97922751278099 Q2=38.57892603459393 Q3=41.95172558721173
> Terminated with exit code 0.
```

The same problem carried out in excel environment for Curchill & Bernstein equation

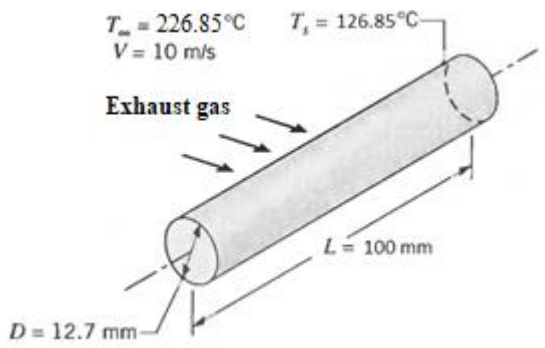
	Heat transfer over a cylinder Churchill & Bernstein equation:		working fluid : air			
M	2.8965000E+01	kg/kmol				
R	2.8705334E+02	J/kgK				
P	1.0132500E+02	kPa				
	ρ	Cp	μ	k	Pr	
	kg/m^3					
a0		1.89668613E+00	-7.00796942E-01	-5.96297846E-01	1.47352515E+00	
a1		-2.23937197E-02	8.85836561E-02	1.00247973E-01	-1.73708682E-02	
a2		2.40885259E-04	-1.20909367E-04	7.32722776E-05	1.80473009E-04	
a3		-1.45254290E-06	1.73074497E-07	-1.02111800E-06	-1.07436904E-06	
a4		5.43552393E-09	-1.45011707E-10	4.21691040E-09	3.99304707E-09	
a5		-1.32310554E-11	-4.33936641E-14	-1.04115589E-11	-9.67956666E-12	
a6		2.13239465E-14	3.14868466E-16	1.67088379E-14	1.55567247E-14	
a7		-2.25744420E-17	-4.41548734E-19	-1.75371889E-17	-1.64359568E-17	
a8		1.50864395E-20	3.25615211E-22	1.16134919E-20	1.09674418E-20	
a9		-5.76865768E-24	-1.28963912E-25	-4.40290627E-24	-4.18869082E-24	
a10		9.61540784E-28	2.16661936E-29	7.28284943E-28	6.97523663E-28	
result:for T	1.0085234E+00	1.0083819E+00	2.0867141E-05	3.0002729E-02	7.0130741E-01	
result: for Ts	8.8245794E-01	1.0140736E+00	2.3055411E-05	3.3453105E-02	6.9887896E-01	
	degree C	degree K				
T	7.6850000E+01	3.5000000E+02	average temperature			
Ts	1.2685000E+02	4.0000000E+02	surface temperature			
Tinf	2.6850000E+01	3.0000000E+02	air stream temp			
P	1.0132500E+00	bar	Pressure			
U	1.0000000E+01	m/s	air velocity			
D	1.2700000E-02	m	Diameter			
L	1.0000000E-01	m	Length			
Re=ρUL/μ	6.1379979E+03		Reynolds' number			
Nu	4.0908036E+01		Nusselt's number			
k	3.0002729E-02	W/mK	thermal conductivity			
h	9.6641947E+01	W/m^2K	thermal convectivity			
A	3.9898227E-03	m^2	surface area			
Q	3.8558423E+01	W	heat transfer			

Problem: Exhaust gas at T_∞=226.85 C and 101.325 kPa pressure flowing through a cylinder with a velocity of 10 m/s. Cylinder has a diameter of D=12.7x10^{-3} m and has a length of 100 mm. Its surface temperature is measured as T_s=126.85 C. Calculate overall heat transfer coefficients and heat transfer by using all three equations given above.

Exhaust gas :

CO2	3	9.81%
H2O	4	13.08%

O2	1	3.27%
N2	22.57143	73.83%




```
public class HT_crossflow_cylinder_exhaust
{  Gmix g;
   public HT_crossflow_cylinder_exhaust(String gname[],double n[])
   {g=new Gmix("exhaust gas",gname,n);}

  public double h1(double V,double Tinf,double Ts,double P,double D)
  {//Nusselt number Hilpert equation
   double Re=Re(V,Tinf,P,D);
   double Pr=g.Prandtl(Tinf);
   double k1=g.k(Tinf);
   double Nu=0.0;
   double C=0;
   double m=0;
   if(Re>0.4 && Re<=4) {C=0.989;m=0.330;}
   else if(Re>4.0 && Re<=40) {C=0.911;m=0.385;}
   else if(Re>40.0 && Re<=4000) {C=0.683;m=0.466;}
   else if(Re>4000.0 && Re<=40000) {C=0.193;m=0.618;}
   else if(Re>40000.0 && Re<=400000) {C=0.027;m=0.805;}
   Nu=C*Math.pow(Re,m)*Math.pow(Pr,(1.0/3.0));
   return Nu*k1/D;
   }
   public double h2(double V,double Tinf,double Ts,double P,double D)
  {//Nusselt number Churchill & Bernstein equation
   double Tf=(Tinf+Ts)/2.0;
   double Re=Re(V,Tf,P,D);
   double Pr=g.Prandtl(Tinf);
   double k1=g.k(Tinf);
   double Nu=0.3+0.62*Math.pow(Re,0.5)*Math.pow(Pr,(1.0/3.0))/Math.pow((1+Math.pow((0.4/Pr),(2.0/3.0))),0.25);
   Nu*=Math.pow((1+Math.pow((Re/282000),(5.0/8.0))),(4.0/5.0));
   return Nu*k1/D;
   }
  public double h3(double V,double Tinf,double Ts,double P,double D)
  {//Nusselt number Zukauskas equation
   double Re=Re(V,Tinf,P,D);
   double Pr=g.Prandtl(Tinf);
   double k1=g.k(Tinf);
   double Prs=g.Prandtl(Ts);
   double Nu=0.0;
   double C=0;
   double m=0;
   double n=0;
   if(Re>1 && Re<=40) {C=0.75;m=0.4;}
   else if(Re>40.0 && Re<=1000) {C=0.51;m=0.5;}
   else if(Re>40.0 && Re<=4000) {C=0.683;m=0.466;}
   else if(Re>1000.0 && Re<=2e5) {C=0.26;m=0.6;}
   else if(Re>2e5 && Re<=1e6) {C=0.076;m=0.7;}
   if(Pr<=10) n=0.37;
   else n=0.36;
   Nu=C*Math.pow(Re,m)*Math.pow(Pr,n)*Math.pow((Pr/Prs),0.25);
   return  Nu*k1/D;
   }
  public  double Re(double V,double T,double P,double D)
  {   double mu=g.vis(T);
      double ro=1.0/g.v(T,P);
      double Re=ro*V*D/mu;
               return Re;
```

```
  }
  public static void main(String arg[])
  { String s1[]={"co2","h2o","o2","n2"};
    double n1[]={0.0981,0.1308,0.0327,0.7383};
     HT_crossflow_cylinder_exhaust ht=new  HT_crossflow_cylinder_exhaust(s1,n1);
              double Ts=126.85+273.15;  //water temp. degree K;
    double Tinf=226.85+273.15; //exhaust gas temperature
    double Tf=(Ts+Tinf)/2.0;
    double V=10.0; //m/s
    double L=5; // m
    double P=1.01325; //bar
    double D=12.7e-3; //m
    double h1=ht.h1(V,Tinf,Ts,P,D);
    double h2=ht.h2(V,Tf,Ts,P,D);
    double h3=ht.h3(V,Tinf,Ts,P,D);
    System.out.println("h1="+h1+" h2="+h2+" h3="+h3);
    double A=Math.PI*D*L;
    double Q1=h1*A*(Tinf-Ts);
    double Q2=h2*A*(Tinf-Ts);
    double Q3=h3*A*(Tinf-Ts);
    System.out.println("Q1="+Q1+" Q2="+Q2+" Q3="+Q3);

}}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_crossflow_cylinder_exhaust
h1=17.411652470236252 h2=18.189882238499077 h3=15.136683528202356
Q1=347.3470287446902 Q2=362.87202260433924 Q3=301.9634154516549
> Terminated with exit code 0.
```

PROBLEM: Water with temperature of T=26.85 ℃ and P=300 kPa is flowing through a Di=25x10^{-3} m , Do=28x10^{-3} m commercial steel pipe(k=36 W/mK) wit a velocity of V=2 m/s. Total pipe length L=200 m. In outside of the pipe exhaust gases is flowing across the pipe with a velocity of V=10 m/s. Exhaust gas temperature is constant and T_∞=226.85 ℃. Find the total heat transfer and pipe exit temperature also find total pressure drop.

Exhaust gas combination is as follows:

CO2	3	9.81%
H2O	4	13.08%
O2	1	3.27%
N2	22.57143	73.83%

```
 import java.io.*;
public class HT_inside_Outside_pipe_exhaust_gas
{ public Gmix g;  //exhaust gas mix
   public steamIAPWS_IF97 st; //steam and water

public  HT_inside_Outside_pipe_exhaust_gas(String gname[],double n[])
{g=new Gmix("exhaust gas",gname,n);
 st=new steamIAPWS_IF97();
}

  public double h1(double V,double Tinf,double Ts,double P,double D)
  {//Nusselt number Hilpert equation
   double Re=Re(V,Tinf,P,D);
   double Pr=g.Prandtl(Tinf);
   double k1=g.k(Tinf);
   double Nu=0.0;
   double C=0;
   double m=0;
   if(Re>0.4 && Re<=4) {C=0.989;m=0.330;}
   else if(Re>4.0 && Re<=40) {C=0.911;m=0.385;}
   else if(Re>40.0 && Re<=4000) {C=0.683;m=0.466;}
   else if(Re>4000.0 && Re<=40000) {C=0.193;m=0.618;}
   else if(Re>40000.0 && Re<=400000) {C=0.027;m=0.805;}
   Nu=C*Math.pow(Re,m)*Math.pow(Pr,(1.0/3.0));
   return Nu*k1/D;
   }
   public double h2(double V,double Tinf,double Ts,double P,double D)
```

```
 {//Nusselt number Churchill & Bernstein equation
  double Tf=(Tinf+Ts)/2.0;
  double Re=Re(V,Tf,P,D);
  double Pr=g.Prandtl(Tinf);
  double k1=g.k(Tinf);
  double Nu=0.3+0.62*Math.pow(Re,0.5)*Math.pow(Pr,(1.0/3.0))/Math.pow((1+Math.pow((0.4/Pr),(2.0/3.0))),0.25);
  Nu*=Math.pow((1+Math.pow((Re/282000),(5.0/8.0))),(4.0/5.0));
  return Nu*k1/D;
  }
 public double h3(double V,double Tinf,double Ts,double P,double D)
 {//Nusselt number Zukauskas equation
  double Re=Re(V,Tinf,P,D);
  double Pr=g.Prandtl(Tinf);
  double k1=g.k(Tinf);
  double Prs=g.Prandtl(Ts);
  double Nu=0.0;
  double C=0;
  double m=0;
  double n=0;
  if(Re>1 && Re<=40) {C=0.75;m=0.4;}
  else if(Re>40.0 && Re<=1000) {C=0.51;m=0.5;}
  else if(Re>40.0 && Re<=4000) {C=0.683;m=0.466;}
  else if(Re>1000.0 && Re<=2e5) {C=0.26;m=0.6;}
  else if(Re>2e5 && Re<=1e6) {C=0.076;m=0.7;}
  if(Pr<=10) n=0.37;
  else n=0.36;
  Nu=C*Math.pow(Re,m)*Math.pow(Pr,n)*Math.pow((Pr/Prs),0.25);
  return  Nu*k1/D;
  }
 public  double Re(double V,double T,double P,double D)
 {   double mu=g.vis(T);
     double ro=1.0/g.v(T,P);
     double Re=ro*V*D/mu;
              return Re;
 }
  //water side (inside pipe)
 public  double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
   //water side (inside pipe)
 public  double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
  double Nu=0;
  double x=Re/1000;
  double f=0;
  if(Re<2300) Nu=3.66;
  else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
  else {f=f_Goudar(Re,eod);
  Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
  }
 return Nu;
  }
public static void main(String arg[]) throws IOException
{       String s1[]={"co2","h2o","o2","n2"};
        double n1[]={0.0981,0.1308,0.0327,0.7383};
        HT_inside_Outside_pipe_exhaust_gas ht=new  HT_inside_Outside_pipe_exhaust_gas(s1,n1);
                double Ti=26.85+273.15;  // degree K
         double Tinf=226.85+273.15; //degree K;
         double To=70.0+273.15;         //guess for outlet temperature To
         double T;
                double P=300.0;         // kPa
                double V=2.0;           // velocity m/s
                double e=0.16e-3;       //surface rougness m
```

```
                double Di=25e-3;        // diameter m
                double Do=28e-3;        // diameter m
                double eod=e/Di;
                double L=200; //m
                double kw=36.0; //W/mK
                double a[];
                double ro=0,Cp=0,k,mu,Pr,Re,f=0,Nu,h1,h2,U,Q1;
                // iteration fr surface temperature T
                for(int i=0;i<5;i++)
                {
                        T=(Ti+To)/2.0;
                a=ht.st.property("tp",T,P);
                ro=a[15];
                Cp=a[7];
                k=a[12];
                mu=a[11];
                Pr=a[14];
                Re=ro*V*Di/mu;
                f=ht.f_Goudar(Re,eod);
                System.out.println("f Goudar = "+f);
                Nu=ht.Nu(Re,eod,Pr);
        h1=Nu*k/Di;
        //outside of the pipe
        h2=ht.h3(V,Tinf,T,P,Do);
        U=1.0/(1.0/(h1*(Math.PI*Di*L))+Math.log(Do/Di)/(2.0*Math.PI*kw*L)+1.0/(h2*(Math.PI*Do*L)));
        Q1=U*(Tinf-T);
         System.out.println("==== iteration i= "+i+"==============");
        System.out.println("Surface Temperature = "+T+" degree K");
        System.out.println("Cp = "+Cp+"J/kgK ");
        System.out.println("Q = "+Q1+"W/m^2K ");
        System.out.println("hi = "+h1+"W/m^2K ");
        System.out.println("ho = "+h2+"W/m^2K ");
        System.out.println("U = "+U+"W/K ");
        To=Ti+Q1/(Cp*1e3);
        T=(h2*Tinf+h1*T)/(h1+h2);
        System.out.println("Ti="+Ti+" degree K To = "+To+"  degree K");
        }
                double A=Math.PI*Di*Di/4.0;
                double m=ro*A*V;  //kg/s
                System.out.println("m = "+m+" kg/s");
                double dP=f*L/Di*ro*V*V/2.0;
                System.out.println("dP = "+dP*1e-3+" kPa");
}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_inside_Outside_pipe_exhaust_gas
f Goudar = 0.03367500764878147
==== iteration i= 0==============
Surface Temperature = 321.575 degree K
Cp = 4.1787826965244275J/kgK
Q = 326893.84212869185W/m^2K
hi = 15582.61533764829W/m^2K
ho = 105.41174252632993W/m^2K
U = 1832.1078443530437W/K
Ti=300.0 degree K To = 378.22704980581443  degree K
f Goudar = 0.03347045133256282
==== iteration i= 1==============
Surface Temperature = 339.1135249029072 degree K
Cp = 4.185253160282877J/kgK
Q = 295192.0122487482W/m^2K
hi = 18388.178143962425W/m^2K
ho = 105.44555900520638W/m^2K
U = 1834.7845092050397W/K
Ti=300.0 degree K To = 370.5314591361055  degree K
f Goudar = 0.03350799800550761
==== iteration i= 2==============
Surface Temperature = 335.26572956805273 degree K
Cp = 4.1832757696945855J/kgK
Q = 302167.3266355612W/m^2K
hi = 17789.875064131727W/m^2K
ho = 105.43850046212334W/m^2K
U = 1834.2711922859328W/K
Ti=300.0 degree K To = 372.2322273909334  degree K
f Goudar = 0.033499399509913846
==== iteration i= 3==============
```

```
Surface Temperature = 336.1161136954667 degree K
Cp = 4.183686088030815J/kgK
Q = 300626.5911366042W/m^2K
hi = 17922.995230472472W/m^2K
ho = 105.4400763851273W/m^2K
U = 1834.3877358264012W/K
Ti=300.0 degree K To = 371.85687090546116  degree K
f Goudar = 0.03350128202455611
==== iteration i= 4==============
Surface Temperature = 335.9284354527306 degree K
Cp = 4.183594226662113J/kgK
Q = 300966.67125708814W/m^2K
hi = 17893.658778422763W/m^2K
ho = 105.43972937523914W/m^2K
U = 1834.3621704806678W/K
Ti=300.0 degree K To = 371.93973768751823  degree K
m = 0.9639246565181816 kg/s
dP = 526.2893776679816 kPa

> Terminated with exit code 0.
```

PROBLEM 1: A combustion of natural gas (%95 CH_4+5% C_2H_6) with 100 % excess air is carried out in a combustion chamber. Air and fuel is at T=298 K at the inlet of the reaction and T=520 K at the exit. Exit exhaust gases flows across a pipe with a diameter of D=10 mm and length L=10m. Inside we have water with inlet temperature of T=300 K and velocity V=1.2 m/s. What is the total heat transfer and exit temperature of water?

PROBLEM 2: Water flows in a pipeline (commercial steel) of diameter D=0.2 m and L=100 m with constant temperature of T=300 K. Inlet pressure is P=20 bar. What is the exit pressure?

PROBLEM: A circular pipe with a diameter of 0.3 m and a length of 1.8 m has a surface temperature of 24 $^{\circ}$C. The air at -10°C moves in cross flow over the pipe with a velocity of 5m/s . What is the heat loss from the pipe?

	Heat transfer over a cylinder		**working fluid : air**						
M	**2.8965000E+01**	**kg/kmo**							
R	**2.8705334E+02**	**J/kgK**							
P	**1.0132500E+02**	**kPa**							
	ρ	**Cp**	**μ**	**k**	**Pr**				
	kg/m^3								
a0		1.8966	-7.007969E-01	-	1.4735				
a1		-	8.858366E-02	1.0024	-				
a2		2.4088	-1.209094E-04	7.3272	1.8047				
a3		-	1.730745E-07	-	-				
a4		5.4355	-1.450117E-10	4.2169	3.9930				
a5		-	-4.339366E-14	-	-				
a6		2.1323	3.148685E-16	1.6708	1.5556				
a7		-	-4.415487E-19	-	-				
a8		1.5086	3.256152E-22	1.1613	1.0967				
a9		-	-1.289639E-25	-	-	ρ	**h**	**u**	**s**
a10		9.6154	2.166619E-29	7.2828	6.9752	1.0080			
result:for T	1.2599792E+00	1.0059	1.7567252E-05	2.4894	7.0985	0.9920	350.	250.	6.85
result: for Ts	1.1878956E+00	1.0059	1.8399751E-05	2.6172	7.0717	v			

	degree C	**degree**	
T	7.0000000E+00	2.8015	air temperature
Ts	2.4000000E+01	2.9715	surface
Tinf	-1.0000000E+01	2.6315	
P	1.0132500E+00	bar	
ro	1.2599792E+00		
U	5.0000000E+00		air velocity
D	3.0000000E-01		diameter
Re=ρUD/μ	1.0758478E+05		Reynolds'

Pr	7.0985568E-01		
Prs	7.0717426E-01		
Nu	2.7059090E+02		Hilbert 7.27
k	2.4894190E-02	W/mK	
L	1.8000000E+00	m	
h	2.2453804E+01	W/m^2	thermal
A	1.6964600E+00	W/m	heat transfer
Q	1.2951274E+03	W	
Nu	2.2619475E+02		Churchill &
h	1.8769783E+01		
Q	1.0826344E+03	W	
C	2.6000000E-01		
m	6.0000000E-01		
n	3.7000000E-01		
Nu	2.3953427E+02		Zukauskas 7.53
h	1.9876706E+01		
Q	1.1464812E+03	W	

```
public class HT_crossflow_cylinder_air1
{
  public static double mu(double T)
  {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double k(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pr(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;

return y;
```

```
}
  public static double ro(double T,double P)
  {   double M=2.8965000E+01;
      double R=8314.5/M;
                return P*1e3/R/T;
  }

   public static double h2(double U,double Tinf,double Ts,double P,double D)
  {//Nusselt number Churchill & Bernstein equation
   double Tf=(Tinf+Ts)/2.0;
   double Re=Re(U,Tf,P,D);
   double Pr=Pr(Tf);
   double k2=k(Tf);
   double Nu=0.3+0.62*Math.pow(Re,0.5)*Math.pow(Pr,(1.0/3.0))/Math.pow((1+Math.pow((0.4/Pr),(2.0/3.0))),0.25);
   Nu*=Math.pow((1+Math.pow((Re/282000),(5.0/8.0))),(4.0/5.0));
   return Nu*k2/D;
   }
  public static double h3(double U,double Tinf,double Ts,double P,double D)
  {//Nusselt number Zukauskas equation
  double T=(Tinf+Ts)/2.0;
   double Re=Re(U,T,P,D);
   double Pr=Pr(T);
   double Prs=Pr(Ts);
    double k3=k(T);
   double Nu=0.0;
   double C=0;
   double m=0;
   double n=0;
   if(Re>1 && Re<=40) {C=0.75;m=0.4;}
   else if(Re>40.0 && Re<=1000) {C=0.51;m=0.5;}
   else if(Re>40.0 && Re<=4000) {C=0.683;m=0.466;}
   else if(Re>1000.0 && Re<=2e5) {C=0.26;m=0.6;}
   else if(Re>2e5 && Re<=1e6) {C=0.076;m=0.7;}
   if(Pr<=10) n=0.37;
   else n=0.36;
   Nu=C*Math.pow(Re,m)*Math.pow(Pr,n)*Math.pow((Pr/Prs),0.25);
   return  Nu*k3/D;
   }
  public static double Re(double U,double T,double P,double D)
  {   double mu=mu(T);
      double ro=ro(T,P);
      double Re=ro*U*D/mu;
      System.out.println("Re="+Re);
                return Re;
  }
  public static void main(String arg[])
  { double Ts=24.0+273.15;
    double Tinf=-10.0+273.15; //air temperature
    double V=5.0; //m/s
    double L=1.8; // m
    double P=101.325; //kPa
    double D=0.3; //m
    double h2=h2(V,Tinf,Ts,P,D);
    double h3=h3(V,Tinf,Ts,P,D);
    System.out.println(" h2="+h2+" h3="+h3);
    double A=Math.PI*D*L;
    double Q2=h2*A*(Ts-Tinf);
    double Q3=h3*A*(Ts-Tinf);
    System.out.println(" Q2="+Q2+" Q3="+Q3);

}}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_crossflow_cylinder_air1
Re=107584.77549676617
Re=107584.77549676617
 h2=18.78019319087609 h3=19.876706041542185
 Q2=1083.234803412284 Q3=1146.4812711220725

> Terminated with exit code 0.
```

12.4 CONVECTIVE HEAT TRANSFER FOR THE SPHERE IN CROSS FLOW

Drag Coefficient in flow across the sphere:

$C_D = \frac{F_D}{A_f(\frac{\rho U^2}{2})} = \frac{24}{Re} \quad Re \geq 0.5$

Whitaker[28] equation:

$$Nu_D = \frac{hD}{k} = 2 + (0.62Re^{0.5} + 0.62Re^{2/3})Pr^{0.4}\left(\frac{\mu}{\mu_s}\right)^{.25} \quad 0.71 \leq Pr \leq 380 \;\; 3.5 \leq$$

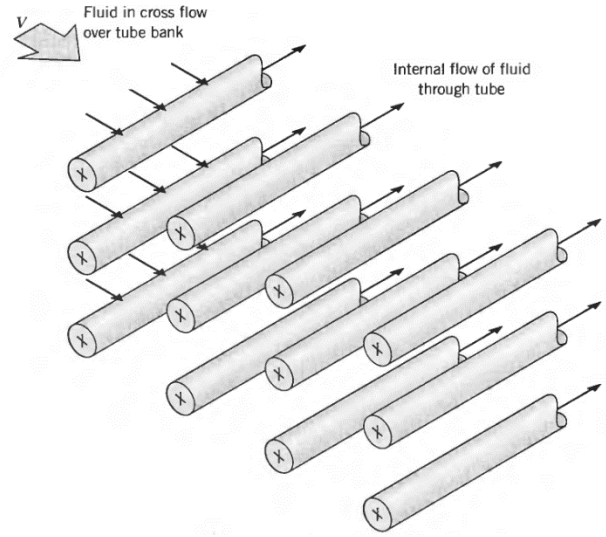
$Pr \leq 7.6x10^4 \quad 1 \leq \frac{\mu}{\mu_s} \leq 3.2$ Properties are evaluated at T_∞ temperature

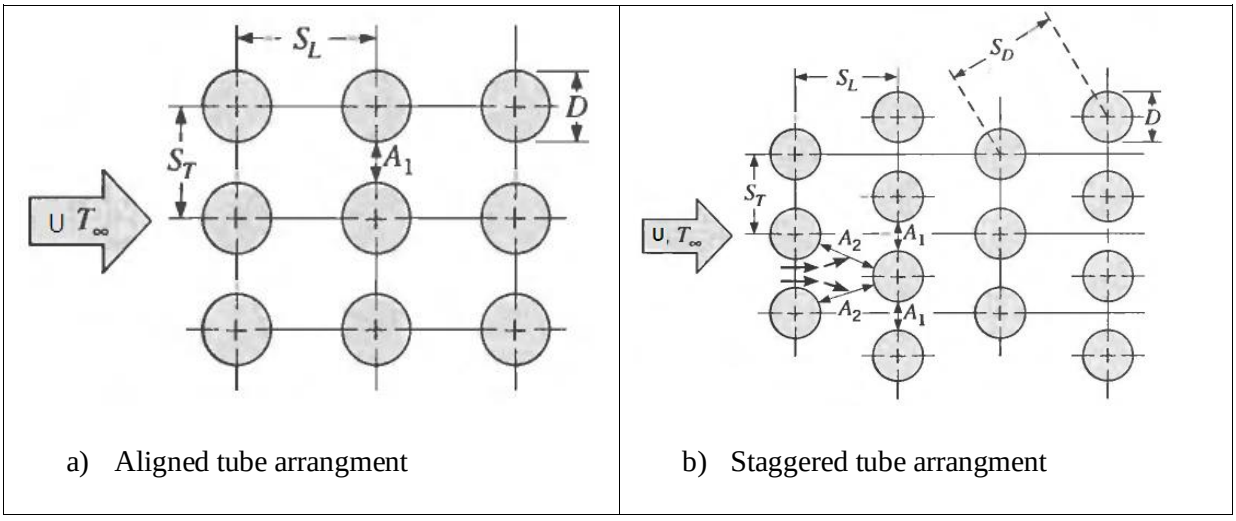
12.5 CONVECTIVE HEAT TRANSFER FOR THE BANK OF PIPES IN CROSS FLOW

When more than one tube existed in the path of cross flow, the flow temperature will also reduced so it will required a different heat transfer mechanism to evaluate this particular case. Tubes palced in the row staggered or aligned geometry. Distances between the pipes are also an important parameter to calculate heat transfer.

Veloctiy in the passage will be higher than the inlet velocity U, therefore should be replaced with the maximum velocity in the passages. For aligned confuguration:

$$U_{max} = \frac{S_T}{S_T - D}U$$





a) Aligned tube arrangment

b) Staggered tube arrangment

For staggered configuration passages can be through crossection A_1 or A_2 therefore

$$S_D = \left[S_L^2 + \left(\frac{S_T}{2}\right)^2\right]^{0.5}$$

if $2(S_D - D) < (S_T - D)$ $U_{max} = \frac{S_T}{2(S_D - D)} U$

else $U_{max} = \frac{S_T}{S_T - D} U$

$Re_{Dmax} = \frac{\rho u_{max} D}{\mu}$ Re_{Dmax} is given as Re in the following equations

Heat transfer equation for the bank of tubes (Zukauskas[27] equation)

$$Nu_D = C_2 C Re^m Pr^{0.36} \left(\frac{Pr}{Pr_s}\right)^{0.25} \quad 0.7 \le Pr \le 500 \quad 1000 \le Re \le 2x10^6$$

Where all properties except Pr_s will be evaluated at inlet properties. Pr_s will be evaluated at surface temperature.

confuguration		Re	C	m
Aligned		10-100	0.8	0.4
Aligned		100-1000	calculate as single cylinder	
Aligned	$\left(\frac{S_T}{S_L}\right) > 0.7$	1e3-2e5	0.27	0.63
Aligned		2e5-2e6	0.021	0.84
Staggered		10-100	0.9	0.4
Staggered		100-1000	calculate as single cylinder	
Staggered	$\left(\frac{S_T}{S_L}\right) < 2$	1e3-2e5	$0.35\left(\frac{S_T}{S_L}\right)^{1/5}$	0.6
Staggered	$\left(\frac{S_T}{S_L}\right) > 2$	1e3-2e5	0.4	0.6
Staggered		2e5-2e6	0.022	0.84

C_2 coefficients are correction for small number of data as given in the following table

N_T	1	2	3	4	5	7	10	13	16	>=20
aligned	0.7	0.8	0.86	0.90	0.92	0.95	0.97	0.98	0.99	1
Staggered	0.64	0.76	0.84	0.89	0.92	0.95	0.97	0.98	0.99	1

Assuming that temperature going into the bank is T_i and temperature going out of the bank is T_0 log mean temperature difference (derived by assuming C_p is constant) is

$$\Delta T_{lm} = \frac{(T_s - T_i) - (T_s - T_0)}{ln\frac{(T_s - T_i)}{(T_s - T_0)}}$$

temperature going out of the bank is T_0 can be estimated from

$\frac{(T_s - T_i)}{(T_s - T_0)} = exp\left(-\frac{(\pi D) N * N_T h}{\rho U N_T S_T C_p}\right)$ once ΔT_{lm} known the heat transfer rate can be calculated

As another approach heat transfer in each row can be calculated and temperature of the exit can be calculated from total heat transfer and process continue columnwise.

PROBLEM : Consider a staggered arrangement for which the tube outside diameter is $16.4x10^{-2}$ m and the longitudinal and tranverse pitches are S_L=34.3 $x10^{-2}$ m and S_T=31.3 $x10^{-3}$m. There are seven rows of tubes in the airflow direction and d 8 tubes per row. Tube surface temperature T_s=72.85 C, while the air upstream temperature,pressure and velocity are 16.85 C, 101.325 kPa and 6 m/s respectively.

public class HT_crossflow_cylinder__bank_air

```
{
  public static double mu(double T)
  {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double k(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pr(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
return y;
}
public static double Cp(double T)
{double a[]={1.89668613E+00,
-2.23937197E-02,
2.40885259E-04,
-1.45254290E-06,
5.43552393E-09,
-1.32310554E-11,
2.13239465E-14,
-2.25744420E-17,
1.50864395E-20,
-5.76865768E-24,
9.61540784E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
return y*1e3;
}


  public static double ro(double T,double P)
  {   double M=2.8965000E+01;
      double R=8314.5/M;
                return P*1e3/R/T;
  }
  public static double Nu4(String ca,double ST,double SL,double U,double Tinf,double Ts,double P,double D,int NL)
  {double Umax=0;
  double SD=0.0;
```

```
double C=0;
double m=0;
double Nu=0;
double T=Tinf;
double Pr=Pr(T);
double Prs=Pr(Ts);
System.out.println("Pr="+Pr);
System.out.println("Prs="+Prs);
double C2=0;
double Re=0;
double nn[]={1,2,3,4,5,7,10,13,16,20,50,100};
if(ca.equals("aligned"))
{Umax=ST/(ST-D)*U;
 double al[]={0.7,0.8,0.86,0.90,0.92,0.95,0.97,0.98,.99,1,1,1};
 CS1 cs=new CS1(nn,al,0.0,0.0);
 C2=cs.func((double)NL);
 }
else if(ca.equals("staggered"))
{SD=Math.sqrt(SL*SL+ST*ST/4.0);
 if(2.0*(SD-D)<(ST-D)) Umax=ST/2.0/(SD-D)*U;
 else Umax=ST/(ST-D)*U;
 double al[]={0.64,0.76,0.84,0.89,0.92,0.95,0.97,0.98,.99,1,1,1};
 CS1 cs=new CS1(nn,al,0.0,0.0);
 C2=cs.func((double)NL);
}

Re=Re(Umax,T,P,D);

if(ca.equals("aligned"))
{if(Re>10 && Re<=100) {C=0.8;m=0.4;Nu=C2*C*Math.pow(Re,m)*Math.pow(Pr,0.36)*Math.pow((Pr/Prs),0.25);}
 else if(Re>100 && Re<=1000) {Nu=C2*Nu2(Umax,Tinf,Ts,P,D);;}
 else if(Re>1000 && Re<=2e5) {C=0.27;m=0.63;Nu=C2*C*Math.pow(Re,m)*Math.pow(Pr,0.36)*Math.pow((Pr/Prs),0.25);}
 else if(Re>2e5 && Re<=2e6) {C=0.021;m=0.84;Nu=C2*C*Math.pow(Re,m)*Math.pow(Pr,0.36)*Math.pow((Pr/Prs),0.25);}
}
else //ca.equals("staggered")
{if(Re>10 && Re<=100) {C=0.9;m=0.4;Nu=C2*C*Math.pow(Re,m)*Math.pow(Pr,0.36)*Math.pow((Pr/Prs),0.25);}
 else if(Re>100 && Re<=1000) {Nu=C2*Nu2(Umax,Tinf,Ts,P,D);;}
 else if(Re>1000 && Re<=2e5)
 {if(ST/SL>2) {C=0.4;m=0.6;Nu=C2*C*Math.pow(Re,m)*Math.pow(Pr,0.36)*Math.pow((Pr/Prs),0.25);}
  else if(ST/SL<2) {C=0.35*Math.pow((ST/SL),0.2);m=0.6;Nu=C*Math.pow(Re,m)*Math.pow(Pr,0.36)*Math.pow((Pr/Prs),0.25);}
 }
 else if(Re>2e5 && Re<=2e6) {C=0.022;m=0.84;Nu=C2*C*Math.pow(Re,m)*Math.pow(Pr,0.36)*Math.pow((Pr/Prs),0.25);}
}
System.out.println("Re="+Re+"Umax="+Umax+"C2="+C2+"C="+C+"m="+m);
return Nu;
}

 public static double Nu2(double U,double Tinf,double Ts,double P,double D)
{//Nusselt number Churchill & Bernstein equation
 double Tf=(Tinf+Ts)/2.0;
 double Re=Re(U,Tf,P,D);
 double Pr=Pr(Tf);
 double Nu=0.3+0.62*Math.pow(Re,0.5)*Math.pow(Pr,(1.0/3.0))/Math.pow((1+Math.pow((0.4/Pr),(2.0/3.0))),0.25);
 Nu*=Math.pow((1+Math.pow((Re/282000),(5.0/8.0))),(4.0/5.0));
 return Nu;
 }

public static double Re(double U,double T,double P,double D)
{   double mu=mu(T);
    double ro=ro(T,P);
    double Re=ro*U*D/mu;
              return Re;
}
public static void main(String arg[])
{ double Tinf=16.85+273.15;  //air temp. degree K;
  double Ts=72.85+273.15;
  double U=6.0; //m/s
  double L=1.0; // m
  double P=101.325; //kPa
  double D=16.4e-3; //m
  double ST=31.3e-3;// m
  double SL=34.3e-3; //m
  int NT=7;
  int N=8;
  double Nu4=Nu4("staggered",ST,SL,U,Tinf,Ts,P,D,NT);
  System.out.println("Nu4="+Nu4);
  double k4=k(Tinf);
```

```
    double h4=Nu4*k4/D;
    double ro=ro(Tinf,P);
    double Cp=Cp(Tinf);
    System.out.println("Cp="+Cp+"J/kgK");
    System.out.println("h4="+h4);
    double y=Math.exp(-(Math.PI*D)*N*NT*h4/(ro*U*NT*ST*Cp));

    double To=Ts-(Ts-Tinf)*y;
    System.out.println("To="+To+"degree K"+(To-273.15)+"degree C");
    double dTlm=((Ts-To)-(Ts-Tinf))/Math.log((Ts-To)/(Ts-Tinf));
    System.out.println("dTlm="+dTlm);
    double Q=N*NT*h4*(Math.PI*D*L)*dTlm;
    System.out.println("Q="+Q);
}}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_crossflow_cylinder__bank_air
Pr=0.7082581146850705
Prs=0.7016206507857993
Re=13937.399561665197Umax=12.604026845637584C2=0.9499999999999998C=0.34365137652399674m=0.6
Nu4=93.26493642157384
Cp=1005.871632990873J/kgK
H4=145.79858759959745
To=302.87989174957degree K29.729891749570015degree C
dTlm=49.27984652584228
Q=20730.243872797408

> Terminated with exit code 0.
```

13. CONVECTIVE HEAT TRANSFER FOR INTERNAL FLOW

13.1 PRESSURE DROP IN PIPES

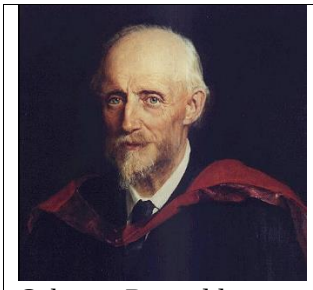
Pressure drop in pipes can be calculated by using Darcy-Weisbach formula. In order to use this formula, Darcy friction factor should be known. The best approximation to Darcy friction factor for turbulent flow is given by Colebrook-White equation. This equation can only be solved by numerical root finding methods which requires relatively costly computer calculations. There are several other approximation equations to Darcy friction factor with some relative error compared to Colebrook-White equation. In some of this equations the error percentage is so small that they can be directly used in place of Colebrook equation. This set of equations will be investigated in here.

Darcy-Weisbach pressure drop formula is given as:

$$\Delta P = f \frac{L}{D} \rho \frac{U^2}{2}$$

In this equation ΔP is the pressure drop, f is the friction factor, L is the length of pipe, U is fluid average velocity, D is pipe diameter. Friction factor f is depends on flow regime. For laminar flow ($Re_{critical}$=2300) Hagen-Poiseuille equation can be used.

$Re = \frac{\rho U D}{\mu}$ where Re is Reynolds' number, μ is dynamic viscosity, D pipe inlet diameter and ρ is the density.



Osborne Reynolds

II. "On the Dynamical Theory of Incompressible Viscous Fluids and the Determination of the Criterion." By OSBORNE REYNOLDS, F.R.S., &c. Received April 25, 1894.

(Abstract.)

The equations of motion of viscous fluid (obtained by grafting on certain terms to the abstract equations of the Eulerian form so as to adapt these equations to the case of fluids subject to stresses depending in some hypothetical manner on the rates of distortion, which equations Navier* seems to have first introduced in 1822, and which were much studied by Cauchy† and Poisson‡) were finally shown by St. Venant§ and Sir Gabriel Stokes,‖ in 1845, to involve no other assumption than that the stresses, other than that of pressure uniform in all directions, are linear functions of the rates of distortion with a co-efficient depending on the physical state of the fluid.

Original paper of Osborne Reynolds

$$f = \frac{64}{Re}$$

For turbulent region Colebrook-White(1937) [4] equation is existed.

$$\frac{1}{\sqrt{f}} = -2log_{10}\left[\frac{(\varepsilon/D)}{3.7} + \frac{2.51}{Re\sqrt{f}}\right]$$

In this equation ε is called surface roughness.

Experiments with Fluid Friction in Roughened Pipes

BY C. F. COLEBROOK AND C. M. WHITE

Imperial College, London

(*Communicated by G. I. Taylor, F.R.S.—Received* 15 *February* 1937)

Nikuradse (1933; Prandtl 1933), experimenting with pipes roughened internally by a uniform layer of sand, found that such pipes were indistinguishable from perfectly smooth ones, provided that the pressure gradient was less than that given by

$$\frac{\rho V_* k}{\mu} = 4,$$

where $V_* = \sqrt{(\tau_0/\rho)}$, τ_0 = shear stress at wall, ρ = density of fluid, μ = viscosity of fluid, k = diameter of roughness grains. With lesser flows neither the resistance nor the distribution of velocity was measurably influenced by the size of the roughness grains, and the observed resistance law was satisfactorily of the type required by the Kármán-Prandtl theory for smooth surfaces. This law is usually expressed in the following form:

$$\frac{1}{\sqrt{\lambda}} = 2\log\frac{\rho U d}{\mu}\sqrt{\lambda} - 0{\cdot}8, \qquad (1)$$

where λ is the coefficient of friction in the formula $h = \lambda\frac{lU^2}{2gd}$, and U = mean

Original paper of Cedric Massey White & Cyril Frank Colebrook



Cyril Frank Colebrook

Surface roughness of some common materials are given in the table below:

name	status	ε
Aluminium, drawn/pressed	new	0.0013 - 0.0015 mm
Aluminium, drawn/pressed	used	to 0.03 mm
Asbestos-cement	new, smooth	0.03 - 0.1 mm
Brass, drawn/pressed	new	0.0013 - 0.0014 mm
Brass, drawn/pressed	used	to 0.03 mm
Cast iron	average city severage	1.2 mm
Cast iron	incrusted	to 3.0 mm
Cast iron	new, bituminized	0.10 - 0.13 mm
Cast iron	new, with skin	0.2 - 0.6 mm
Cast iron	operating several years, cleaned	1.5 mm
Cast iron	slightly rusty	1.0 - 1.5 mm
Clay	new, clay tile	9.0 mm
Clay, Drainage-pipe	new, calcined	0.7 mm
Concrete	new, medium rough	1.0 - 2.0 mm
Concrete	new, rough	2.0 - 3.0 mm
Concrete	new, smooth	0.3 - 0.8 mm
Concrete	operating several years	0.2 - 0.3 mm
Concrete, Centrifugal-	new, smooth plastered	0.1 - 0.15 mm
Concrete, Centrifugal-	new, without plaster	0.2 - 0.8 mm
Concrete, Steel-	new, smooth	01. - 0.15 mm
Copper, drawn/pressed	new	0.0013 - 0.0015 mm
Copper, drawn/pressed	used	to 0.03 mm
Glass, drawn/pressed	new	0.0013 - 0.0015 mm
Glass, drawn/pressed	used	to 0.03 mm
Plastic, drawn/pressed	new	0.0013 - 0.0015 mm
Plastic, drawn/pressed	used	to 0.03 mm
Rubber	new, smoot	0.0016 mm
Steel	after long operation cleaned	0.15 - 0.20 mm
Steel	homogeneous corrosion pits	0.15 mm
Steel	intensely incrusted	2.0 - 4.0 mm
Steel	slightly rusty and incrusted	0.15 - 0.40 mm
Steel, longitudinal welded	new, bituminized	0.01 - 0.05 mm
Steel, longitudinal welded	new, galvanized	0.008 mm
Steel, longitudinal welded	new, rolling skin	0.04 - 0.1 mm
Steel, weldless	new, comm.size galvanized	0.10 - 0.16 mm
Steel, weldless	new, neatly galvanized	0.07 - 0.10 mm
Steel, weldless	new, pickled	0.03 - 0.04 mm
Steel, weldless	new, rolling skin	0.02 - 0.06 mm
Steel, weldless	new, unpickled	0.03 - 0.06 mm
Stoneware		0.25 mm
Wood	after long operating	0.1 mm
Wood	new	0.2 - 1.0 mm

As it is seen from the equation in order to solve this equation numerical root finding methods should be used. As an example Let us investigate a very simple Newton-Raphson Root Finding Method.

$$X = \frac{1}{\sqrt{f}}$$

$$f(X) = X + 2log_{10}\left[\frac{(\varepsilon/D)}{3.7} + \frac{2.51}{Re}X\right]$$

$$\frac{df(X)}{dX} = 1 + 2\frac{\frac{2.51}{Re}}{\left[\frac{(\varepsilon/D)}{3.7} + \frac{2.51}{Re}X\right]}$$

Newton Raphson formula:

$$X_{k+1} = X_k - \frac{f(X_k)}{\frac{df(X_k)}{dX}} \quad k=0,\ldots,n$$

In order to solve the equation some initial guess is also required.

PROBLEM: Water with temperature of T=26.85 C and P=300 kPa is flowing through a D=25x10^{-3} m commercial steel pipe. Find total pressure drop.

```
import javax.swing.*;
/* a[0] "P, pressure        ";
   a[1] "T, temperature     ";
   a[2] "v, specific volume ";
   a[3] "h, enthalpy        ";
   a[4] "u, internal energy ";
   a[5] "s, entropy         ";
   a[6] "x, quality         ";
   a[7] "Cp, specific heat at constant pressure ";
   a[8] "Cv, specific heat at constant volume ";
   a[9] ""+'\u03B1'+" isobaric thermal expansion coefficient ";
   a[10] ""+'\u03BA'+" isothermal compressibility" ;
   a[11] ""+'\u03B7'+" Dynamic viscosity" ;
   a[12] "k Thermal conductivity" ;
   a[13] ""+'\u03C3'+" Surface tension" ;
   a[14] "Prandtl number" ;
   a[15] ""+'\u03C1'+" density" ;
   a[16] " speed of sound" ;
*/
public class water
{ public steamIAPWS_IF97 st;
  public water()
  {st=new steamIAPWS_IF97();}
  public double[] property_TP(double T,double P)
  { // T degree K P kPa
    // 0 T 1 P 2 ro 3 h 4 s 5 Cp 6 k 7 mu 8 alpha 9 Pr
    double a[]=st.property("tp",T,P);
    double b[]=new double[12];
    b[0]=T;
    b[1]=P;
    //double multiply[]={1,1,1,1,1,1,1e-3,1e-6,1e-4,1};
    b[2]=a[15];     //ro kg/m^3
    b[3]=a[3];      //h kJ/kg
    b[4]=a[5];      //s kJ/kgK
    b[5]=a[7];      //Cp kJ/kgK
    b[6]=a[12];     //k W/mK
    b[7]=a[11];     //mu Pas
    b[8]=b[6]/(b[2]*b[5]*1e3); //alpha m^2/s
    b[9]=a[14];  //Pr
    b[10]=a[13];  //surface tension sigma
    b[11]=a[16];   //speed of sound
    return b;
  }

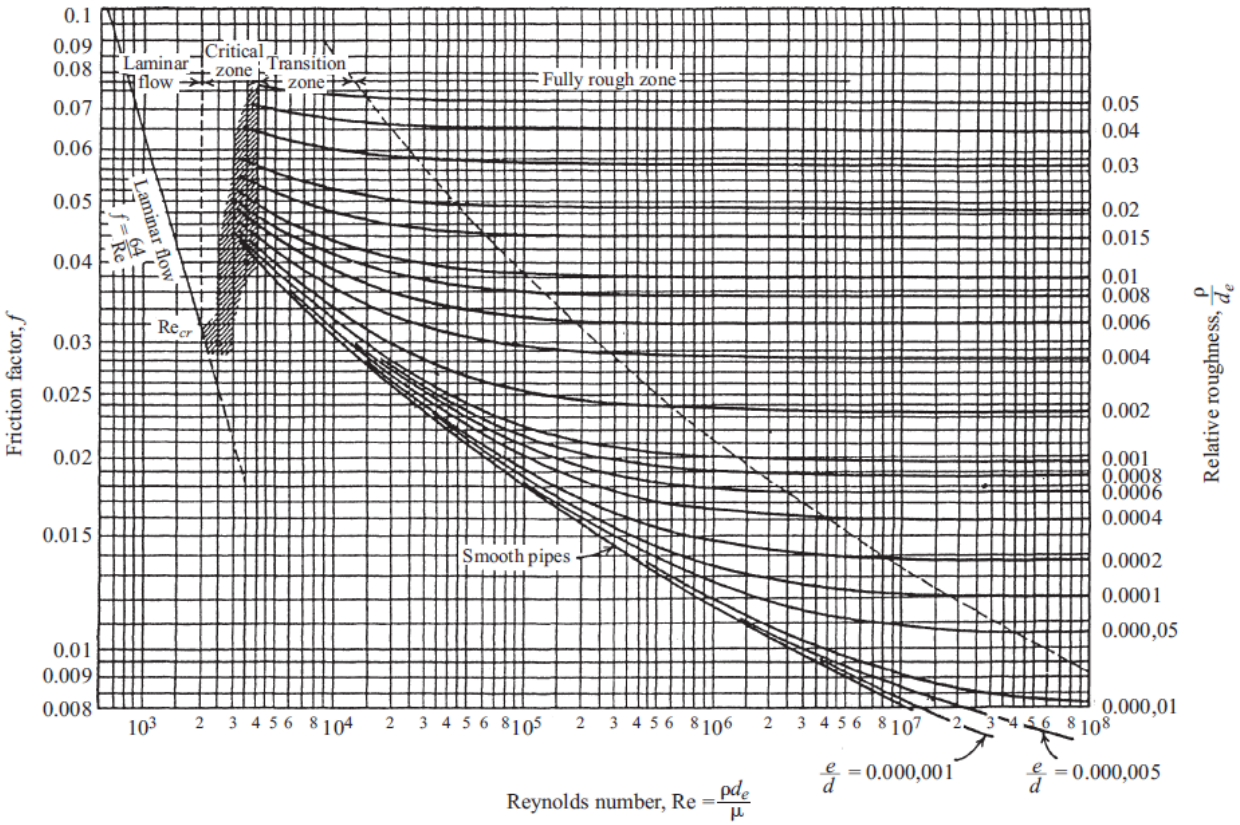
  public static void main(String arg[])
  {water w=new water();
   double T=Double.parseDouble(JOptionPane.showInputDialog("T degree K = "));
   double P=Double.parseDouble(JOptionPane.showInputDialog("P kg/m^3 = "));
   double a[]=w.property_TP(T,P);
   String s[]={"T degree K","P kPa",""+'\u03C1'+" kg/m^3","h kJ/kg","s kJ/kgK","Cp kJ/kgK","k W/mK",""+'\u03BC'+"
Pas",""+'\u03B1'+" m^2/s","Pr",""+'\u03C3'+" N/m","speed of sound m/s"};
   Text.print(a,s,"Thermophysical properties of steam");
  }  }
```

```
public class newtonA
{  public static double newton(if_x f,double x)
   {double eps=1.0e-10;
    while(Math.abs(f.func(x))>eps)
    {x=x-f.func(x)/f.dfunc(x);
    }
   return x;
   }
   public static double newton(if_x f,if_x df,double x)
   {double eps=1.0e-10;
    while(Math.abs(f.func(x))>eps)
    {x=x-f.func(x)/df.func(x);}
   return x;
```

```
}
        public static void main(String arg[])
        {    double T=26.85+273.15;  // degree K
             double P=300.0;         // kPa
             double U=3.0;           // velocity m/s
             double e=0.16e-3;       //surface rougness m
             double D=25e-3;         // diameter m
             double L=1; //m
             water w=new water();
             double a[]=w.property_TP(T,P);
             double mu=a[7];
             double ro=a[2];
             double Re=ro*U*D/mu;
             System.out.println("T="+T+" degree K");
             System.out.println("ro="+ro+" kg/m^3");
             System.out.println("P="+P+" kPa");
             System.out.println("U="+U+" m/s");
             System.out.println("D="+D+" m");
             System.out.println("e="+e+" m");
                    if_x f1=X->X+2.0*Math.log10((e/D)/3.7+2.51/Re*X);
             if_x df1=X->1.0+2.0*(2.51/Re)/((e/D)/3.7+2.51/Re*X);
             double X0=1.0/Math.sqrt(0.001);
             double X1=newton(f1,df1,X0);
             double f=1.0/(X1*X1);
             double A=Math.PI*D*D/4.0;
             double m=ro*A*U;  //kg/s
             double dP=f*L/D*ro*U*U/2.0;
             System.out.println("f = "+f);
             System.out.println("dP = "+dP);
        }}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" newtonA
T=300.0 degree K
ro=996.6470643119458 kg/m^3
P=300.0 kPa
U=3.0 m/s
D=0.025 m
e=1.6E-4 m
f = 0.03368023003583202
dP = 6042.114430301341
> Terminated with exit code 0.
```

In the chart given below laminar and turbulent flow friction coefficient shown graphically.



There are other approximation equations that root finding is not employed. Some of them are listed below:

Haaland equation (1983) [4]

$$\frac{1}{\sqrt{f}} = -1.8log_{10}\left[\left(\frac{(\varepsilon/D)}{3.7}\right)^{1.11} + \frac{6.9}{Re}\right]$$

Moody equation(1944) [9]

$$f = 5.5x10^{-3}\left[1 + \left(2x10^{4}(\varepsilon/D) + \frac{10^{6}}{Re}\right)^{1/3}\right]$$

Wood equation (1966) [18] Valid region: Re>10000 , $10^{-5}<(\varepsilon/D)<0.04$

$$a = 0.53(\varepsilon/D) + 0.094(\varepsilon/D)^{0.225}$$
$$b = 88(\varepsilon/D)^{0.44}$$
$$C = 1.62(\varepsilon/D)^{0.134}$$
$$f = a + bRe^{-C}$$

Churchill equation (1977) [3] Valid region: valid for all regions

$$A = -2log_{10}\left[\left(\frac{(\varepsilon/D)}{3.7}\right) + \left(\frac{7}{Re}\right)^{0.9}\right]^{16}$$
$$B = \left(\frac{37530}{Re}\right)^{16}$$
$$f = 8\left[\left(\frac{8}{Re}\right)^{12} + (A+B)^{-1.5}\right]^{1/12}$$

Chen equation (1979) [2] Valid region: all values

$$A = log_{10}\left[\left(\frac{(\varepsilon/D)^{1.1098}}{2.8257}\right) + \left(\frac{5.8506}{Re^{0.8981}}\right)\right]$$
$$\frac{1}{\sqrt{f}} = -2log_{10}\left[\left(\frac{(\varepsilon/D)}{3.765}\right) + \frac{5.0452A}{Re}\right]$$

Swamee-Jain equation (1976) [14] Valid region: 5000>Re>10^{7} , 0.00004<(ε / D)<0.05

$$A = log_{10}\left[\frac{(\varepsilon/D)}{3.7} + \frac{5.74}{Re^{0.9}}\right]$$
$$f = \frac{0.25}{A^2}$$

Zigrang - Sylvester equation (1982) [20] Valid region: all values

$$A = log_{10}\left[\frac{(\varepsilon/D)}{3.7} + \frac{13}{Re}\right]$$

$$B = log_{10}\left[\frac{(\varepsilon/D)}{3.7} + \frac{5.02A}{Re}\right]$$

$$\frac{1}{\sqrt{f}} = -2log_{10}\left[\left(\frac{(\varepsilon/D)}{3.7}\right) - \frac{5.02B}{Re}\right]$$

Serghides equation (1984) [22] Valid region: all values

$$A = -2log_{10}\left[\frac{(\varepsilon/D)}{3.7} + \frac{12}{Re}\right]$$

$$B = -2log_{10}\left[\frac{(\varepsilon/D)}{3.7} + \frac{2.51A}{Re}\right]$$

$$C = -2log_{10}\left[\frac{(\varepsilon/D)}{3.7} + \frac{2.51B}{Re}\right]$$

$$\frac{1}{\sqrt{f}} = A - \frac{(B-A)^2}{C-2B+A}$$

Goudar- Sonnad equation (2008)[21] Valid region: all values

$$a = \frac{2}{\ln(10)}$$

$$b = \frac{(\varepsilon/D)}{3.7}$$

$$d = \frac{ln(10)}{5.02}Re$$

$$s = bd + \ln\left(\frac{d}{q}\right);$$

$$q = s^{\left(\frac{s}{s+1}\right)}$$

$$g = bd + \ln\left(\frac{d}{q}\right)$$

$$z = \frac{q}{g}$$

$$\delta_{LA} = \frac{g}{g+1}z$$

$$\delta_{CFA} = \delta_{LA}\left(1 + \frac{z/2}{(g+1)^2 + \left(\frac{z}{3}\right)(2g-1)}\right)$$

$$\frac{1}{\sqrt{f}} = a\left[ln\left(\frac{d}{q}\right) + \delta_{CFA}\right]$$

Romeo Equation (2002) [11] Valid region: all values

$$A = log_{10}\left[\left(\frac{(\varepsilon/D)}{7.7918}\right)^{0.9924} + \left(\frac{5.3326}{208.815+Re}\right)^{0.9345}\right]$$

$$B = log_{10}\left[\left(\frac{(\varepsilon/D)}{3.3827}\right) - \frac{4.567A}{Re}\right]$$

$$\frac{1}{\sqrt{f}} = -2log_{10}\left[\left(\frac{(\varepsilon/D)}{3.7065}\right) - \frac{5.0272B}{Re}\right]$$

Out of these equations Goudar-Sonnad & Serghides equations are given such an accurate results that they can be directly replace Colebroke-White equation to calculate the friction factor. Now let us try out different friction equations with the same problem.

PROBLEM: Water with temperature of T=26.85 C and P=300 kPa is flowing through a D=25x10^{-3} commercial steel pipe. Find total pressure drop. **Try all different friction factor equations**

```
 import java.io.*;
public class colebrook2
{

 public static double f_Haaland(double Re,double eod)
 {
 //Haaland equation
 double f1=-1.8*Math.log10(Math.pow((eod/3.7),1.11)+6.9/ Re);
 f1=1.0/(f1*f1);
 return f1;
 }
```

```
public static double f_Moody(double Re,double eod)
{
// Moody equation
// 4000<Re<107 and e/D <0.01
double f1=5.5e-3*(1+Math.pow((2e4*eod+1e6/Re),(1.0/3.0)));
return f1;
}

public static double f_Wood(double Re,double eod)
{
// Wood equation
// Re>10000 and any e/D
double a=0.52*eod+0.094*Math.pow(eod,0.225);
double b=88.0*Math.pow(eod,0.44);
double C=1.62*Math.pow(eod,0.134);
double f1=a+b*Math.pow(Re,-C);
return f1;
}

public static double f_Churchill(double Re,double eod)
{
// Churchill equation
// for all values of Re and e/D
double A=Math.pow((-2.457*Math.log(Math.pow((7.0/Re),0.9)+0.27*eod)),16);
double B=Math.pow((37530.0/Re),16);
double f1=8.0*Math.pow((Math.pow((8.0/Re),12)+1.0/Math.pow((A+B),1.5)),(1.0/12.0));
return f1;
}

public static double f_Chen(double Re,double eod)
{
// Chen equation
// for all values of Re and e/D
double A = Math.log10(Math.pow(eod,1.1098)/2.8257 + (5.8506/Math.pow(Re,0.8981)));
double f1=-2.0*Math.log10(eod/3.7065-5.0452*A/Re);
f1=1.0/(f1*f1);
return f1;
}

public static double f_Swamee(double Re,double eod)
{
//Swamee-Jain equation
double A=Math.log10(eod/3.7+5.74/Math.pow(Re,0.9));
double f1=0.25/(A*A);
return f1;
}

public static double f_Zigrand(double Re,double eod)
{
//Zigrang and Sylvester Equation
// for 4000<Re<108  and 0.00004<e/D<0.05
double A=Math.log10(eod/3.7-5.02/Re*Math.log10(eod/3.7+13/Re));
double f1=-2.0*Math.log10((eod)/3.7-5.02*A/Re);
f1=1.0/(f1*f1);
return f1;
}

public static double f_Romeo(double Re,double eod)
{
// Romeo - Royo - Monzon Equation (2002)

double A=Math.log10(eod/3.827-4.567/Re*Math.log10(Math.pow((eod/7.7918),0.9924)+Math.pow(5.3326/(208.815+Re),0.9345)));
double f1=-2.0*Math.log10((eod)/3.7065-5.0272*A/Re);
f1=1.0/(f1*f1);
return f1;
}

public static double f_Serghides(double Re,double eod)
{
//Serghides equation
// for Re>2100 and any e/D
double A1=-2.0*Math.log10(eod/3.7+12.0/Re);
double B1=-2.0*Math.log10(eod/3.7+2.51*A1/Re);
double C1=-2.0*Math.log10(eod/3.7+2.51*B1/Re);
double f1=A1-((B1-A1)*(B1-A1))/(C1-2.0*B1+A1);
f1=1.0/(f1*f1);
```

```
 return f1;
 }

 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
public static void main(String arg[]) throws IOException
{       double T=26.85+273.15;  // degree K
                double P=300.0;         // kPa
                double U=3.0;           // velocity m/s
                double e=0.16e-3;       //surface rougness m
                double D=25e-3;         // diameter m
                double eod=e/D;
                double L=1; //m
                water w=new water();
                double a[]=w.property_TP(T,P);
                double mu=a[7];
                double ro=a[2];
                double Re=ro*U*D/mu;
                System.out.println("T="+T+" degree K");
                System.out.println("ro="+ro+" kg/m^3");
                System.out.println("P="+P+" kPa");
                System.out.println("U="+U+" m/s");
                System.out.println("D="+D+" m");
                System.out.println("e="+e+" m");
                double f1=f_Haaland(Re,eod);
                double f2=f_Moody(Re,eod);
                double f3=f_Wood(Re,eod);
                double f4=f_Churchill(Re,eod);
                double f5=f_Chen(Re,eod);
                double f6=f_Swamee(Re,eod);
                double f7=f_Zigrand(Re,eod);
                double f8=f_Romeo(Re,eod);
                double f9=f_Serghides(Re,eod);
                double f10=f_Goudar(Re,eod);
                System.out.println("Friction factors:");
                System.out.println("f Haaland = "+f1);
                System.out.println("f Moody = "+f2);
                System.out.println("f Wood = "+f3);
                System.out.println("f Churchill = "+f4);
                System.out.println("f Chen = "+f5);
                System.out.println("f Swamee = "+f6);
                System.out.println("f Zigrand = "+f7);
                System.out.println("f Romeo = "+f8);
                System.out.println("f Serghides = "+f9);
                System.out.println("f Goudar = "+f9);
                double A=Math.PI*D*D/4.0;
                double m=ro*A*U;  //kg/s
                double dP=f9*L/D*ro*U*U/2.0;
                System.out.println("dP = "+dP);

}

//create_data();

}
```

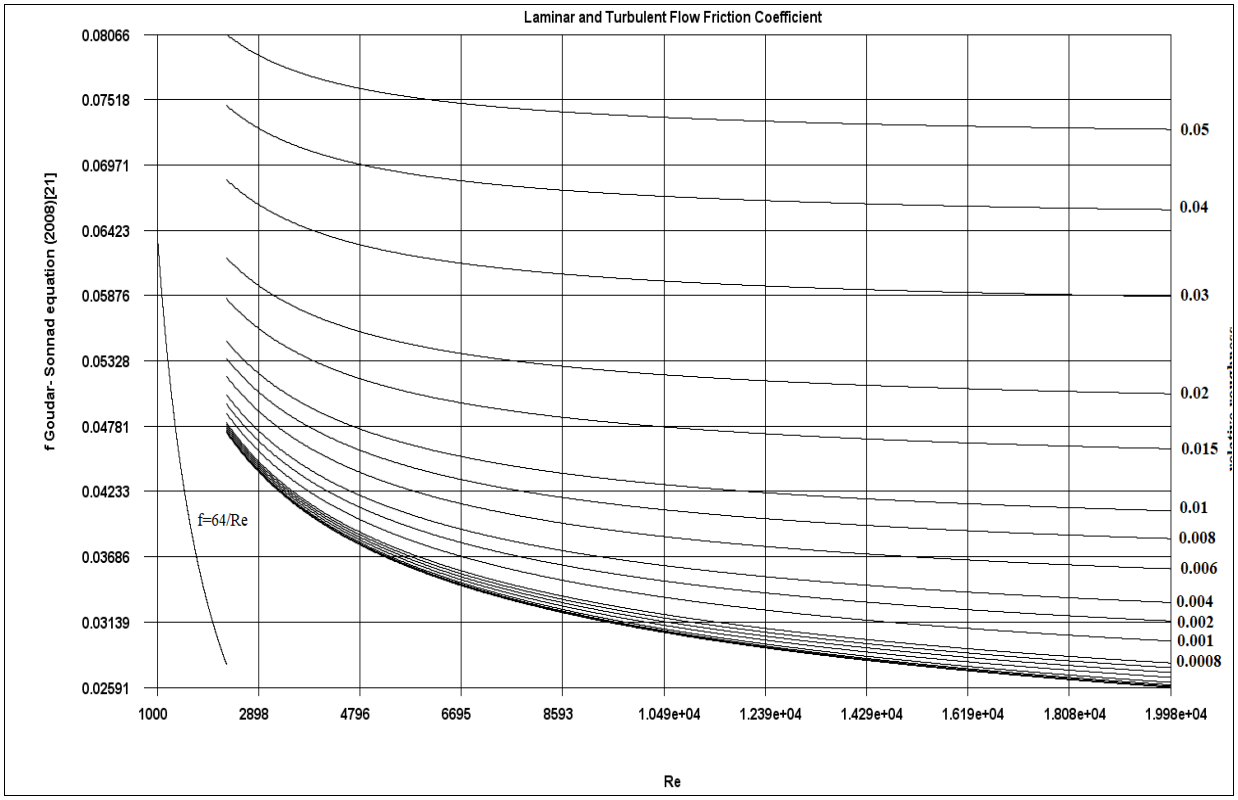
```
 ---------- Capture Output ----------
> "D:\co\java\bin\java.exe" colebrook2
T=300.0 degree K
ro=996.6470643119458 kg/m^3
P=300.0 kPa
```

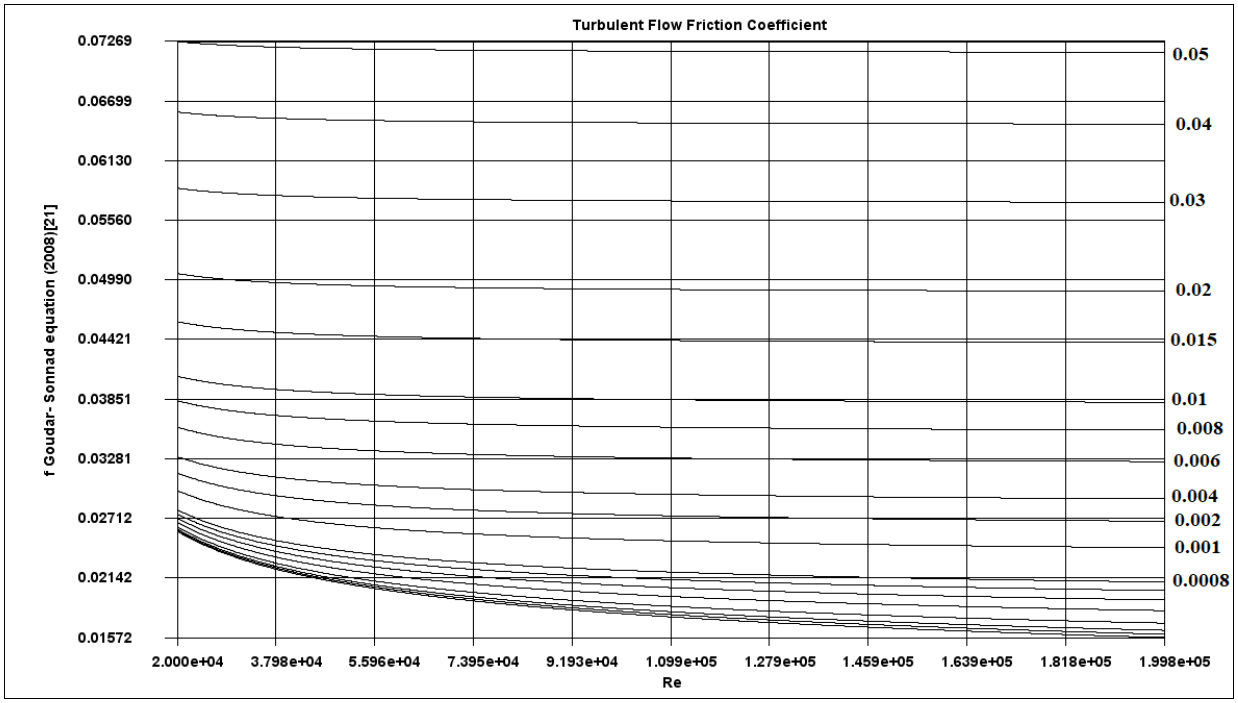
```
U=3.0 m/s
D=0.025 m
e=1.6E-4 m
Friction factors:
f Haaland = 0.03366809407987718
f Moody = 0.034019312456163006
f Wood = 0.03430754588971051
f Churchill = 0.033941360161729314
f Chen = 0.03369873685763093
f Swamee = 0.03395380279310423
f Zigrand = 0.03368022234776756
f Romeo = 0.03366856614181092
f Serghides = 0.03368023003463402
f Goudar = 0.03368023003463402
dP = 6042.114430086424

> Terminated with exit code 0.
```

Goudar-Sonnad plot

```
import java.io.*;
class f4 extends f_x
{  public double func (double Re)
   {return 64.0/Re;}
}
class f3 extends fi_x
{ public double
eod[]={0.00001,0.00005,0.0001,0.0002,0.0004,0.0006,0.0008,0.001,0.002,0.003,0.004,0.006,0.008,0.01,0.015,0.02,0.03,0.04,0.05};
  public int n=eod.length;
double[] func(double Re)
{
double f[]=new double[n];
//f[0]=64.0/Re;
for(int i=0;i<n;i++)
{f[i]=colebrook2.f_Goudar(Re,eod[i]);}
return f;
}}
class PlotT3c
{
public static void main (String args[]) throws IOException
{
f3 ff=new f3();
f4 ff1=new f4();
Plot pp=new Plot(ff,2300.0,20000,1000,ff.n);
pp.addFunction(ff1,1000.0,2300,1000);
//pp.setYlogScaleOn();
pp.setXgrid(5);
pp.setYgrid(20);
pp.setPlabel("Laminar and Turbulent Flow Friction Coefficient");
pp.setXlabel("Re");
pp.setYlabel("f Goudar- Sonnad equation (2008)[21]");
pp.plot();}}
```





An excel version of the f friction factor calculation program pipe_HT.xls

Newton-Raphson root finding Solution of Colebrook-White equation

Re	6000
ε/D eod	0.001
Sergides equation	
A	5.287844876
B	5.210273255
C	5.221702842
X	5.220235048

f sergides	**0.036696098**	

Colebrook-White root finding by Newton-Raphson method

X	**f(x)=X+2log10(eod/3.7+2.51/Re*X)**	**fx/dx=1+2*(2.51/Re)/(eod/3.7+2.51/Re*X)**
5.22023504771383000	8.4445E-06	1.340930432
5.22022875022313000	1.21457E-06	1.340930798
5.22022784445472000	1.74692E-07	1.34093085
5.22022771417782000	2.5126E-08	1.340930858
5.22022769544005000	3.61389E-09	1.340930859
5.22022769274500000	5.19787E-10	1.340930859
5.22022769235736000	7.47606E-11	1.340930859
5.22022769230161000	1.07532E-11	1.340930859
5.22022769229359000	1.54632E-12	1.340930859
5.22022769229244000	2.22933E-13	1.340930859
5.22022769229227000	3.28626E-14	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859
5.22022769229225000	0	1.340930859

f colebrook **0.036696201**

Goudar-Sonnad Equation equation

a	0.868588964
b	0.00027027
d	2752.093737
s	8.663926378
q	6.929186844
g	6.72818391
z	0.02943721
delta_LA	0.025628138
delta_CFA	0.025634441
X	5.220227692
f Goudar-Sonnad	**0.036696201**

13.2 HEAT TRANSFER IN CIRCULAR PIPES

Entry Region for Laminar flow(Re<2300):

$\frac{x_{entry}}{D} \cong 0.05Re$

Entry Region for Turbulent flow(Re>2300):

$10 \leq \frac{x_{entry}}{D} \leq 60$

mean temperature difference (derived by assuming C_p is constant) is

$\Delta T_{lm} = \frac{(T_s - T_i) - (T_s - T_0)}{ln\frac{(T_s - T_i)}{(T_s - T_0)}}$

temperature going out of the bank is T_0 can be estimated from

$\frac{(T_s - T_0)}{(T_s - T_i)} = exp\left(-\frac{PLh}{mC_p}\right)$ once ΔT_{lm} known the heat transfer rate can be calculated

$Q = m * dH = mC_p \Delta T_{lm}$ where dH is enthalpy difference $dH \cong C_p(T) * \Delta T_{lm}$

$m = \rho AU$

Heat transfer equations Fully developed Laminar flow (Re<2300)

Nu=3.66 T_s=const.

Nu=4.66 q_s=const

Heat transfer equations Entry region Laminar flow T_s=const (Re<2300)

Hausen[26] equation:

$Nu = 3.66 + \frac{0.0668\left(\frac{D}{L}\right)RePr}{1+0.04\left[\left(\frac{D}{L}\right)RePr\right]^{2/3}}$

Sieder & Tate Equation[25]

$Nu = 3.66 + 1.86\left[\left(\frac{D}{L}\right)RePr\right]^{1/3}\left(\frac{\mu}{\mu_s}\right)^{0.34}$ $0.48 \leq Pr \leq 16700$ $0.0044 < \left(\frac{\mu}{\mu_s}\right) < 9.75$

Heat transfer equations Fully developed turbulent flow T_s=const (Re>2300)

Dittus-Boelter equation[24]

$Nu = 0.023Re^{4/5}Pr^n$ where n=0.4 for heating and n=0.3 for cooling $0.7 \leq Pr \leq 160$

$Re \geq 1000$ $(\frac{L}{D}) \geq 10$

Sieder-Tate equation[25]

$Nu = 0.023Re^{0.8}Pr^{1/3}\left(\frac{\mu}{\mu_s}\right)^{0.14}$ $0.6 \leq Pr \leq 16700$ $Re \geq 1000$ $(\frac{L}{D}) \geq 60$

Petukhov, Kirillov, and Popov[32] equation

$Nu = \frac{\left(\frac{f}{8}\right)RePr}{1.07+12.7\left(\frac{f}{8}\right)^{.5}\left(Pr^{\frac{2}{3}}-1\right)}$ $1 \leq Pr \leq 2000$ $10^4 \leq Re \leq 510^5$

A modified form of the equation

$Nu = \frac{\left(\frac{f}{8}\right)RePr}{1.07+12.7\left(\frac{f}{8}\right)^{.5}\left(Pr^{\frac{2}{3}}-1\right)}\left(\frac{\mu}{\mu_w}\right)^n$ n=0.11 for heating and n=0.25 for cooling

$0.5 \leq Pr \leq 140$ $10^4 \leq Re \leq 510^5$ $1 \leq \frac{\mu}{\mu_w} \leq 40$

Gnielinski[33] equation

$Nu = \frac{\left(\frac{f}{8}\right)(Re-1000)Pr}{1.07+12.7\left(\frac{f}{8}\right)^{.5}\left(Pr^{\frac{2}{3}}-1\right)}$ $0.5 \leq Pr \leq 2000$ $2300 \leq Re \leq 510^6$

Gnielinski also suggested that

$Nu = 0.0214(Re^{0.8} - 100)Pr^{0.4}$ $0.5 \leq Pr \leq 1.5$ $10^4 \leq Re \leq 510^6$

$Nu = 0.012(Re^{0.87} - 280)Pr^{0.4}$ $0.5 \leq Pr \leq 500$ $3x10^3 \leq Re \leq 10^6$

Sleicher and Rouse[79] equation

$Nu = 5 + 0.015Re^mPr^n$ $m = 0.88 - \frac{0.24}{Pr+4}$ $n = \frac{1}{3} + 0.5\exp(-0.6\text{Pr})$ $1 \leq Pr \leq 1000$

$10^4 \leq Re \leq 10^6$

Heat transfer equations **Fully developed transitional/intermittent region** T_s=const

Abraham-Sparrow-Tong[34] equation

$$Nu = 2.2407\left(\frac{Re}{1000}\right)^4 - 29.499\left(\frac{Re}{1000}\right)^3 + 142.32\left(\frac{Re}{1000}\right)^2 - 292.51\left(\frac{Re}{1000}\right) + 219.88$$

$$2300 \leq Re \leq 3100$$

Abraham recommended Gnilenski equation to be used above Re>3100

Hausen[78] equation for both liquid and gases T_s=const

$$Nu = 0.116(Re^{2/3} - 125)Pr^{1/3}\left(\frac{\mu}{\mu_w}\right)^{0.14}\left[1 + \left(\frac{d}{L}\right)^{2/3}\right] \quad 2300 \leq Re \leq 10000$$

PROBLEM: Water with temperature of T_i =26.85 ℃ and P=300 kPa is flowing through a D=25x10^{-3} commercial steel pipe. Total pipe length L=2 m. Pipe surface temperature are kept constant at T=126.85 ℃. Find the total heat transfer and pipe exit temperature also find total pressure drop. Velocity is U=3 m/s

In the first approach to the problem an initial guess is assigned to the outside temperature and outside temperature is calculated at the end, if it is not resonable iterative process is applied. Abraham-Sparrow-Tong & Gnilenski equations are used .

Algorithm:

Guess T_{output} (can not be bigger than surface temperature)

Take average temperature $T = \frac{T_{input}+T_{output}}{2}$

Evaluate thermophysical properties μ, C_p , k, ρ , Pr, at temperature T

Calculate Re number and decide the flow regime (laminar, Fully developed transitional/intermittent region or turbulent)

Calculate Nusselt number according to flow regime, calculate convective heat transfer coefficient

Calculate heat transfer from

$$A_{surface} = \pi DL$$

$$Q = hA_{surface}(T - T_{surface})$$

Calculate $T_{output} = T_{input} + Q/(mC_p)$

If T_{output} value is very different compare to initial guess repeat the process untill they are relatively close to each other

```
import java.io.*;
public class HT_inside_pipe
{
 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 public static double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
   double Nu=0;
   double x=Re/1000;
   double f=0;
   if(Re<2300) Nu=3.66;
   else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
   else {f=f_Goudar(Re,eod);
   Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
```

```
  }
 return Nu;
  }
public static void main(String arg[]) throws IOException
{       double Ti=26.85+273.15;  // degree K
        double Ts=99.85+273.15; //degree K;
        double To=70+273.15;        //guess for outlet temperature To
        double T=(Ti+To)/2.0;
                double P=300.0;         // kPa
                double U=3.0;           // velocity m/s
                double e=0.16e-3;       //surface rougness m
                double D=25e-3;         // diameter m
                double eod=e/D;
                double L=2; //m
                water w=new water();
                double a[]=w.property_TP(T,P);
                double ro=a[2];
                double Cp=a[5];
                double k=a[6];
                double mu=a[7];
                double Pr=a[9];
                double Re=ro*U*D/mu;
                System.out.println("T="+T+" degree K");
                System.out.println("P="+P+" kPa");
                System.out.println("U="+U+" m/s");
                System.out.println("D="+D+" m");
                System.out.println("e="+e+" m");
                double f=f_Goudar(Re,eod);
                System.out.println("f Goudar = "+f);
                double Nu=Nu(Re,eod,Pr);
        double h=Nu*k/D;
        double A1=Math.PI*D*L;
        double Q1=h*A1*(Ts-T);
        System.out.print("h = "+h+"W/m^2K ");
        System.out.println("To initial guess = "+To+" degree K  = "+(To-273)+" degree C");
        To=Ti+Q1/(Cp*1e3);
        System.out.println(" To = "+To+" degree K  = "+(To-273)+" degree C \nQ="+Q1+" W");
                double A=Math.PI*D*D/4.0;
                double m=ro*A*U;  //kg/s
                System.out.println("m = "+m+"kg/s");
                double dP=f*L/D*ro*U*U/2.0;
                System.out.println("dP = "+dP+" Pa");
}
//create_data();
}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_inside_pipe
T=321.575 degree K
P=300.0 kPa
U=3.0 m/s
D=0.025 m
e=1.6E-4 m
f Goudar = 0.03338070406938855
Nu = 912.2302212643231h = 23310.932651973955W/m^2K To initial guess = 343.15 degree K  = 70.14999999999998 degree C
 To = 345.0613334663793 degree K  = 72.06133346637932 degree C
Q=188301.52057162303 W
m = 1.4561834076054112kg/s
dP = 11882.913018836436 Pa

> Terminated with exit code 0.
```

In the second approach problem is solved in finite difference steps. So that properties are taken as local water temperatures in each step. In this method addition to heat transfer, you will also obtain how temperature is changing in each step of the process. Algorithm of the method is given below:

Take temperature as $T_i = T_{input}$ i=0

Divide total length to N section, calculate step length $\Delta x = \frac{L}{N}$

Take heat transfer Q=0

Assume given velocity U is in at input condition T_{input} calculate mass flow rate $m = \rho UA$

x=0

Loop for N step Δx length increase in each step i=0 to N

$x = i\Delta x$

Evaluate thermophysical properties $\mu(T_i)$, $C_p(T_i)$, k(T_i), $\rho(T_i)$, $Pr(T_i)$, at temperature T_i

Calculate the step velocity $U_i = m/(\rho(T_i)\text{A})$

Calculate Re number and decide the flow regime (laminar, Fully developed transitional/intermittent region or turbulent)

Calculate Nusselt number Nu_i according to flow regime,

calculate convective heat transfer coefficient h_i

Calculate heat transfer from

$$\Delta A_{surface} = \pi D \Delta x$$

$$\Delta Q = h_i \Delta A_{surface}(T_i - T_{surface})$$

$$Q = Q + \Delta Q$$

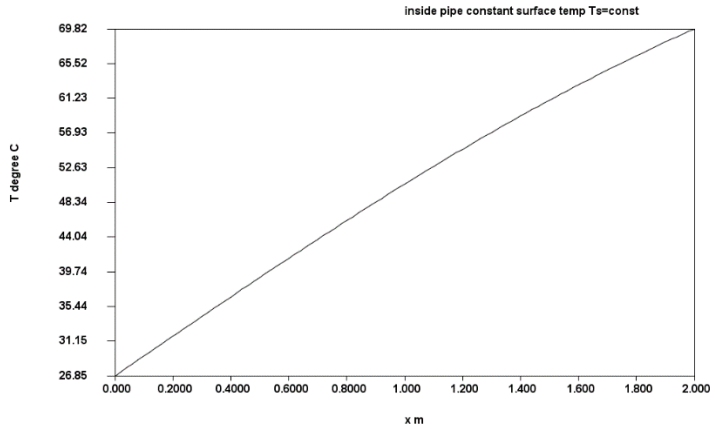
Calculate $T_{i+1} = T_i + \Delta Q/(mC_p(T_i))$

```
 import java.io.*;
public class HT_inside_pipe_FD
{
 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 public static double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
  double Nu=0;
  double x=Re/1000;
  double f=0;
  if(Re<2300) Nu=3.66;
  else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
  else {f=f_Goudar(Re,eod);
  Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
  }
 return Nu;
  }
public static void main(String arg[]) throws IOException
{       double Ti=26.85+273.15;  // degree K
        double Ts=99.85+273.15; //degree K;
        double T=Ti;
                double P=300.0;         // kPa
                double U=3.0;           // velocity m/s
                double e=0.16e-3;       //surface rougness m
                double D=25e-3;         // diameter m
                double eod=e/D;
                double L=2; //m
                int N=100;
                double dx=L/N;
                double x=0;
                double TT[]=new double[N+1];
                double xx[]=new double[N+1];
                water w=new water();
                double ro,Cp,k,mu,Pr,Re,f,Nu,h,Q=0,Q1=0;
                double a[]=w.property_TP(Ti,P);
                ro=a[2];
                double A1=Math.PI*D*dx;
                double A=Math.PI*D*D/4.0;
                double m=ro*U*A;
                for(int i=0;i<=N;i++)
```

```
                { a=w.property_TP(T,P);
                  TT[i]=T-273.15;
                  xx[i]=i*dx;
          ro=a[2];
                  Cp=a[5];
                  k=a[6];
                  mu=a[7];
                  Pr=a[9];
                  U=m/(ro*A);
                  Re=ro*U*D/mu;
          f=f_Goudar(Re,eod);
          Nu=Nu(Re,eod,Pr);
          h=Nu*k/D;
          Q1=h*A1*(Ts-T);
          Q+=Q1;
          T=T+Q1/(Cp*1e3);
          System.out.println("i="+i+" x = "+xx[i]+" U = "+U+" m/s Nu = "+Nu+" h="+h+" T = "+(T-273.15)+" degree C");
        }
        Plot p=new Plot(xx,TT);
        p.setPlabel("inside pipe constant surface temp Ts=const ");
        p.setXlabel("x m");
        p.setYlabel("T degree C");
        p.plot();
        System.out.println("Q="+Q+" Watt");
}
//create_data();
}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_inside_pipe_FD
i=0 x = 0.0 U = 3.0000000000000004 m/s Nu = 729.5552718920588 h=17814.104921096932 T = 27.338623782042816 degree C
i=1 x = 0.02 U = 3.000405839835486 m/s Nu = 733.9790008164412 h=17943.724850960734 T = 27.827530291057656 degree C
i=2 x = 0.04 U = 3.000818566080135 m/s Nu = 738.3994963407229 h=18073.403863451076 T = 28.316670653734775 degree C
i=3 x = 0.06 U = 3.0012380967581764 m/s Nu = 742.8163075807082 h=18203.126697892323 T = 28.805995858315782 degree C
i=4 x = 0.08 U = 3.0016643481086183 m/s Nu = 747.2289849742425 h=18332.878075367964 T = 29.295456775595596 degree C
i=5 x = 0.1 U = 3.002097234616453 m/s Nu = 751.6370804656453 h=18462.642706217048 T = 29.78500418007161 degree C
i=6 x = 0.12 U = 3.0025366690453423 m/s Nu = 756.0401476897645 h=18592.405297559966 T = 30.274588771219783 degree C
i=7 x = 0.14 U = 3.0029825624717215 m/s Nu = 760.437742155354 h=18722.150560842634 T = 30.76416119487618 degree C
i=8 x = 0.16 U = 3.003434824320324 m/s Nu = 764.8294214276168 h=18851.86321939075 T = 31.25367206470372 degree C
i=9 x = 0.18 U = 3.0038933624010746 m/s Nu = 769.2147453098091 h=18981.52801596785 T = 31.743071983721734 degree C
i=10 x = 0.2 U = 3.004358082947343 m/s Nu = 773.5932760236107 h=19111.129720325353 T = 32.23231156587781 degree C
i=11 x = 0.22 U = 3.0048288906555087 m/s Nu = 777.9645783881911 h=19240.65313673889 T = 32.721341457639255 degree C
i=12 x = 0.24 U = 3.005305688725807 m/s Nu = 782.3282199977376 h=19370.083111520937 T = 33.21011235958349 degree C
i=13 x = 0.26 U = 3.005788378904423 m/s Nu = 786.6837713972362 h=19499.404540500316 T = 33.698575047964425 degree C
i=14 x = 0.28 U = 3.006276861526776 m/s Nu = 791.030806256413 h=19628.60237646217 T = 34.186680396233896 degree C
i=15 x = 0.3 U = 3.006771035561984 m/s Nu = 795.3689015415707 h=19757.66163653756 T = 34.67437939649557 degree C
i=16 x = 0.32 U = 3.0072707986584226 m/s Nu = 799.6976376851965 h=19886.56740953561 T = 35.16162318087038 degree C
i=17 x = 0.34 U = 3.007776047190365 m/s Nu = 804.016598753108 h=20015.304863208054 T = 35.6483630427511 degree C
i=18 x = 0.36 U = 3.0082866763056395 m/s Nu = 808.3253726090859 h=20143.85925144108 T = 36.13455045792523 degree C
i=19 x = 0.38 U = 3.00880257997425 m/s Nu = 812.623551076634 h=20272.215921361425 T = 36.620137105544245 degree C
i=20 x = 0.4 U = 3.0093236510379313 m/s Nu = 816.9107300978746 h=20400.360320352578 T = 37.10507488891892 degree C
i=21 x = 0.42 U = 3.009849781260547 m/s Nu = 821.1865098893314 h=20528.27800297171 T = 37.58931595611938 degree C
i=22 x = 0.44 U = 3.010380861379313 m/s Nu = 825.4504950944323 h=20655.95463775865 T = 38.072812720359764 degree C
i=23 x = 0.46 U = 3.0109167811567645 m/s Nu = 829.7022949326335 h=20783.376013930974 T = 38.555517880147306 degree C
i=24 x = 0.48 U = 3.011457429433422 m/s Nu = 833.9415233449243 h=20910.528047955584 T = 39.03738443917615 degree C
i=25 x = 0.5 U = 3.0120026941811 m/s Nu = 838.1677991357006 h=21037.396789992617 T = 39.51836572594675 degree C
i=26 x = 0.52 U = 3.0125524625567714 m/s Nu = 842.380746110703 h=21163.968430200974 T = 39.99841541309195 degree C
i=27 x = 0.54 U = 3.0131066209569863 m/s Nu = 846.5799932110293 h=21290.229304901917 T = 40.47748753639155 degree C
i=28 x = 0.56 U = 3.0136650550727095 m/s Nu = 850.7651746430414 h=21416.165902593384 T = 40.95553651345756 degree C
i=29 x = 0.58 U = 3.014227649944574 m/s Nu = 854.9359300039702 h=21541.764869806673 T = 41.43251716207317 degree C
i=30 x = 0.6 U = 3.0147942900184725 m/s Nu = 859.0919044032786 h=21667.013016803965 T = 41.908384718168236 degree C
i=31 x = 0.62 U = 3.015364859201399 m/s Nu = 863.232748579502 h=21791.897323107143 T = 42.383094853416196 degree C
i=32 x = 0.64 U = 3.0159392409175223 m/s Nu = 867.3581190125384 h=21916.404942853955 T = 42.85660369243624 degree C
i=33 x = 0.66 U = 3.0165173181643947 m/s Nu = 871.4676780313521 h=22040.523209978295 T = 43.32886782958684 degree C
i=34 x = 0.68 U = 3.017098973569252 m/s Nu = 875.561093916886 h=22164.239643206947 T = 43.79984434533594 degree C
i=35 x = 0.7000000000000001 U = 3.0176840894453325 m/s Nu = 879.6380410002159 h=22287.541950871368 T =
44.269490822194996 degree C
i=36 x = 0.72 U = 3.0182725478481696 m/s Nu = 883.6981997558127 h=22410.418035528626 T = 44.737765360204264 degree C
i=37 x = 0.74 U = 3.018864230631771 m/s Nu = 887.7412568898759 h=22532.855998389186 T = 45.20462659195755 degree C
i=38 x = 0.76 U = 3.019459019504656 m/s Nu = 891.766905423704 h=22654.84414354808 T = 45.67003369715519 degree C
i=39 x = 0.78 U = 3.020056796085668 m/s Nu = 895.7748447719999 h=22776.370982015764 T = 46.1339464166752 degree C
i=40 x = 0.8 U = 3.020657441959517 m/s Nu = 899.7647808161686 h=22897.425235548133 T = 46.596325066152986 degree C
i=41 x = 0.8200000000000001 U = 3.0212608387319784 m/s Nu = 903.7364259724443 h=23017.9958402706 T =
47.05713054906113 degree C
i=42 x = 0.84 U = 3.021866868084717 m/s Nu = 907.689499255 h=23138.071950098445 T = 47.516324369280824 degree C
i=43 x = 0.86 U = 3.022475411829671 m/s Nu = 911.623726333876 h=23257.642939948237 T = 47.97386864315854 degree C
```

i=44 x = 0.88 U = 3.0230863519629056 m/s Nu = 915.5388395878006 h=23376.698408742006 T = 48.429726111040736 degree C
i=45 x = 0.9 U = 3.023699570717964 m/s Nu = 919.4345781518875 h=23495.22818220162 T = 48.88386014828211 degree C
i=46 x = 0.92 U = 3.0243149506185816 m/s Nu = 923.3106879602371 h=23613.222315434698 T = 49.33623477572161 degree C
i=47 x = 0.9400000000000001 U = 3.0249323745307555 m/s Nu = 927.1669217834185 h=23730.67109531049 T = 49.786814669622856 degree C
i=48 x = 0.96 U = 3.0255517257141338 m/s Nu = 931.0030392609322 h=23847.56504262756 T = 50.23556517107579 degree C
i=49 x = 0.98 U = 3.0261728878726357 m/s Nu = 934.8188069286402 h=23963.89491407358 T = 50.68245229485689 degree C
i=50 x = 1.0 U = 3.0267957452043 m/s Nu = 938.6139982411444 h=24079.651703975844 T = 51.127442737746435 degree C
i=51 x = 1.02 U = 3.027420182450283 m/s Nu = 942.3883935893303 h=24194.826645848952 T = 51.570503886302106 degree C
i=52 x = 1.04 U = 3.0280460849430044 m/s Nu = 946.141780312953 h=24309.411213735813 T = 52.01160382408847 degree C
i=53 x = 1.06 U = 3.028673338653361 m/s Nu = 949.8739527084206 h=24423.397123347342 T = 52.45071133836308 degree C
i=54 x = 1.08 U = 3.0293018302370016 m/s Nu = 953.5847120318456 h=24536.776333002028 T = 52.887795926220804 degree C
i=55 x = 1.1 U = 3.0299314470796075 m/s Nu = 957.2738664973714 h=24649.541044367288 T = 53.32282780019767 degree C
i=56 x = 1.12 U = 3.0305620773411683 m/s Nu = 960.9412312709369 h=24761.683703006387 T = 53.75577789333778 degree C
i=57 x = 1.1400000000000001 U = 3.0311936099991823 m/s Nu = 964.5866284594891 h=24873.19699873308 T = 54.18661786372576 degree C
i=58 x = 1.16 U = 3.031825934890809 m/s Nu = 968.2098870957799 h=24984.07386577729 T = 54.61532009848963 degree C
i=59 x = 1.18 U = 3.032458942753886 m/s Nu = 971.810843118835 h=25094.307482766297 T = 55.04185771727788 degree C
i=60 x = 1.2 U = 3.033092525266831 m/s Nu = 975.3893393501407 h=25203.891272522633 T = 55.46620457521641 degree C
i=61 x = 1.22 U = 3.0337265750873748 m/s Nu = 978.9452254657426 h=25312.818901685867 T = 55.88833526535103 degree C
i=62 x = 1.24 U = 3.0343609858901317 m/s Nu = 982.4783579642527 h=25421.08428015863 T = 56.30822512058188 degree C
i=63 x = 1.26 U = 3.0349956524029564 m/s Nu = 985.9886001309949 h=25528.68156038502 T = 56.7258502150965 degree C
i=64 x = 1.28 U = 3.0356304704420984 m/s Nu = 989.4758219982881 h=25635.60513646206 T = 57.14118736530918 degree C
i=65 x = 1.3 U = 3.0362653369461277 m/s Nu = 992.9399003020296 h=25741.849643090536 T = 57.55421413031388 degree C
i=66 x = 1.32 U = 3.0369001500086066 m/s Nu = 996.3807184347168 h=25847.409954370036 T = 57.96490881185986 degree C
i=67 x = 1.34 U = 3.0375348089095175 m/s Nu = 999.7981663950054 h=25952.281182442624 T = 58.37325045385791 degree C
i=68 x = 1.36 U = 3.038169214145424 m/s Nu = 1003.1921407338956 h=26056.458675988833 T = 58.779218841426996 degree C
i=69 x = 1.3800000000000001 U = 3.038803267458356 m/s Nu = 1006.5625444977369 h=26159.93801858326 T = 59.18279449949034 degree C
i=70 x = 1.4000000000000001 U = 3.0394368718634297 m/s Nu = 1009.9092871681044 h=26262.715026912163 T = 59.58395869093101 degree C
i=71 x = 1.42 U = 3.040069931675168 m/s Nu = 1013.2322845986982 h=26364.78574885971 T = 59.98269341431734 degree C
i=72 x = 1.44 U = 3.0407023525325583 m/s Nu = 1016.5314589494612 h=26466.146461468936 T = 60.37898140120842 degree C
i=73 x = 1.46 U = 3.041334041422809 m/s Nu = 1019.8067386178865 h=26566.793668779203 T = 60.77280611305048 degree C
i=74 x = 1.48 U = 3.041964906703837 m/s Nu = 1023.0580581678031 h=26666.724099549196 T = 61.16415173767547 degree C
i=75 x = 1.5 U = 3.0425948581254634 m/s Nu = 1026.2853582556404 h=26765.93470486767 T = 61.553003185412706 degree C
i=76 x = 1.52 U = 3.0432238068493453 m/s Nu = 1029.4885855544371 h=26864.422655660404 T = 61.939346084825274 degree C
i=77 x = 1.54 U = 3.04385166546763 m/s Nu = 1032.6676926755229 h=26962.18534009405 T = 62.323166778082566 degree C
i=78 x = 1.56 U = 3.0444783480203568 m/s Nu = 1035.822638088215 h=27059.22036088694 T = 62.7044523159812 degree C
i=79 x = 1.58 U = 3.0451037700115964 m/s Nu = 1038.9533860374825 h=27155.525532528383 T = 63.08319045262573 degree C
i=80 x = 1.6 U = 3.0457278484243546 m/s Nu = 1042.0599064598157 h=27251.09887841406 T = 63.4593696397813 degree C
i=81 x = 1.62 U = 3.046350501734244 m/s Nu = 1045.1421748973548 h=27345.938627901 T = 63.83297902091073 degree C
i=82 x = 1.6400000000000001 U = 3.046971649921937 m/s Nu = 1048.2001724104357 h=27440.043213288256 T = 64.20400842490739 degree C
i=83 x = 1.6600000000000001 U = 3.047591214484422 m/s Nu = 1051.2338854886377 h=27533.4112667273 T = 64.57244835953702 degree C
i=84 x = 1.68 U = 3.0482091184450546 m/s Nu = 1054.2433059604969 h=27626.04161706854 T = 64.93829000459993 degree C
i=85 x = 1.7 U = 3.0488252863624643 m/s Nu = 1057.2284309019724 h=27717.933286647585 T = 65.30152520482642 degree C
i=86 x = 1.72 U = 3.049439644338277 m/s Nu = 1060.189262543752 h=27809.085488016062 T = 65.66214646251694 degree C
i=87 x = 1.74 U = 3.050052120023726 m/s Nu = 1063.1258081776039 h=27899.497620623682 T = 66.02014692993998 degree C
i=88 x = 1.76 U = 3.0506626426251353 m/s Nu = 1066.0380800617309 h=27989.169267452366 T = 66.3755204014991 degree C
i=89 x = 1.78 U = 3.0512711429083024 m/s Nu = 1068.9260953254166 h=28078.100191611942 T = 66.7282613056816 degree C
i=90 x = 1.8 U = 3.0518775532018196 m/s Nu = 1071.7898758729132 h=28166.290332897406 T = 67.07836469680052 degree C
i=91 x = 1.82 U = 3.052481807399332 m/s Nu = 1074.6294482867393 h=28253.739804314286 T = 67.42582624654216 degree C
i=92 x = 1.84 U = 3.0530838409607584 m/s Nu = 1077.4448437305182 h=28340.44888857711 T = 67.77064223533029 degree C
i=93 x = 1.86 U = 3.053683590912523 m/s Nu = 1080.2360978513866 h=28426.418034582824 T = 68.11280954351946 degree C
i=94 x = 1.8800000000000001 U = 3.054280995846789 m/s Nu = 1083.003250682093 h=28511.647853864644 T = 68.4523256424282 degree C
i=95 x = 1.9000000000000001 U = 3.054875995919724 m/s Nu = 1085.7463465429244 h=28596.139117031224 T = 68.78918858522366 degree C
i=96 x = 1.92 U = 3.0554685328488556 m/s Nu = 1088.4654339434512 h=28679.892750191848 T = 69.12339699766898 degree C
i=97 x = 1.94 U = 3.0560585499094843 m/s Nu = 1091.1605654842856 h=28762.909831375157 T = 69.45495006874415 degree C
i=98 x = 1.96 U = 3.0566459919302407 m/s Nu = 1093.83179775886 h=28845.19158694201 T = 69.78384754115143 degree C
i=99 x = 1.98 U = 3.05723080528776 m/s Nu = 1096.4791912553112 h=28926.739387996906 T = 70.11008970171537 degree C
i=100 x = 2.0 U = 3.0578129379005325 m/s Nu = 1099.1028102586188 h=29007.554746802572 **T = 70.43367737168859 degree C Q=182204.6128471073 Watt**

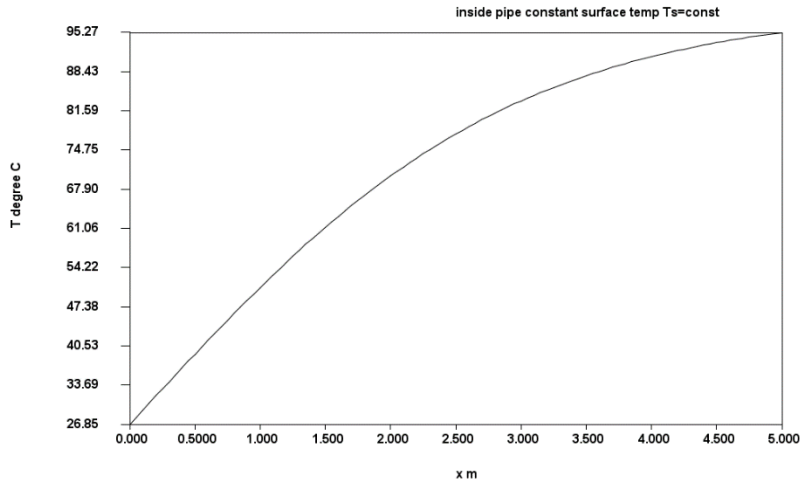


If L=5 m of pipe is taken instead:

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_inside_pipe_FD
i=0 x = 0.0 U = 3.0000000000000004 m/s Nu = 729.5552718920588 h=17814.104921096932 T = 28.07155945510698 degree C
i=1 x = 0.05 U = 3.0010270444823948 m/s Nu = 740.6037416986561 h=18138.124217340912 T = 29.29465626347195 degree C
i=2 x = 0.1 U = 3.002096521336709 m/s Nu = 751.629875776402 h=18462.43049625792 T = 30.51852452164451 degree C
i=3 x = 0.15000000000000002 U = 3.0032071131707667 m/s Nu = 762.6266784097028 h=18786.785368453406 T =
31.742394343233343 degree C
i=4 x = 0.2 U = 3.004357435153133 m/s Nu = 773.5872173627126 h=19110.95029349675 T = 32.965493939065254 degree C
i=5 x = 0.25 U = 3.0055460384154276 m/s Nu = 784.5046417249941 h=19434.687315689567 T = 34.187051718609325 degree C
i=6 x = 0.30000000000000004 U = 3.0067714137928987 m/s Nu = 795.3721995230986 h=19757.759800083837 T =
35.40629839956745 degree C
i=7 x = 0.35000000000000003 U = 3.008031995883049 m/s Nu = 806.1832549869719 h=20079.933163486414 T =
36.622469112332624 degree C
i=8 x = 0.4 U = 3.0093261673984397 m/s Nu = 816.9313053599307 h=20400.975595163716 T = 37.83480548597663 degree C
i=9 x = 0.45 U = 3.01065226378648 m/s Nu = 827.6099971423175 h=20720.65876200876 T = 39.04255770255327 degree C
i=10 x = 0.5 U = 3.012008578085974 m/s Nu = 838.2131416621608 h=21038.75849306218 T = 40.244986506786574 degree C
i=11 x = 0.55 U = 3.0133933659877594 m/s Nu = 848.734729871331 h=21355.055438488598 T = 41.44136515865324 degree C
i=12 x = 0.6000000000000001 U = 3.0148048510645964 m/s Nu = 859.1689462721034 h=21669.335698382736 T =
42.63098131695506 degree C
i=13 x = 0.65 U = 3.0162412301339945 m/s Nu = 869.5101818874411 h=21981.391417130155 T = 43.81313884270105 degree C
i=14 x = 0.7000000000000001 U = 3.0177006787164604 m/s Nu = 879.7530461975945 h=22291.021339447598 T =
44.98715951196522 degree C
i=15 x = 0.75 U = 3.019181356551185 m/s Nu = 889.8923779763218 h=22598.031324684045 T = 46.15238462883991 degree C
i=16 x = 0.8 U = 3.020681413131073 m/s Nu = 899.9232549713342 h=22902.23481645476 T = 47.30817653015026 degree C
i=17 x = 0.8500000000000001 U = 3.0221989932193978 m/s Nu = 909.8410023858063 h=23203.45326520953 T =
48.453919974711084 degree C
i=18 x = 0.9 U = 3.0237322423113753 m/s Nu = 919.6412001299549 h=23501.51650187201 T = 49.58902341107864 degree C
i=19 x = 0.9500000000000001 U = 3.025279312005099 m/s Nu = 929.3196888244304 h=23796.263061245685 T =
50.712920118953036 degree C
i=20 x = 1.0 U = 3.0268383652481337 m/s Nu = 938.8725745495284 h=24087.540454427264 T = 51.82506922060611 degree C
i=21 x = 1.05 U = 3.0284075814280564 m/s Nu = 948.2962323462236 h=24375.205390005816 T = 52.92495655992394 degree C
i=22 x = 1.1 U = 3.0299851612776374 m/s Nu = 957.5873084866555 h=24659.12394434669 T = 54.0120954478478 degree C
i=23 x = 1.1500000000000001 U = 3.031569331567927 m/s Nu = 966.742721542224 h=24939.17168174296 T =
55.08602727415206 degree C
i=24 x = 1.2000000000000002 U = 3.0331583495654293 m/s Nu = 975.7596622874897 h=25215.23372567361 T =
56.14632198660303 degree C
i=25 x = 1.25 U = 3.034750507232409 m/s Nu = 984.6355924868424 h=25487.204782818655 T = 57.19257843958155 degree C
i=26 x = 1.3 U = 3.0363441351525458 m/s Nu = 993.3682426186897 h=25754.989121846636 T = 58.224424615216435 degree C
i=27 x = 1.35 U = 3.037937606167207 m/s Nu = 1001.9556085986046 h=26018.500509310965 T = 59.24151772095462 degree C
i=28 x = 1.4000000000000001 U = 3.0395293387107736 m/s Nu = 1010.3959475682857 h=26277.66210525657 T =
60.24354416828402 degree C
i=29 x = 1.4500000000000002 U = 3.0411177998364805 m/s Nu = 1018.6877728215989 h=26532.406321360275 T =
61.23021943801655 degree C
i=30 x = 1.5 U = 3.04270150792729 m/s Nu = 1026.8298479420093 h=26782.67464459066 T = 62.201287838135045 degree C
i=31 x = 1.55 U = 3.0442790350891364 m/s Nu = 1034.8211802278797 h=27028.417429495625 T = 63.15652216070214 degree C
i=32 x = 1.6 U = 3.0458490092266475 m/s Nu = 1042.6610134832029 h=27269.593662298204 T = 64.09572324472703 degree C
i=33 x = 1.6500000000000001 U = 3.0474101158039777 m/s Nu = 1050.3488202512224 h=27506.170700006514 T =
65.01871945218784 degree C
i=34 x = 1.7000000000000002 U = 3.0489610992957448 m/s Nu = 1057.8842935678629 h=27738.123987737992 T =
65.92536606461528 degree C
i=35 x = 1.75 U = 3.050500764335231 m/s Nu = 1065.2673383100782 h=27965.436757405285 T = 66.81554460776653 degree C
i=36 x = 1.8 U = 3.0520279765689184 m/s Nu = 1072.498062212125 h=28188.099710835984 T = 67.68916211195659 degree C
i=37 x = 1.85 U = 3.053541663228131 m/s Nu = 1079.5767666198678 h=28406.11069028981 T = 68.54615031558023 degree C
i=38 x = 1.9000000000000001 U = 3.055040813430026 m/s Nu = 1086.5039370499153 h=28619.47433920705 T =
69.386464819254 degree C
i=39 x = 1.9500000000000002 U = 3.056524478221396 m/s Nu = 1093.2802336165948 h=28828.201755871734 T =
70.21008419784238 degree C
```

i=40 x = 2.0 U = 3.057991770379817 m/s Nu = 1099.9064813859252 h=29032.310142511094 T = 71.0170090774148 degree C
i=41 x = 2.0500000000000003 U = 3.059441863987397 m/s Nu = 1106.3836607113012 h=29231.82245217391 T = 71.80726118391595 degree C
i=42 x = 2.1 U = 3.060873993793024 m/s Nu = 1112.7128976013696 h=29426.767035549226 T = 72.58088237002818 degree C
i=43 x = 2.15 U = 3.0622874543794083 m/s Nu = 1118.8954541661815 h=29617.17728970107 T = 73.33793362637147 degree C
i=44 x = 2.2 U = 3.0636815991513924 m/s Nu = 1124.9327191830605 h=29803.091310500353 T = 74.07849408282596 degree C
i=45 x = 2.25 U = 3.065055839162083 m/s Nu = 1130.8261988195009 h=29984.551550354783 T = 74.80266000538501 degree C
i=46 x = 2.3000000000000003 U = 3.0664096417932614 m/s Nu = 1136.5775075459705 h=30161.604482648498 T = 75.51054379355634 degree C
i=47 x = 2.35 U = 3.0677425293062406 m/s Nu = 1142.1883592673053 h=30334.300274125017 T = 76.2022729829315 degree C
i=48 x = 2.4000000000000004 U = 3.06905407727904 m/s Nu = 1147.6605586976168 h=30502.69246627912 T = 76.87798925714503 degree C
i=49 x = 2.45 U = 3.070343912945235 m/s Nu = 1152.9959929995557 h=30666.8376666537 T = 77.53784747304655 degree C
i=50 x = 2.5 U = 3.0716117134493146 m/s Nu = 1158.1966237055037 h=30826.795250789604 T = 78.1820147025195 degree C
i=51 x = 2.5500000000000003 U = 3.0728572040327333 m/s Nu = 1163.264478934724 h=30982.6270754281 T = 78.81066929399691 degree C
i=52 x = 2.6 U = 3.074080156164185 m/s Nu = 1168.2016459175031 h=31134.397203433047 T = 79.42399995635708 degree C
i=53 x = 2.6500000000000004 U = 3.0752803856268245 m/s Nu = 1173.0102638344206 h=31282.17164077679 T = 80.02220486752537 degree C
i=54 x = 2.7 U = 3.0764577505745088 m/s Nu = 1177.6925169762849 h=31426.01808582025 T = 80.60549080977097 degree C
i=55 x = 2.75 U = 3.0776121495681803 m/s Nu = 1182.2506282278928 h=31566.00569101715 T = 81.17407233336678 degree C
i=56 x = 2.8000000000000003 U = 3.078743519602874 m/s Nu = 1186.6868528766322 h=31702.204837078974 T = 81.72817094997822 degree C
i=57 x = 2.85 U = 3.079851834134884 m/s Nu = 1191.003472745014 h=31834.686919556945 T = 82.26801435686536 degree C
i=58 x = 2.9000000000000004 U = 3.080937101117884 m/s Nu = 1195.2027906445514 h=31963.52414772544 T = 82.79383569272005 degree C
i=59 x = 2.95 U = 3.0819993610560017 m/s Nu = 1199.287125146839 h=32088.789355586345 T = 83.30587282571844 degree C
i=60 x = 3.0 U = 3.0830386850810365 m/s Nu = 1203.2588056664838 h=32210.555824763313 T = 83.80436767414659 degree C
i=61 x = 3.0500000000000003 U = 3.0840551730602863 m/s Nu = 1207.120167849334 h=32328.89711900546 T = 84.28956555975549 degree C
i=62 x = 3.1 U = 3.085048951740684 m/s Nu = 1210.8735492584933 h=32443.886929982207 T = 84.76171459381823 degree C
i=63 x = 3.1500000000000004 U = 3.0860201729343 m/s Nu = 1214.5212853499381 h=32555.59893402163 T = 85.2210650956979 degree C
i=64 x = 3.2 U = 3.0869690117495288 m/s Nu = 1218.0657057286396 h=32664.106659413184 T = 85.66786904358878 degree C
i=65 x = 3.25 U = 3.087895664871722 m/s Nu = 1221.5091306759768 h=32769.48336388567 T = 86.102379556963 degree C
i=66 x = 3.3000000000000003 U = 3.0888003488963918 m/s Nu = 1224.8538679382837 h=32871.80192184304 T = 86.5248504101425 degree C
i=67 x = 3.35 U = 3.0896832987175267 m/s Nu = 1228.102209766618 h=32971.134720945614 T = 86.93553557631748 degree C
i=68 x = 3.4000000000000004 U = 3.090544765973178 6 m/s Nu = 1231.2564301972373 h=33067.55356760481 T = 87.33468880124894 degree C
i=69 x = 3.45 U = 3.0913850175497735 m/s Nu = 1234.3187825623584 h=33161.12960096765 T = 87.72256320582215 degree C
i=70 x = 3.5 U = 3.0922043341464174 m/s Nu = 1237.2914972206497 h=33251.933214959834 T = 88.09941091655952 degree C
i=71 x = 3.5500000000000003 U = 3.09300300889982 m/s Nu = 1240.1767794969562 h=33340.03398796611 T = 88.46548272315431 degree C
i=72 x = 3.6 U = 3.0937813460703167 m/s Nu = 1242.9768078208729 h=33425.50061972846 T = 88.82102776205016 degree C
i=73 x = 3.6500000000000004 U = 3.0945396597889303 m/s Nu = 1245.693732053793 h=33508.40087505114 T = 89.16629322506367 degree C
i=74 x = 3.7 U = 3.0952782728652757 m/s Nu = 1248.329671994415 h=33588.80153391324 T = 89.50152409202855 degree C
i=75 x = 3.75 U = 3.0959975156557515 m/s Nu = 1250.8867160527525 h=33666.76834759767 T = 89.82696288642916 degree C
i=76 x = 3.8000000000000003 U = 3.0966977249912433 m/s Nu = 1253.3669200829936 h=33742.366000458984 T = 90.1428494529863 degree C
i=77 x = 3.85 U = 3.097379243163409 m/s Nu = 1255.772306365829 h=33815.65807696418 T = 90.44942075616069 degree C
i=78 x = 3.9000000000000004 U = 3.098042416968313 m/s Nu = 1258.104862731127 h=33886.70703365604 T = 90.74691069854617 degree C
i=79 x = 3.95 U = 3.0986875968061987 m/s Nu = 1260.3665418122132 h=33955.574175700676 T = 91.0355499581371 degree C
i=80 x = 4.0 U = 3.0993151358358553 m/s Nu = 1262.5592604231613 h=34022.31963769513 T = 91.31556584346976 degree C
i=81 x = 4.05 U = 3.0999253891820904 m/s Nu = 1264.6848990510482 h=34087.00236842798 T = 91.58718216565836 degree C
i=82 x = 4.1000000000000005 U = 3.100518713194639 m/s Nu = 1266.7453014552652 h=34149.68011929668 T = 91.85061912636712 degree C
i=83 x = 4.15 U = 3.1010954647567925 m/s Nu = 1268.742274366414 h=34210.40943610295 T = 92.10609322078608 degree C
i=84 x = 4.2 U = 3.1016560006419853 m/s Nu = 1270.6775872776284 h=34269.245653959966 T = 92.35381715470544 degree C
i=85 x = 4.25 U = 3.102200676916483 m/s Nu = 1272.552972321455 h=34326.242895060204 T = 92.59399977481092 degree C
i=86 x = 4.3 U = 3.102729848386407 m/s Nu = 1274.3701242257944 h=34381.454069064894 T = 92.82684601135372 degree C
i=87 x = 4.3500000000000005 U = 3.103243868087166 m/s Nu = 1276.1307003427612 h=34434.930875893006 T = 93.05255683237903 degree C
i=88 x = 4.4 U = 3.103743086813476 m/s Nu = 1277.8363207444813 h=34486.723810695294 T = 93.2713292087289 degree C
i=89 x = 4.45 U = 3.104227852688122 m/s Nu = 1279.4885683804148 h=34536.88217081764 T = 93.48335608906723 degree C
i=90 x = 4.5 U = 3.1046985107675877 m/s Nu = 1281.0889892907871 h=34585.45406456448 T = 93.68882638420712 degree C
i=91 x = 4.55 U = 3.1051554026827968 m/s Nu = 1282.6390928713006 h=34632.486421589325 T = 93.88792496005226 degree C
i=92 x = 4.6000000000000005 U = 3.105598866313128 m/s Nu = 1284.1403521843279 h=34678.025004747076 T = 94.08083263849738 degree C
i=93 x = 4.65 U = 3.106029235491992 m/s Nu = 1285.594204312263 h=34722.114423256426 T = 94.26772620566265 degree C
i=94 x = 4.7 U = 3.1064468397422536 m/s Nu = 1287.002050748816 h=34764.79814702824 T = 94.44877842686998 degree C
i=95 x = 4.75 U = 3.106852004039828 m/s Nu = 1288.3652578244194 h=34806.118522028264 T = 94.62415806779802 degree C
i=96 x = 4.800000000000001 U = 3.107245048603834 m/s Nu = 1289.6851571620246 h=34846.116786548984 T = 94.79402992128337 degree C

```
i=97 x = 4.8500000000000005 U = 3.107626288711764 m/s Nu = 1290.9630461599413 h=34884.83308827671 T =
94.95855483926482 degree C
i=98 x = 4.9 U = 3.1079960345381306 m/s Nu = 1292.2001884985063 h=34922.30650204727 T = 95.11788976939482 degree C
i=99 x = 4.95 U = 3.1083545910151233 m/s Nu = 1293.3978146675233 h=34958.57504818994 T = 95.27218779587025 degree C
i=100 x = 5.0 U = 3.1087022577139116 m/s Nu = 1294.5571225118356 h=34993.67571137076 T = 95.42159818406122 degree C
Q=287101.32401790976 Watt
```



PROBLEM: Water is flowing through 40-mm diameter 4-m long tube with a rate of **2kg/s**. Tube surface temperature is maintained at 100 ℃ and the water enters the tube at 27℃

a) What is the outlet temperature of the water?

b) What is the corresponding heat transfer rate?

```
import java.io.*;
public class HT_inside_pipe1
{
 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 public static double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
  double Nu=0;
  double x=Re/1000;
  double f=0;
  if(Re<2300) Nu=3.66;
  else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
  else {f=f_Goudar(Re,eod);
  Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
  }
 return Nu;
  }
public static void main(String arg[]) throws IOException
{       double Ti=26.85+273.15;  // degree K
        double Ts=99.85+273.15; //degree K;
        double To=90+273.15;        //guess for outlet temperature To
        double T=(Ti+To)/2.0;
                double P=300.0;        // kPa
                double m=2.0;          // mass flow rate kg/s
                double e=0.16e-3;      //surface rougness m
                double D=40e-3;        // diameter m
```

```
                double eod=e/D;
                double L=4; //m
                water w=new water();
                double a[]=w.property_TP(T,P);
                double ro=a[2]; //density
                double Cp=a[5]; //Specific heat
                double k=a[6]; //thermal conductivity
                double mu=a[7]; //viscosity
                double Pr=a[9]; //Prandtl number
                double A=Math.PI*D*D/4.0;  //cross sectional area
                double U=m/(ro*A);   //velocity m/s
                double Re=ro*U*D/mu; //Reynols number
                System.out.println("T="+T+" degree K");
                System.out.println("P="+P+" kPa");
                System.out.println("U="+U+" m/s");
                System.out.println("D="+D+" m");
                System.out.println("e="+e+" m");
                double f=f_Goudar(Re,eod);
                System.out.println("f Goudar = "+f);
                double Nu=Nu(Re,eod,Pr);
        double h=Nu*k/D;
        double A1=Math.PI*D*L;
        double Q1=h*A1*(Ts-T);
         System.out.print("Nu = "+Nu);
        System.out.print("h = "+h+"W/m^2K ");
        System.out.println("To initial guess = "+To+" degree K  = "+(To-273)+" degree C");
        To=Ti+Q1/(Cp*1e3);
        System.out.println(" To = "+To+" degree K  = "+(To-273)+" degree C \nQ="+Q1+" W");
                System.out.println("m = "+m+"kg/s");
                double dP=f*L/D*ro*U*U/2.0;
                System.out.println("dP = "+dP+" Pa");
}
//create_data();
}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_inside_pipe1
T=331.575 degree K
P=300.0 kPa
U=1.6172641911803445 m/s
D=0.04 m
e=1.6E-4 m
f Goudar = 0.029239926249620383
Nu = 772.3100468151199h = 12538.2467247622W/m^2K To initial guess = 363.15 degree K  = 90.14999999999998 degree C
 To = 362.43373182911387 degree K  = 89.43373182911387 degree C
Q=261077.34286248437 W
m = 2.0kg/s
dP = 3763.114038848259 Pa

> Terminated with exit code 0.
```

In the second approach problem is solved in finite difference steps. So that properties are taken as local water temperatures in each step.

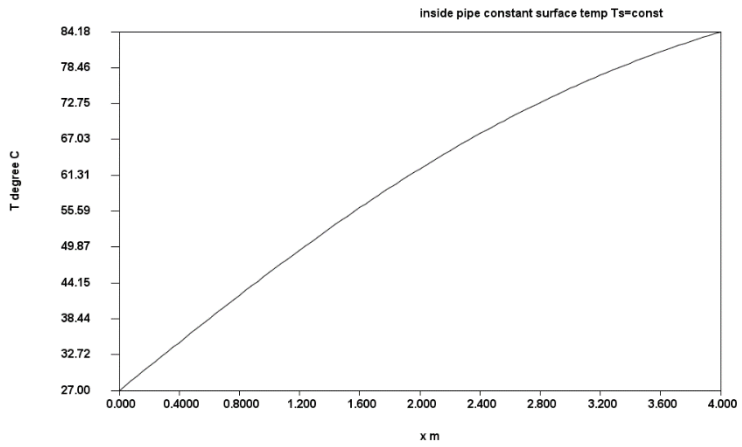
```
 import java.io.*;
public class HT_inside_pipe_FD1
{
 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 public static double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
  double Nu=0;
```

```
   double x=Re/1000;
   double f=0;
   if(Re<2300) Nu=3.66;
   else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
   else {f=f_Goudar(Re,eod);
   Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
   }
 return Nu;
   }
public static void main(String arg[]) throws IOException
{       double Ti=27+273.15;  // degree K
        double Ts=100+273.15; //degree K;
        double T=Ti;
                double P=300.0;        // kPa
                double m=2.0;          //kg/s mass flow rate
                double e=0.16e-3;      //surface rougness m
                double D=40e-3;        // diameter m
                double eod=e/D;
                double L=4; //m
                int N=100;
                double dx=L/N;
                double x=0;
                double TT[]=new double[N+1];
                double xx[]=new double[N+1];
                water w=new water();
                double ro,Cp,k,mu,Pr,Re,f,Nu,h,Q=0,Q1=0;
                double A1=Math.PI*D*dx;
                double A=Math.PI*D*D/4.0;
                double U=0;
                for(int i=0;i<=N;i++)
                { double a[]=w.property_TP(T,P);
                  TT[i]=T-273.15;
                  xx[i]=i*dx;
         ro=a[2];
                   Cp=a[5];
                   k=a[6];
                   mu=a[7];
                   Pr=a[9];
                   U=m/(ro*A);
                   Re=ro*U*D/mu;
         f=f_Goudar(Re,eod);
         Nu=Nu(Re,eod,Pr);
         h=Nu*k/D;
         Q1=h*A1*(Ts-T);
         Q+=Q1;
         T=T+Q1/(Cp*1e3);
         System.out.println("i="+i+" x = "+xx[i]+" U = "+U+" m/s Nu = "+Nu+" h="+h+" T = "+(T-273.15)+" degree C");
        }
        Plot p=new Plot(xx,TT);
        p.setPlabel("inside pipe constant surface temp Ts=const ");
        p.setXlabel("x m");
        p.setYlabel("T degree C");
        p.plot();
        System.out.println("Q="+Q+" Watt");
}
//create_data();
}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_inside_pipe_FD1
i=0 x = 0.0 U = 1.5969696861233507 m/s Nu = 571.7259160230003 h=8728.40153846963 T = 27.766127974982908 degree C
i=1 x = 0.04 U = 1.5973116849962452 m/s Nu = 576.8903698793106 h=8823.823715114035 T = 28.5325552703664 degree C
i=2 x = 0.08 U = 1.5976624608862633 m/s Nu = 582.0426311620275 h=8919.199934637085 T = 29.299095221864093 degree C
i=3 x = 0.12 U = 1.598021832902932 m/s Nu = 587.1814668268387 h=9014.503104025613 T = 30.065560985719117 degree C
i=4 x = 0.16 U = 1.5983896151098347 m/s Nu = 592.305657334445 h=9109.70624270242 T = 30.83176573386328 degree C
i=5 x = 0.2 U = 1.5987656167099766 m/s Nu = 597.4139978000148 h=9204.782514345292 T = 31.59752284932972 degree C
i=6 x = 0.24 U = 1.599149642240665 m/s Nu = 602.5052991181368 h=9299.705258399857 T = 32.362646121446744 degree C
i=7 x = 0.28 U = 1.5995414917774735 m/s Nu = 607.57838906093 h=9394.448021204063 T = 33.12694994034064 degree C
i=8 x = 0.32 U = 1.5999409611468334 m/s Nu = 612.6321133466438 h=9488.984586637696 T = 33.890249490275835 degree C
i=9 x = 0.36 U = 1.6003478421467388 m/s Nu = 617.6653366764489 h=9583.289006216217 T = 34.65236094136469 degree C
i=10 x = 0.4 U = 1.6007619227750416 m/s Nu = 622.6769437370292 h=9677.33562854771 T = 35.413101639184845 degree C
i=11 x = 0.44 U = 1.601182987464745 m/s Nu = 627.6658401667137 h=9771.09912807494 T = 36.172290291850686 degree C
i=12 x = 0.48 U = 1.6016108173257004 m/s Nu = 632.6309534829595 h=9864.554533026476 T = 36.92974715409423 degree C
i=13 x = 0.52 U = 1.6020451903920903 m/s Nu = 637.5712339691864 h=9957.677252505218 T = 37.685294207924244 degree C
i=14 x = 0.56 U = 1.6024858818750163 m/s Nu = 642.4856555189754 h=10050.443102644947 T = 38.43875533944572 degree C
i=15 x = 0.6 U = 1.6029326644195596 m/s Nu = 647.3732164358904 h=10142.82833177026 T = 39.1899565114374 degree C
i=16 x = 0.64 U = 1.60338530836559 m/s Nu = 652.2329401873169 h=10234.809644500134 T = 39.938725931303566 degree C
i=17 x = 0.68 U = 1.6038435820116574 m/s Nu = 657.06387611078 h=10326.364224737948 T = 40.68489421403473 degree C
```

i=18 x = 0.72 U = 1.6043072518812396 m/s Nu = 661.8651000715112 h=10417.469757498031 T = 41.42829453983239 degree C
i=19 x = 0.76 U = 1.6047760829906534 m/s Nu = 666.6357150701634 h=10508.104449523049 T = 42.168762806075904 degree C
i=20 x = 0.8 U = 1.6052498391179066 m/s Nu = 671.3748517996329 h=10598.247048650153 T = 42.906137773331466 degree C
i=21 x = 0.84 U = 1.6057282830717985 m/s Nu = 676.0816691504122 h=10687.876861893115 T = 43.64026120512767 degree C
i=22 x = 0.88 U = 1.6062111769605658 m/s Nu = 680.755354663784 h=10776.973772208941 T = 44.37097800124792 degree C
i=23 x = 0.92 U = 1.6066982824593818 m/s Nu = 685.3951249325779 h=10865.518253926286 T = 45.09813632431366 degree C
i=24 x = 0.96 U = 1.6071893610760402 m/s Nu = 690.0002259492672 h=10953.491386816486 T = 45.82158771946018 degree C
i=25 x = 1.0 U = 1.607684174414173 m/s Nu = 694.5699334015127 h=11040.87486879573 T = 46.54118722693278 degree C
i=26 x = 1.04 U = 1.608182484433347 m/s Nu = 699.103552915272 h=11127.6510272502 T = 47.256793487456605 degree C
i=27 x = 1.08 U = 1.608684053705441 m/s Nu = 703.6004202459076 h=11213.802828982687 T = 47.9682688402616 degree C
i=28 x = 1.12 U = 1.6091886456667017 m/s Nu = 708.0599014178981 h=11299.313888785402 T = 48.675479413669564 degree C
i=29 x = 1.16 U = 1.6096960248649232 m/s Nu = 712.4813928138217 h=11384.16847664635 T = 49.37829520817667 degree C
i=30 x = 1.2 U = 1.6102059572012006 m/s Nu = 716.8643212135714 h=11468.351523604328 T = 50.076590171990745 degree C
i=31 x = 1.24 U = 1.610718210165771 m/s Nu = 721.2081437847996 h=11551.848626269919 T = 50.77024226900767 degree C
i=32 x = 1.28 U = 1.611232553067446 m/s Nu = 725.5123480258413 h=11634.646050036798 T = 51.45913353923601 degree C
i=33 x = 1.32 U = 1.611748757256225 m/s Nu = 729.7764516624114 h=11716.730731009393 T = 52.1431501517024 degree C
i=34 x = 1.36 U = 1.6122665963386478 m/s Nu = 734.0000024995124 h=11798.090276678719 T = 52.8221824498936 degree C
i=35 x = 1.4000000000000001 U = 1.612785846385546 m/s Nu = 738.1825782301589 h=11878.712965381163 T = 53.4961249898127 degree C
i=36 x = 1.44 U = 1.6133062861318386 m/s Nu = 742.3237862025235 h=11958.587744577948 T = 54.164876570747595 degree C
i=37 x = 1.48 U = 1.613827697168084 m/s Nu = 746.4232631472918 h=12037.704227996726 T = 54.82834025887041 degree C
i=38 x = 1.52 U = 1.6143498641235066 m/s Nu = 750.4806748670442 h=12116.05269167935 T = 55.48642340380411 degree C
i=39 x = 1.56 U = 1.6148725748402915 m/s Nu = 754.4957158894875 h=12193.624068980478 T = 56.13903764831019 degree C
i=40 x = 1.6 U = 1.6153956205389348 m/s Nu = 758.4681090865986 h=12270.409944567595 T = 56.78609893126753 degree C
i=41 x = 1.6400000000000001 U = 1.6159187959745056 m/s Nu = 762.3976052615463 h=12346.402547470032 T = 57.42752748412721 degree C
i=42 x = 1.68 U = 1.6164418995836798 m/s Nu = 766.2839827054529 h=12421.594743229343 T = 58.06324782104099 degree C
i=43 x = 1.72 U = 1.6169647336224742 m/s Nu = 770.1270467260192 h=12495.98002520293 T = 58.69318872287374 degree C
i=44 x = 1.76 U = 1.6174871042946035 m/s Nu = 773.9266291500813 h=12569.552505074642 T = 59.317283215320515 degree C
i=45 x = 1.8 U = 1.6180088218704487 m/s Nu = 777.6825878020676 h=12642.30690262451 T = 59.93546854135843 degree C
i=46 x = 1.84 U = 1.618529700796626 m/s Nu = 781.3948059604755 h=12714.238534813063 T = 60.547686128271664 degree C
i=47 x = 1.8800000000000001 U = 1.6190495597962116 m/s Nu = 785.0631917943161 h=12785.343304232392 T = 61.15388154949483 degree C
i=48 x = 1.92 U = 1.619568221959649 m/s Nu = 788.6876777815529 h=12855.617686978612 T = 61.75400448152499 degree C
i=49 x = 1.96 U = 1.6200855148264568 m/s Nu = 792.2682201114781 h=12925.058719997758 T = 62.34800865615779 degree C
i=50 x = 2.0 U = 1.6206012704578319 m/s Nu = 795.8047980729358 h=12993.663987957625 T = 62.935851808305245 degree C
i=51 x = 2.04 U = 1.6211153255002826 m/s Nu = 799.2974134302776 h=13061.431609696729 T = 63.517495619656245 degree C
i=52 x = 2.08 U = 1.6216275212404538 m/s Nu = 802.7460897888625 h=13128.360224300524 T = 64.09290565844037 degree C
i=53 x = 2.12 U = 1.6221377036513114 m/s Nu = 806.1508719518679 h=13194.448976853797 T = 64.66205131555631 degree C
i=54 x = 2.16 U = 1.6226457234298899 m/s Nu = 809.5118252701021 h=13259.697503916246 T = 65.22490573732551 degree C
i=55 x = 2.2 U = 1.6231514360268093 m/s Nu = 812.829034986499 h=13324.105918768064 T = 65.78144575512869 degree C
i=56 x = 2.24 U = 1.6236547016677785 m/s Nu = 816.102605576804 h=13387.674796468735 T = 66.33165181218124 degree C
i=57 x = 2.2800000000000002 U = 1.6241553853673445 m/s Nu = 819.3326600880275 h=13450.405158772503 T = 66.87550788769977 degree C
i=58 x = 2.32 U = 1.6246533569351174 m/s Nu = 822.5193394760416 h=13512.298458940666 T = 67.41300141870693 degree C
i=59 x = 2.36 U = 1.6251484909747493 m/s Nu = 825.6628019437187 h=13573.356566489905 T = 67.94412321971788 degree C
i=60 x = 2.4 U = 1.6256406668759324 m/s Nu = 828.763222280882 h=13633.58175191337 T = 68.46886740054612 degree C
i=61 x = 2.44 U = 1.6261297687996878 m/s Nu = 831.8207912072977 h=13692.976671409968 T = 68.98723128245928 degree C
i=62 x = 2.48 U = 1.6266156856572431 m/s Nu = 834.8357147198157 h=13751.544351654058 T = 69.49921531291062 degree C
i=63 x = 2.52 U = 1.6270983110827806 m/s Nu = 837.8082134447943 h=13809.288174637948 T = 70.00482297906387 degree C
i=64 x = 2.56 U = 1.6275775434003465 m/s Nu = 840.7385219967206 h=13866.211862614864 T = 70.50406072032195 degree C
i=65 x = 2.6 U = 1.6280532855852206 m/s Nu = 843.6268883440142 h=13922.319463170666 T = 70.99693784006291 degree C
i=66 x = 2.64 U = 1.6285254452200486 m/s Nu = 846.4735731828247 h=13977.615334448608 T = 71.48346641677887 degree C
i=67 x = 2.68 U = 1.6289939344460194 m/s Nu = 849.278849319659 h=14032.104130551641 T = 71.96366121480395 degree C
i=68 x = 2.72 U = 1.629458669909396 m/s Nu = 852.0430010634842 h=14085.79078714223 T = 72.43753959481177 degree C
i=69 x = 2.7600000000000002 U = 1.6299195727036861 m/s Nu = 854.7663236280008 h=14138.680507260162 T = 72.905121424252 degree C
i=70 x = 2.8000000000000003 U = 1.6303765683077522 m/s Nu = 857.4491225446972 h=14190.778747376458 T = 73.3664289878891 degree C
i=71 x = 2.84 U = 1.6308295865201345 m/s Nu = 860.0917130871298 h=14242.091203698232 T = 73.82148689859662 degree C
i=72 x = 2.88 U = 1.631278561389889 m/s Nu = 862.694419707002 h=14292.623798740482 T = 74.27032200855314 degree C
i=73 x = 2.92 U = 1.6317234311441966 m/s Nu = 865.2575754823621 h=14342.38266817643 T = 74.71296332097626 degree C
i=74 x = 2.96 U = 1.6321641381130343 m/s Nu = 867.7815215783068 h=14391.374147977822 T = 75.14944190252413 degree C
i=75 x = 3.0 U = 1.6326006286511714 m/s Nu = 870.2666067205515 h=14439.604761856113 T = 75.57979079648447 degree C
i=76 x = 3.04 U = 1.633032853057737 m/s Nu = 872.7131866820198 h=14487.081209011194 T = 76.00404493686415 degree C
i=77 x = 3.08 U = 1.6334607654936253 m/s Nu = 875.1216237827523 h=14533.810352196238 T = 76.42224106348345 degree C
i=78 x = 3.12 U = 1.6338843238969898 m/s Nu = 877.4922864032515 h=14579.79920610325 T = 76.83441763817189 degree C
i=79 x = 3.16 U = 1.634303489897033 m/s Nu = 879.8255485114095 h=14625.054926075045 T = 77.24061476215468 degree C
i=80 x = 3.2 U = 1.6347182287263595 m/s Nu = 882.1217892031017 h=14669.584797146206 T = 77.6408740947117 degree C
i=81 x = 3.24 U = 1.6351285091320849 m/s Nu = 884.3813922564765 h=14713.396223415639 T = 78.03523877318355 degree C
i=82 x = 3.2800000000000002 U = 1.635534303285926 m/s Nu = 886.6047456999939 h=14756.49671775258 T = 78.42375333439185 degree C
i=83 x = 3.3200000000000003 U = 1.6359355866934744 m/s Nu = 888.7922413941773 h=14798.893891836133 T = 78.80646363753533 degree C
i=84 x = 3.36 U = 1.6363323381028365 m/s Nu = 890.9442746270025 h=14840.595446527723 T = 79.18341678861543 degree C
i=85 x = 3.4 U = 1.6367245394128527 m/s Nu = 893.0612437229292 h=14881.609162576106 T = 79.55466106644025 degree C
i=86 x = 3.44 U = 1.6371121755810374 m/s Nu = 895.1435496653813 h=14921.942891651986 T = 79.92024585024802 degree C

```
i=87 x = 3.48 U = 1.637495234531443 m/s Nu = 897.1915957326629 h=14961.604547710971 T = 80.28022154898781 degree C
i=88 x = 3.52 U = 1.637873707062584 m/s Nu = 899.2057871470571 h=15000.602098680067 T = 80.63463953228728 degree C
i=89 x = 3.56 U = 1.638247586755576 m/s Nu = 901.1865307370375 h=15038.943558465498 T = 80.98355206313391 degree C
i=90 x = 3.6 U = 1.638616869882645 m/s Nu = 903.1342346123727 h=15076.636979276605 T = 81.32701223229043 degree C
i=91 x = 3.64 U = 1.638981555316127 m/s Nu = 905.0493078519299 h=15113.69044426141 T = 81.66507389446048 degree C
i=92 x = 3.68 U = 1.63934164443809 m/s Nu = 906.9321602040058 h=15150.112060449055 T = 81.99779160621631 degree C
i=93 x = 3.72 U = 1.6396971410506902 m/s Nu = 908.7832017989045 h=15185.909951992766 T = 82.32522056569599 degree C
i=94 x = 3.7600000000000002 U = 1.6400480512873818 m/s Nu = 910.6028428736025 h=15221.092253708699 T = 82.64741655407374
degree C
i=95 x = 3.8000000000000003 U = 1.640394383525074 m/s Nu = 912.3914935082041 h=15255.667104903376 T = 82.96443587880339
degree C
i=96 x = 3.84 U = 1.6407361482973337 m/s Nu = 914.1495633739846 h=15289.642643484509 T = 83.27633531863125 degree C
i=97 x = 3.88 U = 1.6410733582087156 m/s Nu = 915.8774614927394 h=15323.02700034794 T = 83.5831720703718 degree C
i=98 x = 3.92 U = 1.6414060278503044 m/s Nu = 917.5755960071565 h=15355.82829403381 T = 83.88500369743622 degree C
i=99 x = 3.96 U = 1.6417341737165483 m/s Nu = 919.2443739620478 h=15388.054625646473 T = 84.18188808010143 degree C
i=100 x = 4.0 U = 1.6420578141234263 m/s Nu = 920.8842010960258 h=15419.714074029369 T = 84.47388336750441 degree C
Q=240447.53616958222 Watt
```



PROBLEM: Water with temperature of T=26.85 ℃ and P=300 kPa is flowing through a Di=25x10^{-3} m , Do=28x10^{-3} m commercial steel pipe(k=36 W/mK). Total pipe length L=200 m. In outside of the pipe air is flowing across the pipe with a velocity of V=20 m/s. An air temperature is constant and T_∞=226.85 ℃. Find the total heat transfer and pipe exit temperature also find total pressure drop.

```
 import java.io.*;
public class HT_inside_Outside_pipe
{
            //air side
              public static double mu(double T)
 {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double k(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
```

```
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pr(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;

return y;
}
  public static double ro(double T,double P)
  {   double M=2.8965000E+01;
      double R=8314.5/M;
                return P*1e3/R/T;
  }
  public static double h1(double U,double Tinf,double Ts,double P,double D)
  {//Nusselt number Hilpert equation
   double Re=Re(U,Tinf,P,D);
   double Pr=Pr(Tinf);
   double k1=k(Tinf);
   double Nu=0.0;
   double C=0;
   double m=0;
   if(Re>0.4 && Re<=4) {C=0.989;m=0.330;}
   else if(Re>4.0 && Re<=40) {C=0.911;m=0.385;}
   else if(Re>40.0 && Re<=4000) {C=0.683;m=0.466;}
   else if(Re>4000.0 && Re<=40000) {C=0.193;m=0.618;}
   else if(Re>40000.0 && Re<=400000) {C=0.027;m=0.805;}
   Nu=C*Math.pow(Re,m)*Math.pow(Pr,(1.0/3.0));
   return Nu*k1/D;
   }
   public static double h2(double U,double Tinf,double Ts,double P,double D)
   double Nu=0.3+0.62*Math.pow(Re,0.5)*Math.pow(Pr,(1.0/3.0))/Math.pow((1+Math.pow((0.4/Pr),(2.0/3.0))),0.25);
   Nu*=Math.pow((1+Math.pow((Re/282000),(5.0/8.0))),(4.0/5.0));
   return Nu*k2/D;
   }
  {//Nusselt number Churchill & Bernstein equation
   double Tf=(Tinf+Ts)/2.0;
   double Re=Re(U,Tf,P,D);
   double Pr=Pr(Tf);
   double k2=k(Tf);

  public static double h3(double U,double Tinf,double Ts,double P,double D)
  {//Nusselt number Zukauskas equation
   double Re=Re(U,Tinf,P,D);
   double Pr=Pr(Tinf);
    double k3=k(Tinf);
   double Prs=Pr(Ts);
   double Nu=0.0;
   double C=0;
   double m=0;
   double n=0;
   if(Re>1 && Re<=40) {C=0.75;m=0.4;}
   else if(Re>40.0 && Re<=1000) {C=0.51;m=0.5;}
   else if(Re>40.0 && Re<=4000) {C=0.683;m=0.466;}
   else if(Re>1000.0 && Re<=2e5) {C=0.26;m=0.6;}
   else if(Re>2e5 && Re<=1e6) {C=0.076;m=0.7;}
   if(Pr<=10) n=0.37;
   else n=0.36;
   Nu=C*Math.pow(Re,m)*Math.pow(Pr,n)*Math.pow((Pr/Prs),0.25);
   return  Nu*k3/D;
   }
  public static double Re(double U,double T,double P,double D)
```

```
  {   double mu=mu(T);
      double ro=ro(T,P);
      double Re=ro*U*D/mu;
                return Re;
  }
  //water side
 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/(((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 public static double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
   double Nu=0;
   double x=Re/1000;
   double f=0;
   if(Re<2300) Nu=3.66;
   else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
   else {f=f_Goudar(Re,eod);
   Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
    }
 return Nu;
   }
public static void main(String arg[]) throws IOException
{        double Ti=26.85+273.15;  // degree K
         double Tinf=226.85+273.15; //degree K;
         double To=70.0+273.15;         //guess for outlet temperature To
         double T;
                  double P=300.0;         // kPa
                  double V=3.0;           // velocity m/s
                  double e=0.16e-3;       //surface rougness m
                  double Di=25e-3;         // diameter m
                  double Do=28e-3;         // diameter m
                  double eod=e/Di;
                  double L=200; //m
                  double kw=36.0; //W/mK
                  water w=new water();
                  double a[];
                  double ro=0,Cp=0,k,mu,Pr,Re,f=0,Nu,h1,h2,U,Q1;
                  for(int i=0;i<5;i++)
                  {
                           T=(Ti+To)/2.0;
                  a=w.property_TP(T,P);
                  ro=a[2];
                  Cp=a[5];
                  k=a[6];
                  mu=a[7];
                  Pr=a[9];
                  Re=ro*V*Di/mu;
                  f=f_Goudar(Re,eod);
                  System.out.println("f Goudar = "+f);
                  Nu=Nu(Re,eod,Pr);
         h1=Nu*k/Di;
         h2=h3(V,Tinf,T,P,Do);
         U=1.0/(1.0/(h1*(Math.PI*Di*L))+Math.log(Do/Di)/(2.0*Math.PI*kw*L)+1.0/(h2*(Math.PI*Do*L)));
         Q1=U*(Tinf-T);
         System.out.println("Cp = "+Cp+"J/kgK ");
         System.out.println("Q = "+Q1+"W/m^2K ");
         System.out.println("Q = "+Q1+"W/m^2K ");
         System.out.println("hi = "+h1+"W/m^2K ");
         System.out.println("ho = "+h2+"W/m^2K ");
         System.out.println("U = "+U+"W/K ");
         To=Ti+Q1/(Cp*1e3);
         System.out.println("Ti="+Ti+"To = "+To);
         }
```

```
                double A=Math.PI*Di*Di/4.0;
                double m=ro*A*V;  //kg/s
                System.out.println("m = "+m+"kg/s");
                double dP=f*L/Di*ro*V*V/2.0;
                System.out.println("dP = "+dP);
}
//create_data();
}
```

```
 ---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_inside_Outside_pipe
f Goudar = 0.03338070406938855
Cp = 4.1787826965244275J/kgK
Q = 195962.93069258792W/m^2K
Q = 195962.93069258792W/m^2K
hi = 23310.932651973955W/m^2K
ho = 62.7892161297526W/m^2K
U = 1098.293012148454W/K
Ti=300.0To = 346.8947406275933
f Goudar = 0.033362633010319886
Cp = 4.1791591397112455J/kgK
Q = 193932.2230216538W/m^2K
Q = 193932.2230216538W/m^2K
hi = 23772.986235864304W/m^2K
ho = 62.793905023299324W/m^2K
U = 1098.438597976933W/K
Ti=300.0To = 346.40460354305947
f Goudar = 0.03336494616129356
Cp = 4.179105504031251J/kgK
Q = 194198.0942413714W/m^2K
Q = 194198.0942413714W
hi = 23712.674455772434W/m^2K
ho = 62.79329735296755W/m^2K
U = 1098.4198108191156W/K
Ti=300.0To = 346.46881828038175
f Goudar = 0.03336464223092693
Cp = 4.179112456453951J/kgK
Q = 194163.26276201953W/m^2K
Q = 194163.26276201953W
hi = 23720.578993397186W/m^2K
ho = 62.79337706978085W/m^2K
U = 1098.4222767428098W/K
Ti=300.0To = 346.4604063147825
f Goudar = 0.03336468202995698
Cp = 4.179111544423709J/kgK
Q = 194167.82561914567W/m^2K
Q = 194167.82561914567W
hi = 23719.54356829523W/m^2K
ho = 62.7933666288612W/m^2K
U = 1098.4219537912786W/K
Ti=300.0To = 346.46150827877005
m = 1.4550938255872268kg/s
dP = 1186832.2406123942

> Terminated with exit code 0.
```

PROBLEM

Water at 27 ℃ flows with a mean velocity of 1 m/s through a 1000 m long pipe of 0.25 m inside diameter.

a) Determine the pressure drop over the pipe length and the corresponding pump power requirement, if the pipe surface is smooth

b) If the pipe is made of cast iron and its surface is clean, determine the pressure drop and pump power requirement

```
 import java.io.*;
public class HT_inside_pipe
{
 public static double f_Goudar(double Re,double eod)
```

```
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 public static double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
  double Nu=0;
  double x=Re/1000;
  double f=0;
  if(Re<2300) Nu=3.66;
  else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
  else {f=f_Goudar(Re,eod);
  Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
  }
 return Nu;
  }
public static void main(String arg[]) throws IOException
{       double Ti=27+273.15;  // degree K
        double Ts=57+273.15; //degree K;
        double To=70+273.15;        //guess for outlet temperature To
        double T=(Ti+To)/2.0;
                double P=120.;        // kPa
                double U=1;          // velocity m/s
                double e=0;//0.26e-3;      //surface rougness m cast iron
                double D=20e-3;        // diameter m
                double eod=e/D;
                double L=1000; //m
                water w=new water();
                double a[]=w.property_TP(T,P);
                double ro=a[2];
                double Cp=a[5];
                double k=a[6];
                double mu=a[7];
                double Pr=a[9];
                double Re=ro*U*D/mu;
                System.out.println("T="+T+" degree K");
                System.out.println("P="+P+" kPa");
                System.out.println("U="+U+" m/s");
                System.out.println("D="+D+" m");
                System.out.println("e="+e+" m");
                double f=f_Goudar(Re,eod);
                System.out.println("f Goudar = "+f);
                double Nu=Nu(Re,eod,Pr);
        double h=Nu*k/D;
        double A1=Math.PI*D*L;
        //cross sectional area

        double Q1=h*A1*(Ts-T);
        System.out.print("h = "+h+"W/m^2K ");
        System.out.println("To initial guess = "+To+" degree K  = "+(To-273)+" degree C");
        To=Ti+Q1/(Cp*1e3);
        System.out.println(" To = "+To+" degree K  = "+(To-273)+" degree C \nQ="+Q1+" W");
                double A=Math.PI*D*D/4.0;
                double V=A*U;  //volumetric flow rate m^3/s
                double m=ro*V;  //kg/s
                System.out.println("m = "+m+"kg/s");
                double dP=f*L/D*ro*U*U/2.0;
                System.out.println("dP = "+dP+" Pa");
                double Pow=V*dP;
                System.out.println("Power requirment= "+Pow+" W");

}
//create_data();
}
```

```
 import java.io.*;
```

```
public class HT_inside_pipe
{
 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 public static double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
  double Nu=0;
  double x=Re/1000;
  double f=0;
  if(Re<2300) Nu=3.66;
  else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
  else {f=f_Goudar(Re,eod);
  Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
  }
 return Nu;
  }
public static void main(String arg[]) throws IOException
{       double Ti=27+273.15;  // degree K
        double Ts=57+273.15; //degree K;
        double To=70+273.15;        //guess for outlet temperature To
        double T=(Ti+To)/2.0;
                double P=120.;        // kPa
                double U=1;          // velocity m/s
                double e=0.26e-3;      //surface rougness m cast iron
                double D=20e-3;        // diameter m
                double eod=e/D;
                double L=1000; //m
                water w=new water();
                double a[]=w.property_TP(T,P);
                double ro=a[2];
                double Cp=a[5];
                double k=a[6];
                double mu=a[7];
                double Pr=a[9];
                double Re=ro*U*D/mu;
                System.out.println("T="+T+" degree K");
                System.out.println("P="+P+" kPa");
                System.out.println("U="+U+" m/s");
                System.out.println("D="+D+" m");
                System.out.println("e="+e+" m");
                double f=f_Goudar(Re,eod);
                System.out.println("f Goudar = "+f);
                double Nu=Nu(Re,eod,Pr);
        double h=Nu*k/D;
        double A1=Math.PI*D*L;
        //cross sectional area

        double Q1=h*A1*(Ts-T);
        System.out.print("h = "+h+"W/m^2K ");
        System.out.println("To initial guess = "+To+" degree K  = "+(To-273)+" degree C");
        To=Ti+Q1/(Cp*1e3);
        System.out.println(" To = "+To+" degree K  = "+(To-273)+" degree C \nQ="+Q1+" W");
                double A=Math.PI*D*D/4.0;
                double V=A*U;  //volumetric flow rate m^3/s
                double m=ro*V;  //kg/s
                System.out.println("m = "+m+"kg/s");
                double dP=f*L/D*ro*U*U/2.0;
                System.out.println("dP = "+dP+" Pa");
                double Pow=V*dP;
                System.out.println("Power requirment= "+Pow+" W");

}
```

```
//create_data();
}
---------- Capture Output ---------- SMOOTH PİPE
> "C:\Users\Turhan Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-
0933\jre\bin\javaw.exe" HT_inside_pipe
T=321.65 degree K
P=120.0 kPa
U=1.0 m/s
D=0.02 m
e=2.6E-4 m
f Goudar = 0.04290973460937848
h = 9156.94740111877W/m^2K To initial guess = 343.15 degree K  = 70.14999999999998 degree C
 To = 1470.3354371916762 degree K  = 1197.3354371916762 degree C
Q=4890457.77639269 W
m = 0.3106174540979485kg/s
dP = 1060649.3258406469 Pa
Power requirment= 333.21281300959436 W
> Terminated with exit code 0.
```

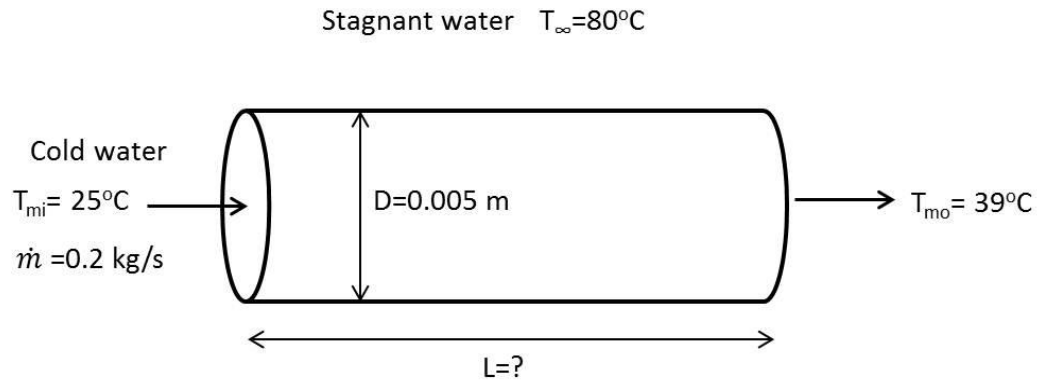
PROBLEM Cold water moves at a rate of m =0.2 kg/s through a 5 mm diameter horizontal tube submerged in a large hot water bath maintained at 80◦C. Convection heat transfer coefficient outside of the tube is 1658 W/m^{2}K. The cold water enters the tube at 25◦C and leaves at 39◦C. (Tube is made of commercial steel)

a) Estimate the length of the tube

b) What is the average surface temperature of the tube, Ts ?

c) What is the required pump power to move cold water through the tube?

Hint: Neglect tube wall thickness, assume fully developed flow, you can assume Ts between 30◦C-40◦C



Cold water properties at

$$T_m = \frac{T_{mi} + T_{mo}}{2} = \frac{25 + 39}{2} = 32\ ℃ + 273.15 = 305.15\ K$$

Water properties (Form steamTAbleIF97.java you can obtain form any other source)

Properties of water and steam
International Association for the Properties of Water and Steam (IAPWS)Industrial formulation 1997

Dr. M. Turhan Çoban,
Ege University, School of Engineering, Mechanical Eng. Dept., Izmir, Turkey
email : turhan.coban@turhancoban.tr
web : www.turhancoban.com

Reference: International Steam Tables, Wolfgang Wagner, Hans-Joachim Kretzschamar
ISBN 978-3-540-21419-9, Springer 2008
273.15 K <= T <= 1073.15 K && 0< P <= 100 MPa
1073.15 K <= T <= 2273.15 K && 0< p <= 50 MPa

select known property pair :	tp	tp
Temperature	32	degree C
Pressure	101.325	kPa

ref no	Property	Value	Units
0	P, pressure	101.325	kPA
1	T, temperature	305.15	deg K
2	v, specific volume	0.0010049930667963...	m³/kg
3	h, enthalpy	134.19321774319454	KJ/kg
4	u, internal energy	134.0913868207014	KJ/kg
5	s, entropy	0.46424831659896804	KJ/kgK
6	x, quality	-1.0	kg vap/kg mix
7	Cp, specific heat at co...	4.179503124066132	KJ/kgK
8	Cv, specific heat at co...	4.108831471595188	KJ/kgK
9	α isobaric thermal exp...	3.202480367382756E-4	1/K
10	κ isothermal compres...	4.4504522496866907E-7	1/Pa
11	η Dynamic viscosity	7.644069730329813E-4	Pa.s
12	k Thermal conductivity	0.6178686153119193	W/mK
13	σ Surface tension	0.07087901096028078	N/m
14	Prandtl number	5.170745450853537	
15	ρ density	995.031740057369	kg/m³
16	speed of sound	1515.5931118537292	m/s

$\dot{m}$=0.2 kg/s

$T_\infty = 80$ ℃

$$h_o = 1658\ W/m^2K$$

D=0.005 m

P=101.325 kPa (cold water pressure. It can be taken higher such as 200 kPa difference will be small)

$\varepsilon = 46x10^{-6}\ m$ (commersial steel)

$$\frac{\varepsilon}{D} = \frac{46x10^{-6}}{0.005} = 0.0092$$

$A = \frac{\pi D^2}{4} = \frac{\pi 0.005^2}{4} = 1.96349x10^{-5}\ m^2$ (Cross sectonal area)

$\dot{m} = \rho VA \quad V = \frac{\dot{m}}{\rho A} = \frac{0.2}{995.031740057369x1.96349x10^{-5}} = 10.236775318638607$ m/s

$$Re = \frac{\rho VD}{\mu} = \frac{995.031740057369x10.236775318638607x0.005}{7.644069730329813x10^{-4}} = 66626.26530908044$$

Flow is Turbulent (Re>2300)

from **Goudar- Sonnad equation (2008)[21]** Valid region: all values

$$f = 0.037797$$

Gnielinski[33] equation

$$Nu = \frac{\left(\frac{f}{8}\right)(Re-1000)Pr}{1.07+12.7\left(\frac{f}{8}\right)^{.5}(Pr^{\frac{2}{3}}-1)} \qquad 0.5 \le Pr \le 2000 \quad 2300 \le Re \le 510^6$$

$$Nu = \frac{\left(\frac{0.037797}{8}\right)(66626.26530908044 - 1000)Pr}{1.07 + 12.7\left(\frac{0.037797}{8}\right)^{.5}(5.1707454508535373^{\frac{2}{3}} - 1)} = 571.0870401009162$$

$$h_i = \frac{Nuk}{D} = \frac{571.0870401009162\text{x}0.6178686153119193}{0.005} = 70571.35173794712\ \text{W/(m}^2\text{K)}$$

$$\frac{1}{U} = \frac{1}{h_i} + \frac{1}{h_o} = \frac{1}{70571.35173794712} + \frac{1}{1658}$$

$$U = 1619.9411785672198\ \text{W/(m}^2\text{K)}$$

$$\Delta T_1 = T_\infty - T_{mi} = 80\ ℃ - 25\ ℃ = 55\ K$$

$$\Delta T_2 = T_\infty - T_{mo} = 80\ ℃ - 39\ ℃ = 41\ K$$

$$LMTD = \frac{\Delta T_1 - \Delta T_2}{ln\left(\frac{\Delta T_1}{\Delta T_2}\right)} = \frac{55 - 41}{ln\left(\frac{55}{41}\right)} = 47.65777060675854$$

$Q = \dot{m}C_p(T_{mo} - T_{mi}) = 0.2\ \text{x}4.179503124066132\text{x}10^3(39 - 25)$ =11702.60875W

$$Q = UA_pLMTD$$

$$A_p = \pi DL = \frac{Q}{UxLMTD} = \frac{11702.60875}{1619.9411785672198\ \text{x}47.65777060675854}$$

$$= 0.15158272767700642\ m^2$$

$$L = \frac{A_p}{\pi D} = \frac{0.15158272767700642}{\pi 0.005} = 9.65005615885929\ \text{m}$$

b) What is the average surface temperature of the tube, Ts ?

$q = \frac{Q}{A_p} = \frac{11702.61\ \text{W}}{0.15158272767700642\ m^2}$ =270.4793742 $\frac{W}{m^2}$

$$q = h_o(T_\infty - T_s)$$

$270.4793742\frac{W}{m^2} = 1658(80 - T_s)$

$$T_s = 33.43619717454847\ ℃$$

c) What is the required pump power to move cold water through the tube?

Darcy-Weisbach pressure drop formula is given as:

$\Delta P = f\frac{L}{D}\rho\frac{V^2}{2}$

$$\Delta P = 0.037797\frac{9.65005615885929}{0.005}995.031740057369\frac{10.236775318638607^{\ 2}}{2}$$

$$\Delta P = 3803216.6694378927\text{Pa}$$

$$Power = \Delta P\frac{\dot{m}}{\rho} = 3803216.6694378927\frac{0.1}{995.031740057369} = 764.4412768619053\ \text{W}$$

```
/* steam property vector
  a[0]="P, pressure       ";
  a[1]="T, temperature    ";
  a[2]="v, specific volume ";
  a[3]="h, enthalpy       ";
  a[4]="u, internal energy ";
  a[5]="s, entropy        ";
  a[6]="x, quality        ";
  a[7]="Cp, specific heat at constant pressure ";
  a[8]="Cv, specific heat at constant volume ";
  a[9]="isobaric thermal expansion coefficient ";
```

```
   a[10]="" isothermal compressibility" ;
   a[11]=" Dynamic viscosity" ;
   a[12]="k Thermal conductivity" ;
   a[13]=" Surface tension" ;
   a[14]="Prandtl number" ;
   a[15]=" density" ;
   a[16]=" speed of sound" ;
*/
public class HT_2023_Q1
{
public static double f(double Re,double eod)
 {
 //Goudar equation
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
public static double Nu(double Re,double Pr,double f)
{double xx=(f/8.0)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8.0)*(Math.pow(Pr,(2.0/3.0))-1));
   return xx;
}
public static void main(String arg[])
{ steamIAPWS_IF97 water=new steamIAPWS_IF97();
  double Tmi=25.0;//Degree C
  double Tmo=39.0;//Degree C
  double m=0.2;//kg/s  mass flow rate
  double Tinf=80.0;//degree C
  double D=0.005;//m  diameter
  double Tm=(Tmi+Tmo)/2.0; //degree C
  double TmK=Tm+273.15;// degree K
  double P=101.325; //kPa
  double eps=46e-6;//m
  double eod=eps/D;
  double a[]=water.property("tp",Tm+273.15,P);
  double ro=a[15];
  double Cp=a[7];
  double mu=a[11];
  double k=a[12];
  double Pr=a[14];
  double A=Math.PI*D*D/4.0;   // m^2 cross sectional area
  double V=m/(ro*A);   // m/s
  System.out.println("Tm="+TmK+"degree
K\nro="+ro+"kg/m^3\nCp="+Cp+"kJ/kgK\nmu="+mu+"Pas\nk="+k+"W/mK\nPr="+Pr);
```

```
  double Re=ro*V*D/mu;
  System.out.println("A="+A+"m^2\n"+"V="+V+"m/s\nRe="+Re);
  double f=f(Re,eod);
  double Nu=Nu(Re,Pr,f);
  double hi=Nu*k/D;
  System.out.println("eod="+eod+"\nf="+f+"\nNu="+Nu+"\nhi="+hi+"W\\mK");
  double ho=1658.0;//W/m^'K
  double U=1.0/(1.0/hi+1/ho);
  System.out.println("U="+U+"W/m^2K");
  double Q=m*Cp*(Tmo-Tmi)*1e3;  //W
  double dT1=Tinf-Tmo;
  double dT2=Tinf-Tmi;
  double LMTD=(dT1-dT2)/Math.log(dT1/dT2);
  double A1=Q/(LMTD*U);
  //a) Length of tube
  double L=A1/(Math.PI*D);
  System.out.println("L="+L+"m \nA1="+A1+"m^2\nLMTD="+LMTD+"degree K");
  //b)  surface temp. Q=h0(Tinf-Ts)
  double q=Q/A1;
  double Ts=Tinf-q/ho;
  System.out.println("Ts="+Ts+"degree C");
  //c) Pump power
  double dP=f*L/D*ro*V*V/2.0;
  double Pow=dP*m/ro;
  System.out.println("dP="+dP+"Pa\nPower="+Pow+" W");
}
}
```

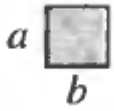
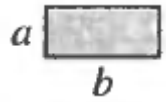
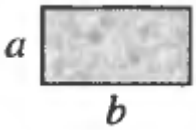
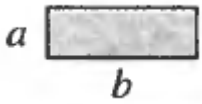
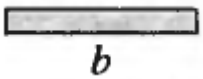
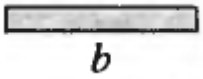
```
---------- Capture Output ----------
> "C:\Users\Turhan
Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-
0933\jre\bin\javaw.exe" HT_2023_Q1
Tm=305.15degree K
ro=995.031740057369kg/m^3
Cp=4.179503124066132kJ/kgK
mu=7.644069730329813E-4Pas
k=0.6178686153119193W/mK
Pr=5.170745450853537
A=1.963495408493621E-5m^2
V=10.236775318638607m/s
Re=66626.26530908044
eod=0.0092
f=0.037797053921779004
Nu=571.0870401009162
hi=70571.35173794712W\mK
U=1619.9411785672198W/m^2K
L=9.65005615885929m
A1=0.15158272767700642m^2
LMTD=47.65777060675854degree K
Ts=33.43619717454847degree C
dP=3803216.6694378927Pa
Power=764.4412768619053 W
```

> Terminated with exit code 0.

13.3 INTERNAL FLOW HEAT TRANSFER IN NON_CIRCULAR TUBES

In order to avaluate non-circular tubes hydrolic diameter concept can be used.

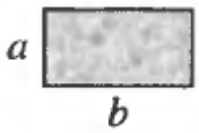
$D_h = \frac{4A_c}{P}$ where A_c is cross-sectional area and P is the perimeter. For turbulent flow the same equation will be used by taking hydrolic diameter instead of circular diameter in the equations. If it is Laminar flow constant Nu numbers are given

Cross section	b/a	Uniform qs	Uniform Ts
		**4.36**	**3.66**
	**1**	**3.61**	**2.98**
	**1.43**	**3.73**	**3.08**
	**2**	**4.12**	**3.39**
	**3**	**4.79**	**3.96**
	**4**	**5.33**	**4.44**
	**8**	**6.49**	**5.60**
	∞	**8.23**	**7.54**
		**3.11**	**2.49**

Laminar rectangular Nusselt Number polynomial curve-fitting coefficients:

$$Nu = a[0] + a[1] * \left(\frac{b}{a}\right) + a[2] * \left(\frac{b}{a}\right)^2 + a[3] * \left(\frac{b}{a}\right)^3 + a[4] * \left(\frac{b}{a}\right)^4 + a[5] * \left(\frac{b}{a}\right)^5 + a[6] * \left(\frac{b}{a}\right)^6$$

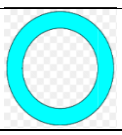
a[0]= .754000000000580E+01;
a[1]= -.193789080028494E+02;
a[2]= .333861212302836E+02;
a[3]= -.170496495070800E+02 ;
a[4]= -.304226226912837E+02;
a[5]= .495297442580787E+02;
a[6]= -.206246852871550E+02;



Laminar concentric annulus Nusselt Number polynomial curve-fitting coefficients:

$$Nu = a[0] + a[1] * \left(\frac{d2}{d1}\right) + a[2] * \left(\frac{d2}{d1}\right)^2 + a[3] * \left(\frac{d2}{d1}\right)^3$$

a[0]= .401698044199492E+01;
a[1]= .921803331830132E+00;
a[2]= -.307845192109755E+00;
a[3]= .229092480046921E+00;



PROBLEM:

Water at an inlet temperature of 27 ℃ enters into a pipe with 2.5x10-2m in diameter end exit at a temperature of 67 C. The water velocity at the pipe inlet is u=3 m/s. If the surface temperature of the pipe is Ts=107 C, calculate

a) the heat transfer coefficient through the pipe

b) Pipe length

c) Total heat transfer rate through the pipe

```
import java.io.*;
public class HT_inside_waterHW2

{
 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 public static double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
  double Nu=0;
  double x=Re/1000;
  double f=0;
  if(Re<2300) Nu=3.66;
  else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
  else {f=f_Goudar(Re,eod);
  Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
  }
 return Nu;
  }
public static void main(String arg[]) throws IOException
{       double Ti=27+273.15;  // degree K
        double Ts=107+273.15; //degree K;
        double To=67+273.15;        //guess for outlet temperature To
        double T=(Ti+To)/2.0;
                double P=300.0;        // kPa
                double U=3.0;          // velocity m/s
                double e=0;      //surface rougness m
                double D=25e-3;        // diameter m
                double eod=e/D;
                water w=new water();
                double L=2.573;//m
                double a[]=w.property_TP(T,P);
                double ro=a[2];
                double Cp=a[5];
                double k=a[6];
                double mu=a[7];
                double Pr=a[9];
                double Re=ro*U*D/mu;
                System.out.println("T="+T+" degree K");
                System.out.println("P="+P+" kPa");
                System.out.println("U="+U+" m/s");
                System.out.println("D="+D+" m");
                System.out.println("e="+e+" m");
                double f=f_Goudar(Re,eod);
                System.out.println("f Goudar = "+f);
                double Nu=Nu(Re,eod,Pr);
        double h=Nu*k/D;
        double A1=Math.PI*D*L;
        double Q1=h*A1*(Ts-T);
```

```
        System.out.print("h = "+h+"W/m^2K ");
        System.out.println("To initial guess = "+To+" degree K  = "+(To-273)+" degree C");
        To=Ti+Q1/(Cp*1e3);
        System.out.println(" To = "+To+" degree K  = "+(To-273)+" degree C \nQ="+Q1+" W");
                double A=Math.PI*D*D/4.0;
                double m=ro*A*U;  //kg/s
                System.out.println("m = "+m+"kg/s");
                double dP=f*L/D*ro*U*U/2.0;
                System.out.println("dP = "+dP+" Pa");
}
//create_data();
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-
0933\jre\bin\javaw.exe" HT_inside_waterHW2
T=320.15 degree K
P=300.0 kPa
U=3.0 m/s
D=0.025 m
e=0.0 m
f Goudar = 0.017069367893462542
h = 13785.032004431088W/m^2K To initial guess = 340.15 degree K  = 67.14999999999998 degree C
 To = 340.15030542716926 degree K  = 67.15030542716926 degree C
Q=167143.1938823993 W
m = 1.4570998307033105kg/s
dP = 7822.1814211405235 Pa
> Terminated with exit code 0.
```

```
import java.io.*;
public class HT_inside_waterHW2

{
 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 public static double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
  double Nu=0;
  double x=Re/1000;
  double f=0;
  if(Re<2300) Nu=3.66;
  else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
  else {f=f_Goudar(Re,eod);
  Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
  }
 return Nu;
  }
public static void main(String arg[]) throws IOException
{       double Ti=27+273.15;  // degree K
        double Ts=107+273.15; //degree K;
        double To=67+273.15;        //guess for outlet temperature To
        double T=(Ti+To)/2.0;
                double P=300.0;        // kPa
                double U=3.0;          // velocity m/s
                double e=0.16e-3;      //surface rougness m
                double D=25e-3;        // diameter m
                double eod=e/D;
                water w=new water();
                double L=1.55;//m
                double a[]=w.property_TP(T,P);
                double ro=a[2];
```

```
                double Cp=a[5];
                double k=a[6];
                double mu=a[7];
                double Pr=a[9];
                double Re=ro*U*D/mu;
                System.out.println("T="+T+" degree K");
                System.out.println("P="+P+" kPa");
                System.out.println("U="+U+" m/s");
                System.out.println("D="+D+" m");
                System.out.println("e="+e+" m");
                double f=f_Goudar(Re,eod);
                System.out.println("f Goudar = "+f);
                double Nu=Nu(Re,eod,Pr);
        double h=Nu*k/D;
        double A1=Math.PI*D*L;
        double Q1=h*A1*(Ts-T);
        System.out.print("h = "+h+"W/m^2K ");
        System.out.println("To initial guess = "+To+" degree K  = "+(To-273)+" degree C");
        To=Ti+Q1/(Cp*1e3);
        System.out.println(" To = "+To+" degree K  = "+(To-273)+" degree C \nQ="+Q1+" W");
                double A=Math.PI*D*D/4.0;
                double m=ro*A*U;  //kg/s
                System.out.println("m = "+m+"kg/s");
                double dP=f*L/D*ro*U*U/2.0;
                System.out.println("dP = "+dP+" Pa");
}
//create_data();
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-
0933\jre\bin\javaw.exe" HT_inside_waterHW2
T=320.15 degree K
P=300.0 kPa
U=3.0 m/s
D=0.025 m
e=1.6E-4 m
f Goudar = 0.03339509480178225
h = 22957.376943128136W/m^2K To initial guess = 340.15 degree K  = 67.14999999999998 degree C
 To = 340.2800504533178 degree K  = 67.28005045331781 degree C
Q=167685.3396942686 W
m = 1.4570998307033105kg/s
dP = 9219.025967160564 Pa
> Terminated with exit code 0.
```

```
import java.io.*;
public class HT_inside_pipe_FD
{
 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 public static double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
   double Nu=0;
   double x=Re/1000;
   double f=0;
   if(Re<2300) Nu=3.66;
   else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
   else {f=f_Goudar(Re,eod);
   Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
```

```
   }
 return Nu;
   }
public static void main(String arg[]) throws IOException
{        double Ti=27+273.15;  // degree K
         double Ts=107+273.15; //degree K;
         double T=Ti;
                  double P=300.0;         // kPa
                  double U=3.0;           // velocity m/s
                  double e=0;//0.16e-3;       //surface rougness m
                  double D=25e-3;         // diameter m
                  double eod=e/D;
                  double L=2.65; //m
                  int N=100;
                  double dx=L/N;
                  double x=0;
                  double TT[]=new double[N+1];
                  double xx[]=new double[N+1];
                  water w=new water();
                  double ro,Cp,k,mu,Pr,Re,f,Nu,h,Q=0,Q1=0;
                  double a[]=w.property_TP(Ti,P);
                  ro=a[2];
                  double A1=Math.PI*D*dx;
                  double A=Math.PI*D*D/4.0;
                  double m=ro*U*A;
                  for(int i=0;i<=N;i++)
                  { a=w.property_TP(T,P);
                    TT[i]=T-273.15;
                    xx[i]=i*dx;
          ro=a[2];
                    Cp=a[5];
                    k=a[6];
                    mu=a[7];
                    Pr=a[9];
                    U=m/(ro*A);
                    Re=ro*U*D/mu;
          f=f_Goudar(Re,eod);
          Nu=Nu(Re,eod,Pr);
          h=Nu*k/D;
          Q1=h*A1*(Ts-T);
          Q+=Q1;
          T=T+Q1/(Cp*1e3);
          System.out.println("i="+i+" x = "+xx[i]+" U = "+U+" m/s Nu = "+Nu+" h="+h+" T = "+(T-273.15)+" degree C");
         }
         Plot p=new Plot(xx,TT);
         p.setPlabel("inside pipe constant surface temp Ts=const ");
         p.setXlabel("x m");
         p.setYlabel("T degree C");
         p.plot();
         System.out.println("Q="+Q+" Watt");
}
//create_data();
}
```

```
---------- Capture Output ---------- SMOOTH TUBE FINITE ELEMENT
> "C:\Users\Turhan Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-
0933\jre\bin\javaw.exe" HT_inside_pipe_FD
i=0 x = 0.0 U = 3.0 m/s Nu = 479.8711572957336 h=11721.723378903203 T = 27.466864765414016 degree C
i=1 x = 0.0265 U = 3.0003895643531027 m/s Nu = 481.5898692240505 h=11777.225010744687 T = 27.933222251319762 degree C
i=2 x = 0.053 U = 3.000784750068775 m/s Nu = 483.2953298611546 h=11832.403753081282 T = 28.399036014280966 degree C
i=3 x = 0.0795 U = 3.001185461695994 m/s Nu = 484.9875165615732 h=11887.256521643563 T = 28.864269991103754 degree C
i=4 x = 0.106 U = 3.001591603491813 m/s Nu = 486.66641090793047 h=11941.780370785478 T = 29.32888850785804 degree C
i=5 x = 0.1325 U = 3.0020030794573676 m/s Nu = 488.3319986275554 h=11995.972492430416 T = 29.792856288549388 degree C
i=6 x = 0.159 U = 3.0024197933736385 m/s Nu = 489.9842695079562 h=12049.83021493273 T = 30.256138463439697 degree C
i=7 x = 0.1855 U = 3.0028416488369123 m/s Nu = 491.6232173112756 h=12103.351001858231 T = 30.718700577016307 degree C
i=8 x = 0.212 U = 3.0032685492939626 m/s Nu = 493.24883968786895 h=12156.532450686967 T = 31.180508595607876 degree C
i=9 x = 0.2385 U = 3.0037003980768735 m/s Nu = 494.8611380891396 h=12209.37229144257 T = 31.641528914647893 degree C
i=10 x = 0.265 U = 3.004137098437538 m/s Nu = 496.4601176796814 h=12261.868385249412 T = 32.1017283655853 degree C
i=11 x = 0.2915 U = 3.0045785535817497 m/s Nu = 498.045787248935 h=12314.018722823743 T = 32.561074222442926 degree C
i=12 x = 0.318 U = 3.0050246667029388 m/s Nu = 499.61815912239 h=12365.821422899584 T = 33.01953420802511 degree C
i=13 x = 0.3445 U = 3.005475341015445 m/s Nu = 501.1772490724889 h=12417.274730594832 T = 33.477076499775364 degree C
i=14 x = 0.371 U = 3.005930479787388 m/s Nu = 502.72307622930157 h=12468.37701571918 T = 33.93366973528623 degree C
i=15 x = 0.39749999999999996 U = 3.006389986373061 m/s Nu = 504.2556629911055 h=12519.126771028641 T = 34.38928301746313
degree C
i=16 x = 0.424 U = 3.0068537642448545 m/s Nu = 505.7750349349514 h=12569.522610429209 T = 34.843885919344984 degree C
i=17 x = 0.4505 U = 3.007321717024672 m/s Nu = 507.2812207272925 h=12619.563267132839 T = 35.29744848858411 degree C
i=18 x = 0.477 U = 3.007793748514859 m/s Nu = 508.77425203481766 h=12669.247591769812 T = 35.74994125158861 degree C
i=19 x = 0.5035 U = 3.008269762728566 m/s Nu = 510.2541634355163 h=12718.574550459698 T = 36.20133521733072 degree C
```

i=20 x = 0.53 U = 3.008749663919608 m/s Nu = 511.7209923300968 h=12767.543222844506 T = 36.6516018808249 degree C
i=21 x = 0.5565 U = 3.0092333566117264 m/s Nu = 513.1747788538445 h=12816.15280008778 T = 37.10071322627937 degree C
i=22 x = 0.583 U = 3.0097207456273014 m/s Nu = 514.6155657889648 h=12864.402582841552 T = 37.54864172992603 degree C
i=23 x = 0.6094999999999999 U = 3.010211736115464 m/s Nu = 516.0433984775115 h=12912.291979185022 T = 37.995360362532494 degree C
i=24 x = 0.636 U = 3.010706233579622 m/s Nu = 517.4583247349669 h=12959.820502537616 T = 38.44084259160178 degree C
i=25 x = 0.6625 U = 3.011204143904364 m/s Nu = 518.8603947645241 h=13006.987769549256 T = 38.88506238326397 degree C
i=26 x = 0.689 U = 3.0117053733817465 m/s Nu = 520.2496610721772 h=13053.793497971355 T = 39.327994203865956 degree C
i=27 x = 0.7155 U = 3.0122098287369536 m/s Nu = 521.6261783826156 h=13100.23750450995 T = 39.769613021263694 degree C
i=28 x = 0.742 U = 3.012717417153317 m/s Nu = 522.9900035560405 h=13146.319702665041 T = 40.209894305823696 degree C
i=29 x = 0.7685 U = 3.013228046296691 m/s Nu = 524.3411955059365 h=13192.040100558419 T = 40.6488140311385 degree C
i=30 x = 0.7949999999999999 U = 3.0137416243391693 m/s Nu = 525.6798151178288 h=13237.39879875184 T = 41.086348674462954 degree C
i=31 x = 0.8215 U = 3.014258059982159 m/s Nu = 527.0059251690955 h=13282.395988058679 T = 41.5224752168769 degree C
i=32 x = 0.848 U = 3.0147772624787654 m/s Nu = 528.319590249894 h=13327.03194735198 T = 41.95717114318086 degree C
i=33 x = 0.8744999999999999 U = 3.0152991416555333 m/s Nu = 529.6208766852023 h=13371.307041369928 T = 42.39041444153082 degree C
i=34 x = 0.901 U = 3.0158236079335112 m/s Nu = 530.909852458041 h=13415.221718521861 T = 42.822183602818825 degree C
i=35 x = 0.9275 U = 3.0163505723486326 m/s Nu = 532.1865871339173 h=13458.77650869726 T = 43.25245761980585 degree C
i=36 x = 0.954 U = 3.0168799465714358 m/s Nu = 533.4511517864958 h=13501.972021078813 T = 43.68121598601357 degree C
i=37 x = 0.9804999999999999 U = 3.0174116429261026 m/s Nu = 534.7036189245586 h=13544.808941962694 T = 44.1084386943819 degree C
i=38 x = 1.007 U = 3.0179455744088157 m/s Nu = 535.9440624202565 h=13587.288032587277 T = 44.53410623569897 degree C
i=39 x = 1.0334999999999999 U = 3.018481654705454 m/s Nu = 537.1725574386853 h=13629.410126972207 T = 44.95819959681046 degree C
i=40 x = 1.06 U = 3.0190197982086033 m/s Nu = 538.3891803688067 h=13671.176129769838 T = 45.38070025861521 degree C
i=41 x = 1.0865 U = 3.0195599200339083 m/s Nu = 539.5940087557465 h=13712.587014130859 T = 45.80159019385371 degree C
i=42 x = 1.113 U = 3.0201019360357377 m/s Nu = 540.7871212344578 h=13753.643819585179 T = 46.2208518646969 degree C
i=43 x = 1.1395 U = 3.0206457628222 m/s Nu = 541.968597464791 h=13794.347649940084 T = 46.638468220141476 degree C
i=44 x = 1.166 U = 3.0211913177694 97 m/s Nu = 543.1385180679686 h=13834.699671196973 T = 47.05442269321952 degree C
i=45 x = 1.1925 U = 3.0217385190356083 m/s Nu = 544.2969645644813 h=13874.70110948815 T = 47.4686991980285 degree C
i=46 x = 1.2189999999999999 U = 3.022287285573348 m/s Nu = 545.4440193133992 h=13914.35324903449 T = 47.88128212658893 degree C
i=47 x = 1.2455 U = 3.0228375371427476 m/s Nu = 546.5797654531193 h=13953.65743012601 T = 48.29215634553 64 degree C
i=48 x = 1.272 U = 3.0233891943228266 m/s Nu = 547.7042868435573 h=13992.61504712603 T = 48.70130719265495 degree C
i=49 x = 1.2985 U = 3.0239421785227076 m/s Nu = 548.8176680097514 h=14031.227546499691 T = 49.1087204732583 degree C
i=50 x = 1.325 U = 3.0244964119921334 m/s Nu = 549.9199940869203 h=14069.496424868497 T = 49.514382456425494 degree C
i=51 x = 1.3515 U = 3.0250518178313 36 m/s Nu = 551.0113507669433 h=14107.42322709143 T = 49.91827987109809 degree C
i=52 x = 1.378 U = 3.0256083200003 29 m/s Nu = 552.0918242462739 h=14145.009544373379 T = 50.32039990204498 degree C
i=53 x = 1.4044999999999999 U = 3.0261658433275627 m/s Nu = 553.1615011752732 h=14182.257012402202 T = 50.72073018570126 degree C
i=54 x = 1.431 U = 3.0267243135180286 m/s Nu = 554.2204686089577 h=14219.167309514301 T = 51.11925880588802 degree C
i=55 x = 1.4575 U = 3.0272836571607504 m/s Nu = 555.2688139591683 h=14255.742154890506 T = 51.51597428941875 degree C
i=56 x = 1.484 U = 3.0278438017357 12 m/s Nu = 556.3066249481093 h=14291.983306781443 T = 51.910865601599085 degree C
i=57 x = 1.5105 U = 3.0284046756202296 m/s Nu = 557.333989563308 h=14327.8925607645 T = 52.303922141625435 degree C
i=58 x = 1.537 U = 3.0289662080947557 m/s Nu = 558.3509960139146 h=14363.471748031325 T = 52.69513373788919 degree C
i=59 x = 1.5635 U = 3.029528329348156 m/s Nu = 559.3577326884009 h=14398.72273370785 T = 53.08449064319154 degree C
i=60 x = 1.5899999999999999 U = 3.0300909704824464 m/s Nu = 560.354288113559 h=14433.647415205254 T = 53.47198352987556 degree C
i=61 x = 1.6165 U = 3.0306540635170047 m/s Nu = 561.3407509148797 h=14468.247720604553 T = 53.857603484880485 degree C
i=62 x = 1.643 U = 3.0312175413922815 m/s Nu = 562.3172097782043 h=14502.525607072708 T = 54.24134200472423 degree C
i=63 x = 1.6695 U = 3.031781337973005 m/s Nu = 563.2837534127037 h=14536.483059312337 T = 54.623190990419175 degree C
i=64 x = 1.696 U = 3.032345388050897 m/s Nu = 564.2404705151172 h=14570.122088043965 T = 55.003142742326986 degree C
i=65 x = 1.7225 U = 3.0329096273469163 m/s Nu = 565.1874497352604 h=14603.444728521532 T = 55.38118995495722 degree C
i=66 x = 1.7489999999999999 U = 3.033473992513029 m/s Nu = 566.1247796427702 h=14636.45303908103 T = 55.757325711714884 degree C
i=67 x = 1.7754999999999999 U = 3.034038421133535 m/s Nu = 567.0525486950734 h=14669.149099722348 T = 56.131543479602044 degree C
i=68 x = 1.802 U = 3.0346028517259436 m/s Nu = 567.9708452065571 h=14701.535010724352 T = 56.503837103878084 degree C
i=69 x = 1.8285 U = 3.0351672237414182 m/s Nu = 568.8797573189098 h=14733.612891292905 T = 56.87420080268345 degree C
i=70 x = 1.855 U = 3.0357314775648048 m/s Nu = 569.7793729726329 h=14765.384878242248 T = 57.24262916163127 degree C
i=71 x = 1.8815 U = 3.036295554514249 m/s Nu = 570.6697798796695 h=14796.853124708792 T = 57.609117128371224 degree C
i=72 x = 1.908 U = 3.0368593968404225 m/s Nu = 571.5510654971707 h=14828.019798898298 T = 57.97366000713026 degree C
i=73 x = 1.9344999999999999 U = 3.0374229477253576 m/s Nu = 572.4233170023316 h=14858.887082865103 T = 58.33625345323378 degree C
i=74 x = 1.9609999999999999 U = 3.0379861512808994 m/s Nu = 573.2866212683017 h=14889.457171323933 T = 58.69689346761203 degree C
i=75 x = 1.9875 U = 3.0385489525468334 m/s Nu = 574.1410648411517 h=14919.732270493841 T = 59.055576391294665 degree C
i=76 x = 2.014 U = 3.0391112974886085 m/s Nu = 574.9867339178388 h=14949.714596973803 T = 59.4122988998983 degree C
i=77 x = 2.0404999999999998 U = 3.0396731329947615 m/s Nu = 575.8237143252014 h=14979.4063766503 T = 59.76705799810969 degree C
i=78 x = 2.0669999999999997 U = 3.040234406873995 m/s Nu = 576.6520914998963 h=15008.809843635456 T = 60.11985101416883 degree C
i=79 x = 2.0935 U = 3.0407950678519513 m/s Nu = 577.4719504693156 h=15037.927239236635 T = 60.4706755943547 degree C
i=80 x = 2.12 U = 3.041355065567658 m/s Nu = 578.2833758334214 h=15066.760810956532 T = 60.81952969747766 degree C
i=81 x = 2.1465 U = 3.0419143505697046 m/s Nu = 579.0864517474746 h=15095.312811522805 T = 61.1664115893808 degree C
i=82 x = 2.173 U = 3.0424728743121094 m/s Nu = 579.881261905678 h=15123.585497948656 T = 61.511319837453925 degree C

```
i=83 x = 2.1995 U = 3.0430305891499376 m/s Nu = 580.6678895256372 h=15151.581130621345 T = 61.854253305162786 degree C
i=84 x = 2.226 U = 3.0435874483346383 m/s Nu = 581.4464173336962 h=15179.301972420863 T = 62.19521114659619 degree C
i=85 x = 2.2525 U = 3.0441434060091317 m/s Nu = 582.2169275510595 h=15206.750287866398 T = 62.53419280103412 degree C
i=86 x = 2.279 U = 3.0446984172026585 m/s Nu = 582.9795018807223 h=15233.928342291238 T = 62.87119798753889 degree C
i=87 x = 2.3055 U = 3.045252437825395 m/s Nu = 583.7342214951536 h=15260.838401044897 T = 63.20622669957197 degree C
i=88 x = 2.332 U = 3.045805424662841 m/s Nu = 584.4811670247447 h=15287.482728722916 T = 63.539279199639 degree C
i=89 x = 2.3585 U = 3.0463573353700046 m/s Nu = 585.2204185469651 h=15313.863588422839 T = 63.87035601396491 degree C
i=90 x = 2.385 U = 3.0469081284653687 m/s Nu = 585.9520555762252 h=15339.98324102662 T = 64.19945792720154 degree C
i=91 x = 2.4114999999999998 U = 3.0474577633246804 m/s Nu = 586.6761570544272 h=15365.843944508746 T = 64.52658597716936
degree C
i=92 x = 2.4379999999999997 U = 3.048006200174543 m/s Nu = 587.3928013421662 h=15391.447953269484 T = 64.85174144963565
degree C
i=93 x = 2.4645 U = 3.048553400085835 m/s Nu = 588.1020662105749 h=15416.797517492907 T = 65.17492587313063 degree C
i=94 x = 2.491 U = 3.0490993249669733 m/s Nu = 588.8040288337869 h=15441.894882529068 T = 65.49614101380342 degree C
i=95 x = 2.5175 U = 3.049643937556994 m/s Nu = 589.4987657819956 h=15466.74228830008 T = 65.8153888703194 degree C
i=96 x = 2.544 U = 3.0501872014185127 m/s Nu = 590.1863530150949 h=15491.34196872922 T = 66.13267166880047 degree C
i=97 x = 2.5705 U = 3.0507290809305236 m/s Nu = 590.8668658768723 h=15515.696151192771 T = 66.44799185780096 degree C
i=98 x = 2.597 U = 3.0512695412810675 m/s Nu = 591.5403790897462 h=15539.807055994243 T = 66.761352103381 degree C
i=99 x = 2.6235 U = 3.0518085484597854 m/s Nu = 592.206966750025 h=15563.676895860131 T = 67.07275528409752 degree C
i=100 x = 2.65 U = 3.0523460692503446 m/s Nu = 592.8667023236603 h=15587.30787545695 T = 67.38220448621587 degree C
Q=168801.50875150986 Watt
```

PROBLEM:

Water flows with a mean velocity of Um= 3 m/s inside a circular pipe of diameter D=5x10-2 m. The pipe is commercial steel, and its Wall is maintained at a uniform temperature Tw=100 C. Pipe length is L=20 m, and pipe inlet temperature Ti=20 C. Calculate

a) the heat transfer coefficient through the pipe

b) The exit temperature T0

c) Total heat transfer rate through the pipe

For this problem only numerical (finite difference method can give proper answer because pipe is very long)

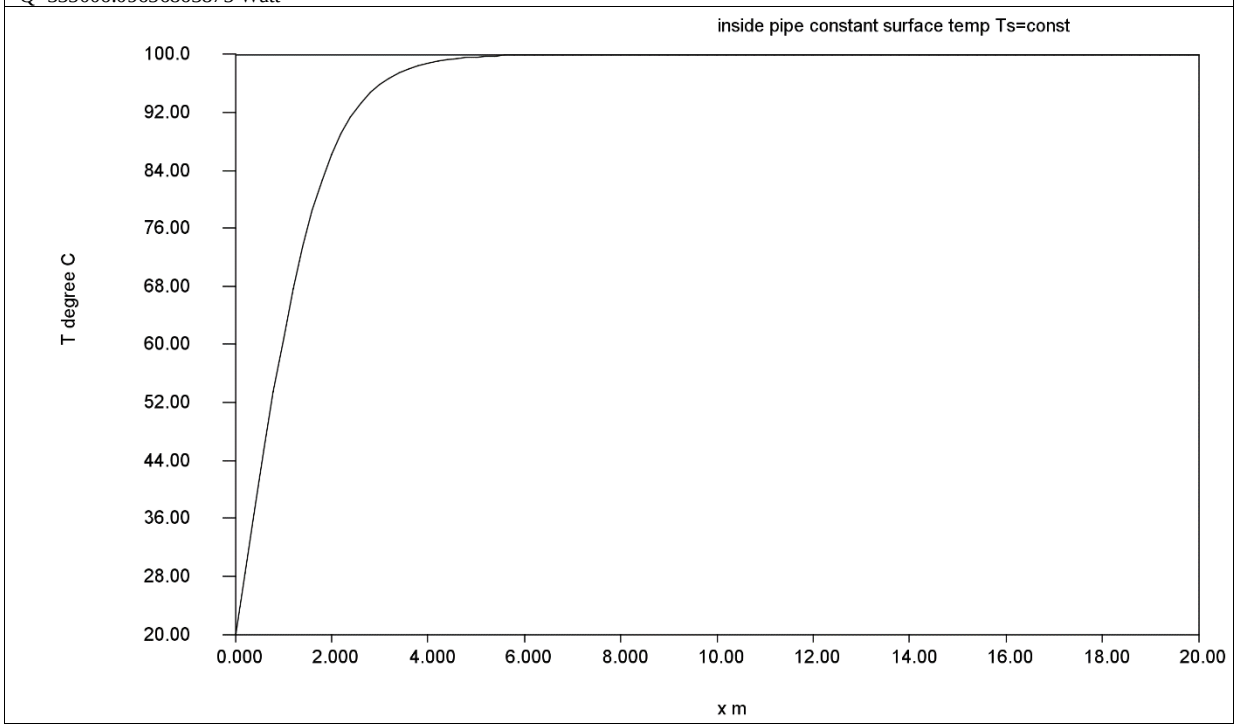
```
import java.io.*;
public class HT_inside_pipe_FD
{
 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 public static double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
   double Nu=0;
   double x=Re/1000;
   double f=0;
   if(Re<2300) Nu=3.66;
   else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
   else {f=f_Goudar(Re,eod);
   Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
   }
 return Nu;
   }
public static void main(String arg[]) throws IOException
```

```
{       double Ti=20+273.15;  // degree K
        double Ts=100+273.15; //degree K;
        double T=Ti;
                double P=300.0;         // kPa
                double U=3.0;           // velocity m/s
                double e=0.16e-3;       //surface rougness m
                double D=50e-3;         // diameter m
                double eod=e/D;
                double L=20; //m
                int N=100;
                double dx=L/N;
                double x=0;
                double TT[]=new double[N+1];
                double xx[]=new double[N+1];
                water w=new water();
                double ro,Cp,k,mu,Pr,Re,f,Nu,h,Q=0,Q1=0;
                double a[]=w.property_TP(Ti,P);
                ro=a[2];
                double A1=Math.PI*D*dx;
                double A=Math.PI*D*D/4.0;
                double m=ro*U*A;
                for(int i=0;i<=N;i++)
                { a=w.property_TP(T,P);
                  TT[i]=T-273.15;
                  xx[i]=i*dx;
         ro=a[2];
                   Cp=a[5];
                   k=a[6];
                   mu=a[7];
                   Pr=a[9];
                   U=m/(ro*A);
                   Re=ro*U*D/mu;
         f=f_Goudar(Re,eod);
         Nu=Nu(Re,eod,Pr);
         h=Nu*k/D;
         Q1=h*A1*(Ts-T);
         Q+=Q1;
         T=T+Q1/(Cp*1e3);
         System.out.println("i="+i+" x = "+xx[i]+" U = "+U+" m/s Nu = "+Nu+" h="+h+" T = "+(T-273.15)+" degree C");
        }
        Plot p=new Plot(xx,TT);
        p.setPlabel("inside pipe constant surface temp Ts=const ");
        p.setXlabel("x m");
        p.setYlabel("T degree C");
        p.plot();
        System.out.println("Q="+Q+" Watt");
}
//create_data();
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-
0933\jre\bin\javaw.exe" HT_inside_pipe_FD
i=0 x = 0.0 U = 3.0 m/s Nu = 1170.9320219868757 h=14042.65339957574 T = 28.4348839068208 degree C
i=1 x = 0.2 U = 3.006309010993253 m/s Nu = 1298.0373316343919 h=15909.109859486894 T = 36.991943238961426 degree C
i=2 x = 0.4 U = 3.0147089478067 m/s Nu = 1422.814735647276 h=17779.722482175905 T = 45.41520983838359 degree C
i=3 x = 0.6000000000000001 U = 3.0247278776923148 m/s Nu = 1541.6425090238117 h=19589.97363819338 T = 53.455127372300296
degree C
i=4 x = 0.8 U = 3.0357718762125216 m/s Nu = 1651.4832320488854 h=21282.447420799017 T = 60.90022597239738 degree C
i=5 x = 1.0 U = 3.047205809244079 m/s Nu = 1750.1954704382715 h=22814.232604284363 T = 67.60017723869345 degree C
i=6 x = 1.2000000000000002 U = 3.0584415068603112 m/s Nu = 1836.658065709462 h=24160.566434192697 T = 73.47481221510918
degree C
i=7 x = 1.4000000000000001 U = 3.069006981707657 m/s Nu = 1910.6990038971937 h=25314.261786639294 T = 78.50934634670358
degree C
i=8 x = 1.6 U = 3.0785810420530977 m/s Nu = 1972.8899861262312 h=26282.12005636172 T = 82.74039308399233 degree C
i=9 x = 1.8 U = 3.0869930482197185 m/s Nu = 2024.2916909267637 h=27080.165019497163 T = 86.23858749806982 degree C
i=10 x = 2.0 U = 3.094198453202458 m/s Nu = 2066.217305820006 h=27729.18066077875 T = 89.09237759576058 degree C
i=11 x = 2.2 U = 3.1002437480020704 m/s Nu = 2100.0492229860715 h=28251.303620105602 T = 91.39537802846934 degree C
i=12 x = 2.4000000000000004 U = 3.105231619654837 m/s Nu = 2127.1163066936447 h=28667.80771988594 T = 93.23784478503171
degree C
i=13 x = 2.6 U = 3.1092923383160103 m/s Nu = 2148.623605351318 h=28997.88858783107 T = 94.70175695590058 degree C
i=14 x = 2.8000000000000003 U = 3.1125632813161417 m/s Nu = 2165.6208928253222 h=29258.15772422188 T = 95.85858917386776
degree C
i=15 x = 3.0 U = 3.1151760033438682 m/s Nu = 2178.9967448184484 h=29462.580023075447 T = 96.76886164631082 degree C
i=16 x = 3.2 U = 3.117249201924596 m/s Nu = 2189.4875630509787 h=29622.65558306972 T = 97.48273353444955 degree C
i=17 x = 3.4000000000000004 U = 3.1188857752728727 m/s Nu = 2197.694081265818 h=29747.71372534775 T = 98.04111982277186
degree C
i=18 x = 3.6 U = 3.1201724460710514 m/s Nu = 2204.1005572916188 h=29845.239505815993 T = 98.47699755640372 degree C
```

i=19 x = 3.8000000000000003 U = 3.1211808241433143 m/s Nu = 2209.0938370921385 h=29921.188996843466 T = 98.81670565798674 degree C
i=20 x = 4.0 U = 3.1219691519426855 m/s Nu = 2212.9808180763407 h=29980.272339811316 T = 99.0811360931391 degree C
i=21 x = 4.2 U = 3.1225842656517107 m/s Nu = 2216.003682409879 h=30026.19695519694 T = 99.28677280828703 degree C
i=22 x = 4.4 U = 3.1230635074505226 m/s Nu = 2218.352764476373 h=30061.87062006084 T = 99.44656903789053 degree C
i=23 x = 4.6000000000000005 U = 3.123436456422184 m/s Nu = 2220.177180657332 h=30089.56773847028 T = 99.57067177986727 degree C
i=24 x = 4.800000000000001 U = 3.1237264258910793 m/s Nu = 2221.5934718967687 h=30111.063594751402 T = 99.66701062261598 degree C
i=25 x = 5.0 U = 3.1239517205668896 m/s Nu = 2222.692549258867 h=30127.741666409387 T = 99.74177080340633 degree C
i=26 x = 5.2 U = 3.124126670392869 m/s Nu = 2223.5452276805845 h=30140.678766978835 T = 99.79977000784652 degree C
i=27 x = 5.4 U = 3.1242624682622293 m/s Nu = 2224.2066063983657 h=30150.712224995914 T = 99.8447565714709 degree C
i=28 x = 5.6000000000000005 U = 3.1243678415040006 m/s Nu = 2224.7195194536225 h=30158.49266810322 T = 99.87964433340454 degree C
i=29 x = 5.800000000000001 U = 3.124449585819044 m/s Nu = 2225.1172435628728 h=30164.525365552137 T = 99.90669691672787 degree C
i=30 x = 6.0 U = 3.124512987292497 m/s Nu = 2225.425617131679 h=30169.202531119292 T = 99.92767191829455 degree C
i=31 x = 6.2 U = 3.1245621544281663 m/s Nu = 2225.664694841743 h=30172.828516376907 T = 99.94393348593854 degree C
i=32 x = 6.4 U = 3.1246002784855818 m/s Nu = 2225.850037428729 h=30175.639431271586 T = 99.95654006791739 degree C
i=33 x = 6.6000000000000005 U = 3.1246298370618897 m/s Nu = 2225.9937157837803 h=30177.81840803413 T = 99.9663127226201 degree C
i=34 x = 6.800000000000001 U = 3.124652752968815 m/s Nu = 2226.1050918731917 h=30179.50746583323 T = 99.97388824270502 degree C
i=35 x = 7.0 U = 3.1246705180282532 m/s Nu = 2226.1914256143555 h=30180.816727210404 T = 99.97976043802697 degree C
i=36 x = 7.2 U = 3.124684289416963 m/s Nu = 2226.258346216199 h=30181.83157380666 T = 99.98431219784487 degree C
i=37 x = 7.4 U = 3.1246949645779982 m/s Nu = 2226.3102180846745 h=30182.61819998485 T = 99.9878403804646 degree C
i=38 x = 7.6000000000000005 U = 3.1247032394253687 m/s Nu = 2226.3504247770343 h=30183.22792175829 T = 99.99057512807309 degree C
i=39 x = 7.800000000000001 U = 3.1247096535428924 m/s Nu = 2226.3815892984635 h=30183.70051921533 T = 99.99269485133016 degree C
i=40 x = 8.0 U = 3.124714625268371 m/s Nu = 2226.4057449754564 h=30184.066828765433 T = 99.99433785207344 degree C
i=41 x = 8.200000000000001 U = 3.1247184789175844 m/s Nu = 2226.424467972023 h=30184.350753298266 T = 99.99561133692379 degree C
i=42 x = 8.4 U = 3.1247214659036535 m/s Nu = 2226.438980045919 h=30184.570820797024 T = 99.99659840660462 degree C
i=43 x = 8.6 U = 3.1247237811174284 m/s Nu = 2226.4502282202266 h=30184.74139273489 T = 99.99736347506064 degree C
i=44 x = 8.8 U = 3.124725575630318 m/s Nu = 2226.458946551774 h=30184.873600889707 T = 99.99795647081311 degree C
i=45 x = 9.0 U = 3.124726966544049 m/s Nu = 2226.465704016426 h=30184.976073574926 T = 99.99841609401165 degree C
i=46 x = 9.200000000000001 U = 3.124728044627441 m/s Nu = 2226.470941630016 h=30185.055498603335 T = 99.99877234130975 degree C
i=47 x = 9.4 U = 3.1247288802369817 m/s Nu = 2226.4750012236464 h=30185.11705967462 T = 99.99904846308698 degree C
i=48 x = 9.600000000000001 U = 3.1247295279066445 m/s Nu = 2226.4781477489682 h=30185.164774637582 T = 99.99926248059779 degree C
i=49 x = 9.8 U = 3.1247300299058627 m/s Nu = 2226.48058656791 h=30185.20175768794 T = 99.99942836197107 degree C
i=50 x = 10.0 U = 3.124730418997601 m/s Nu = 2226.4824768543854 h=30185.230422602206 T = 99.99955693375682 degree C
i=51 x = 10.200000000000001 U = 3.12473072057623 m/s Nu = 2226.4839419821124 h=30185.2526402662 T = 99.99965658747988 degree C
i=52 x = 10.4 U = 3.124730954324717 m/s Nu = 2226.485077576387 h=30185.269860776287 T = 99.9997338272949 degree C
i=53 x = 10.600000000000001 U = 3.124731135499108 m/s Nu = 2226.4859577549273 h=30185.2832080794 T = 99.99979369447482 degree C
i=54 x = 10.8 U = 3.124731275924167 m/s Nu = 2226.486639965206 h=30185.293553328545 T = 99.99984009643026 degree C
i=55 x = 11.0 U = 3.1247313847651 08 m/s Nu = 2226.487168733827 h=30185.301571739805 T = 99.99987606173067 degree C
i=56 x = 11.200000000000001 U = 3.124731469125741 m/s Nu = 2226.487578572586 h=30185.307786661342 T = 99.99990393776847 degree C
i=57 x = 11.4 U = 3.12473153451212 m/s Nu = 2226.48789623098 h=30185.31260373108 T = 99.99992554396658 degree C
i=58 x = 11.600000000000001 U = 3.124731585191895 m/s Nu = 2226.488142442071 h=30185.316337351764 T = 99.99994229052737 degree C
i=59 x = 11.8 U = 3.1247316244728514 m/s Nu = 2226.488333275684 h=30185.319231211182 T = 99.99995527047315 degree C
i=60 x = 12.0 U = 3.1247316549187865 m/s Nu = 2226.4884811872384 h=30185.32147418732 T = 99.99996533098573 degree C
i=61 x = 12.200000000000001 U = 3.124731678516864 m/s Nu = 2226.4885958307054 h=30185.323212675925 T = 99.99997312870005 degree C
i=62 x = 12.4 U = 3.1247316968072902 m/s Nu = 2226.4886846887016 h=30185.324560145757 T = 99.99997917256184 degree C
i=63 x = 12.600000000000001 U = 3.124731710983857 m/s Nu = 2226.488753560861 h=30185.32560454424 T = 99.99998385704532 degree C
i=64 x = 12.8 U = 3.124731721971848 m/s Nu = 2226.4888069423773 h=30185.326414037863 T = 99.99998748790028 degree C
i=65 x = 13.0 U = 3.124731730488434 m/s Nu = 2226.4888483173863 h=30185.327041461114 T = 99.9999903021075 degree C
i=66 x = 13.200000000000001 U = 3.1247317370894776 m/s Nu = 2226.488880386372 h=30185.327527765006 T = 99.99999248334649 degree C
i=67 x = 13.4 U = 3.1247317422058223 m/s Nu = 2226.488905242439 h=30185.32790469002 T = 99.99999417398368 degree C
i=68 x = 13.600000000000001 U = 3.1247317461714053 m/s Nu = 2226.488924507904 h=30185.328196837407 T = 99.99999548436472 degree C
i=69 x = 13.8 U = 3.124731749245055 m/s Nu = 2226.488939440206 h=30185.328423275372 T = 99.99999650001632 degree C
i=70 x = 14.0 U = 3.1247317516273814 m/s Nu = 2226.488951013952 h=30185.32859878319 T = 99.99999728722872 degree C
i=71 x = 14.200000000000001 U = 3.124731753473877 m/s Nu = 2226.4889599845474 h=30185.32873481605 T = 99.99999789738217 degree C
i=72 x = 14.4 U = 3.1247317549050617 m/s Nu = 2226.4889669374766 h=30185.328840252398 T = 99.9999983703006 degree C
i=73 x = 14.600000000000001 U = 3.1247317560143464 m/s Nu = 2226.4889723265665 h=30185.32892197419 T = 99.99999873685078 degree C

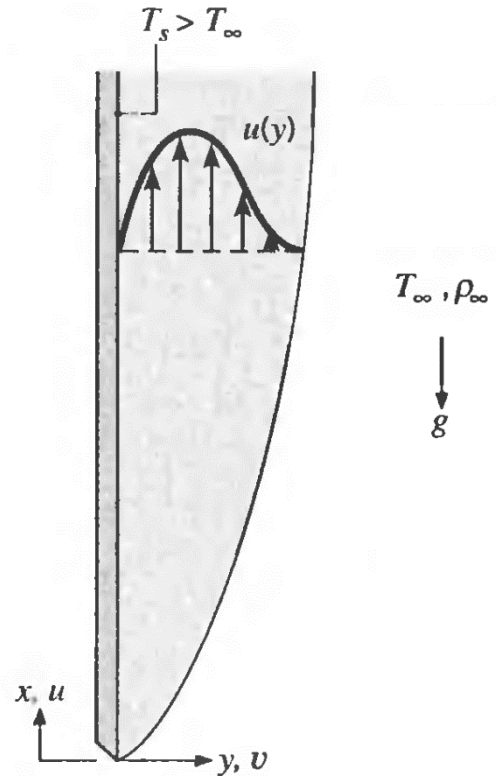
i=74 x = 14.8 U = 3.124731756874132 m/s Nu = 2226.4889765035487 h=30185.32898531522 T = 99.99999902095686 degree C
i=75 x = 15.0 U = 3.124731757540535 m/s Nu = 2226.488979741049 h=30185.329034409682 T = 99.99999924116213 degree C
i=76 x = 15.200000000000001 U = 3.124731758057052 m/s Nu = 2226.4889822503774 h=30185.329072461867 T = 99.9999994118391 degree C
i=77 x = 15.4 U = 3.124731758457395 m/s Nu = 2226.488984195307 h=30185.329101955376 T = 99.9999995441276 degree C
i=78 x = 15.600000000000001 U = 3.1247317587676915 m/s Nu = 2226.4889857027824 h=30185.32912481522 T = 99.99999964666193 degree C
i=79 x = 15.8 U = 3.124731759008198 m/s Nu = 2226.4889868712 h=30185.329142533446 T = 99.99999972613432 degree C
i=80 x = 16.0 U = 3.12473175919461 m/s Nu = 2226.4889877768223 h=30185.329156266573 T = 99.9999997877319 degree C
i=81 x = 16.2 U = 3.1247317593390944 m/s Nu = 2226.4889884787517 h=30185.329166910844 T = 99.99999983547502 degree C
i=82 x = 16.400000000000002 U = 3.1247317594510813 m/s Nu = 2226.4889890228023 h=30185.329175160994 T = 99.9999998724798 degree C
i=83 x = 16.6 U = 3.12473175953788 m/s Nu = 2226.4889894444827 h=30185.3291815555 T = 99.99999990116152 degree C
i=84 x = 16.8 U = 3.1247317596051567 m/s Nu = 2226.488989771322 h=30185.32918651178 T = 99.99999992339218 degree C
i=85 x = 17.0 U = 3.1247317596573 m/s Nu = 2226.4889900246512 h=30185.329190353354 T = 99.99999994062273 degree C
i=86 x = 17.2 U = 3.1247317596977173 m/s Nu = 2226.4889902210043 h=30185.329193330883 T = 99.9999999539778 degree C
i=87 x = 17.400000000000002 U = 3.124731759729043 m/s Nu = 2226.4889903731873 h=30185.329195638642 T = 99.99999996432905 degree C
i=88 x = 17.6 U = 3.124731759753323 m/s Nu = 2226.488990491142 h=30185.32919742735 T = 99.9999999723521 degree C
i=89 x = 17.8 U = 3.1247317597721427 m/s Nu = 2226.4889905825708 h=30185.329198813786 T = 99.99999997857066 degree C
i=90 x = 18.0 U = 3.124731759786728 m/s Nu = 2226.488990653432 h=30185.329199888372 T = 99.99999998339052 degree C
i=91 x = 18.2 U = 3.124731759798034 m/s Nu = 2226.488990708358 h=30185.32920072127 T = 99.99999998712633 degree C
i=92 x = 18.400000000000002 U = 3.124731759806796 m/s Nu = 2226.488990750929 h=30185.329201366832 T = 99.99999999002188 degree C
i=93 x = 18.6 U = 3.124731759813588 m/s Nu = 2226.4889907839224 h=30185.329201867164 T = 99.99999999226617 degree C
i=94 x = 18.8 U = 3.1247317598188524 m/s Nu = 2226.4889908094974 h=30185.329202254998 T = 99.99999999400563 degree C
i=95 x = 19.0 U = 3.124731759822932 m/s Nu = 2226.4889908293158 h=30185.32920255554 T = 99.9999999953539 degree C
i=96 x = 19.200000000000003 U = 3.1247317598260955 m/s Nu = 2226.488990844684 h=30185.32920278857 T = 99.99999999639891 degree C
i=97 x = 19.400000000000002 U = 3.124731759828546 m/s Nu = 2226.4889908565883 h=30185.329202969104 T = 99.99999999720887 degree C
i=98 x = 19.6 U = 3.124731759830447 m/s Nu = 2226.48899086582 h=30185.329203109075 T = 99.99999999783665 degree C
i=99 x = 19.8 U = 3.1247317598319193 m/s Nu = 2226.4889908729733 h=30185.32920321754 T = 99.99999999832323 degree C
i=100 x = 20.0 U = 3.1247317598330597 m/s Nu = 2226.4889908785144 h=30185.329203301597 T = 99.99999999870039 degree C
Q=335006.09656863875 Watt



14.0 NATURAL CONVECTION

14.1 NATURAL CONVECTION SIMILARITY SOLUTION

The equations governing external free convection are essentially the same as those for forced convection. A boundary layer for free convection of a vertical flow is shown in the figure. Velocity in x direction is assumed to be zero at the wall and again is zero out of boundary layer.



Conservation of mass:

$$\frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} = 0$$

Conservation of momentum

$$\rho u \frac{\partial u}{\partial x} + \rho v \frac{\partial u}{\partial y} = \frac{\partial}{\partial y}\left(\mu \frac{\partial u}{\partial y}\right) - \frac{\partial P}{\partial x} - \rho g$$

Conservation of energy

$$\rho u C_p \frac{\partial T}{\partial x} + \rho v C_p \frac{\partial T}{\partial y} = \frac{\partial}{\partial y}\left(k \frac{\partial T}{\partial y}\right)$$

The boundary conditions are

u=0, v=0, $T=T_s(x)$ at y=0

$u = 0$, $T = T_\infty$at x=0

$u = 0$, $T = T_\infty$as $y \to \infty$

Outside of the boundary layer, the only hydrostatic terms of the boundary layer will be left so:

$$\frac{\partial P}{\partial x} = -\rho_\infty g$$

if this substituted the previous momentum equation it becomes:

$$\rho u \frac{\partial u}{\partial x} + \rho v \frac{\partial u}{\partial y} = \frac{\partial}{\partial y}\left(\mu \frac{\partial u}{\partial y}\right) + g(\rho_\infty - \rho)$$

Furthermore, if Definition of the thermodynamic property of thermal expansion coefficient is taken into consideration:

$$\beta = -\frac{1}{\rho}\left(\frac{\partial \rho}{\partial T}\right)_P \cong \frac{1}{\rho}\left(\frac{\rho_\infty - \rho}{T_\infty - T}\right)$$
$$(\rho_\infty - \rho) = \rho\beta(T_\infty - T)$$

Equation becomes:

$$\rho u\frac{\partial u}{\partial x} + \rho v\frac{\partial u}{\partial y} = \frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right) + \rho g\beta(T - T_\infty)$$
$$\frac{\rho}{\rho}u\frac{\partial u}{\partial x} + \frac{\rho}{\rho}v\frac{\partial u}{\partial y} = \frac{\partial}{\partial y}\left(\frac{\mu}{\rho}\frac{\partial u}{\partial y}\right) + \frac{\rho}{\rho}g\beta(T - T_\infty)$$
$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = \frac{\partial}{\partial y}\left(\frac{\mu}{\rho}\frac{\partial u}{\partial y}\right) + g\beta(T - T_\infty)$$

$$\upsilon = \frac{\mu}{\rho}$$
$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = \upsilon\frac{\partial^2 u}{\partial y^2} + g\beta(T - T_\infty)$$

Boundary layer equations can be converted by using stream function concept. Let $u = \frac{\partial \varphi}{\partial y}$ $v = -\frac{\partial \varphi}{\partial x}$ where $\varphi(x,y)$ is stream function. If such a function existed (for constant ρ) , then conservation of mass is automatically satisfied.

$$\frac{\partial(u)}{\partial x} + \frac{\partial(v)}{\partial y} = 0$$
$$\frac{\partial^2\varphi(x,y)}{\partial x\partial y} - \frac{\partial^2\varphi(x,y)}{\partial y\partial x} = 0$$
$$\theta(x,y) = \frac{T(x,y) - T_\infty}{T_s - T_\infty}$$
$$(T(x,y) - T_\infty) = (T_s - T_\infty)\theta(x,y)$$

Then equations become:

(Momentum equation)

$$\frac{\partial\varphi(x,y)}{\partial y}\frac{\partial^2\varphi(x,y)}{\partial x\partial y} - \frac{\partial\varphi(x,y)}{\partial x}\frac{\partial^2\varphi(x,y)}{\partial y^2} = \nu\frac{\partial^3\varphi(x,y)}{\partial y^3} + g\beta(T_s - T_\infty)\theta(x,y)$$

(Energy equation)

$$\frac{\partial\varphi(x,y)}{\partial y}\frac{\partial\theta(x,y)}{\partial x} - \frac{\partial\varphi(x,y)}{\partial x}\frac{\partial\theta(x,y)}{\partial y} = \alpha\frac{\partial^2\varphi(x,y)}{\partial y^2} + \theta(x,y)\frac{\partial\varphi(x,y)}{\partial y}\frac{\partial T_s}{\partial x}$$

A similarity transformation will be applied to this equations in the form of

$$\eta(x,y) = yH(x)$$

Where

$$H(x) = \frac{1}{x}\left(\frac{1}{4}Gr_x\right)^{1/4}$$

Where $\mathrm{Gr}_x = \frac{g\beta(T_s(x) - T_\infty)x^3}{\nu^2}$ is Grashoff number, Then $\eta(x,y)$ becomes:

$$\eta(x,y) = yH(x) = \frac{y}{x}\left(\frac{1}{4}Gr_x\right)^{1/4}$$

And a separation of variable can be applied to the equations in the form of

$$\varphi(x,\eta) = \nu F(\eta)G(x)$$

Where

$$G(x) = \left(\frac{1}{4} Gr_x\right)^{1/4}$$

For the case of $T_s(x)$=constant, the equation becomes:

$$\frac{d^3F(\eta)}{d\eta^3} + 3F(\eta)\frac{dF(\eta)}{d\eta} - 2\left(\frac{dF(\eta)}{d\eta}\right)^2 + \theta(\eta) = 0$$

$$\frac{d^3\theta(\eta)}{d\eta^3} + 3PrF(\eta)\frac{d\theta(\eta)}{d\eta} = 0$$

The boundary conditions on these equations

u=0 at y=0	$F'(0) = 0$
v=0 ay y=0	$F(0) = 0$
u=0 $y \to \infty$	$F'(\infty) = 0$
T=T_s y=0	$\theta(0) = 1$
$T \to T_\infty \quad y \to \infty$	$\theta(\infty) = 0$

This set of differential equation can only be solved by using numerical methods. Furthermore, it should be solved for different Prandtl number. In order to obtain thermal conductivity coefficient or Nussel number from the numerical solutions, following convertions can be carried out.
Local convective heat transfer coefficient:

$$q_s^{"} = h_x(T_s - T_\infty)$$

$$h_x = \frac{q_s^{"}}{T_s - T_\infty}$$

Form conduction heat transfer:

$$q_s^{"} = -k\frac{dT}{dy}\bigg|_{y=0}$$

$$q_s^{"} = -k\frac{dT}{dy}\bigg|_{y=0} = -k(T_s - T_\infty)\left[\frac{d\theta(\eta)}{d\eta}\frac{d\eta}{dy}\right]_{\eta=0} = h_x(T_s - T_\infty)$$

$$q_s^{"} = -k(T_s - T_\infty)\frac{d\theta(0)}{d\eta}H(x)$$

Then the local Nusselt number will be defined as:

$$Nu_x = \frac{xh_x}{k} = -\frac{d\theta(0)}{d\eta}xH(x) = -\frac{1}{\sqrt{2}}\frac{d\theta(0)}{d\eta}Gr_x^{1/4}$$

So we will seek solutions for $\theta'(0)$for different Pr numbers. Since relation might not be linear curve fitting of the obtained solutions should be applied.
As it is seen from boundary conditions, two given conditions are not initial conditions. One way of solving boundary conditions is using non-linear system of equation solving methods. In order to solve differential equation set 6th order Runge-Kutta Method is used. In order to solve boundary value nonlinear system of equation Nelder & Mead optimization method is used. Now that basic mathematical calculation methods are established, The equation can be solved by using differential equation boundary value solving method.

```
 import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

class f2 extends f_xj
{//adaptation function
```

```
fi_xi ff1;
public f2(fi_xi ffi) {ff1=ffi;}

public double func(double x[])
{
double ff=0.0;
double fa[]=ff1.func(x);
for(int i=0;i<fa.length;i++)  ff+=fa[i]*fa[i];
return ff;
}
}

class fa extends fi_xi
{ // Nonlinear system of equation
double y[];
double w;
fm2 b3;
double a[][];
public fa(double Pr,double yi[],double wi)
{ b3=new fm2(Pr);
  y=new double[yi.length];
  for(int i=0;i<y.length;i++) y[i]=yi[i];w=wi;
}

public double[] func(double x[])
{
// shooting method
y[2]=x[0];
y[4]=x[1];
a=RK6(b3,0.0,w,y,10000);
double ff[]=new double[2];
ff[0]=a[2][10000]-0.0;
ff[1]=a[4][10000]-0.0;
return ff;
}
 //differenetial equation solution
public static double[][] RK6(f_xi fp,double x0,double xn,double f0[],int N)
{
//6th order Runge Kutta Method
//fp  : given set of derivative functions dfj/dxi(fj,x)
// xo : initial value of the independent variable
// xn : final value of the independent variable
// f0 : initial value of the dependent variable
// N  : number of dependent variable to be calculated
// fi : dependent variable
double h=(xn-x0)/N;
int M=f0.length;
double fi[][];
fi=new double[M][N+1];
double xi[]=new double[M+1];
double k[]=new double[6];
int i,j;
double x;
for(j=0;j<M;j++)
  {
  fi[j][0]=f0[j];
  xi[j+1]=f0[j];
  }
for(x=x0,i=0;i<N;x+=h,i++)
  {
  for(j=1;j<=M;j++)
    {
    xi[0]=x;
    xi[j]=fi[j-1][i];
    k[0]=h*fp.func(xi,j-1);
    xi[0]=x+h/2.0;
    xi[j]=fi[j-1][i]+k[0]/2;
    k[1]=h*fp.func(xi,j-1);
    xi[0]=x+h/2.0;
    xi[j]=fi[j-1][i]+k[0]/4.0+k[1]/4.0;
    k[2]=h*fp.func(xi,j-1);
    xi[0]=x+h;
    xi[j]=fi[j-1][i]-k[1]+2.0*k[2];
    k[3]=h*fp.func(xi,j-1);
    xi[0]=x+2.0/3.0*h;
    xi[j]=fi[j-1][i]+7.0/27.0*k[0]+10.0/27.0*k[1]+1.0/27.0*k[3];
    k[4]=h*fp.func(xi,j-1);
```

```
    xi[0]=x+1.0/5.0*h;
    xi[j]=fi[j-1][i]+28.0/625.0*k[0]-1.0/5.0*k[1]+546.0/625.0*k[2]+54.0/625.0*k[3]-378/625.0*k[4];
    k[5]=h*fp.func(xi,j-1);
    fi[j-1][i+1]=fi[j-1][i]+k[0]/24.0+5.0*k[3]/48.0+27.0*k[4]/56.0+125.0*k[5]/336.0;
    xi[j]=fi[j-1][i];
    }
  }
double a[][]=new double[M+1][N+1];
for(x=x0,i=0;i<=N;x+=h,i++)
{
  a[0][i]=x;
  for(j=1;j<=M;j++)
  {
  a[j][i]=fi[j-1][i];
  }
}
return a;
}
}

//Blassius flat plate differential equation
class fm2 extends f_xi
{ double Pr;
  fm2(double Pri) {Pr=Pri;}
// multivariable function
 double func(double x[],int x_ref)
 {
   //Natural convection similarity solution
   double a=0;
   if(x_ref==4) a= -3.0*Pr*x[1]*x[5];
   if(x_ref==3) a= x[5];
   if(x_ref==2) a= -3.0*x[1]*x[3]+2.0*x[2]*x[2]-x[4];
   if(x_ref==1) a= x[3];
   if(x_ref==0) a= x[2];
   return a;
 }
}
class fb extends f_xr
{
                double func(double Pr,int i)
                {
     double xx=0.0;
      if(i==0) xx=Math.pow(Pr,0.25);
      else if(i==1) xx=Math.sqrt(Pr);
      else if(i==2) xx=Math.pow(Pr,0.75);
      else if(i==3) xx=Pr;
      else if(i==4) xx=Math.pow(Pr,1.25);
      else if(i==5) xx=Math.pow(Pr,1.5);
      else if(i==6) xx=Math.pow(Pr,1.75);
   return xx;
                }
}

public class natural_convection_similarity
{
                 public static double[] inputdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }
public static String toString(double x[])
{ String s="";
int n=x.length;
  for(int k=0;k<n;k++)
  {s+=x[k]+" ";}
  return s;
}
```

```
public static double[] nelder(f_xj fnelder,double a[],double da[],int maxiteration,double tolerance,int printlist)
{
              int i,j;
double x[][]=new double[a.length+1][a.length];
double p[][]=new double[a.length+1][a.length+1];
for(i=0;i<x.length;i++)
  {for(j=0;j<x[0].length;j++)
    {if(i==j){x[i][j]=a[j]+da[j];p[i][j]=x[i][j];}
     else    {x[i][j]=a[j];p[i][j]=x[i][j]; }
    }
   p[i][j] = fnelder.func(p[i]);
  }


  // Inlet variable definitions
  // fnelder : abstract multivariable function f(x)
  // x : independent variable set of n+1 simplex elements
  // maxiteration : maximum iteration number
  // tolerance :
      int NDIMS = x.length-1;
      int NPTS = x.length;
      int FUNC = NDIMS;
      int ncalls = 0;
        ////// construct the starting simplex //////////////////
        //double p[][]=new double[NPTS][NPTS]; // [row][col] = [whichvx][coord,FUNC]
        double z[]=new double[NDIMS];
        double best = 1E99;
        /////////////// calculate the first function values for the simplex /////////////////

        int iter=0;

        for (iter=1; iter<maxiteration; iter++)
        {
                  ///////////  define lo, nhi, hi (low high next_to_high //////////////
          int  ilo=0, ihi=0, inhi = -1; // -1 means missing
          double flo = p[0][FUNC];
          double fhi = flo;
          double pavg,sterr;
          for (i=1; i<NPTS; i++)
          {
            if (p[i][FUNC] < flo)
              {flo=p[i][FUNC]; ilo=i;}
            if (p[i][FUNC] > fhi)
              {fhi=p[i][FUNC]; ihi=i;}
          }
          double fnhi = flo;
          inhi = ilo;
          for (i=0; i<NPTS; i++)
           if ((i != ihi) && (p[i][FUNC] > fnhi))
             {fnhi=p[i][FUNC]; inhi=i;}
          ////////// exit criteria //////////////
          if ((iter % 4*NDIMS) == 0)
          {
            // calculate the avarage (including maximum value)
            pavg=0;
            for(i=0;i<NPTS;i++)
              pavg+=p[i][FUNC];
            pavg/=NPTS;
            double tot=0;
            if(printlist!=0)
            {  System.out.print(iter);
               for (j=0; j<=NDIMS; j++)
                  { System.out.print(p[ilo][j]+" ");}
               System.out.println("");
            }
            for(i=0;i<NPTS;i++)
            { tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg);}
            sterr=Math.sqrt(tot/NPTS);
            //if(sterr < tolerance)
              { for (j=0; j<NDIMS; j++)
                   { z[j]=p[ilo][j];}
                //break;
              }
            best = p[ilo][FUNC];
          }

          ///// calculate avarage without maximum value //////
```

```
double ave[] = new double[NDIMS];
for (j=0; j<NDIMS; j++)
  ave[j] = 0;
for (i=0; i<NPTS; i++)
  if (i != ihi)
    for (j=0; j<NDIMS; j++)
      ave[j] += p[i][j];
for (j=0; j<NDIMS; j++)
   ave[j] /= (NPTS-1);


//////// reflect ////////////////

double r[] = new double[NDIMS];
for (j=0; j<NDIMS; j++)
  r[j] = 2*ave[j] - p[ihi][j];
double fr = fnelder.func(r);

if ((flo <= fr) && (fr < fnhi))  // in zone: accept
{
   for (j=0; j<NDIMS; j++)
     p[ihi][j] = r[j];
   p[ihi][FUNC] = fr;
   continue;
}

if (fr < flo)  //// expand
{
   double e[] = new double[NDIMS];
   for (j=0; j<NDIMS; j++)
     e[j] = 3*ave[j] - 2*p[ihi][j];
   double fe = fnelder.func(e);
   if (fe < fr)
   {
      for (j=0; j<NDIMS; j++)
        p[ihi][j] = e[j];
      p[ihi][FUNC] = fe;
      continue;
   }
   else
   {
      for (j=0; j<NDIMS; j++)
        p[ihi][j] = r[j];
      p[ihi][FUNC] = fr;
      continue;
   }
}

////////////// shrink:

if (fr < fhi)
{
   double c[] = new double[NDIMS];
   for (j=0; j<NDIMS; j++)
     c[j] = 1.5*ave[j] - 0.5*p[ihi][j];
   double fc = fnelder.func(c);
   if (fc <= fr)
   {
      for (j=0; j<NDIMS; j++)
        p[ihi][j] = c[j];
      p[ihi][FUNC] = fc;
      continue;
   }
   else   /////// daralt
   {
      for (i=0; i<NPTS; i++)
        if (i != ilo)
        {
           for (j=0; j<NDIMS; j++)
             p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
           p[i][FUNC] = fnelder.func(p[i]);
        }
      continue;
   }
}
```

```
            if (fr >= fhi)   ///
            {
               double cc[] = new double[NDIMS];
               for (j=0; j<NDIMS; j++)
                 cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
               double fcc = fnelder.func(cc);
               if (fcc < fhi)
               {
                  for (j=0; j<NDIMS; j++)
                    p[ihi][j] = cc[j];
                  p[ihi][FUNC] = fcc;
                  continue;
               }
               else    /////////
               {
                  for (i=0; i<NPTS; i++)
                    if (i != ilo)
                    {
                       for (j=0; j<NDIMS; j++)
                         p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                       p[i][FUNC] = fnelder.func(p[i]);
                    }
               }
            }
         }
      return z;
}
    public static double[] nelder(f_xj fnelder,double a[],double da[],double tolerance)
    {return nelder(fnelder,a,da,500,tolerance,0);}

    public static double[] nelder(f_xj fnelder,double a[],double da[])
    {return nelder(fnelder,a,da,500,1.0e-10,0);}

    public static double[] nelder(f_xj fnelder,double a[])
    {
                    double [] da=new double[a.length];
          for(int i=0;i<a.length;i++) da[i]=0.1*a[i];
          return nelder(fnelder,a,da);
    }
//general least square curve fitting
  public static double[] gausswithpartialpivot(double a[][],double b[])
  { //Gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //partial pivoting
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
              //
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
          b[i]   =b[i]   -carpan*b[k];
              }
  }
```

```
  //backward substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}


public static double[] GeneralLeastSquare(double xi[],double yi[],int n)
{
fb f=new fb();
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  B[i]=0;
                for(j=0;j<np1;j++)
    {
                A[i][j]=0.0;
    for(k=0;k<l;k++) A[i][j]+=f.func(xi[k],i)*f.func(xi[k],j);
    }
    for(k=0;k<l;k++) B[i]+= f.func(xi[k],i)*yi[k];
  }
//System.out.println("A = \n"+Matrix.toString(A));
//System.out.println("B = \n"+Matrix.toStringT(B));
X=gausswithpartialpivot(A,B);
//X=B/A;
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
Text.printT(X);
return X;
}

public static double funcGeneralLeastSquare(double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
fb f=new fb();
int n=e.length;
double ff=0;
if(n!=0.0)
  {
  for(int i=n-1;i>=0;i--)
   {ff+=e[i]*f.func(x,i);}
  }
return ff;
}

public static double hata(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcGeneralLeastSquare(e,x[k]))*(y[k]-funcGeneralLeastSquare(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

  public static double[][]  funcGeneralLeastSquare(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
```

```
 {
              //aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
              // ara değer sayisi 2 dir
               int n=xi.length;
              int nn=(n-1)*(aradegersayisi+1)+1;
              double z[][]=new double[2][nn];
   double E[]=GeneralLeastSquare(xi,yi,polinomkatsayisi);
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
     {z[0][k]=xi[i];z[1][k]=funcGeneralLeastSquare(E,z[0][k]);k++;
     for(int j=0;j<aradegersayisi;j++)
     {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
     z[0][k]=z[0][k-1]+dx;z[1][k]=funcGeneralLeastSquare(E,z[0][k]);k++;}
     }
     z[0][k]=xi[i];z[1][k]=funcGeneralLeastSquare(E,z[0][k]);
   return z;
 }


public static void main(String arg[]) throws IOException
{ //initial guess for differential equation set
  double y[]={0,0,0.7,1.0,-0.5};
 //String s="leave a space between initial root guess values: xmin0 xmin1.. ";
 //double x[]=inputdata(s);
 double x[]={0.7,-0.5};//inputdata(s);
 fa ff;
 f2 f;
 double r1[];
 //selected Prandtl number values
 double
Pr[]={0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.1,0.2,0.3,0.4,0.5,0.6,0.8,1.0,1.2,1.4,1.6,1.8,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10,11,12,13,14,15};
 //selected values for maximum eta(limit is not taken infinite)
 // corresponding to the Pr numbers
 double
w[]={16.0,15.0,15.0,15.0,15.0,15.0,15.0,15.0,15.0,10.0,10.0,9.0,8.0,8.0,9.0,7.0,7.0,7.0,6.0,5.0,5.0,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5
,4.5};
 //output data for Nusselt's number curve fitting
 double xi[]=new double[Pr.length];
 double yi[]=new double[Pr.length];
 //Initial calculations
 //non linear eqation function
 ff=new fa(Pr[0],y,w[0]);
 //optimisation function
 f=new f2(ff);
 r1=nelder(f,x);
 xi[0]=Pr[0];
 yi[0]=-r1[1];
 Plot pp=new Plot();
 Plot pp1=new Plot();
 for(int i=0;i<xi.length;i++)
 {ff=new fa(Pr[i],y,w[i]);
  f=new f2(ff);
  r1=nelder(f,x);
   xi[i]=Pr[i];
   yi[i]=-r1[1];
  System.out.println("wall surface Pr="+Pr[i]+" eta="+ff.a[0][0]+" f= "+ff.a[1][0]+" f'= "+ff.a[2][0]+" f''= "+ff.a[3][0]+" T = "+ff.a[4][0]+" T' =
"+ff.a[5][0]+" ");
  System.out.println("boundary layer limit Pr= "+Pr[i]+" eta = "+ff.a[0][10000]+" f = "+ff.a[1][10000]+" f' = "+ff.a[2][10000]+" f'' =
"+ff.a[3][10000]+" T = "+ff.a[4][10000]+" T' = "+ff.a[5][10000]+" ");
   String s1[]={""+'\u03B7',"f","f'","f''",""+'\u03B8',""+'\u03B8'+"'"};
   if(i==0) {Plot pp2=new Plot(ff.a[0],ff.a[2]);pp=pp2;
            Plot pp3=new Plot(ff.a[0],ff.a[4]);pp1=pp3;
           }
   else
   { pp.addData(ff.a[0],ff.a[2]);
     pp1.addData(ff.a[0],ff.a[4]);
   }
 }
  double gPr1[]=new double[Pr.length];
  double gPr2[]=new double[Pr.length];
  double egPr1[]=new double[Pr.length];
  double egPr2[]=new double[Pr.length];
  double b[]=GeneralLeastSquare(xi,yi,6);
  for(int i=0;i<Pr.length;i++)
```

```
{gPr1[i]=b[0]*Math.pow(Pr[i],0.25)+b[1]*Math.pow(Pr[i],0.5)+b[2]*Math.pow(Pr[i],0.75)+b[3]*Pr[i]+b[4]*Math.pow(Pr[i],1.25)+b[5]*Math.po
w(Pr[i],1.5)+b[6]*Math.pow(Pr[i],1.75);
   gPr2[i]=0.75*Math.sqrt(Pr[i])/Math.pow((0.609+1.221*Math.sqrt(Pr[i])+1.238*Pr[i]),0.25);
   egPr1[i]=(gPr1[i]-yi[i])/yi[i]*100;
   egPr2[i]=(gPr2[i]-yi[i])/yi[i]*100;
   }
  Text.printT(b);
  double d[][]={xi,yi};
  Text.printT(d,"d");
  System.out.println("f(Pr) ="+b[0]+b[1]+"*Pr^0.25+"+b[2]+"*Pr^0.5+"+b[3]+"*Pr^0.75+"+b[4]+"*Pr +"+b[5]+"*Pr^1.25+"+b[6]+"*Pr^1.5");
 pp.plot();
 pp1.plot();
 Plot pp5=new Plot(xi,yi);
 pp5.addData(xi,gPr1);
 pp5.setColor(1,0,0,255);
 pp5.setPlotType(1,23);
 pp5.addData(xi,gPr2);
  pp5.setColor(2,255,0,0);
 pp5.setPlotType(2,28);
 pp5.plot();
 Plot pp6=new Plot(xi,egPr1);
 pp6.setColor(1,0,0,255);

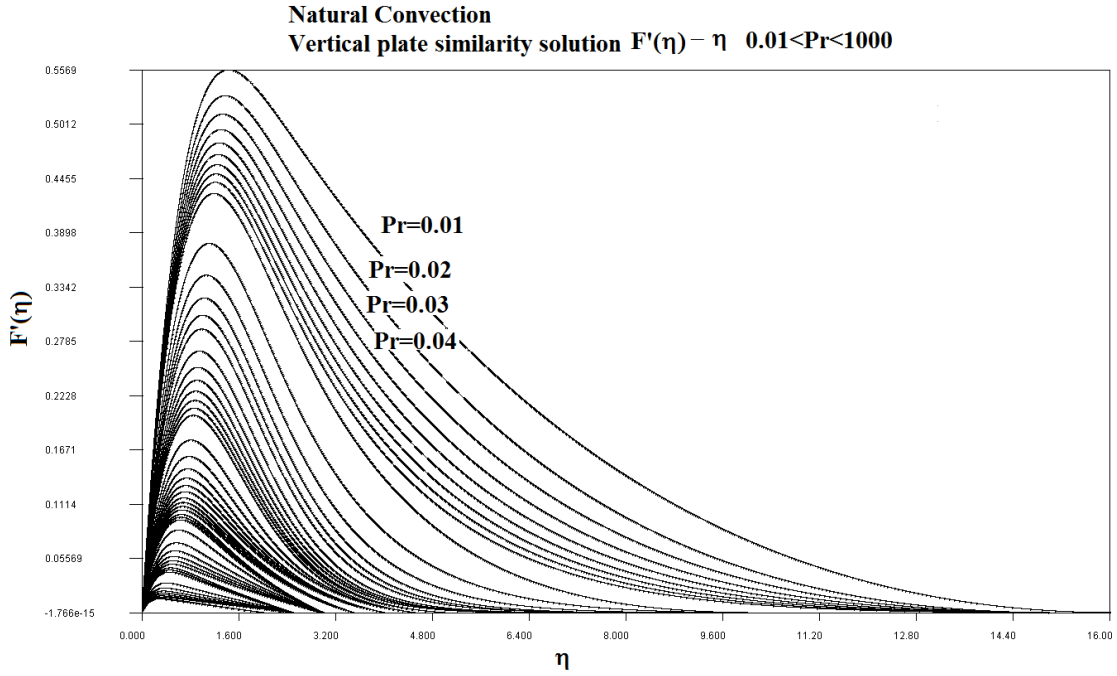
 pp6.addData(xi,egPr2);
  pp6.setColor(2,255,0,0);
 pp6.plot();
}
}
```

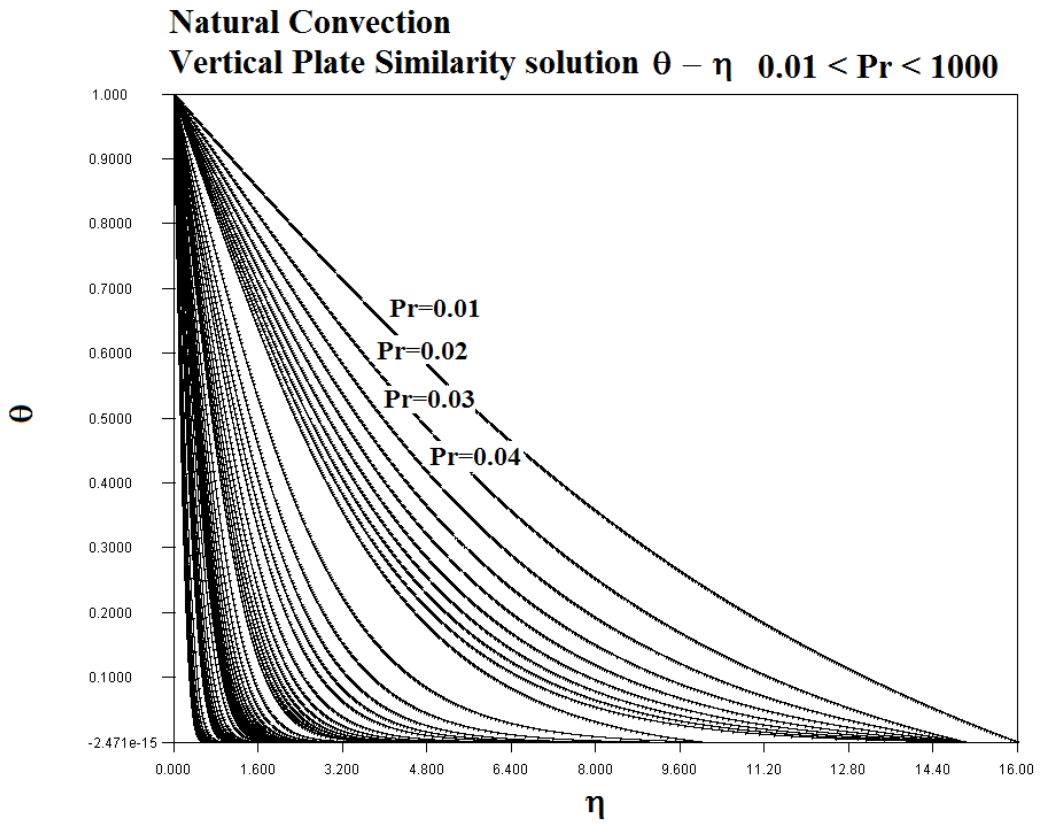
Program investigates a wide range of Prandtl numbers and as a results:

Results for $\left.\frac{\partial\theta(\eta)}{\partial\eta}\right|_{\eta=0}$

Pr	θ(η)
0.01	0.091093502
0.02	0.116834579
0.03	0.136995175
0.04	0.154605985
0.05	0.170229134
0.06	0.184286962
0.07	0.19708927
0.08	0.208865088
0.09	0.219786662
0.1	0.231500963
0.2	0.307792789
0.3	0.362168135
0.4	0.4050892
0.5	0.440866301
0.6	0.471765381
0.8	0.523754747
1	0.56672913
1.2	0.603623472
1.4	0.636067035
1.6	0.664867361
1.8	0.691266588
2	0.715036357
3	0.813788218
4	0.889502603
5	0.950493977
6	1.003583771
7	1.050193207
8	1.09189393
9	1.127210608
10	1.161864748
11	1.193933662

12	1.223822704
13	1.246961238
14	1.273319569
15	1.298263279
20	1.406640617
30	1.572188344
40	1.699540604
50	1.804447867
60	1.894382556
70	1.973529018
80	2.034457087
90	2.098909425
100	2.158133722
200	2.587798708
300	2.874897742
400	3.096602009
500	3.279691561
600	3.436935238
700	3.575508896
800	3.699882055
900	3.796072907
1000	3.900184715



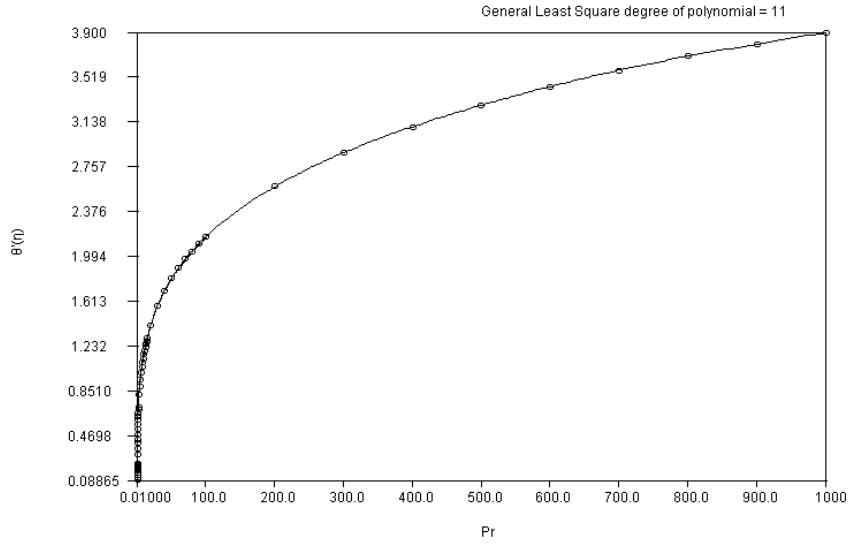


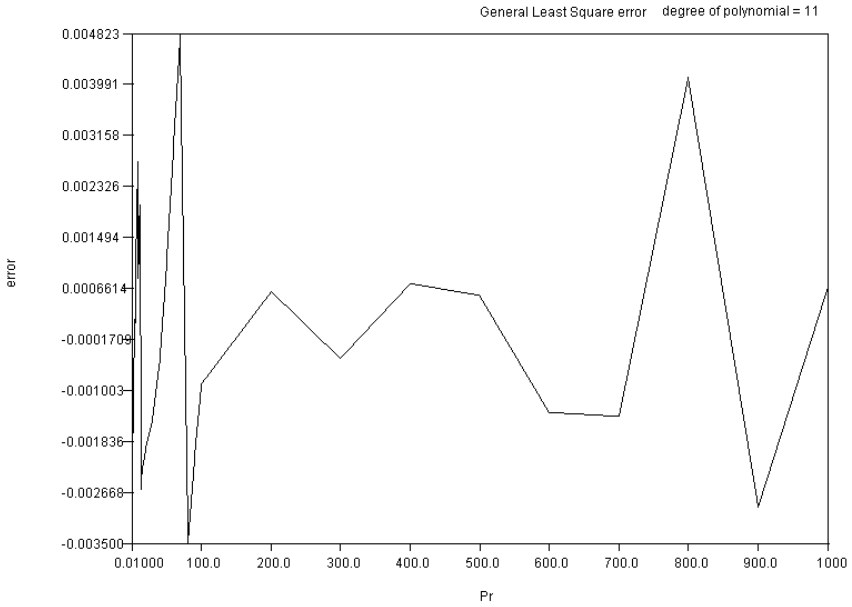
After establishing data, different curve fitting methods can be used to get Nusselt number correlations. The simplest correlation could be a linear polynomial such as an 11th degree polynomial

$$\left.\frac{\partial\theta}{\partial y}\right|_{y=0} = \theta' = a_0 Pr^{1/4} + a_1 Pr^{1/2} + a_2 Pr^{3/4} + a_3 Pr + a_4 Pr^{5/4} + a_5 Pr^{3/2} + a_6 Pr^{7/4} + a_7 Pr^2 + a_8 Pr^{9/4} + a_9 Pr^{5/2} + a_{10} Pr^{11/4} + a_{11} Pr^3$$

a_0	2.0989129231E-01
a_1	-3.0051209224E-01
a_2	2.6523436090E+00
a_3	-3.9991683936E+00
a_4	3.0819446934E+00
a_5	-1.4187784421E+00
a_6	4.0479301621E-01
a_7	-6.9334038631E-02
a_8	6.0379609656E-03
a_9	-3.3673993063E-05
a_{10}	-3.6909765186E-05
a_{11}	2.0994279162E-06

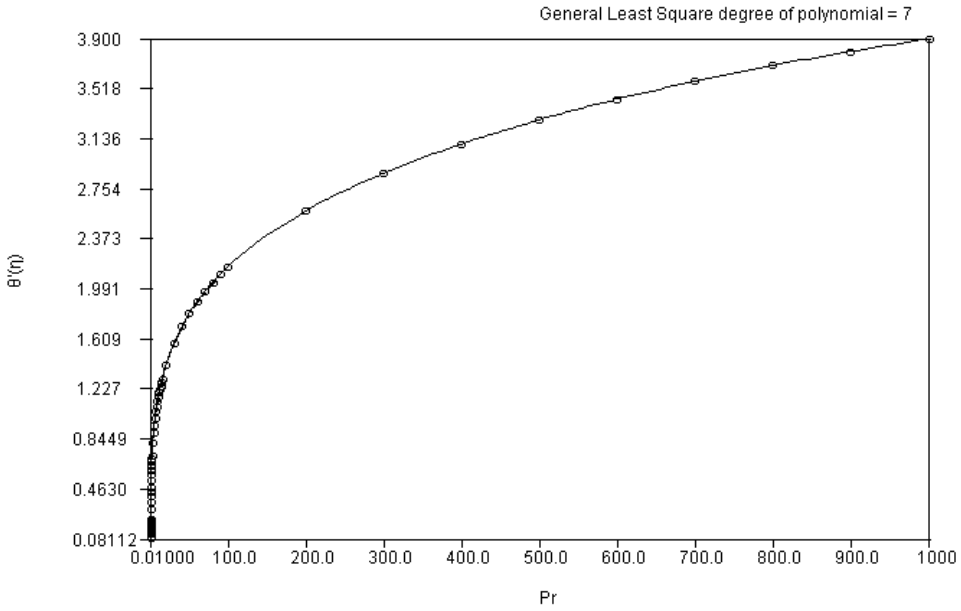
For this polynomial data and fitted line and error shown as follows

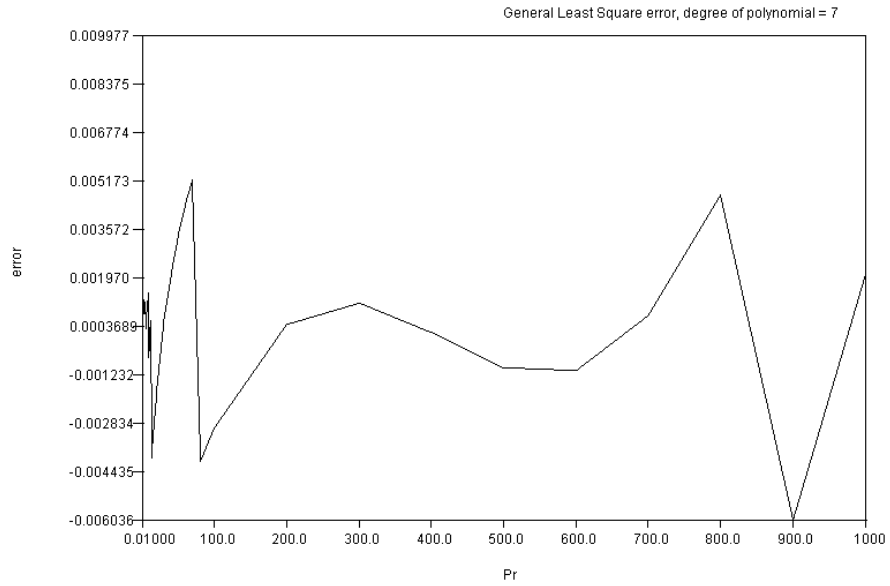




$$\left.\frac{\partial\theta}{\partial y}\right|_{y=0} = \theta' = a_0 Pr^{1/4} + a_1 Pr^{1/2} + a_2 Pr^{3/4} + a_3 Pr + a_4 Pr^{5/4} + a_5 Pr^{3/2} + a_6 Pr^{7/4} + a_7 Pr^2$$

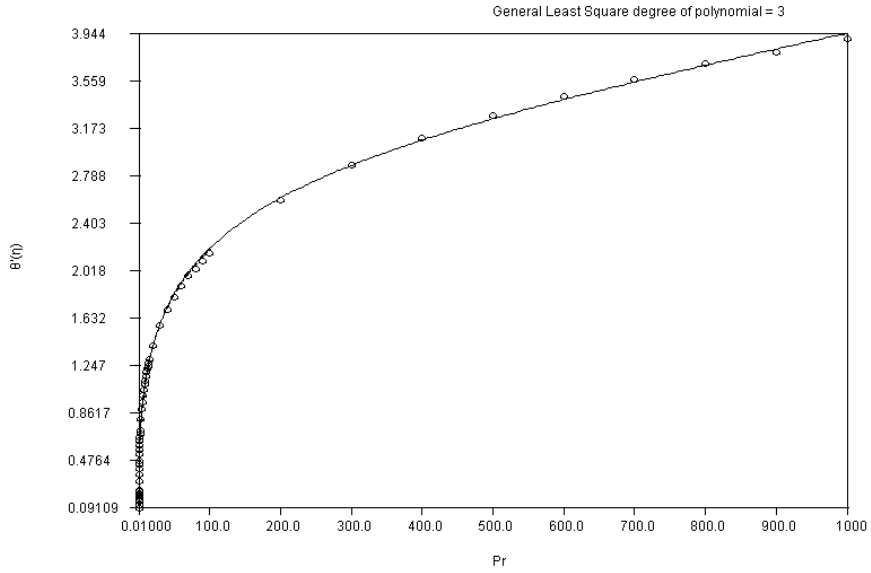
a0	-7.1616625047E-02
a1	1.3294314755E+00
a2	-1.0629736882E+00
a3	4.8311043084E-01
a4	-1.3183854844E-01
a5	2.1366843379E-02
a6	-1.8917609206E-03
a7	7.0288177613E-05

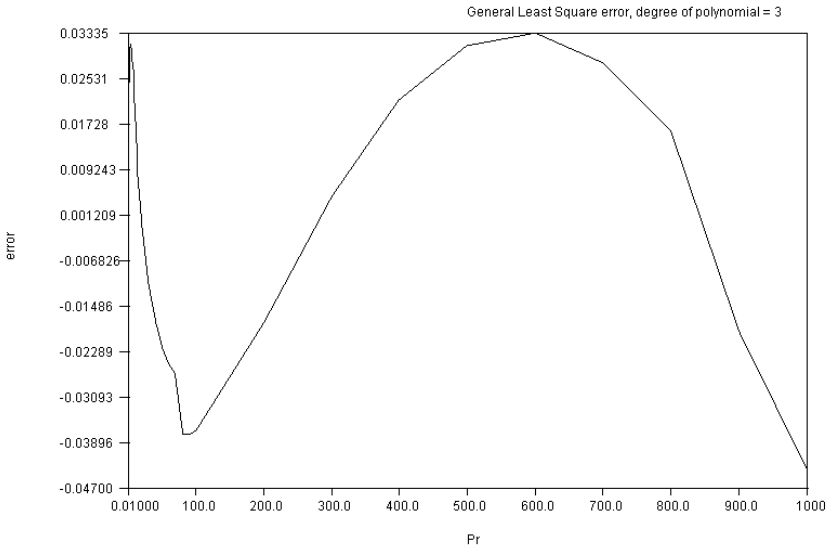




$$\left.\frac{\partial\theta}{\partial y}\right|_{y=0} = \theta' = a_0 Pr^{1/4} + a_1 Pr^{1/2} + a_2 Pr^{3/4} + a_3 Pr$$

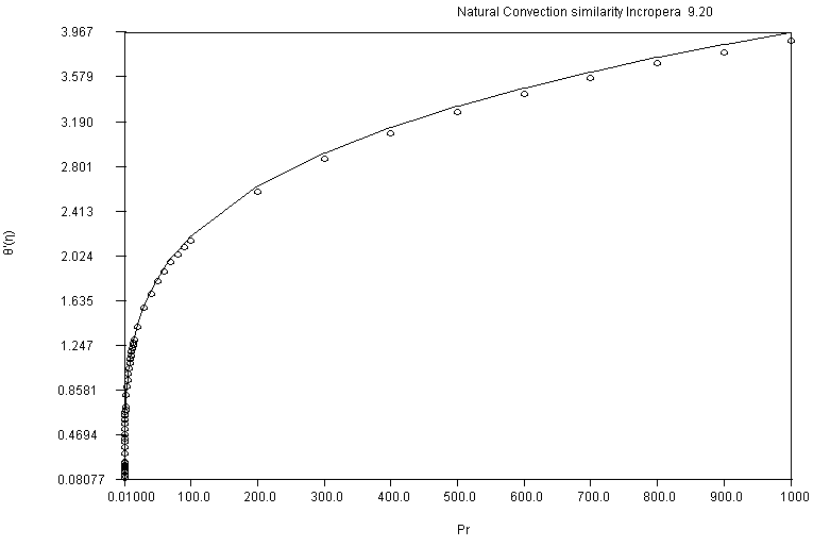
a0	0.337767076
a1	0.279380627
a2	-0.071379462
a3	0.005903166



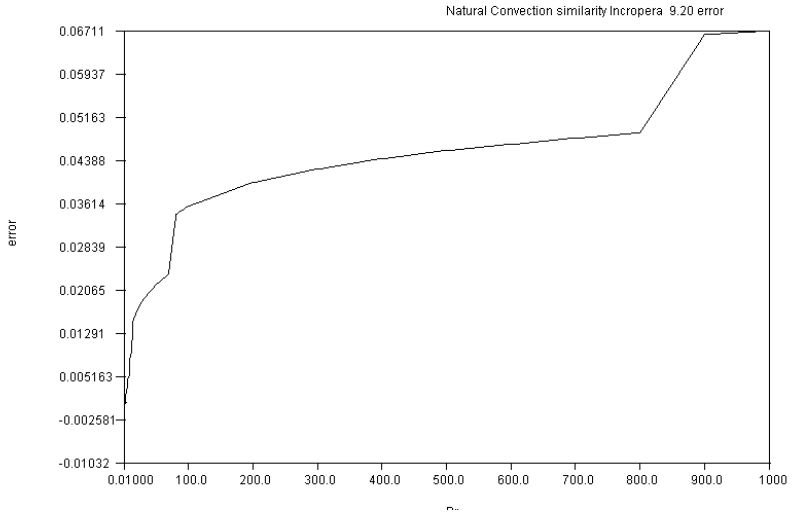


As a solution of this problem following equation is given in Fundamentals of Heat and Mass Transfer, Incropera/DeWitt/Bergman/Lavine, sixth edition as:

$$\left.\frac{\partial\theta}{\partial y}\right|_{y=0} = \theta' = \frac{0.75Pr^{1/2}}{(0.609 + 1.221Pr^{1/2} + 1.238Pr)^{1/4}}$$



Error level of the equation is as follow:



Pr	θ'(η) calculated	θ'(η) Incropera 9.20	Error
0.01	0.091093502	0.081214605	-0.009878897
0.02	0.116834579	0.112577416	-0.004257163
0.03	0.136995175	0.135789496	-0.001205679
0.04	0.154605985	0.154806134	2.00E-04
0.05	0.170229134	0.171157064	9.28E-04
0.06	0.184286962	0.185625468	0.001338506
0.07	0.19708927	0.198675703	0.001586433
0.08	0.208865088	0.210609701	0.001744613
0.09	0.219786662	0.221636726	0.001850064
0.1	0.231500963	0.231908792	4.08E-04
0.2	0.307792789	0.309682066	0.001889277
0.3	0.362168135	0.363931695	0.00176356
0.4	0.4050892	0.406583847	0.001494647
0.5	0.440866301	0.442124632	0.001258331
0.6	0.471765381	0.472793688	0.001028307
0.8	0.523754747	0.524235458	4.81E-04
1	0.56672913	0.566769318	4.02E-05
1.2	0.603623472	0.603273793	-3.50E-04
1.4	0.636067035	0.635395565	-6.71E-04
1.6	0.664867361	0.66417209	-6.95E-04
1.8	0.691266588	0.690302464	-9.64E-04
2	0.715036357	0.714281819	-7.55E-04
3	0.813788218	0.812086974	-0.001701244
4	0.889502603	0.887072196	-0.002430407
5	0.950493977	0.948637499	-0.001856478
6	1.003583771	1.001262098	-0.002321673
7	1.050193207	1.047456694	-0.002736513
8	1.09189393	1.088780187	-0.003113743
9	1.127210608	1.126271668	-9.39E-04
10	1.161864748	1.16066093	-0.001203818
11	1.193933662	1.192481565	-0.001452097
12	1.223822704	1.222136374	-0.00168633
13	1.246961238	1.249937427	0.002976189
14	1.273319569	1.276131774	0.002812205
15	1.298263279	1.300918579	0.0026553
20	1.406640617	1.408592852	0.001952235
30	1.572188344	1.572996939	8.09E-04
40	1.699540604	1.699415203	-1.25E-04
50	1.804447867	1.80352129	-9.27E-04
60	1.894382556	1.892748434	-0.001634122
70	1.973529018	1.971257678	-0.00227134

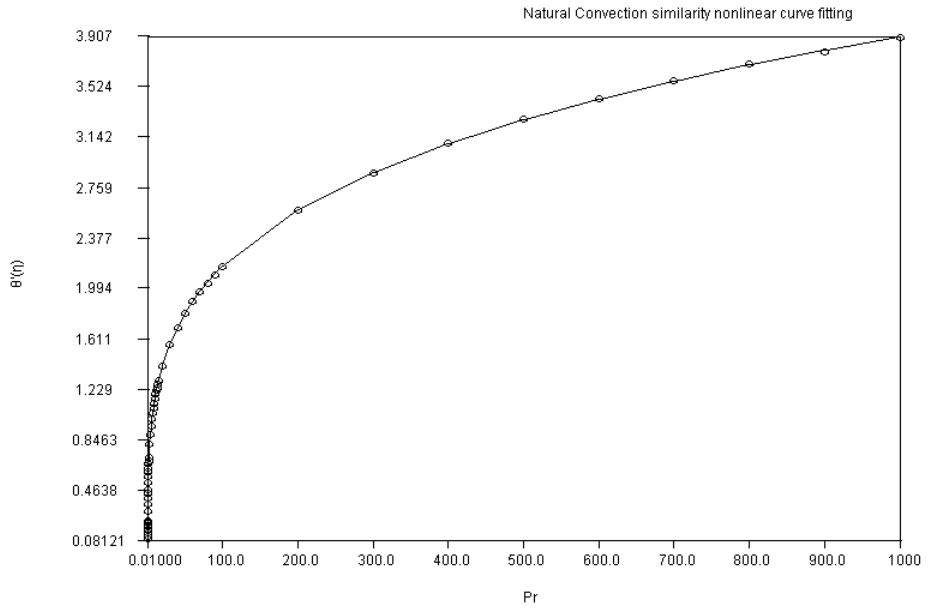
80	2.034457087	2.0416329	0.007175813
90	2.098909425	2.105597706	0.006688281
100	2.158133722	2.164363939	0.006230217
200	2.587798708	2.590451013	0.002652305
300	2.874897742	2.874952717	5.50E-05
400	3.096602009	3.094562562	-0.002039447
500	3.279691561	3.275874082	-0.003817479
600	3.436935238	3.431560807	-0.005374431
700	3.575508896	3.568742382	-0.006766514
800	3.699882055	3.691852017	-0.008030038
900	3.796072907	3.803853475	0.007780568
1000	3.900184715	3.906834359	0.006649644

The same accuracy equation is curve fitted by using Nelder_& Mead nonlinear optimization method.

$$\left.\frac{\partial\theta}{\partial y}\right|_{y=0} = \theta' = \frac{a_0 Pr^{1/2}}{(a_1 + a_2 + a_3 Pr)^{1/4}}$$

$$\left.\frac{\partial\theta}{\partial y}\right|_{y=0} = \theta' = \frac{a_0\ \mathrm{Pr}^{1/2}}{(a_1 + a_2\ \mathrm{Pr}^{1/2} + a_3\ \mathrm{Pr})^{1/4}}$$

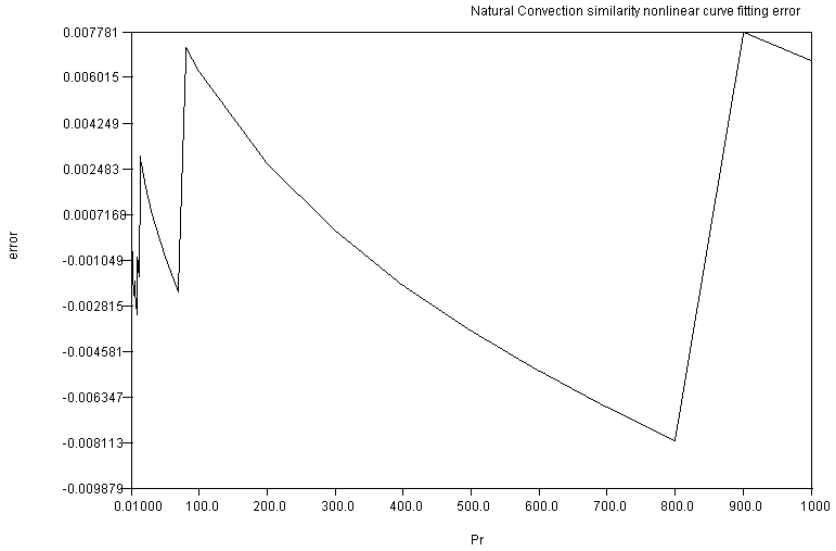
a0	0.756705949
a1	0.621261022
a2	1.187008971
a3	1.369213226



The error of these curve fitting value:

Pr	θ'(η) calculated	θ'(η) Non Lin. C.F.	Error
0.01	0.0910935	0.081214605	-0.009878897
0.02	0.11683458	0.112577416	-0.004257163
0.03	0.13699518	0.135789496	-0.001205679
0.04	0.15460599	0.154806134	2.00E-04
0.05	0.17022913	0.171157064	9.28E-04
0.06	0.18428696	0.185625468	0.001338506
0.07	0.19708927	0.198675703	0.001586433
0.08	0.20886509	0.210609701	0.001744613
0.09	0.21978666	0.221636726	0.001850064
0.1	0.23150096	0.231908792	4.08E-04

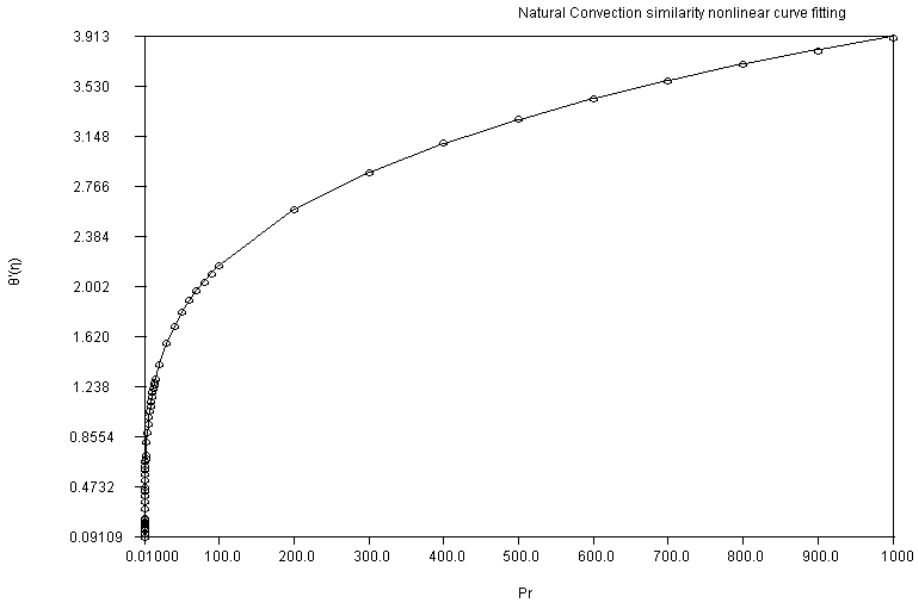
0.2	0.30779279	0.309682066	0.001889277
0.3	0.36216814	0.363931695	0.00176356
0.4	0.4050892	0.406583847	0.001494647
0.5	0.4408663	0.442124632	0.001258331
0.6	0.47176538	0.472793688	0.001028307
0.8	0.52375475	0.524235458	4.81E-04
1	0.56672913	0.566769318	4.02E-05
1.2	0.60362347	0.603273793	-3.50E-04
1.4	0.63606704	0.635395565	-6.71E-04
1.6	0.66486736	0.66417209	-6.95E-04
1.8	0.69126659	0.690302464	-9.64E-04
2	0.71503636	0.714281819	-7.55E-04
3	0.81378822	0.812086974	-0.001701244
4	0.8895026	0.887072196	-0.002430407
5	0.95049398	0.948637499	-0.001856478
6	1.00358377	1.001262098	-0.002321673
7	1.05019321	1.047456694	-0.002736513
8	1.09189393	1.088780187	-0.003113743
9	1.12721061	1.126271668	-9.39E-04
10	1.16186475	1.16066093	-0.001203818
11	1.19393366	1.192481565	-0.001452097
12	1.2238227	1.222136374	-0.00168633
13	1.24696124	1.249937427	0.002976189
14	1.27331957	1.276131774	0.002812205
15	1.29826328	1.300918579	0.0026553
20	1.40664062	1.408592852	0.001952235
30	1.57218834	1.572996939	8.09E-04
40	1.6995406	1.699415203	-1.25E-04
50	1.80444787	1.80352129	-9.27E-04
60	1.89438256	1.892748434	-0.001634122
70	1.97352902	1.971257678	-0.00227134
80	2.03445709	2.0416329	0.007175813
90	2.09890943	2.105597706	0.006688281
100	2.15813372	2.164363939	0.006230217
200	2.58779871	2.590451013	0.002652305
300	2.87489774	2.874952717	5.50E-05
400	3.09660201	3.094562562	-0.002039447
500	3.27969156	3.275874082	-0.003817479
600	3.43693524	3.431560807	-0.005374431
700	3.5755089	3.568742382	-0.006766514
800	3.69988206	3.691852017	-0.008030038
900	3.79607291	3.803853475	0.007780568
1000	3.90018472	3.906834359	0.006649644



As it is seen from the results, error level of this equation is one order better than the Incropera equation. Error level can be further reduced in nonlinear equations by increasing number of terms as well. Let us assume the equation in the form of

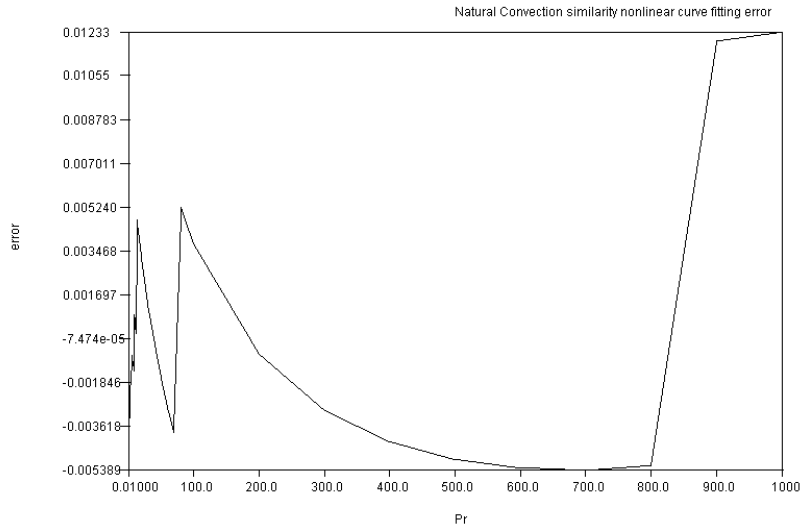
$$\left.\frac{\partial\theta}{\partial y}\right|_{y=0} = \theta' = \frac{a_0 + a_1\,\mathrm{Pr}^{1/2} + a_2\,\mathrm{Pr}}{(a_3 + a_4\,\mathrm{Pr}^{1/2} + a_5\,\mathrm{Pr})^{1/4}}$$

a0	0.022830145
a1	0.752624416
a2	2.08E-04
a3	1.222236219
a4	1.010421489
a5	1.389905434



Pr	θ'(η) calculated	θ'(η) Non Lin. C.F.	Error
0.01	0.091093502	0.091221691	1.28E-04
0.02	0.116834579	0.118992716	0.002158137
0.03	0.136995175	0.139872692	0.002877517
0.04	0.154605985	0.157180232	0.002574247
0.05	0.170229134	0.172202802	0.001973668

0.06	0.184286962	0.185601606	0.001314644
0.07	0.19708927	0.19776992	6.81E-04
0.08	0.208865088	0.208964269	9.92E-05
0.09	0.219786662	0.219363018	-4.24E-04
0.1	0.231500963	0.22909615	-0.002404813
0.2	0.307792789	0.304102599	-0.00369019
0.3	0.362168135	0.35757659	-0.004591545
0.4	0.4050892	0.400125376	-0.004963824
0.5	0.440866301	0.435841974	-0.005024327
0.6	0.471765381	0.466810409	-0.004954972
0.8	0.523754747	0.518973737	-0.00478101
1	0.56672913	0.562235954	-0.004493176
1.2	0.603623472	0.59941289	-0.004210582
1.4	0.636067035	0.632137656	-0.003929379
1.6	0.664867361	0.661449436	-0.003417925
1.8	0.691266588	0.688053664	-0.003212924
2	0.715036357	0.712452737	-0.00258362
3	0.813788218	0.811759058	-0.00202916
4	0.889502603	0.887621384	-0.001881219
5	0.950493977	0.949722148	-7.72E-04
6	1.003583771	1.002679508	-9.04E-04
7	1.050193207	1.04907898	-0.001114227
8	1.09189393	1.090522895	-0.001371035
9	1.127210608	1.128077086	8.66E-04
10	1.161864748	1.162488589	6.24E-04
11	1.193933662	1.194302537	3.69E-04
12	1.223822704	1.223929691	1.07E-04
13	1.246961238	1.25168775	0.004726512
14	1.273319569	1.277827826	0.004508257
15	1.298263279	1.302552067	0.004288788
20	1.406640617	1.409850676	0.003210059
30	1.572188344	1.57346755	0.001279206
40	1.699540604	1.699202238	-3.38E-04
50	1.804447867	1.802740682	-0.001707185
60	1.894382556	1.891497867	-0.002884689
70	1.973529018	1.969616401	-0.003912617
80	2.034457087	2.039665331	0.005208244
90	2.098909425	2.10335679	0.004447365
100	2.158133722	2.161893751	0.003760029
200	2.587798708	2.587090857	-7.08E-04
300	2.874897742	2.871901695	-0.002996047
400	3.096602009	3.092319163	-0.004282846
500	3.279691561	3.274700561	-0.004991
600	3.436935238	3.431612916	-0.005322322
700	3.575508896	3.570119596	-0.0053893
800	3.699882055	3.694620897	-0.005261158
900	3.796072907	3.808059921	0.011987014
1000	3.900184715	3.912510613	0.012325898



Non linear curve fitting program:

```
import java.io.*;
import javax.swing.*;

abstract class f_xj
{
  // single function  multi independent variable
  // a single value is returned  (
  // example f=x[0]+sin(x[1])
  // func(x) returns the value of f
  abstract public double func(double x[]);
}
     class yy extends f_xj
     {
               double xi[]; // independent variable data set
     double yi[]; //dependent variable data set
     double a[];  //fit function coefficient set
     int nn;
               public yy(String filename,double ia[])
     {
       //read the data to curvefit
       //get the data file and initial fit coefficient when class is defined
                 xi=new double[1001];
       yi=new double[1001];
       int n=ia.length;
       a=new double[n];
       seta(ia);
       int i=-1;
       try{
                  BufferedReader fin=new BufferedReader(new FileReader(filename));
       try {
       while(fin != null)
       {
       i++;
       xi[i]=Text.readDouble(fin);
       yi[i]=Text.readDouble(fin);
       }
       } catch(EOFException e_eof) {System.out.println("end of file"); }
       }catch(IOException e_io) {System.out.println("dosya bulunamadı"); }
     nn=i;
     a=ia;
     }

     public void seta(double ia[])
     {
                  //assign new fit coefficient set
                   for(int ii=0;ii<nn;ii++)
           a[ii]=ia[ii];
                }

               public double[] geta()
     {
                  // return fit coefficient set
                  return a;
```

```
                }

    double Ps(double Pr,double ai[])
    {a=ai;
    double pi=a[0]*Math.pow(Pr,0.5)/Math.pow((a[1]+a[2]*Math.pow(Pr,0.5)+a[3]*Pr),0.25);
    return pi;
    }

    public double func(double ai[])
    {
    double ff=0;
    double w;
    double yy;
     for(int i=0;i<nn;i++)
                {w=Ps(xi[i],ai);
                yy=yi[i];
      //System.out.println("x="+xi[i]+"w="+w+"yy="+yy+"ff="+ff);
                w-=yy;
      ff+=w*w;
      //System.out.println("x="+xi[i]+"w="+w+"yy="+yy+"ff="+ff);
                }
    return ff;
                }
}

class zz extends yy
{    public zz(String filename,double ia[])
    {super(filename,ia);}
    public double func(double ai[])
    {
    return -super.func(ai);
                }
}

public class NA50
{
   public static void main(String args[]) throws IOException
   {
   //String in_name=JOptionPane.showInputDialog(" enter name of the input file : ");
   String in_name="Pr_t.txt";
   double p[];
   p=new double[4];
   p[0] =0.75;
   p[1] =0.609;
   p[2] =1.221;
   p[3] =1.238;

   double da[];
   da=new double[4];
   da[0] =0.1*p[0];
   da[1] =0.1*p[1];
   da[2] =0.1*p[2];
   da[3] =0.1*p[3];

   yy f=new yy(in_name,p);
   System.out.println("book function = ");
   double aa[][]=new double[4][53];
   for(int i=0;i<53;i++)
   {aa[0][i]=f.xi[i];aa[1][i]=f.yi[i];aa[2][i]=f.Ps(f.xi[i],p);aa[3][i]=aa[2][i]-aa[1][i];
   System.out.println("i="+i+"x="+aa[0][i]+"yi="+aa[1][i]+"y=f(xi)="+aa[2][i]+"error="+aa[3][i]);};
   Text.printT(aa,"incropera 9.20");
   Plot pp=new Plot(aa[0],aa[1]);
  pp.setPlabel("Natural Convection similarity Incropera  9.20 ");
  pp.setXlabel("Pr");
  pp.setYlabel(""+'\u03B8'+"'("+'\u03B7'+")");
  pp.addData(aa[0],aa[2]);
  pp.setPlotType(0,22);
  pp.plot();
  Plot pp1=new Plot(aa[0],aa[3]);
  pp1.setPlabel("Natural Convection similarity Incropera  9.20 error");
  pp1.setXlabel("Pr");
  pp1.setYlabel("error");
  pp1.plot();


   p=NA41.nelder(f,p,da,1e-15);
```

```
    for(int i=0;i<53;i++)
    {aa[0][i]=f.xi[i];aa[1][i]=f.yi[i];aa[2][i]=f.Ps(f.xi[i],p);aa[3][i]=aa[2][i]-aa[1][i];
    System.out.println("i="+i+"x="+aa[0][i]+"yi="+aa[1][i]+"y=f(xi)="+aa[2][i]+"error="+aa[3][i]);};
  Text.printT(aa,"nonlinear curve fitting");
  Text.printT(p,"coefficients of non-linear eqaution");
    pp=new Plot(aa[0],aa[1]);
  pp.setPlabel("Natural Convection similarity nonlinear curve fitting ");
  pp.setXlabel("Pr");
  pp.setYlabel(""+'\u03B8'+"'("+'\u03B7'+")");
  pp.addData(aa[0],aa[2]);
  pp.setPlotType(0,22);
  pp.plot();
  pp1=new Plot(aa[0],aa[3]);
  pp1.setPlabel("Natural Convection similarity nonlinear curve fitting error");
  pp1.setXlabel("Pr");
  pp1.setYlabel("error");
  pp1.plot();
        String s1=" coefficient of the curve fitting equation  : \n"+Matrix.toStringT(p)+"'\n";
    String s2="Nelder-Mead nonlinear least square curve fitting : ";
    JOptionPane.showMessageDialog(null,s1,s2,JOptionPane.PLAIN_MESSAGE);
    }

}
```

It should also be note that in order to obtain average heat transfer coefficients, these local heat transfer coefficients should be integrated over the total distance.

$$\bar{h} = \frac{1}{L}\int_0^L h dx = \frac{k}{L}\left[\frac{g\beta(T_s - T_\infty)}{4\nu^2}\right]^{1/4} \frac{d\theta(0)}{d\eta} \int_0^L \frac{dx}{x^{1/4}}$$

And

$$\overline{Nu} = \frac{\bar{h}L}{k} = \frac{4}{3}Nu_x = \frac{4}{3}\left(\frac{\mathrm{Gr}_L}{4}\right)^{1/4} \frac{d\theta(0)}{d\eta}$$

(Several different curve fitting equations for $\theta'^{(0)}$and their error level compare to results obtained from the program is given above)

Results and discussion

Most of the equations given in the text books related to this topic is listed and derived for hand calculations. In recent times the basic method of calculations are through computer programs. Therefore more accurate equations containing more terms can be easily utilized for this type of equations.

14.2 NATURAL CONVECTION HEAT TRANSFER EQUATIONS IN VERTICAL PLATE

Grashoff Number: $\mathrm{Gr}_x = \frac{g\beta(T_s - T_\infty)x^3}{\nu^2}$ where $\nu = \frac{\mu}{\rho}$ $\beta = -\frac{1}{\rho}\left(\frac{\partial\rho}{\partial T}\right)_P$ is the volumetric thermal expansion coefficient. This term may be approximated as: $\beta \approx -\frac{1}{\rho}\left(\frac{\rho_\infty - \rho}{T_\infty - T}\right)$. For a perfect gas: $\rho = P/RT$ therefore: $\beta = -\frac{1}{\rho}\left(\frac{\partial\rho}{\partial T}\right)_P = \frac{1}{\rho}\frac{P}{RT^2} = \frac{1}{T}$ 1/K



Franz Grashoff

Rayleigh Number: $\mathrm{Ra}_x = \mathrm{Gr}_x Pr = \frac{g\beta(T_s - T_\infty)x^3}{\nu\alpha}$ $\alpha = \frac{k}{\rho C_p}$

Critical Rayleigh Number $\mathrm{Ra}_{x,critical} = 10^9$

$\mathrm{Nu}_L = 4/3\mathrm{Nu}_x$



Lord(John William Stuart) Rayleigh

Laminar free convection: Incropera-DeWitt equation[35] similarity solution

$$\mathrm{Nu}_L = \frac{4}{3}\left(\frac{\mathrm{Gr}_L}{4}\right)^{1/4} \frac{0.75Pr^{1/2}}{(0.609+1.221Pr^{\frac{1}{2}}+1.238\mathrm{Pr})^{1/4}} \quad 0 \le Pr \le \infty \quad 0 \le Ra_L \le 10^9$$

Laminar free convection: Coban-2 equation similarity solution (see previous subchapter for calculation details)

$$\mathrm{Nu}_L = \frac{4}{3}\left(\frac{\mathrm{Gr}_L}{4}\right)^{1/4} \frac{(0.022830145+0.752624416Pr^{\frac{1}{2}}+2.08\mathrm{E}{-4}\mathrm{Pr})}{(1.222236219+1.010421489Pr^{\frac{1}{2}}+1.389905434\mathrm{Pr})^{1/4}} \; 0.01 \le Pr \le 1000 \quad 0 \le Ra_L \le 10^9$$

Churchill & Chu 1Equation for all Ra range[36]

$$\mathrm{Nu}_L = \left\{0.825 + \frac{0.387Ra_L^{1/6}}{\left[1+\left(\frac{0.492}{\mathrm{Pr}}\right)^{9/16}\right]^{8/27}}\right\}^2$$

Churchill & Chu 2 Equation for Laminar Ra range[36]

$$\mathrm{Nu}_L = 0.68 + \frac{0.67Ra_L^{1/4}}{\left[1+\left(\frac{0.492}{\mathrm{Pr}}\right)^{9/16}\right]^{4/9}} \; 0 \le Ra_L \le 10^9$$

PROBLEM : air temperature of a room is T_∞=26.85 C and room pressure is P=101.325 kPa. Room has a heating plate of height **L=0.5** m and width W=1 m. Plate surface temperature is T_s=126.85 C. Find the heat transfer from natural convection.

```
 public class HT_natural_vertical_plate_air
{
  public static double mu(double T)
  {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
```

```
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double k(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pr(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;

return y;
}
public static double Cp(double T)
{double a[]={1.89668613E+00,
-2.23937197E-02,
2.40885259E-04,
-1.45254290E-06,
5.43552393E-09,
-1.32310554E-11,
2.13239465E-14,
-2.25744420E-17,
1.50864395E-20,
-5.76865768E-24,
9.61540784E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
return y*1e3;
}

  public static double ro(double T,double P)
  {   double M=2.8965000E+01;
      double R=8314.5/M;
                return P*1e3/R/T;
  }
  public static double Nu(double Ts,double Tinf,double P,double L,int select)
  {//Nusselt number
   double T=(Tinf+Ts)/2.0;
   double Ra=Ra(Ts,Tinf,P,L);
   double Pr=Pr(T);
   System.out.println("Pr="+Pr);
   double Gr=Ra/Pr;
   double Nu=0.0;
   double Racritic=1e9;
   double Pr_05=Math.sqrt(Pr);
   if(Ra>Racritic || select==4 )
{Nu=(0.825+0.387*Math.pow(Ra,(1.0/6.0))/Math.pow((1+Math.pow((0.492/Pr),(9.0/16.0))),(8.0/27.0)));
                Nu=Nu*Nu;
```

```
                }
   else
   {if(select==1) Nu=4.0/3.0*Math.pow((Gr/4.0),(1.0/4.0))*0.75*Pr_05/Math.pow((0.609+1.221*Pr_05+1.238*Pr),(1.0/4.0));
    else if(select==2) Nu=4.0/3.0*Math.pow((Gr/4.0),(1.0/4.0))*(0.022830145+0.752624416*Pr_05+2.04e-
4*Pr)/Math.pow((1.222236219+1.010421489*Pr_05+1.389905434*Pr),(1.0/4.0));
    else if(select==3) Nu=(0.68+0.67*Math.pow(Ra,(1.0/4.0))/Math.pow((1+Math.pow((0.492/Pr),(9.0/16.0))),(4.0/9.0)));
   }
   return Nu;
   }

  public static double Ra(double Ts,double Tinf,double P,double L)
  {   double T=(Ts+Tinf)/2.0;
                double mu=mu(T);
     double ro=ro(T,P);
     double Cp=Cp(T);
     double k=k(T);
     double beta=1/T;
     double nu=mu/ro;
     double alpha=k/(ro*Cp);
     double g=9.806; //m/s^2
     double Ra=g*beta*(Ts-Tinf)*L*L*L/alpha/nu;
                return Ra;
  }
  public static void main(String arg[])
  { double Tinf=26.85+273.15;  //air temp. degree K;
    double Ts=126.85+273.15;
    double L=0.5; // m
    double W=1.0;
    double P=101.325; //kPa
    double Nu1=Nu(Ts,Tinf,P,L,1);
    double Nu2=Nu(Ts,Tinf,P,L,2);
    double Nu3=Nu(Ts,Tinf,P,L,3);
    double Nu4=Nu(Ts,Tinf,P,L,4);
    System.out.println("Incropera-DeWitt eqn : Nu1="+Nu1+"\nCoban-2 Eqn  Nu2="+Nu2+"\nChurchill & Chu eqn
Nu3="+Nu3+"\nChurchill & Chu eqn  Nu4="+Nu4);
    double T=(Tinf+Ts)/2.0;
    double k1=k(T);
    double h1=Nu1*k1/L;
    double h2=Nu2*k1/L;
    double h3=Nu3*k1/L;
    double h4=Nu4*k1/L;
    System.out.println("h1="+h1+" h2="+h2+" h3="+h3+" h4="+h4);
    double A=W*L;
    double Q1=h1*A*(Ts-Tinf);
    double Q2=h2*A*(Ts-Tinf);
    double Q3=h3*A*(Ts-Tinf);
    double Q4=h4*A*(Ts-Tinf);
    System.out.println("Q1="+Q1+" Q2="+Q2+" Q3="+Q3+"Q4 = "+Q4);
}}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_natural_vertical_plate_air
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_natural_vertical_plate_air
Ra=5.737293044140159E8
Ra=5.737293044140159E8
Ra=5.737293044140159E8
Ra=5.737293044140159E8
Incropera-DeWitt eqn : Nu1=79.64626650993371
Coban-2 Eqn  Nu2=78.86293521532025
Churchill & Chu eqn  Nu3=80.1577851580879
Churchill & Chu eqn  Nu4=103.39052157236225
h1=4.7792106730770305 h2=4.732206520242007 h3=4.809904583662677 h4=6.203995564964818
Q1=238.9605336538515 Q2=236.61032601210036 Q3=240.49522918313383Q4 = 310.1997782482409
```

Note $Ra < Ra_{critical} = 10^9$

	Heat transfer over a flat plate		**working fluid : air**		
M	**2.8965000E+01**	**kg/kmol**			
R	**2.8705334E+02**	**J/kgK**			
P	**1.0132500E+02**	**kPa**			
	ρ	Cp	μ	k	Pr
	kg/m^3				

a0		1.89668613E+00	-7.00796942E-01	-5.96297846E-	1.47352515E+00
a1		-2.23937197E-02	8.85836561E-02	1.00247973E-	-1.73708682E-02
a2		2.40885259E-04	-1.20909367E-04	7.32722776E-	1.80473009E-04
a3		-1.45254290E-06	1.73074497E-07	-1.02111800E-	-1.07436904E-06
a4		5.43552393E-09	-1.45011707E-10	4.21691040E-	3.99304707E-09
a5		-1.32310554E-11	-4.33936641E-14	-1.04115589E-	-9.67956666E-12
a6		2.13239465E-14	3.14868466E-16	1.67088379E-	1.55567247E-14
a7		-2.25744420E-17	-4.41548734E-19	-1.75371889E-	-1.64359568E-17
a8		1.50864395E-20	3.25615211E-22	1.16134919E-	1.09674418E-20
a9		-5.76865768E-24	-1.28963912E-25	-4.40290627E-	-4.18869082E-24
a10		9.61540784E-28	2.16661936E-29	7.28284943E-	6.97523663E-28
result:for T	1.0085234E+00	1.0083819E+00	2.0867141E-05	3.0002729E-	7.0130741E-01
result: for	8.8245794E-01	1.0140736E+00	2.3055411E-05	3.3453105E-	6.9887896E-01

	degree C	**degree K**	
T	7.6850000E+01	3.5000000E+02	average temp
Ts	1.2685000E+02	4.0000000E+02	surface
Tinf	2.6850000E+01	3.0000000E+02	air temp
P	1.0132500E+00	bar	air pressure
ρ	1.0085234E+00		air density kg/m^3
W	1.0000000E+00		weight of plate
L	5.0000000E-01		length of plate
k	3.0002729E-02		air thermal
Cp	1.0083819E+03		air specific heat
Pr	7.0130741E-01		air Prandtl
α	2.9501884E-05	W/mK	air thermal
g	9.8060000E+00	m/s^2	gravitational
β	2.8571429E-03	W/m	thermal expansion
μ	2.0867141E-05	Pas	dynamic viscosity
ν	2.0690786E-05	m^2/s	kinematic viscosity
Ra	5.6888063E+08		Rayleigh number
Gr	8.1117157E+08		
Incropera-	7.9477454E+01		Nusselt number
h1	4.7690810E+00	W/(m2K)	thermal
Q1	**2.3845405E+02**	W	Rayleigh number
Coban-2	7.8696121E+01		Nusselt number
h2	4.7221968E+00	W/(m2K)	thermal
Q2	**2.3610984E+02**	W	Rayleigh number
Churchil &	7.9989330E+01		Nusselt number
h3	4.7997963E+00	W/(m2K)	thermal
Q3	**2.3998982E+02**	W	Rayleigh number
Churchil &	1.0312212E+02		Nusselt number
h4	6.1878901E+00	W/(m2K)	thermal
Q4	3.0939451E+02	W	Rayleigh number

PROBLEM : air temperature of a room is T_∞=26.85 C and room pressure is P=101.325 kPa. Room has a heating plate of heigh **L=1 m** and width W=1 m. Plate surface temperature is T_s=126.85 C. Find the heat transfer from natural convection.

```
Incropera-DeWitt eqn : Nu1=197.06985451111964
Coban-2 Eqn  Nu2=197.06985451111964
Churchill & Chu eqn  Nu3=197.06985451111964
Churchill & Chu eqn  Nu4=197.06985451111964
h1=5.912633405759276 h2=5.912633405759276 h3=5.912633405759276 h4=5.912633405759276
Q1=591.2633405759276 Q2=591.2633405759276 Q3=591.2633405759276Q4 = 591.2633405759276

> Terminated with exit code 0.
```

Note: Note $Ra>Ra_{critical}=10^9$ laminar equations are bypassed and case 4 Churchill & Chu equation is used regarless of selection

	degree C	**degree K**	
T	7.6850000E+01	3.5000000E+02	average temp
Ts	1.2685000E+02	4.0000000E+02	surface temperature
Tinf	2.6850000E+01	3.0000000E+02	air temp
P	1.0132500E+00	bar	air pressure

ρ	1.0085234E+00		air density kg/m^3
W	1.0000000E+00		weight of plate
L	1.0000000E+00		length of plate
k	3.0002729E-02		air thermal conductivity
Cp	1.0083819E+03		air specific heat
Pr	7.0130741E-01		air Prandtl number
α	2.9501884E-05	W/mK	air thermal diffusivity
g	9.8060000E+00	m/s^2	gravitational acceleration cof.
β	2.8571429E-03	W/m	thermal expansion coeff.
μ	2.0867141E-05	Pas	dynamic viscosity
ν	2.0690786E-05	m^2/s	kinematic viscosity
Ra	4.5510451E+09		Rayleigh number
Gr	6.4893726E+09		
Churchil & Chu 1	1.9654582E+02		Nusselt number
h4	5.8969110E+00	W/(m2K)	thermal convection coef.
Q4	5.8969110E+02	W	Rayleigh number

PROBLEM: Calculate the heat transfer rates by free convection from a 0.3 m high vertical plate maintained at a uniform temperature Tw=80 C to an ambient at T_{inf}=23 C containing air at 1 atm.

```
public class HT_natural_vertical_plate_air
{
  public static double mu(double T)
  {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double k(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pr(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;

return y;
}
```

```
public static double Cp(double T)
{double a[]={1.89668613E+00,
-2.23937197E-02,
2.40885259E-04,
-1.45254290E-06,
5.43552393E-09,
-1.32310554E-11,
2.13239465E-14,
-2.25744420E-17,
1.50864395E-20,
-5.76865768E-24,
9.61540784E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
return y*1e3;
}

 public static double ro(double T,double P)
 {   double M=2.8965000E+01;
     double R=8314.5/M;
                return P*1e3/R/T;
 }

 public static double Nu(double Ts,double Tinf,double P,double L,int select)
 {//Nusselt number
  double T=(Tinf+Ts)/2.0;
  double Ra=Ra(Ts,Tinf,P,L);
  double Pr=Pr(T);
  double Gr=Ra/Pr;
  double Nu=0.0;
  double Racritic=1e9;
  double Pr_05=Math.sqrt(Pr);
  if(Ra>Racritic || select==4 ) {Nu=(0.825+0.387*Math.pow(Ra,(1.0/6.0))/Math.pow((1+Math.pow((0.492/Pr),(9.0/16.0))),(8.0/27.0)));
                Nu=Nu*Nu;
               }
  else
  {if(select==1) Nu=4.0/3.0*Math.pow((Gr/4.0),(1.0/4.0))*0.75*Pr_05/Math.pow((0.609+1.221*Pr_05+1.238*Pr),(1.0/4.0));
   else if(select==2) Nu=4.0/3.0*Math.pow((Gr/4.0),(1.0/4.0))*(0.022830145+0.752624416*Pr_05+2.04e-
4*Pr)/Math.pow((1.222236219+1.010421489*Pr_05+1.389905434*Pr),(1.0/4.0));
   else if(select==3) Nu=(0.68+0.67*Math.pow(Ra,(1.0/4.0))/Math.pow((1+Math.pow((0.492/Pr),(9.0/16.0))),(4.0/9.0)));
  }
  return Nu;
  }

 public static double Ra(double Ts,double Tinf,double P,double L)
 {   double T=(Ts+Tinf)/2.0;
                double mu=mu(T);
    double ro=ro(T,P);
    double Cp=Cp(T);
    double k=k(T);
    double beta=1/T;
    double nu=mu/ro;
    double alpha=k/(ro*Cp);
    double g=9.806; //m/s^2
    double Ra=g*beta*(Ts-Tinf)*L*L*L/alpha/nu;
    System.out.println("Ra="+Ra);
                return Ra;
 }
 public static void main(String arg[])
 { double Tinf=23+273.15;  //air temp. degree K;
  double Ts=80+273.15;
  double L=0.3; // m
  double W=1.0;
  double P=101.325; //kPa
  double Nu1=Nu(Ts,Tinf,P,L,1);
  double Nu2=Nu(Ts,Tinf,P,L,2);
  double Nu3=Nu(Ts,Tinf,P,L,3);
  double Nu4=Nu(Ts,Tinf,P,L,4);
  System.out.println("Nu1="+Nu1+" Nu2="+Nu2+" Nu3="+Nu3+" Nu4="+Nu4);
  double T=(Tinf+Ts)/2.0;
  double k1=k(Tinf);
  double h1=Nu1*k1/L;
  double h2=Nu2*k1/L;
  double h3=Nu3*k1/L;
  double h4=Nu4*k1/L;
  System.out.println("h1="+h1+" h2="+h2+" h3="+h3+" h4="+h4);
  double A=W*L;
```

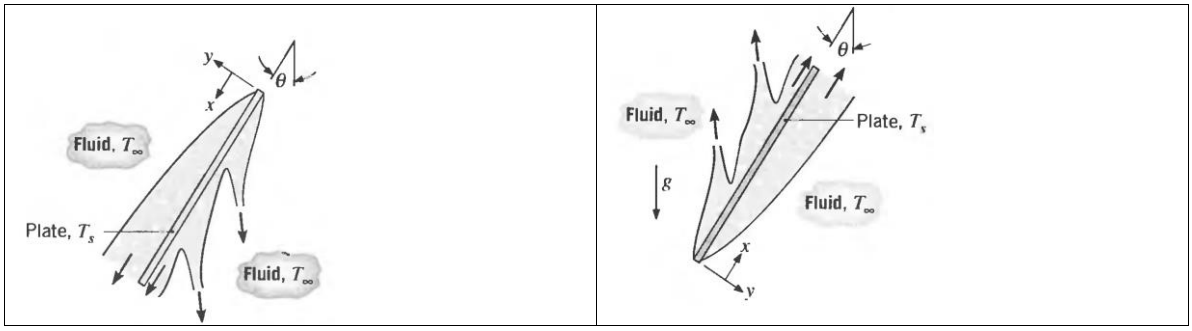
```
    double Q1=h1*A*(Ts-Tinf);
    double Q2=h2*A*(Ts-Tinf);
    double Q3=h3*A*(Ts-Tinf);
    double Q4=h4*A*(Ts-Tinf);
    System.out.println("Q1="+Q1+" Q2="+Q2+" Q3="+Q3+"Q4 = "+Q4);
}}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_natural_vertical_plate_air
Ra=9.958635104221523E7
Ra=9.958635104221523E7
Ra=9.958635104221523E7
Ra=9.958635104221523E7
Nu1=51.42810895822179 Nu2=50.92328414255047 Nu3=51.9994124399301 Nu4=60.916383627909866
h1=4.473866569836352 h2=4.429950530296759 h3=4.523565763522867 h4=5.29927655884891
Q1=76.50311834420161 Q2=75.75215406807459 Q3=77.35297455624102Q4 = 90.61762915631637

> Terminated with exit code 0.
```

T	5.1500000E+01	3.2465000E+02	average temp
Ts	8.0000000E+01	3.5315000E+02	surface temperature
Tinf	2.3000000E+01	2.9615000E+02	air temp
P	1.0132500E+00	bar	air pressure
ρ	1.0872730E+00		air density kg/m^3
W	1.0000000E+00		weight of plate
L	3.0000000E-01		length of plate
k	2.8191011E-02		air thermal conductivity
Cp	1.0066983E+03		air specific heat
Pr	7.0364658E-01		air Prandtl number
α	2.5755663E-05	W/mK	air thermal diffusivity
g	9.8060000E+00	m/s^2	gravitational acceleration cof.
β	3.0802403E-03	W/m	thermal expansion coeff.
μ	1.9705211E-05	Pas	dynamic viscosity
ν	1.8123517E-05	m^2/s	kinematic viscosity
Ra	9.1592789E+07		Rayleigh number
Gr	1.3016874E+08		
Incropera-DeWitt **Nu1**	5.0363501E+01		Nusselt number
h1	4.7326601E+00	W/(m2K)	thermal convection coef.
Q1	**8.0928488E+01**	W	Rayleigh number
Coban-2 **Nu2**	4.9869341E+01		Nusselt number
h2	4.6862239E+00	W/(m2K)	thermal convection coef.
Q2	**8.0134428E+01**	W	Rayleigh number
Churchil & Chu 2 **Nu3**	5.0937055E+01		Nusselt number
h3	4.7865570E+00	W/(m2K)	thermal convection coef.
Q3	**8.1850124E+01**	W	Rayleigh number
Churchil & Chu 1 **Nu4**	5.9416852E+01		Nusselt number
h4	5.5834038E+00	W/(m2K)	thermal convection coef.
Q4	9.5476205E+01	W	Rayleigh number

14.3 INCLINED PLATES NATURAL CONVECTION



Grashoff Number: $\mathrm{Gr}_x = \frac{g\cos(\theta)\beta(T_s - T_\infty)x^3}{\nu^2}$

Rayleigh Number: $\mathrm{Ra}_x = \mathrm{Gr}_x Pr = \frac{g\cos(\theta)\beta(T_s - T_\infty)x^3}{\nu\alpha}$

$$\mathrm{Nu}_L = \left\{0.825 + \frac{0.387 Ra_L^{1/6}}{\left[1+\left(\frac{0.492}{\mathrm{Pr}}\right)^{9/16}\right]^{8/27}}\right\}^2$$

PROBLEM : air temperature of a room is T_∞=26.85 C and room pressure is P=101.325 kPa. Room has an inclined heating plate of height **L=0.5** m, inclination angle of 45 degree and width W=1 m. Plate surface temperature is T_s=126.85 C. Find the heat transfer from natural convection.

Heat transfer natural convection inclined surface **working fluid**

M	**2.8965000E+01**	**kg/kmol**
R	**2.8705334E+02**	**J/kgK**
P	**1.0132500E+02**	**kPa**

	ρ	Cp	μ	k	Pr
	kg/m^3				
a0		1.89668613E+0	-	-	1.47352515E+0
a1		-2.23937197E-	8.85836561E-	1.00247973E-	-1.73708682E-
a2		2.40885259E-	-	7.32722776E-	1.80473009E-
a3		-1.45254290E-	1.73074497E-	-	-1.07436904E-
a4		5.43552393E-	-	4.21691040E-	3.99304707E-
a5		-1.32310554E-	-	-	-9.67956666E-
a6		2.13239465E-	3.14868466E-	1.67088379E-	1.55567247E-
a7		-2.25744420E-	-	-	-1.64359568E-
a8		1.50864395E-	3.25615211E-	1.16134919E-	1.09674418E-
a9		-5.76865768E-	-	-	-4.18869082E-
a10		9.61540784E-	2.16661936E-	7.28284943E-	6.97523663E-
result:for T	1.0085234E+00	1.0083819E+00	2.0867141E-	3.0002729E-	7.0130741E-01
result: for Tw	8.8245794E-01	1.0140736E+00	2.3055411E-	3.3453105E-	6.9887896E-01

	degree C	**degree K**	
T	7.6850000E+01	3.5000000E+02	average temp
Ts	1.2685000E+02	4.0000000E+02	surface
Tinf	2.6850000E+01	3.0000000E+02	air temp
P	1.0132500E+00	bar	air pressure
ρ	1.0085234E+00		air density
W	1.0000000E+00		weight of
L	5.0000000E-01		length of
θ	7.8539816E-01	45.0	
k	3.0002729E-02		air thermal
Cp	1.0083819E+03		air specific
Pr	7.0130741E-01		air Prandtl
α	2.9501884E-05	W/mK	air thermal
g	9.8060000E+00	m/s^2	gravitational
β	2.8571429E-03	W/m	thermal
μ	2.0867141E-05	Pas	dynamic
ν	2.0690786E-05	m^2/s	kinematic

Ra	4.0225935E+08		Rayleigh
Gr	5.7358492E+08		
Churchil & Chu 1 Nu4	**9.2761148E+01**		**Nusselt**
h4	**5.5661752E+00**	**W/(m2K)**	**thermal**
Q4	**2.7830876E+02**	**W**	**Heat**

```
 public class HT_natural_inclined_plate_air
{
  public static double mu(double T)
  {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double k(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pr(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;

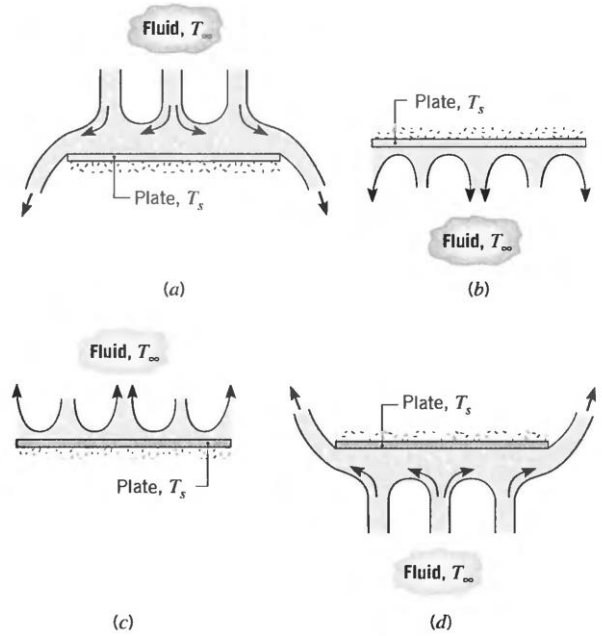
return y;
}
public static double Cp(double T)
{double a[]={1.89668613E+00,
-2.23937197E-02,
2.40885259E-04,
-1.45254290E-06,
5.43552393E-09,
-1.32310554E-11,
2.13239465E-14,
-2.25744420E-17,
1.50864395E-20,
-5.76865768E-24,
9.61540784E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
return y*1e3;
}
```

```
  public static double ro(double T,double P)
  {   double M=2.8965000E+01;
      double R=8314.5/M;
                return P*1e3/R/T;
  }
  public static double Nu(double Ts,double Tinf,double P,double L,double teta)
  {//Nusselt number
   double T=(Tinf+Ts)/2.0;
   double Ra=Ra(Ts,Tinf,P,L,teta);
   double Pr=Pr(T);
   double Gr=Ra/Pr;
   double Nu=0.0;
   double Racritic=1e9;
   double Pr_05=Math.sqrt(Pr);
  Nu=(0.825+0.387*Math.pow(Ra,(1.0/6.0))/Math.pow((1+Math.pow((0.492/Pr),(9.0/16.0))),(8.0/27.0)));
   Nu=Nu*Nu;
   return Nu;
   }

  public static double Ra(double Ts,double Tinf,double P,double L,double teta)
  {   double T=(Ts+Tinf)/2.0;
                double mu=mu(T);
     double ro=ro(T,P);
     double Cp=Cp(T);
     double k=k(T);
     double beta=1/T;
     double nu=mu/ro;
     double alpha=k/(ro*Cp);
     double g=9.806; //m/s^2
     double Ra=g*Math.cos(teta)*beta*(Ts-Tinf)*L*L*L/alpha/nu;
     System.out.println("Ra="+Ra);
                return Ra;
  }
  public static void main(String arg[])
  { double Tinf=26.85+273.15;  //air temp. degree K;
    double Ts=126.85+273.15;
    double L=0.5;
    double teta=45.0*Math.PI/180.0;
    System.out.println("teta="+teta);
    double W=1.0;
    double P=101.325; //kPa
    double Nu=Nu(Ts,Tinf,P,L,teta);
    System.out.println("Nu="+Nu);
    double T=(Tinf+Ts)/2.0;
    double k=k(T);;
    double h=Nu*k/L;
    System.out.println("h="+h);
    double A=W*L;
    double Q=h*A*(Ts-Tinf);
    System.out.println("Q="+Q);
}}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_natural_inclined_plate_air
teta=0.7853981633974483
Ra=4.0568788171659166E8
Nu=93.00141918885832
h=5.580592721735006
Q=279.0296360867503
```

14.4 HORIZONTAL PLATES NATURAL CONVECTION



$L_c = \frac{A_s}{P}$ where A_s is the plate surface area and P is the perimeter

$$\mathrm{Ra}_x = \mathrm{Gr}_x Pr = \frac{g\beta(T_s - T_\infty)L_c^3}{\nu\alpha}$$

Upper surface of Hot Plate or Lower surface of Cold Plate

$\mathrm{Nu}_L = 0.54 Ra_L^{1/4} \quad 10^4 \le Ra_L \le 10^7$

$\mathrm{Nu}_L = 0.15 Ra_L^{1/3} \quad 10^7 \le Ra_L \le 10^{11}$

Lower surface of Hot Plate or Upper surface of Cold Plate

$\mathrm{Nu}_L = 0.27 Ra_L^{1/4} \quad 10^5 \le Ra_L \le 10^{10}$

PROBLEM : air temperature of a room is T_∞=26.85 C and room pressure is P=101.325 kPa. Room has a horizontal heating plate of height **L=0.5** m, and width **W=0.5 m**. Plate surface temperature is T_s=126.85 C. Find the heat transfer from natural convection.

```
public class HT_natural_horizontal_plate_air
{
  public static double mu(double T)
  {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double k(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
-4.40290627E-24,
7.28284943E-28};
```

```
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pr(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;

return y;
}
public static double Cp(double T)
{double a[]={1.89668613E+00,
-2.23937197E-02,
2.40885259E-04,
-1.45254290E-06,
5.43552393E-09,
-1.32310554E-11,
2.13239465E-14,
-2.25744420E-17,
1.50864395E-20,
-5.76865768E-24,
9.61540784E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
return y*1e3;
}

  public static double ro(double T,double P)
  {   double M=2.8965000E+01;
      double R=8314.5/M;
                return P*1e3/R/T;
  }
  public static double[] Nu(double Ts,double Tinf,double W,double L,double P)
  {//Nusselt number
   double T=(Tinf+Ts)/2.0;
   double A=W*L;
   double Pe=2.0*W+2.0*L;
   double Ls=A/Pe;
   double Ra=Ra(Ts,Tinf,Ls,P);
   double Pr=Pr(T);
   double Gr=Ra/Pr;
   double Nu[]=new double[2];
   double Racritic=1e9;
   double Pr_05=Math.sqrt(Pr);
   if(Ts>Tinf)
   {System.out.println("Hot plate \n");
              if(Ra>1e4 && Ra<=1e7) Nu[0]=0.54*Math.pow(Ra,0.25);
    else if (Ra>1e7 && Ra<=1e11) Nu[0]=0.15*Math.pow(Ra,(1.0/3.0));
    Nu[1]=0.27*Math.pow(Ra,(1.0/3.0));
   }
   else
   {System.out.println("Cold plate \n");
     if(Ra>1e4 && Ra<=1e7) Nu[1]=0.54*Math.pow(Ra,0.25);
    else if (Ra>1e7 && Ra<=1e11) Nu[1]=0.15*Math.pow(Ra,(1.0/3.0));
    Nu[0]=0.27*Math.pow(Ra,(1.0/3.0));
    }
   return Nu;
   }

  public static double Ra(double Ts,double Tinf,double L,double P)
  {   double T=(Ts+Tinf)/2.0;
                double mu=mu(T);
     double ro=ro(T,P);
     double Cp=Cp(T);
     double k=k(T);
```

```
    double beta=1/T;
    double nu=mu/ro;
    double alpha=k/(ro*Cp);
    double g=9.806; //m/s^2
    double Ra=g*beta*(Ts-Tinf)*L*L*L/alpha/nu;
            return Ra;
  }
  public static void main(String arg[])
  { double Tinf=26.85+273.15;  //air temp. degree K;
    double Ts=126.85+273.15;
    double L=0.5;
    double W=0.5;
    double A=L*W;
    double Pe=2.0*L+2.0*W;
    double Ls=A/Pe;
    System.out.println("Area = "+A+" Perimeter = "+Pe+"Ls = "+Ls);
    double P=101.325; //kPa
    double Nu[]=Nu(Ts,Tinf,W,L,P);
    System.out.println("Nu  upper surface  of the plate ="+Nu[0]+"\nNu lower surface of the plate ="+Nu[1]);
    double T=(Tinf+Ts)/2.0;
    double k=k(T);;
    double h[]=new double[2];
    h[0]=Nu[0]*k/Ls;
    h[1]=Nu[1]*k/Ls;
    System.out.println("h  upper surface  of the plate ="+h[0]+"W/m2K\nh lower surface of the plate ="+h[1]+" W/m2K");
    double Q1=h[0]*A*(Ts-Tinf);
    double Q2=h[1]*A*(Ts-Tinf);
    double Q=Q1+Q2;
    System.out.println("Q total ="+Q+"W Q upper surface ="+Q1+"W Q lower surface ="+Q2+"W ");
}}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_natural_horizontal_plate_air
Area = 0.25 Perimeter = 2.0Ls = 0.125
Hot plate

Nu  upper surface  of the plate =29.547825432626695
Nu lower surface of the plate =56.08836545050797
h  upper surface  of the plate =7.0921231521235795W/m2K
h lower surface of the plate =13.462432153707027 W/m2K
Q total =513.8638826457652W Q upper surface =177.30307880308948W Q lower surface =336.5608038426757W

> Terminated with exit code 0.
```

PROBLEM : A circular hot plate D=0.25 m in diameter with both surfaces are maintained at a uniform temparature of100 C is suspended in horizontal position in atmospheric air at 27 C. Determine the heat transfer rate by free convection from the plate into the atmosphere.

```
public class HT_natural_horizontal_plate_air1
{
  public static double mu(double T)
  {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double k(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
```

```
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pr(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;

return y;
}
public static double Cp(double T)
{double a[]={1.89668613E+00,
-2.23937197E-02,
2.40885259E-04,
-1.45254290E-06,
5.43552393E-09,
-1.32310554E-11,
2.13239465E-14,
-2.25744420E-17,
1.50864395E-20,
-5.76865768E-24,
9.61540784E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
return y*1e3;
}

  public static double ro(double T,double P)
  {   double M=2.8965000E+01;
      double R=8314.5/M;
                return P*1e3/R/T;
  }
  public static double[] Nu(double Ts,double Tinf,double W,double L,double P)
  {//Nusselt number
   double T=(Tinf+Ts)/2.0;
   double A=W*L;
   double Pe=2.0*W+2.0*L;
   double Ls=A/Pe;
   double Ra=Ra(Ts,Tinf,Ls,P);
   double Pr=Pr(T);
   double Gr=Ra/Pr;
   double Nu[]=new double[2];
   double Racritic=1e9;
   double Pr_05=Math.sqrt(Pr);
   if(Ts>Tinf)
   {System.out.println("Hot plate \n");
              if(Ra>1e4 && Ra<=1e7) Nu[0]=0.54*Math.pow(Ra,0.25);
    else if (Ra>1e7 && Ra<=1e11) Nu[0]=0.15*Math.pow(Ra,(1.0/3.0));
    Nu[1]=0.27*Math.pow(Ra,(1.0/3.0));
   }
   else
   {System.out.println("Cold plate \n");
     if(Ra>1e4 && Ra<=1e7) Nu[1]=0.54*Math.pow(Ra,0.25);
    else if (Ra>1e7 && Ra<=1e11) Nu[1]=0.15*Math.pow(Ra,(1.0/3.0));
    Nu[0]=0.27*Math.pow(Ra,(1.0/3.0));
    }
   return Nu;
   }

  public static double Ra(double Ts,double Tinf,double L,double P)
  {   double T=(Ts+Tinf)/2.0;
                double mu=mu(T);
     double ro=ro(T,P);
```

```
    double Cp=Cp(T);
    double k=k(T);
    double beta=1/T;
    double nu=mu/ro;
    double alpha=k/(ro*Cp);
    double g=9.806; //m/s^2
    double Ra=g*beta*(Ts-Tinf)*L*L*L/alpha/nu;
            return Ra;
  }
  public static void main(String arg[])
  { double Tinf=27+273.15;  //air temp. degree K;
    double Ts=100+273.15;
    double D=0.25;
    double W=0.5;
    double A=Math.PI*D*D/4.0;
    double Pe=Math.PI*D;
    double Lc=A/Pe;
    System.out.println("Area = "+A+" Perimeter = "+Pe+"Lc = "+Lc);
    double P=101.325; //kPa
    double Nu[]=Nu(Ts,Tinf,W,Lc,P);
    System.out.println("Nu  upper surface  of the plate ="+Nu[0]+"\nNu lower surface of the plate ="+Nu[1]);
    double T=(Tinf+Ts)/2.0;
    double k=k(T);;
    double h[]=new double[2];
    h[0]=Nu[0]*k/Lc;
    h[1]=Nu[1]*k/Lc;
    System.out.println("h  upper surface  of the plate ="+h[0]+"W/m2K\nh lower surface of the plate ="+h[1]+" W/m2K");
    double Q1=h[0]*A*(Ts-Tinf);
    double Q2=h[1]*A*(Ts-Tinf);
    double Q=Q1+Q2;
    System.out.println("Q total ="+Q+"W Q upper surface ="+Q1+"W Q lower surface ="+Q2+"W ");
}}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_natural_horizontal_plate_air1
Area = 0.04908738521234052 Perimeter = 0.7853981633974483Lc = 0.0625
Hot plate

Nu  upper surface  of the plate =9.240595251299844
Nu lower surface of the plate =11.906168242534513
h  upper surface  of the plate =4.295655076137223W/m2K
h lower surface of the plate =5.534794096862192 W/m2K
Q total =35.22622631167103W Q upper surface =15.392960708703647W Q lower surface =19.83326560296738W

> Terminated with exit code 0.
```

14.5 LONG HORIZONTAL CYLINDER NATURAL CONVECTION

Grashoff Number: $\mathrm{Gr}_D = \frac{g\beta(T_s - T_\infty)D^3}{\nu^2}$

Rayleigh Number: $\mathrm{Ra}_D = \mathrm{Gr}_D Pr = \frac{g\beta(T_s - T_\infty)D^3}{\nu\alpha}$

Morgan equation[37]

$\mathrm{Nu}_D = \frac{hD}{k} = CRa_D^n$

Ra_D	C	n
10^{-10}-10^{-2}	0.675	0.058
10^{-2}-10^{2}	1.02	0.148
10^{2}-10^{4}	0.850	0.188
10^{4}-10^{7}	0.480	0.250
10^{7}-10^{12}	0.125	0.333

Churchill & Chu equation[38]

$$\mathrm{Nu}_D = \left\{0.60 + \frac{0.387Ra_D^{1/6}}{\left[1+\left(\frac{0.559}{\mathrm{Pr}}\right)^{9/16}\right]^{8/27}}\right\}^2 \quad Ra_D \le 10^{12}$$

PROBLEM : air temperature of a room is T_∞=26.85 C and room pressure is P=101.325 kPa. Room has a horizontal pipe of length L=10 m and diameter D=0.01 m. Pipe surface temperature is constant and T_s=126.85 C. Find the heat transfer from naural convection.

```
public class HT_natural_horizontal_cylinder_air
```

```
{
  public static double mu(double T)
  {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double k(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pr(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;

return y;
}
public static double Cp(double T)
{double a[]={1.89668613E+00,
-2.23937197E-02,
2.40885259E-04,
-1.45254290E-06,
5.43552393E-09,
-1.32310554E-11,
2.13239465E-14,
-2.25744420E-17,
1.50864395E-20,
-5.76865768E-24,
9.61540784E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
return y*1e3;
}
  public static double ro(double T,double P)
  {   double M=2.8965000E+01;
      double R=8314.5/M;
                return P*1e3/R/T;
  }
  public static double Nu(double Ts,double Tinf,double P,double D,int select)
  {//Nusselt number
   double T=(Tinf+Ts)/2.0;
   double Ra=Ra(Ts,Tinf,P,D);
```

```
   System.out.println("Ra="+Ra);
   double Pr=Pr(T);
   double Gr=Ra/Pr;
   double Nu=0.0;
   double Racritic=1e9;
   double C=0,n=0;
   double Pr_05=Math.sqrt(Pr);
   if(select==1) {
                double x1=0.387*Math.pow(Ra,(1.0/6.0));
                Nu=(0.60+0.387*Math.pow(Ra,(1.0/6.0))/Math.pow((1+Math.pow((0.559/Pr),(9.0/16.0))),(8.0/27.0)));
     Nu=Nu*Nu;
   }
   else if(select==2)
   {  if(Ra>=1e-10 && Ra<1e-2)    {C=0.675;n=0.058;}
      else if(Ra>=1e-2 && Ra<1e2) {C=1.02;n=0.148;}
      else if(Ra>=1e2 && Ra<1e4) {C=0.85;n=0.188;}
      else if(Ra>=1e4 && Ra<1e7)  {C=0.480;n=0.250;}
      else if(Ra>=1e7 && Ra<=1e12){C=0.125;n=0.333;}
                Nu=C*Math.pow(Ra,n);
                System.out.println("C="+C+"n="+n);
   }
   return Nu;
   }

  public static double Ra(double Ts,double Tinf,double P,double D)
  {   double T=(Ts+Tinf)/2.0;
                double mu=mu(T);
     double ro=ro(T,P);
     double Cp=Cp(T);
     double k=k(T);
     double beta=1/T;
     double nu=mu/ro;
     double alpha=k/(ro*Cp);
     double g=9.806; //m/s^2
     double Ra=g*beta*(Ts-Tinf)*D*D*D/alpha/nu;

                return Ra;
  }
  public static void main(String arg[])
  { double Tinf=26.85+273.15;  //air temp. degree K;
    double Ts=126.85+273.15;
    double D=0.01; // m
    double L=10.0; //m
    double P=101.325; //kPa
    double Nu1=Nu(Ts,Tinf,P,D,1);
    double Nu2=Nu(Ts,Tinf,P,D,2);
    System.out.println("Nu1="+Nu1+" Nu2="+Nu2);
    double k1=k(Tinf);
    double h1=Nu1*k1/D;
    double T=(Tinf+Ts)/2.0;
    double k2=k(T);
    double h2=Nu2*k2/D;
    System.out.println("h1="+h1+" h2="+h2);
    double A=Math.PI*D*L;
    double Q1=h1*A*(Ts-Tinf);
    double Q2=h2*A*(Ts-Tinf);
    System.out.println("Q1="+Q1+" Q2="+Q2);
}}
```

```
 ---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_natural_horizontal_cylinder_air
Ra=4589.834435312127
Ra=4589.834435312127
C=0.85n=0.188
Nu1=3.6422471233262153 Nu2=4.147981067277606
h1=9.609798786306515 h2=12.445075115970486
Q1=301.90073269536657 Q2=390.97356557706024

> Terminated with exit code 0.
```

Heat transfer natural convection horizontal cylinder			**working fluid**		
M	**2.8965000E+01**	**kg/kmol**			
R	**2.8705334E+02**	**J/kgK**			
P	**1.0132500E+02**	**kPa**			
	ρ	Cp	μ	k	Pr

	kg/m^3				
a0		1.89668613E+0	-	-	1.47352515E+0
a1		-2.23937197E-	8.85836561E	1.00247973E	-1.73708682E-
a2		2.40885259E-	-	7.32722776E	1.80473009E-
a3		-1.45254290E-	1.73074497E	-	-1.07436904E-
a4		5.43552393E-	-	4.21691040E	3.99304707E-
a5		-1.32310554E-	-	-	-9.67956666E-
a6		2.13239465E-	3.14868466E	1.67088379E	1.55567247E-
a7		-2.25744420E-	-	-	-1.64359568E-
a8		1.50864395E-	3.25615211E	1.16134919E	1.09674418E-
a9		-5.76865768E-	-	-	-4.18869082E-
a10		9.61540784E-	2.16661936E	7.28284943E	6.97523663E-
result:for T	1.0085234E+00	1.0083819E+00	2.0867141E-	3.0002729E-	7.0130741E-01
result: for Tw	8.8245794E-01	1.0140736E+00	2.3055411E-	3.3453105E-	6.9887896E-01

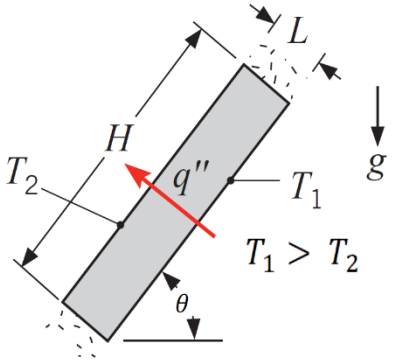
	degree C	**degree K**	
T	7.6850000E+01	3.5000000E+02	average
Ts	1.2685000E+02	4.0000000E+02	surface
Tinf	2.6850000E+01	3.0000000E+02	air temp
P	1.0132500E+00	bar	air pressure
ρ	1.0085234E+00		air density
L	1.0000000E+01		length of
D	1.0000000E-02		diameter of
A	3.1415927E-01		
k	3.0002729E-02		air thermal
Cp	1.0083819E+03		air specific
Pr	7.0130741E-01		air Prandtl
α	2.9501884E-05	W/mK	air thermal
g	9.8060000E+00	m/s^2	gravitational
β	2.8571429E-03	W/m	thermal
μ	2.0867141E-05	Pas	dynamic
ν	2.0690786E-05	m^2/s	kinematic
Ra	4.5898353E+03		Rayleigh
Gr	6.5446838E+03		
Churchil & Chu **Nu1**	3.6422472E+00		Nusselt
h1	1.0927736E-02	W/(m2K)	thermal
Q1	3.4330494E-01	W	Rayleigh
C	**8.5000000E-01**		
n	**1.8800000E-01**		
Morgan **Nu2**	4.1479812E+00		Nusselt
h1	1.2445076E-02	W/(m2K)	thermal
Q1	1.9548679E-01	W	Rayleigh

14.5 NATURAL CONVECTION HEAT TRANSFER IN ENCLOSURES

Horizontal cavity (θ = 0)

Globe and Dropkin equation[61]

$$\mathrm{Nu}_L = 0.069\mathrm{Ra}_L^{1/3} Pr^{0.074}$$



vertical rectangular cavity (θ = 90)

Catton[62] equations

$\mathrm{Nu}_L = 0.22\left(\frac{Pr}{0.2+Pr}Ra_L\right)^{0.28}\left(\frac{H}{L}\right)^{-1/4} \quad 2 \le \left(\frac{H}{L}\right) \le 10 \quad Pr \le 10^5 \quad 10^3 \le Ra_L \le 10^{10}$

$\mathrm{Nu}_L = 0.18\left(\frac{Pr}{0.2+Pr}Ra_L\right)^{0.29} \quad 1 \le \left(\frac{H}{L}\right) \le 2 \quad 10^{-5} \le Pr \le 10^5 \quad 10^3 \le \frac{Ra_L Pr}{0.2+Pr}$

McGregor[63] equations

$\mathrm{Nu}_L = 0.42(Ra_L)^{1/4}\left(\frac{H}{L}\right)^{-0.3} \quad 10 \le \left(\frac{H}{L}\right) \le 40 \quad 1 \le Pr \le 2x10^4 \quad 10^4 \le Ra_L \le 10^7$

$\mathrm{Nu}_L = 0.046(Ra_L)^{1/3} \quad 1 \le \left(\frac{H}{L}\right) \le 40 \quad 1 \le Pr \le 20 \quad 10^6 \le Ra_L \le 10^9$

inclined rectangular cavity

Hollands at al.[64] equation

$X = \left[1 - \frac{1708}{Ra_L \cos(\theta)}\right] \quad Y = \left[\left(\frac{Ra_L \cos(\theta)}{5830}\right) - 1\right] \quad \theta^*$

If (X<0) X=0 If (Y<0) Y=0

$\mathrm{Nu}_L = 1 + 1.44X\left[1 - \frac{1708(\sin(1.8\theta)^{1.6}}{Ra_L \cos(\theta)}\right] + Y \quad 12 \le \left(\frac{H}{L}\right) \quad 0 \le \theta \le \theta^*$

Where

$\left(\frac{H}{L}\right)$	θ^* (radian)	θ^* (degree)
1	0.436332313	25
3	0.925024504	53
6	1.047197551	60
12	1.169370599	67
20	1.221730476	70
30	1.221730476	70
50	1.221730476	70
60	1.221730476	70

Equation is valid when θ is not exceeded critical tilt angle θ^* which is given in the table above as a function of H/L. Beyond critical tilt angle, θ^*, The following correlations can be used:

Ayyaswamy and Catton[66] equation

$\mathrm{Nu}_L = \mathrm{Nu}_L(\theta = 90°)[\sin(\theta)]^{1/4} \quad \theta^* \le \theta \le 90°$

Arnold et al. [65] equation

$\mathrm{Nu}_L = 1 + [\mathrm{Nu}_L(\theta = 90°) - 1]\sin(\theta) \quad 90° \le \theta \le 180°$

Free convection in annular space between long horizontal cylinders

$L_C = \frac{2\left[\ln\left(\frac{r_o}{r_i}\right)\right]^{4/3}}{\left(r_i^{-3/5} + r_o^{-3/5}\right)^{5/3}}$

Rayleigh Number: $\mathrm{Ra}_{L_C} = \mathrm{Gr}_{L_C} Pr = \frac{g\beta(T_i - T_o)L_C^3}{\nu\alpha}$

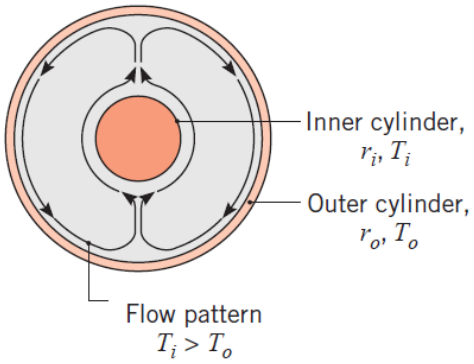
$q = \frac{2\pi L k_{eff}(T_i - T_o)}{\ln\left(\frac{r_o}{r_i}\right)} \quad 0.7 \le Pr \le 6000 \quad \mathrm{Ra}_{L_C} \le 10^7$

$R = 0.386\left(\frac{Pr}{0.861 + Pr}\right)^{1/4} Ra_{L_C}^{1/4}$

If R >1 $k_{eff} = kR$

Else $k_{eff} = k$

Properties should be calculated at mean value $T_m = \frac{T_i + T_o}{2}$



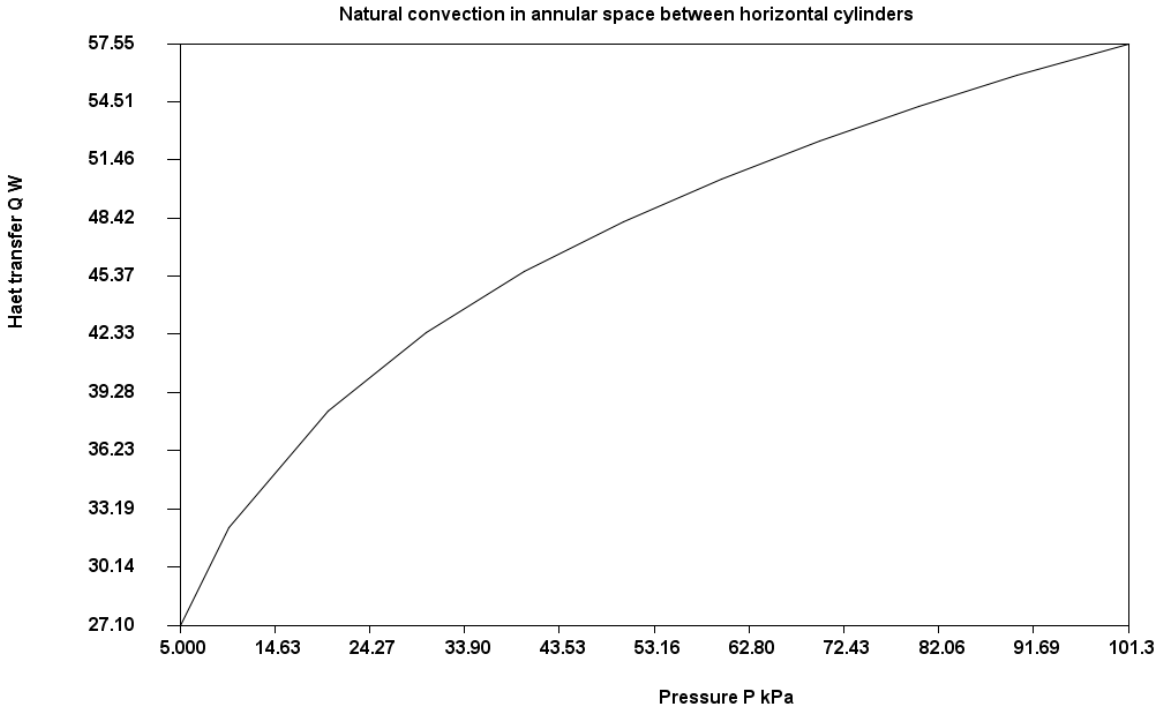
PROBLEM: A solar heater consist of two annular concentric cyclinders with radius r_0=0.08 m and r_i=0.05 m respectively. Temperature of the inner cyclinder is 80 ℃ and outside cylinder is 25 ℃. There is air in between cylinders. Calculate heat transfer for different air pressures in the annular space.

```
public class HT_natural_annular_space_betwen_horizontal_cylinders_air
{  public air ai;
  // for air T degree K P kPa
  // 0 T 1 P 2 ro 3 h 4 s 5 Cp 6 k 7 mu 8 alpha 9 Pr
  public double ri,ro,L;
  public double Lc;
  public double R;
  public double mu,rod,Cp,k,beta,nu,alpha,g,Ra,Pr,keff,q;
  public HT_natural_annular_space_betwen_horizontal_cylinders_air(double rii,double roi,double Li)
  {ri=rii;ro=roi;L=Li;Lc=2.0*Math.pow(Math.log(ro/ri),(4.0/3.0))/(1.0/Math.pow(ri,(3.0/5.0))+1.0/Math.pow(ro,(3.0/5.0)));
   ai=new air();}
  public double Ra(double Ti,double To,double P)
  {  //Ti and To in degree K
              double T=(Ti+To)/2.0;
              double b[]=ai.property_TP(T,P);
              mu=b[7];
    rod=b[2];
    Cp=b[5];
    Pr=b[9];
    k=b[6];
    beta=1/T;
    nu=mu/ro;
    alpha=k/(rod*Cp*1e3);
    g=9.806; //m/s^2
    double Ra=g*beta*(Ti-To)*Lc*Lc*Lc*Lc/alpha/nu;
              return Ra;
  }
  public double keff(double Ti,double To,double P)
  {Ra=Ra(Ti,To,P);
   R=0.386*Math.pow((Pr*Ra/(0.861+Pr)),0.25);
   if(R>1) keff=R*k;
   else keff=k;
  return keff;
  }
  public double q(double Ti,double To,double P)
  {keff(Ti,To,P);
   q=2.0*Math.PI*L*keff*(Ti-To)/Math.log(ro/ri);
   return q;
  }

  public static void main(String arg[])
  { double To=25.0+273.15;  //air temp. degree K;
    double Ti=80.0+273.15;  //air temp. degree K;
    double ro=0.08; // m
    double ri=0.05; //m
    double L=1.0; //m
    double P[]={5,10.0,20.0,30.0,40.0,50.0,60.0,70.0,80.0,90.0,101.325}; //kPa
    int n=P.length;
    double Q[]=new double[n];
    HT_natural_annular_space_betwen_horizontal_cylinders_air hnc=
    new HT_natural_annular_space_betwen_horizontal_cylinders_air(ri,ro,L);
    for(int i=0;i<n;i++)
    {Q[i]=hnc.q(Ti,To,P[i]);
     System.out.println("P= "+P[i]+"kPa Q="+Q[i]+" W ");
    }
```

```
    Plot pp=new Plot(P,Q);
    pp.setPlabel("Natural convection in annular space between horizontal cylinders");
    pp.setXlabel("Pressure P kPa");
    pp.setYlabel("Heat transfer Q W");
    pp.plot();
}}
```

```
 ---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_natural_annular_space_betwen_horizontal_cylinders_air
P= 5.0kPa Q=27.097997041922884 W
P= 10.0kPa Q=32.22714154872674 W
P= 20.0kPa Q=38.32918526528419 W
P= 30.0kPa Q=42.42275732781902 W
P= 40.0kPa Q=45.590889325339255 W
P= 50.0kPa Q=48.21141280439816 W
P= 60.0kPa Q=50.465046118766324 W
P= 70.0kPa Q=52.45343211577841 W
P= 80.0kPa Q=54.23990146700251 W
P= 90.0kPa Q=55.86681311951242 W
P= 101.325kPa Q=57.55394830809767 W
```

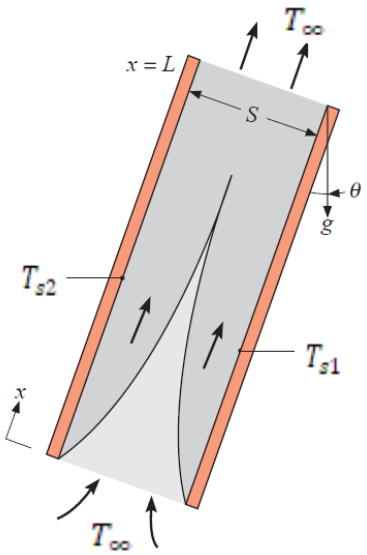


14.6 NATURAL CONVECTION HEAT TRANSFER IN CHANNELS

For symmetrical heated, isothermal plates Elenbaas[58] equation is as folows:

$$\mathrm{Nu}_S = \left(\frac{q/A}{T_s - T_\infty}\right)\frac{S}{k} = \frac{1}{24} Ra_S \left(\frac{S}{L}\right)\left[1 - exp\left(-\frac{35}{Ra_S\left(\frac{S}{L}\right)}\right)\right]^{3/4}$$

where **Rayleigh Number:** $\mathrm{Ra}_S = \frac{g\beta(T_s - T_\infty)S^3}{\nu\alpha}$ $\quad 10^{-1} \leq \mathrm{Ra}_S \leq 10^5$



For constant heat flux cases

$\mathrm{Nu}_S = \left(\frac{q"_s}{T_s - T_\infty}\right)\frac{S}{k} \quad \mathrm{Ra}_S^* = \frac{g\beta q"_s S^4}{\mathrm{k}\nu\alpha}$

for symmetric fully developed constant heat flux

$\mathrm{Nu}_{SL} = 0.144[\mathrm{Ra}_S^*(S/L)]^{1/2}$

for asymmetric fully developed constant heat flux

$\mathrm{Nu}_{SL} = 0.204[\mathrm{Ra}_S^*(S/L)]^{1/2}$

Bar-Cohen-Rohsenow[59] equation:

For isothermal plates

$$\mathrm{Nu}_{SL} = \left[\frac{576}{\left(Ra_S\left(\frac{S}{L}\right)^2\right)} + \frac{2.87}{\left(Ra_S\left(\frac{S}{L}\right)^{1/2}\right)}\right]^{-1/2} \quad 10 \le \mathrm{Ra}_S \le 100 \quad T_{s1} = T_{s2} \text{ symmetric isothermal}$$

$$\mathrm{Nu}_{SL} = \left[\frac{144}{\left(Ra_S\left(\frac{S}{L}\right)^2\right)} + \frac{2.87}{\left(Ra_S\left(\frac{S}{L}\right)^{1/2}\right)}\right]^{-1/2} \quad 10 \le \mathrm{Ra}_S \le 100 \quad T_{s1},\ q_{s2}" = 0 \text{ isothermal adiabatic}$$

For constant heat flux

$$\mathrm{Nu}_{SL} = \left[\frac{48}{\mathrm{Ra}_S^*\left(\frac{S}{L}\right)} + \frac{2.51}{\left(\mathrm{Ra}_S^*\left(\frac{S}{L}\right)^{2/5}\right)}\right]^{-1/2} \quad 10 \le \mathrm{Ra}_S^* \le 100 \quad q"_{s1} = q"_{s2}$$

$$\mathrm{Nu}_{SL} = \left[\frac{24}{\mathrm{Ra}_S^*\left(\frac{S}{L}\right)} + \frac{2.51}{\left(\mathrm{Ra}_S^*\left(\frac{S}{L}\right)^{2/5}\right)}\right]^{-1/2} \quad 10 \le \mathrm{Ra}_S^* \le 100 \quad q"_{s1}=\mathrm{q}"_{s2} = 0$$

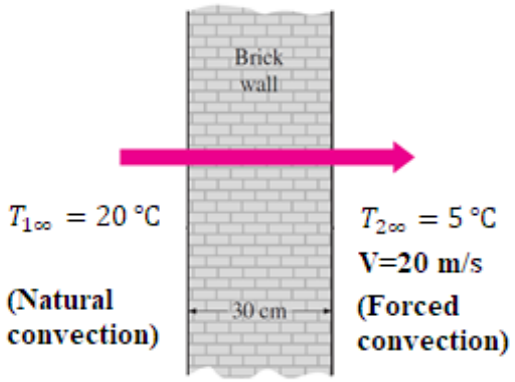
Inclined channels

Azevedo, L. F. A., and E. M. Sparrow[60] equation

$$\mathrm{Nu}_{SL} = 0.645\left[Ra_S\left(\frac{S}{L}\right)\right]^{1/4} \quad 200 \le Ra_S\left(\frac{S}{L}\right) \quad 0 \le \theta \le 45°$$

Where fluid properties are evaluated at $\mathrm{T}_f = \frac{T_s + T_\infty}{2}$

PROBLEM: Surface of a brick wall of a 5 m x 5 m of thikness 0.3 m and thermal conductivity k=0.7 W/mK exposed of still air $T_{1\infty}$ =20 C on one side (inside of the room) and $T_{2\infty}$ =5 C and wind speed V=20 m/s on the other side(outside). Determine the heat transfer through the wall.




```
public class HT_wall_natural_forced
{   //air propertiws 101.325 kPa
public static double mu(double T)
{double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double k(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pr(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;

return y;
}
public static double Cp(double T)
{double a[]={1.89668613E+00,
-2.23937197E-02,
2.40885259E-04,
-1.45254290E-06,
5.43552393E-09,
-1.32310554E-11,
```

```
2.13239465E-14,
-2.25744420E-17,
1.50864395E-20,
-5.76865768E-24,
9.61540784E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
return y*1e3;
}

  public static double ro(double T,double P)
  {   double M=2.8965000E+01;
     double R=8314.5/M;
                return P*1e3/R/T;
  }
  //natural convection
  public static double h_natural(double Ts,double Tinf,double P,double L,int select)
  {//Nusselt number
   double T=(Tinf+Ts)/2.0;
   double Ra=Ra(Ts,Tinf,P,L);
   double Pr=Pr(T);
   double k=k(T);
   double Gr=Ra/Pr;
   double Nu=0.0;
   double Racritic=1e9;
   double Pr_05=Math.sqrt(Pr);
   if(Ra>Racritic || select==4 ) {Nu=(0.825+0.387*Math.pow(Ra,(1.0/6.0))/Math.pow((1+Math.pow((0.492/Pr),(9.0/16.0))),(8.0/27.0)));
                Nu=Nu*Nu;
               }
   else
   {if(select==1) Nu=4.0/3.0*Math.pow((Gr/4.0),(1.0/4.0))*0.75*Pr_05/Math.pow((0.609+1.221*Pr_05+1.238*Pr),(1.0/4.0));
    else if(select==2) Nu=4.0/3.0*Math.pow((Gr/4.0),(1.0/4.0))*(0.022830145+0.752624416*Pr_05+2.04e-
4*Pr)/Math.pow((1.222236219+1.010421489*Pr_05+1.389905434*Pr),(1.0/4.0));
    else if(select==3) Nu=(0.68+0.67*Math.pow(Ra,(1.0/4.0))/Math.pow((1+Math.pow((0.492/Pr),(9.0/16.0))),(4.0/9.0)));
   }
   return Nu*k/L;
   }

static double Ra(double Ts,double Tinf,double P,double L)
  {   double T=(Ts+Tinf)/2.0;
                double mu=mu(T);
     double ro=ro(T,P);
     double Cp=Cp(T);
     double k=k(T);
     double beta=1/T;
     double nu=mu/ro;
     double alpha=k/(ro*Cp);
     double g=9.806; //m/s^2
     double Ra=g*beta*(Tinf-Ts)*L*L*L/alpha/nu;
                return R  public a;
  }
  //forced convection
  public static double hc(double T,double P,double V,double L)
  { double Rec=5e5;
    double ro=ro(T,P); //kg/m^3
    double mu=mu(T);
    double k=k(T); //W/mK
    double Pr=Pr(T);
    double Nu=0.0;
    double Re=V*L*ro/mu;
    double Nu1=0.664*Math.sqrt(Re)*Math.pow(Pr,(1.0/3.0));
    double Nu2=0.036*(Math.pow(Re,0.8)-9200.0)*Math.pow(Pr,0.43);
    if(Re<Rec) Nu=Nu1;
    else       Nu=Nu2;
    double h1=Nu*k/L;
    return h1;

  }
  public static void main(String arg[])
  { //Assume wall temperature for the first iteration:
    double T1w=10.0+273.15; //Degree Kelvin
    double T2w=10.0+273.15; //Degree Kelvin
    double T2inf=20.0+273.15; //Degree K
    double T1inf=5.0+273.15; //Degree K
    double V=10.0; // m/s;
    double kw=0.7; // W/mK
    double P=101.325; //kPa
```

```
    double L=5.0; //m
    double W=5.0; //m
    double dl=0.3; //m wall thickness
    double T1,T2;
    double Ts,Tw;
    double h1,h2,q;
    for(int i=0;i<10;i++)
    { T1=(T1inf+T1w)/2.0;
      T2=(T1inf+T1w)/2.0;
      h1=hc(T1,P,V,L);
      h2=h_natural(T2w,T2inf,P,L,3);
      q=(T2inf-T1inf)/(1/h1+dl/kw+1/h2);
      System.out.println(i+"th iteration q = "+q+" W/m^2"+" h1="+h1+" W/m^2K"+" h2="+h2+"W/m^2K");
      T1w=T1inf+q/h1;
      T2w=T2inf-q/h2;
    }
  }
}
```

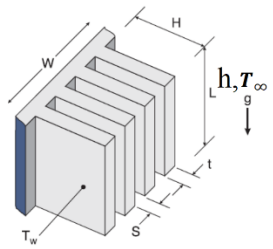
```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_wall_natural_forced
0th iteration q = 18.69720100944359 W/m^2 h1=25.656013858739414 W/m^2K h2=2.987656732151236W/m^2K
1th iteration q = 17.479844876767306 W/m^2 h1=25.790802530817484 W/m^2K h2=2.558943705860056W/m^2K
2th iteration q = 17.708521476882193 W/m^2 h1=25.792421798856104 W/m^2K h2=2.6336073981509065W/m^2K
3th iteration q = 17.667363989114605 W/m^2 h1=25.792141687974084 W/m^2K h2=2.6199946810550387W/m^2K
4th iteration q = 17.674819839975015 W/m^2 h1=25.792192109119913 W/m^2K h2=2.622454925397789W/m^2K
5th iteration q = 17.673470760107552 W/m^2 h1=25.79218297474933 W/m^2K h2=2.6220095745099843W/m^2K
6th iteration q = 17.673714917586533 W/m^2 h1=25.79218462753219 W/m^2K h2=2.6220901682776803W/m^2K
7th iteration q = 17.67367073147667 W/m^2 h1=25.792184328409938 W/m^2K h2=2.6220755827142694W/m^2K
8th iteration q = 17.67367872806091 W/m^2 h1=25.7921843825432 W/m^2K h2=2.622078222330986W/m^2K
9th iteration q = 17.67367728088033 W/m^2 h1=25.792184372746483 W/m^2K h2=2.622077744626547W/m^2K
```

14.7 SOME SIMPLIFIED NATURAL CONVECTION EQUATIONS FOR ATHMOSPHERIC AIR

In most of the cases we are dealing atmospheric air at reasonabley low temperatures. If that is the case some simplified formulations can be given for average air heat transfer for 101.325 kPa pressure.

		Ra	
Vertical Plates	Laminar	$10^4 – 10^9$	$h = 1.42\left(\frac{\Delta T}{L}\right)^{1/4}$
	Turbulent	$10^9 – 10^{13}$	$h = 1.31\Delta T^{1/3}$
Horizontal Cylinders	Laminar	$10^4 – 10^9$	$h = 1.32\left(\frac{\Delta T}{D}\right)^{1/4}$
	Turbulent	$10^9 – 10^{12}$	$h = 1.24\Delta T^{1/3}$
Horizontal plates Upper surface hot or lower surface cold L=A/P A:area P: perimeter	Laminar	$10^5 – 2x10^7$	$h = 1.32\left(\frac{\Delta T}{L}\right)^{1/4}$
	Turbulent	$2x10^7 – 3x10^{10}$	$h = 1.52\Delta T^{1/3}$
Horizontal plates Upper surface cold or lower surface hot	Laminar	$3x10^5 – 3x10^7$	$h = 0.59\left(\frac{\Delta T}{L}\right)^{1/4}$

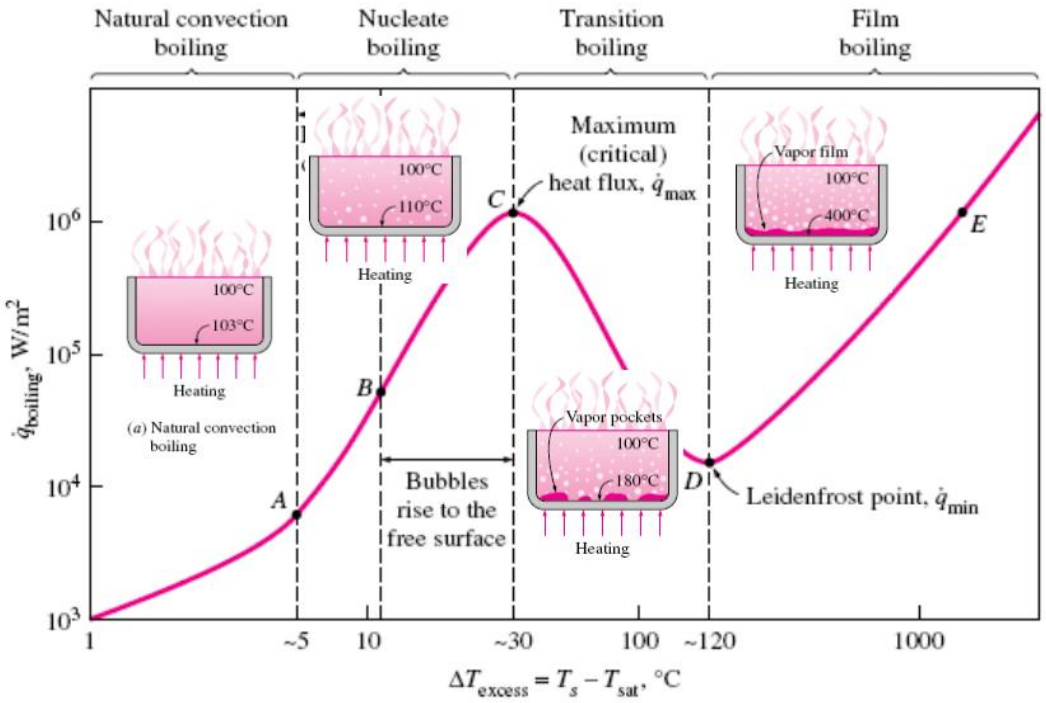
For a natural convective heat sink:

	$S_{opt} = 2.714 \frac{L}{Ra^{1/4}}$ $h = 1.31 \frac{k}{S_{opt}}$

15.0 BOILING AND CONDENSATION HEAT TRANSFER

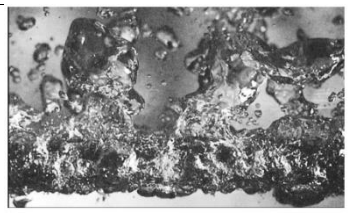
15.1 POOL BOILING HEAT TRANSFER

When a liquid started to heat up by using a wire or surface warmer than the boiling temperature of the liquid, it wil first heat up untill it reaches to the boiling temperature. The bubles of vapor forms around heated surface and rise upwards. This phase of the boiling process is called nucleate boiling. When heating surface covered completely with the vapor phase, heat transfer dropes due to lower heat transfer coefficient of vapor compare to liquid, when the temperature difference increases radiation heat transfer take over and heat transfer started to rise again due to radiative heat transfer. This region is called film boiling.

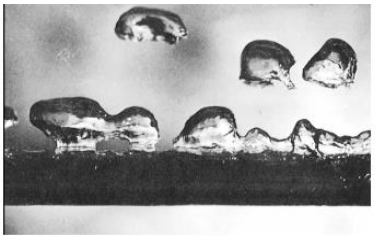




Nucleate boling



Transition boiling



Film boiling

Figre 15.1 Pool boilibg heat transfer zones

In the free convective heating region natural convection equations will be used. Nucleate boiling region is quite important and desirable region for heat transfer applications due to high heat transfer coefficient in this region. Rohsenow[45] equation is widely used heat transfer equation for the nucleate boiling region.

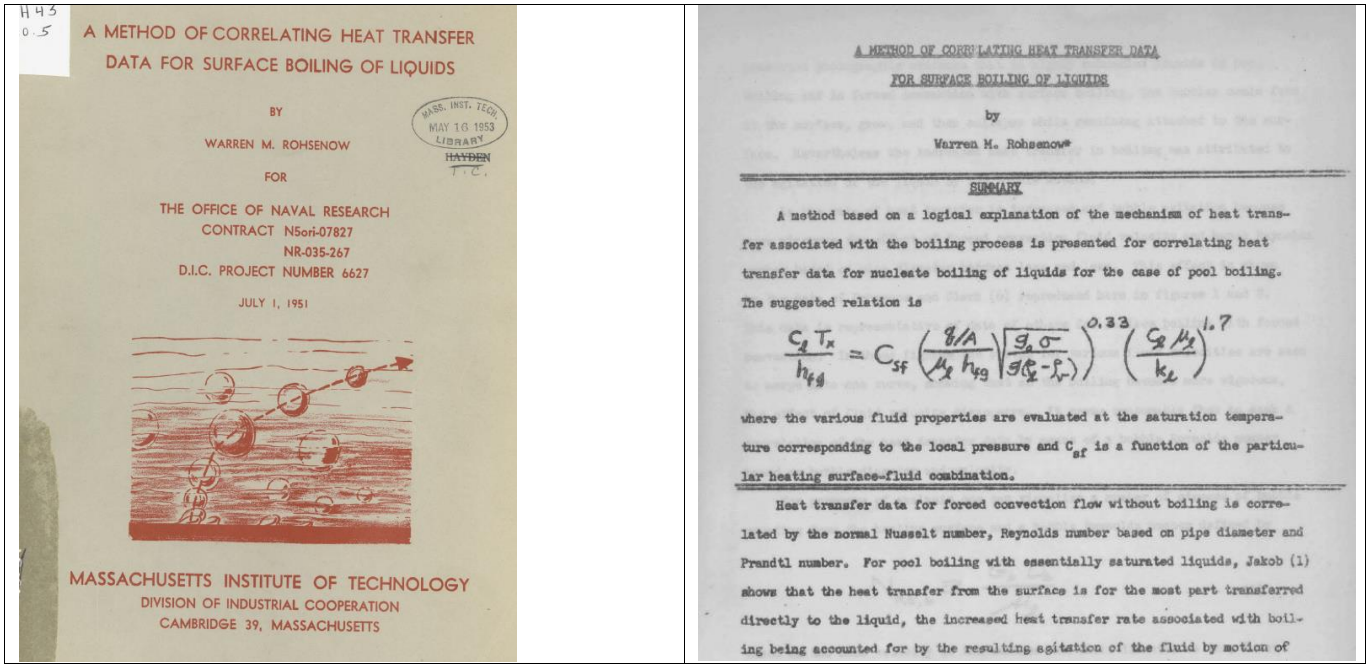
$$\frac{C_{pl}\Delta T_e}{h_{fg}} = C_{sf}\left[\frac{q}{h_{fg}\mu_l}\left(\frac{\sigma}{g(\rho_l-\rho_v)}\right)^{1/2}\right]^m Pr_l^n$$

Where μ_l is saturated liquid viscosity at boiling temperature, h_{fg} is saturation entalphy difference of liquid and vapor phases. g is the gravitational constant (can be taken as 9.806 m/s^2) , ρ_l and ρ_v saturated liquid and vapor densities. σ is surface tension. C_{pl} is specific heat of saturated liquid. Pr_l is Parndtl number of saturated liquid. $\Delta T_e = T_w - T_s$ temperature difference between Wall and satuartion fluid. The coefficient C_{sf} and n depends on the solid liquid combination. Pioro[46] given a detailed list of C_{sf} and n coefficients. Some additional values of the C_{sf} and n coefficients are taken from Jabardo et al.[50]. m coefficient in the equation was given as 0.33 in Rohsenow's original paper and n coeefficient was taken as 1 for water and 1.7 for other liquids.

C_{sf} and n coefficients

Fluid/surface	**C_{sf}**	**n**	**m**
Water:Copper (plate oxidized Ra=1.37 Rq=1.73)	0.015	0.81	0.33
Water:Copper (thin circular plate emery polished and paraffin treated)	0.0147	1	0.33
Water (int forced convection)/Copper (vertical tube)	0.013	1.7	0.33
Water:Copper (circular plate emery polished)	0.0128	1	0.33
Water:Copper (circular plate scored)	0.0068	1	0.33
Water:Aluminum (circulate plate polished R_q=0.33)	0.011	1	0.33
Water:Aluminum (plate oxidized\ Ra3.61 Rq=4.52)	0.011	1.26	0.33
Water/Brass(plate R_a=0.47, R_q=0.66	0.015	0.81	0.33
Water/Brass(tube ext)	0.009	1.1	0.33
Water:Chromium (polished thin layer on circular copper plate)	0.019	0.45	0.33
Water:Platinum (wire)	0.013	1	0.33
Water:St. St. (plate,Ra=0.75 Rq=1.2)	0.015	0.69	0.33
Water:St.St. (circular plate polished Rq=0.13)	0.01	1	0.33
Water:St.St. (circular plate grounded Rq=0.53)	0.008	1	0.33
Water:St.St. (circular plate grounded Rq=3.6)	0.007	1	0.33
Water:St[St[(thin strip pitted with Teflon coating in pitts)	0.0058	1	0.33
Ethanol:Copper (plate oxidized Ra=0.47, Rq=0.88)	0.079	2.3	0.33
Ethanol:Aluminum (plate oxidized, Ra=3.61, Rq=4.52)	0.08	1.18	0.33
Ethanol:Brass (plate Ra=0.47, Rq=0.66)	0.011	0.92	0.33
Ethanol:Chromium (polished thin layer on circular copper plate)	0.0045	1.47	0.33
Ethanol:St. St. (plate Ra=0.75 Rq=1.2)	0.00053	2.28	0.33
Methanol:Brass (tube ext)	0.0026	1.7	0.33
Iso-Propanol (int forced conv):Copper (vertical tube)	0.0022	1.7	0.33
n-Butanol(int forced conv)/Copper(vertical tube)	0.003	1.7	0.33
R-11:Copper (plate oxidized Ra=1.35 Rq=1.73)	0.009	3.47	0.33
R-12:Copper (tube ext)	0.0016	1.7	0.33
R-113:Copper plate (oxidized Ra=1.37 Rq=1.73)	0.0022	2.25	0.33
R-113:Aliminium plate (oxidized Ra=3.6 Rq=4.52)	0.013	1.20	0.33
CCl_4 (int forced conv):Copper (vertical tube)	0.013	1.7	0.33
CCl_4:Copper (circular plate emery polished)	0.007	1.7	0.33
CCl_4:Copper (circular plate lapped)	0.031	1.7	0.33
CCl_4:Brass (tube ext)	0.0022	2.1	0.33
Propane:Chromium (polished thin layer on circular copper plate)	0.0069	2.17	0.33
n-Pentane:Copper(circularplatemirror_finishing)	0.0171	1.7	0.33

n-Pentane:Copper (plate emery polished Rq=0.15)	0.0154	1.7	0.33
n-Pentane:Copper (circular plate emery rubbed)	0.0074	1.7	0.33
n-Pentane:Copper (circular plate lapped)	0.0049	1.7	0.33
n-Pentane:Zinc (circular plate polished)	0.0088	1.7	0.33
n-Pentane:Chromium (polished thin layer on circular copper plate)	0.015	1.7	0.33
n-Pentane:Nickel (circular plate mirror _finishing	0.0154	1.7	0.33
n-Pentane:Nickel (plate emery polished	0.0127	1.7	0.33
n-Pentane:Nickel (circular plate lapped)	0.0043	1.7	0.33
n-Pentane:Inconel(circularplatemirror_finishing)	0.018	1.7	0.33
n-Pentane:Inconel (circular plate lapped)	0.0072	1.7	0.33
Benzene:Chromium (polished thin layer circular copper plate)	0.01	1.7	0.33
Benzene:Platinium(polished)	0.019	1.7	0.33
n-Heptane:Chromium (polished thin layer on circular copper plate)	0.0014	1.37	0.33
Acetone:Copper (circular plate emery polished)	0.0096	1.7	0.33
35% K_2CO_3 (int forced conv):Copper (vertical tube)	0.0054	1.7	0.33
50% K_2CO_3 (int forced conv):Copper (vertical tube)	0.0028	1.7	0.33
R134a:Brass (Ra=0.07)	0.006	1.7	0.33
R134a:Stainless Steel (Ra=0.08)	0.008	1.7	0.33
R134a:Copper(Ra=0.07)	0.0073	1.7	0.33
R134a:Copper(Ra=0.5)	0.0053	1.7	0.33
R134a:Copper(Ra=2.5)	0.0032	1.7	0.33
R123:Brass (Ra=0.16)	0.007	1.7	0.33
R123:Stainless Steel (Ra=0.08)	0.008	1.7	0.33
R123:Copper(Ra=0.07)	0.007	1.7	0.33
R123:Copper(Ra=0.5)	0.0058	1.7	0.33
R123:Copper(Ra=2.5)	0.0036	1.7	0.33



H43
0.5

A METHOD OF CORRELATING HEAT TRANSFER DATA FOR SURFACE BOILING OF LIQUIDS

BY

WARREN M. ROHSENOW

FOR

THE OFFICE OF NAVAL RESEARCH
CONTRACT N5ori-07827
NR-035-267
D.I.C. PROJECT NUMBER 6627

JULY 1, 1951

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DIVISION OF INDUSTRIAL COOPERATION
CAMBRIDGE 39, MASSACHUSETTS

MASS. INST. TECH. MAY 16 1953 LIBRARY HAYDEN T.C.

A METHOD OF CORRELATING HEAT TRANSFER DATA
FOR SURFACE BOILING OF LIQUIDS

by

Warren M. Rohsenow

SUMMARY

A method based on a logical explanation of the mechanism of heat transfer associated with the boiling process is presented for correlating heat transfer data for nucleate boiling of liquids for the case of pool boiling. The suggested relation is

$$\frac{C_{\ell} T_x}{h_{fg}} = C_{sf} \left(\frac{q/A}{\mu_{\ell} h_{fg}} \sqrt{\frac{g_c \sigma}{g(\rho_{\ell} - \rho_v)}} \right)^{0.33} \left(\frac{C_{\ell} \mu_{\ell}}{k_{\ell}} \right)^{1.7}$$

where the various fluid properties are evaluated at the saturation temperature corresponding to the local pressure and C_{sf} is a function of the particular heating surface-fluid combination.

Heat transfer data for forced convection flow without boiling is correlated by the normal Nusselt number, Reynolds number based on pipe diameter and Prandtl number. For pool boiling with essentially saturated liquids, Jakob (1) shows that the heat transfer from the surface is for the most part transferred directly to the liquid, the increased heat transfer rate associated with boiling being accounted for by the resulting agitation of the fluid by motion of

Figure 15.2 Original Rohsenow paper: an important part of the history of the heat transfer

PROBLEM

Saturated water at T_{sat}= 100 C is boiled inside a emery polished copper pan having a heating surface A=2x10^{-2} m^2 which is maintained at a uniform temperature Tw = 108 ^{o}C. Calculate
(a)the suface heat flux and (b) the rate of evaporation.

$$\frac{C_{pl}\Delta T_e}{h_{fg}} = C_{sf}\left[\frac{q}{h_{fg}\mu_l}\left(\frac{\sigma}{g(\rho_l-\rho_v)}\right)^{1/2}\right]^m Pr_l^n$$

Temperature, T (K)	Pressure, p (bars)[b]	Specific Volume (m^3/kg) $v_f \cdot 10^3$	Specific Volume (m^3/kg) v_g	Heat of Vaporization, h_{fg} (kJ/kg)	Specific Heat (kJ/kg · K) $c_{p,f}$	Specific Heat (kJ/kg · K) $c_{p,g}$	Viscosity (N · s/m^2) $\mu_f \cdot 10^6$	Viscosity (N · s/m^2) $\mu_g \cdot 10^6$	Thermal Conductivity (W/m · K) $k_f \cdot 10^3$	Thermal Conductivity (W/m · K) $k_g \cdot 10^3$	Prandtl Number Pr_f	Prandtl Number Pr_g	Surface Tension, $\sigma_f \cdot 10^3$ (N/m)	Expansion Coefficient, $\beta_f \cdot 10^6$ (K^{-1})	Temperature, T (K)
345	0.3372	1.024	4.683	2329	4.191	1.941	389	10.89	668	22.6	2.45	0.933	64.1	595.4	345
350	0.4163	1.027	3.846	2317	4.195	1.954	365	11.09	668	23.0	2.29	0.942	63.2	624.2	350
355	0.5100	1.030	3.180	2304	4.199	1.968	343	11.29	671	23.3	2.14	0.951	62.3	652.3	355
360	0.6209	1.034	2.645	2291	4.203	1.983	324	11.49	674	23.7	2.02	0.960	61.4	697.9	360
365	0.7514	1.038	2.212	2278	4.209	1.999	306	11.69	677	24.1	1.91	0.969	60.5	707.1	365
370	0.9040	1.041	1.861	2265	4.214	2.017	289	11.89	679	24.5	1.80	0.978	59.5	728.7	370
373.15	1.0133	1.044	1.679	2257	4.217	2.029	279	12.02	680	24.8	1.76	0.984	58.9	750.1	373.15
375	1.0815	1.045	1.574	2252	4.220	2.036	274	12.09	681	24.9	1.70	0.987	58.6	761	375
380	1.2869	1.049	1.337	2239	4.226	2.057	260	12.29	683	25.4	1.61	0.999	57.6	788	380
385	1.5233	1.053	1.142	2225	4.232	2.080	248	12.49	685	25.8	1.53	1.004	56.6	814	385

Fluid/surface	**C_{sf}**	**n**	**m**
Water:Copper (plate oxidized Ra=1.37 Rq=1.73)	0.015	0.81	0.33
Water (int forced convection)/Copper (vertical tube)	0.013	1.7	0.33
Water:Copper (circular plate emery polished)	0.0128	1	0.33
Water:Copper (circular plate scored)	0.0068	1	0.33

```
/* a[0] "P, pressure        ";
   a[1] "T, temperature     ";
   a[2] "v, specific volume ";
   a[3] "h, enthalpy        ";
   a[4] "u, internal energy ";
   a[5] "s, entropy         ";
   a[6] "x, quality         ";
   a[7] "Cp, specific heat at constant pressure ";
   a[8] "Cv, specific heat at constant volume ";
   a[9] ""+'\u03B1'+" isobaric thermal expansion coefficient ";
   a[10] ""+'\u03BA'+" isothermal compressibility" ;
   a[11] ""+'\u03B7'+" Dynamic viscosity" ;
   a[12] "k Thermal conductivity" ;
   a[13] ""+'\u03C3'+" Surface tension" ;
   a[14] "Prandtl number" ;
   a[15] ""+'\u03C1'+" density" ;
   a[16] " speed of sound" ;
*/
public class HT_pool_boiling_water
{ public static void main(String arg[])
  { steamIAPWS_IF97 st=new steamIAPWS_IF97();
    double Ts=373.15;       //steam saturation temp. degree K;
    double Tw=108.0+273.15;   //wall temp. degree K
    double P=101.325; //kPa
    double al[]=st.property("tx",Ts,0.0);
    double av[]=st.property("tx",Ts,1.0);
    double kl=al[12];   // W/(mK)
    double mul=al[11];  //kg/(m.s)=Pa.s
    double muv=av[11];  //kg/(m.s)=Pa.s
    double Prl=al[14];
    double Cpl=al[7]*1e3;  //J/kgK
    double hfg=(av[3]-al[3])*1e3; //J/kg
    double rol=al[15];  //kg/m^3
    double rov=av[15];  //kg/m^3
    double sigma=al[13];  //N/m
    double Psat=al[0]*1e3; //Pa
    double g=9.806; //m/s^2
    double Csf=0.0128;
    double n=1;
    double m=0.33;
    double dTe=Tw-Ts;
    double q=Math.pow((Cpl*dTe/(hfg*Csf*Prl)),1.0/m)*hfg*mul*Math.sqrt(g*(rol-rov)/sigma); //m
   System.out.println("q="+q+"W/m^2");
}}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_pool_boiling_water
q=74253.99827665798W/m^2

> Terminated with exit code 0.
```

Excel version of the calculations (utilising polynomial curve fittings for property evaluation):

Excel version of the calculations (utilising polynomial curve fittings for property evaluation):

	1	2.00E+00	3.00E+00	4.00E+00	5.00E+00	6.00E+00	7.00E+00
	Psat(T)	ρl(T)	ρv(T)	hl(t)	hv(T)	si(T)	sv(T)
	6.76E-04	1.00E+03	6.16E-02	-1.43E-02	2.50E+03	1.06E-05	9.16E+00
	6.00E-06	1.16E-01	-3.22E-02	4.20E+00	1.85E+00	1.54E-02	-2.67E-02
	5.18E-06	-1.33E-02	3.10E-03	-1.73E-05	-2.30E-03	-2.61E-05	1.14E-04
	-1.19E-07	2.53E-04	-1.17E-04	-2.82E-05	7.55E-05	-1.04E-07	-1.08E-07
	3.22E-09	-4.08E-06	2.28E-06	8.21E-07	-1.63E-06	4.06E-09	-5.85E-09
	-3.19E-11	4.24E-08	-2.58E-08	-1.09E-08	1.90E-08	-5.22E-11	7.96E-11
	2.52E-13	-2.80E-10	1.78E-10	8.35E-11	-1.38E-10	3.83E-13	-5.74E-13
	-1.12E-15	1.16E-12	-7.64E-13	-3.86E-13	6.26E-13	-1.70E-15	2.50E-15
	2.95E-18	-2.95E-15	1.98E-15	1.07E-15	-1.71E-15	4.51E-18	-6.56E-18
	-4.28E-21	4.17E-18	-2.86E-18	-1.62E-18	2.58E-18	-6.60E-21	9.51E-21
	2.68E-24	-2.52E-21	1.76E-21	1.05E-21	-1.67E-21	4.09E-24	-5.86E-24
Multiply with	1.00E+05	1.00E+00	1.00E+00	1.00E+03	1.00E+03	1.00E+03	1.00E+03
f(T)	1.01E+04	9.58E+02	6.21E-01	4.19E+05	2.68E+06	1.31E+03	7.35E+03
f(TA)	1.34E+04	9.52E+02	8.11E-01	4.53E+05	2.69E+06	1.40E+03	7.26E+03

8.00E+00	9.00E+00	1.00E+01	1.10E+01	1.20E+01	1.30E+01	1.40E+01	1.50E+01	1.60E+01	1.70E+01
CPl(T)	Cpv(t)	k(T)	al(T)	av(T)	μl(T)	μv(T)	kl(T)	kv(T)	σ(T)
4.23E+00	1.96E+00	1.54E+00	1.40E+03	4.09E+02	1.79E+03	9.22E+00	5.61E+02	1.72E+01	7.57E+01
-7.89E-03	-4.18E-02	-1.13E-01	5.13E+00	8.43E-01	-6.00E+01	2.03E-02	1.88E+00	-4.68E-02	-1.40E-01
5.57E-04	4.06E-03	1.03E-02	-6.45E-02	-1.17E-02	1.40E+00	3.99E-04	4.23E-03	9.71E-03	-2.58E-04
-1.93E-05	-1.52E-04	-3.73E-04	5.55E-04	3.97E-04	-2.30E-02	-1.09E-05	-2.80E-04	-3.57E-04	-1.08E-06
3.70E-07	2.95E-06	6.98E-06	-5.75E-06	-7.47E-06	2.60E-04	1.99E-07	2.49E-06	6.92E-06	2.26E-08
-4.18E-09	-3.30E-08	-7.55E-08	5.16E-08	8.09E-08	-1.98E-06	-2.20E-09	-7.77E-09	-7.78E-08	-2.36E-10
2.92E-11	2.26E-10	4.99E-10	-3.31E-10	-5.37E-10	1.02E-08	1.51E-11	-2.34E-11	5.35E-10	1.55E-12
-1.27E-13	-9.61E-13	-2.05E-12	1.38E-12	2.22E-12	-3.42E-11	-6.42E-14	2.85E-13	-2.29E-12	-6.41E-15
3.36E-16	2.47E-15	5.08E-15	-3.57E-15	-5.54E-15	7.26E-14	1.66E-16	-1.02E-15	5.91E-15	1.62E-17
-4.93E-19	-3.52E-18	-6.97E-18	5.15E-18	7.66E-18	-8.77E-17	-2.38E-19	1.71E-18	-8.48E-18	-2.29E-20
3.08E-22	2.13E-21	4.06E-21	-3.17E-21	-4.50E-21	4.60E-20	1.45E-22	-1.13E-21	5.18E-21	1.39E-23
1.00E+03	1.00E+03	1.00E+00	1.00E+00	1.00E+00	1.00E-06	1.00E-06	1.00E-03	1.00E-03	1.00E-03
4.22E+03	2.11E+03	1.39E+00	1.54E+03	4.72E+02	2.81E-04	1.23E-05	6.79E-01	2.52E-02	5.89E-02
4.23E+03	2.16E+03	1.45E+00	1.54E+03	4.76E+02	2.59E-04	1.25E-05	6.81E-01	2.61E-02	5.74E-02

T	1.00E+02	degree C
TA	1.08E+02	
dTe	8.00E+00	degree C
Csf	1.28E-02	
n	1.00E+00	
m	3.30E-01	
g	9.81E+00	m/s^2
q"	**7.51E+04**	**W/m^2**

Prl	1.74E+00
kl	6.79E-01
mv	1.23E-05
ml	2.81E-04
Cpl	4.22E+03
rol	9.58E+02
sigma	5.89E-02
dT	8.00E+00
hl	4.19E+05

hv	2.68E+06	
hfg	2.26E+06	
rov	6.21E-01	

PROBLEM:

Saturated water at Tw=60 ℃ is boiled with a copper heating element, having a heating surface A=5x10^{-3} m^2 and is maintained at a constant temperature T_w=70 ℃. Calculate heat transfer and rate of evaporation

```
/* a[0] "P, pressure        ";
   a[1] "T, temperature     ";
   a[2] "v, specific volume ";
   a[3] "h, enthalpy        ";
   a[4] "u, internal energy ";
   a[5] "s, entropy         ";
   a[6] "x, quality         ";
   a[7] "Cp, specific heat at constant pressure ";
   a[8] "Cv, specific heat at constant volume ";
   a[9] ""+'\u03B1'+" isobaric thermal expansion coefficient ";
   a[10] ""+'\u03BA'+" isothermal compressibility" ;
   a[11] ""+'\u03B7'+" Dynamic viscosity" ;
   a[12] "k Thermal conductivity" ;
   a[13] ""+'\u03C3'+" Surface tension" ;
   a[14] "Prandtl number" ;
   a[15] ""+'\u03C1'+" density" ;
   a[16] " speed of sound" ;
*/
public class HT_pool_boiling_water
{ public static void main(String arg[])
  { steamIAPWS_IF97 st=new steamIAPWS_IF97();
    double Ts=60.0+273.15;       //steam saturation temp. degree K;
    double Tw=70.0+273.15;   //wall temp. degree K
    double A=5e-3;  //m^2

    double al[]=st.property("tx",Ts,0.0);
    double av[]=st.property("tx",Ts,1.0);
     double Ps=al[0];
     System.out.println("Ts = "+(Ts-273.15)+" degree C Ps = "+Ps+" kPa");
    double kl=al[12];   // W/(mK)
    double mul=al[11];  //kg/(m.s)=Pa.s
    double muv=av[11];  //kg/(m.s)=Pa.s
    double Prl=al[14];
    double Cpl=al[7]*1e3;  //J/kgK
    double hfg=(av[3]-al[3])*1e3; //J/kg
    double rol=al[15];  //kg/m^3
    double rov=av[15];  //kg/m^3
    double sigma=al[13];  //N/m
    double Psat=al[0]*1e3; //Pa
    double g=9.806; //m/s^2
    double Csf=0.0128;
    double n=1;
    double m=0.33;
    double dTe=Tw-Ts;
    double q=Math.pow((Cpl*dTe/(hfg*Csf*Prl)),1.0/m)*hfg*mul*Math.sqrt(g*(rol-rov)/sigma); //m

    System.out.println("hfg="+hfg+" J/kg");
```

```
    System.out.println("sigma="+sigma+" N/m");
    System.out.println("rol="+rol+" kg/m^3");
    System.out.println("rov="+rov+" kg/m^3");
    System.out.println("Cp="+Cpl+" J/(kgK)");
   double Q=q*A;
   double mm=Q/hfg;
   System.out.println("heat transfer q = "+q+" W/m^2");
   System.out.println("total heat transfer Q = "+Q+" W\nrate of evaporation m="+mm+" kg/s =
"+mm*3600+" kg/h");

}}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_pool_boiling_water
Ts = 60.0 degree C Ps = 19.945801924678744 kPa
hfg=2357691.011563888 J/kg
sigma=0.06623826252330559 N/m
rol=983.1751288349705 kg/m^3
rov=0.13041794752318994 kg/m^3
Cp=4182.945029312105 J/(kgK)
heat transfer q = 40568.41327023788 W/m^2
total heat transfer Q = 202.8420663511894 W
rate of evaporation m=8.603420268232756E-5 kg/s = 0.3097231296563792 kg/h

> Terminated with exit code 0.
```

PROBLEM: During the boiling of saturated water at T_s=100 ℃ with an electric heating element, aheat flux of q=5e5 W/m^2 is achived. Electric heating element diameter diameter is 0.05 m and length is L=10 m. Find the surface temperature of the pipe. And total water evaporation rate.

```
/* a[0] "P, pressure        ";
   a[1] "T, temperature     ";
   a[2] "v, specific volume ";
   a[3] "h, enthalpy        ";
   a[4] "u, internal energy ";
   a[5] "s, entropy         ";
   a[6] "x, quality         ";
   a[7] "Cp, specific heat at constant pressure ";
   a[8] "Cv, specific heat at constant volume ";
   a[9] ""+'\u03B1'+" isobaric thermal expansion coefficient ";
   a[10] ""+'\u03BA'+" isothermal compressibility" ;
   a[11] ""+'\u03B7'+" Dynamic viscosity" ;
   a[12] "k Thermal conductivity" ;
   a[13] ""+'\u03C3'+" Surface tension" ;
   a[14] "Prandtl number" ;
   a[15] ""+'\u03C1'+" density" ;
   a[16] " speed of sound" ;
*/
public class HT_pool_boiling_water1
{
public static double newton_bisection(if_x f,double x1, double x2)
{
double xacc=1.0e-10; //accuracy
double h=0.0001; //derivative function step
int MAXIT=100;  //Maximum number of iterations
int j;
double dfr,dx,dxold,fr,fh,fl;
double temp,b,a,r;
fl=f.func(x1);
fh=f.func(x2);
//Check if a root is existed in the given region
if ((fl > 0.0 && fh > 0.0) || (fl < 0.0 && fh < 0.0))
System.out.println("There are no root in the given region or double root?");
if (fl == 0.0) return x1;
if (fh == 0.0) return x2;
if (fl < 0.0)   { a=x1;b=x2;}
```

```
else          { b=x1;a=x2;}
r=0.5*(x1+x2); //midpoint value
dxold=Math.abs(x2-x1);
dx=dxold;
fr=f.func(r);   //function value at midpoint
dfr=f.dfunc(r); //derivative of function value at midpoint
for (j=1;j<=MAXIT;j++)
{
if ((((r-b)*dfr-fr)*((r-a)*dfr-fr) > 0.0) || (Math.abs(2.0*fr) > Math.abs(dxold*dfr)))
   { //bisection step
                dxold=dx;
     dx=0.5*(b-a);
     r=a+dx;
     if (a == r) return r; //çözüm!!!!
   }
else{ //Newton-Raphson (Secant) step
                dxold=dx;
     dx=fr/dfr;
     temp=r;
     r -= dx;
     if (temp == r) return r;  //solution!!!!!
   }
if (Math.abs(dx) < xacc) return r; //solution!!!!!
fr=f.func(r);
dfr=f.dfunc(r);
if (fr < 0.0)
a=r;
else
b=r;
}
System.out.println("Maximum number of iterations are exceeded");
return 0.0; //program should never reach here
          //dummy return
}
  public static void main(String arg[])
  { steamIAPWS_IF97 st=new steamIAPWS_IF97();
    double Ts=373.15;       //steam saturation temp. degree K;
    double P=101.325; //kPa
    double D=0.05; // m
    double L=1.0;   //m
    double al[]=st.property("tx",Ts,0.0);
    double av[]=st.property("tx",Ts,1.0);
    double kl=al[12];   // W/(mK)
    double mul=al[11];  //kg/(m.s)=Pa.s
    double muv=av[11];  //kg/(m.s)=Pa.s
    double Prl=al[14];
    double Cpl=al[7]*1e3;  //J/kgK
    double hfg=(av[3]-al[3])*1e3; //J/kg
    double rol=al[15];  //kg/m^3
    double rov=av[15];  //kg/m^3
    double sigma=al[13];  //N/m
    double Psat=al[0]*1e3; //Pa
    double g=9.806; //m/s^2
    double Csf=0.0128;
    double n=1;
    double m=0.33;
    double q=5.0e5;
   if_x f=T->Math.pow((Cpl*(T-Ts)/(hfg*Csf*Prl)),1.0/m)*hfg*mul*Math.sqrt(g*(rol-rov)/sigma)-q;
   double Tw=newton_bisection(f,Ts,(Ts+20.0));
   System.out.println("Tw="+Tw+"degree K ="+(Tw-273.15)+" degree C");
   double A=Math.PI*D*L;
   double Q=q*A;
   double mfr=q/hfg;
    System.out.println("Q="+Q+" W  m ="+mfr+" kg/s");
}}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_pool_boiling_water1
Tw=388.16110075725857degree K =115.01110075725859 degree C
Q=78539.81633974482 W  m =0.22158475987536186 kg/s
> Terminated with exit code 0.
```

PROBLEM:

An electrically heated copper kettle with a flat bottom of diameter D =0.25 m is to boiI water. Barometer at the room reads Ps=97 kPa. Water is boiling at a flow rate of m=2.5 kg/h. What is the temperature of the bottom surface of the kettle? What is the total heat transfer in W

```
/* a[0] "P, pressure        ";
   a[1] "T, temperature     ";
   a[2] "v, specific volume ";
   a[3] "h, enthalpy        ";
   a[4] "u, internal energy ";
   a[5] "s, entropy         ";
   a[6] "x, quality         ";
   a[7] "Cp, specific heat at constant pressure ";
   a[8] "Cv, specific heat at constant volume ";
   a[9] ""+'\u03B1'+" isobaric thermal expansion coefficient ";
   a[10] ""+'\u03BA'+" isothermal compressibility" ;
   a[11] ""+'\u03B7'+" Dynamic viscosity" ;
   a[12] "k Thermal conductivity" ;
   a[13] ""+'\u03C3'+" Surface tension" ;
   a[14] "Prandtl number" ;
   a[15] ""+'\u03C1'+" density" ;
   a[16] " speed of sound" ;
*/
public class HT_pool_boiling_water2
{
public static double newton_bisection(if_x f,double x1, double x2)
{
double xacc=1.0e-10; //accuracy
double h=0.0001; //derivative function step
int MAXIT=100;  //Maximum number of iterations
int j;
double dfr,dx,dxold,fr,fh,fl;
double temp,b,a,r;
fl=f.func(x1);
fh=f.func(x2);
//Check if a root is existed in the given region
if ((fl > 0.0 && fh > 0.0) || (fl < 0.0 && fh < 0.0))
System.out.println("There are no root in the given region or double root?");
if (fl == 0.0) return x1;
if (fh == 0.0) return x2;
if (fl < 0.0)   { a=x1;b=x2;}
else            { b=x1;a=x2;}
r=0.5*(x1+x2); //midpoint value
dxold=Math.abs(x2-x1);
dx=dxold;
fr=f.func(r);   //function value at midpoint
dfr=f.dfunc(r); //derivative of function value at midpoint
for (j=1;j<=MAXIT;j++)
{
if ((((r-b)*dfr-fr)*((r-a)*dfr-fr) > 0.0) || (Math.abs(2.0*fr) > Math.abs(dxold*dfr)))
   { //bisection step
            dxold=dx;
     dx=0.5*(b-a);
     r=a+dx;
     if (a == r) return r; //çözüm!!!!
   }
else{ //Newton-Raphson (Secant) step
            dxold=dx;
     dx=fr/dfr;
     temp=r;
     r -= dx;
```

```
      if (temp == r) return r;  //solution!!!!!
    }
if (Math.abs(dx) < xacc) return r; //solution!!!!!
fr=f.func(r);
dfr=f.dfunc(r);
if (fr < 0.0)
a=r;
else
b=r;
}
System.out.println("Maximum number of iterations are exceeded");
return 0.0; //program should never reach here
          //dummy return
}
  public static void main(String arg[])
  { steamIAPWS_IF97 st=new steamIAPWS_IF97();
    //double Ts=373.15;       //steam saturation temp. degree K;
    double Ps=97.0 ; //kPa
    double a[]=st.property("px",Ps,1.0);
    double Ts=a[1];
    System.out.println("Ts="+Ts+" degree K = "+(Ts-273.15)+" degree C");
    double D=0.25; // m
    double al[]=st.property("tx",Ts,0.0);
    double av[]=st.property("tx",Ts,1.0);
    double kl=al[12];   // W/(mK)
    double mul=al[11];  //kg/(m.s)=Pa.s
    double muv=av[11];  //kg/(m.s)=Pa.s
    double Prl=al[14];
    double Cpl=al[7]*1e3;  //J/kgK
    double hfg=(av[3]-al[3])*1e3; //J/kg
    double rol=al[15];  //kg/m^3
    double rov=av[15];  //kg/m^3
    double sigma=al[13];  //N/m
    double Psat=al[0]*1e3; //Pa
    double g=9.806; //m/s^2
    double Csf=0.0128;
    double n=1;
    double m=0.33;
    double mfr=2.5/3600.0;  //kg/s
    double A=Math.PI*D*D/4.0;  //kettle surface area
    double Q=mfr*hfg;  //Total heat transfer W
    double q=Q/A;       //Heat transfer per unit area W/m^2
    //q_calculated-q_given=0
   if_x f=T->Math.pow((Cpl*(T-Ts)/(hfg*Csf*Prl)),1.0/m)*hfg*mul*Math.sqrt(g*(rol-rov)/sigma)-q;
   double Tw=newton_bisection(f,Ts,(Ts+20.0));
   System.out.println("Tw="+Tw+" degree K ="+(Tw-273.15)+" degree C");
    System.out.println("Q="+Q+" W  m ="+mfr+" kg/s = "+mfr*3600+" kg/h");
}}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_pool_boiling_water2
Ts=371.90666948549483 degree K = 98.75666948549485 degree C
Tw=378.03170245007954 degree K =104.88170245007956 degree C
Q=1569.2718097643035 W  m =6.944444444444445E-4 kg/s = 2.5 kg/h

> Terminated with exit code 0.
```

PROBLEM: During the boiling of saturated n-Butane at T_s=80 ℃ with an electric heating element with surface temperature Tw=85 ℃,. Electric heating element diameter diameter is 0.05 m and length is L=10 m. Find the total heat flux. And total water evaporation rate.

```
 /* a[0] "P, pressure        ";
   a[1] "T, temperature     ";
   a[2] "v, specific volume ";
   a[3] "h, enthalpy        ";
   a[4] "u, internal energy ";
   a[5] "s, entropy         ";
   a[6] "x, quality         ";
   a[7] "Cp, specific heat at constant pressure ";
   a[8] "Cv, specific heat at constant volume ";
   a[9] ""+'\u03B1'+" isobaric thermal expansion coefficient ";
   a[10] ""+'\u03BA'+" isothermal compressibility" ;
   a[11] ""+'\u03B7'+" Dynamic viscosity" ;
   a[12] "k Thermal conductivity" ;
   a[13] ""+'\u03C3'+" Surface tension" ;
   a[14] "Prandtl number" ;
   a[15] ""+'\u03C1'+" density" ;
   a[16] " speed of sound" ;
*/
public class HT_pool_boiling_butane
{
public static double newton_bisection(if_x f,double x1, double x2)
{
double xacc=1.0e-10; //accuracy
double h=0.0001; //derivative function step
int MAXIT=100;  //Maximum number of iterations
int j;
double dfr,dx,dxold,fr,fh,fl;
double temp,b,a,r;
fl=f.func(x1);
fh=f.func(x2);
//Check if a root is existed in the given region
if ((fl > 0.0 && fh > 0.0) || (fl < 0.0 && fh < 0.0))
System.out.println("There are no root in the given region or double root?");
if (fl == 0.0) return x1;
if (fh == 0.0) return x2;
if (fl < 0.0)   { a=x1;b=x2;}
else            { b=x1;a=x2;}
r=0.5*(x1+x2); //midpoint value
dxold=Math.abs(x2-x1);
dx=dxold;
fr=f.func(r);   //function value at midpoint
dfr=f.dfunc(r); //derivative of function value at midpoint
for (j=1;j<=MAXIT;j++)
{
if ((((r-b)*dfr-fr)*((r-a)*dfr-fr) > 0.0) || (Math.abs(2.0*fr) > Math.abs(dxold*dfr)))
   { //bisection step
            dxold=dx;
     dx=0.5*(b-a);
     r=a+dx;
     if (a == r) return r; //solution!!!!!
   }
```

```
else{ //Newton-Raphson (Secant) step
            dxold=dx;
     dx=fr/dfr;
     temp=r;
     r -= dx;
     if (temp == r) return r;  //solution!!!!!
    }
if (Math.abs(dx) < xacc) return r; //solution!!!!!
fr=f.func(r);
dfr=f.dfunc(r);
if (fr < 0.0)
a=r;
else
b=r;
}
System.out.println("Maximum number of iterations are exceeded");
return 0.0; //program should never reach here
          //dummy return
}
  public static void main(String arg[])
  { refrigerant_coolProp st=new  refrigerant_coolProp("n-Butane");
    double Ts=80+273.15;       //steam saturation temp. degree K;
    double Tw=85+273.15;    //wall temperature
    double P=101.325; //kPa
    double D=0.25; // m
    double al[]=st.property("tx",Ts,0.0);
    double av[]=st.property("tx",Ts,1.0);
    double kl=al[12];   // W/(mK)
    double mul=al[11];  //kg/(m.s)=Pa.s
    double muv=av[11];  //kg/(m.s)=Pa.s
    double Prl=al[14];
    double Cpl=al[7]*1e3;  //J/kgK
    double hfg=(av[3]-al[3])*1e3; //J/kg
    double rol=al[15];  //kg/m^3
    double rov=av[15];  //kg/m^3
    double sigma=al[13];  //N/m
    double Psat=al[0]*1e3; //Pa
    double g=9.806; //m/s^2
    double Csf=0.0128;
    double n=1;
    double m=0.33;
   double q=Math.pow((Cpl*(Tw-Ts)/(hfg*Csf*Prl)),1.0/m)*hfg*mul*Math.sqrt(g*(rol-rov)/sigma);
   double A=Math.PI*D*D/4.0;
   System.out.println("A="+A+" m^2");
   double Q=q*A;
   double mfr=q/hfg;
    System.out.println("Q="+Q+" W  m ="+mfr+" kg/s");
}}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan
Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v20230129-
1123\jre\bin\javaw.exe" HT_pool_boiling_butane
A=0.04908738521234052 m^2
Q=2034.0987451894746 W  m =0.14148552203292586 kg/s

> Terminated with exit code 0.
```

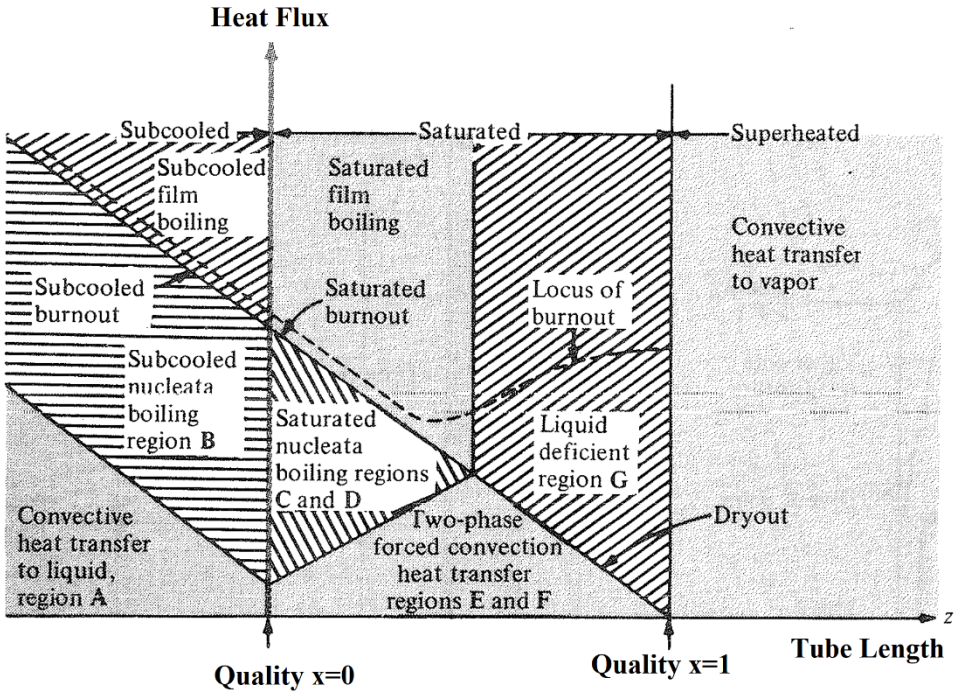
15.2 INSIDE PIPES CONVECTIVE BOILING HEAT TRANSFER

Basic mechanisms of the flow for vertical and horizontal tubes are shown in the figure. Flow is relatively symetric in vertical flow cases, liquid parts deviates to the bottom of pipe in the horizontal flow cases. In the figure below horizontal and vertical flow boling regimes are shown.

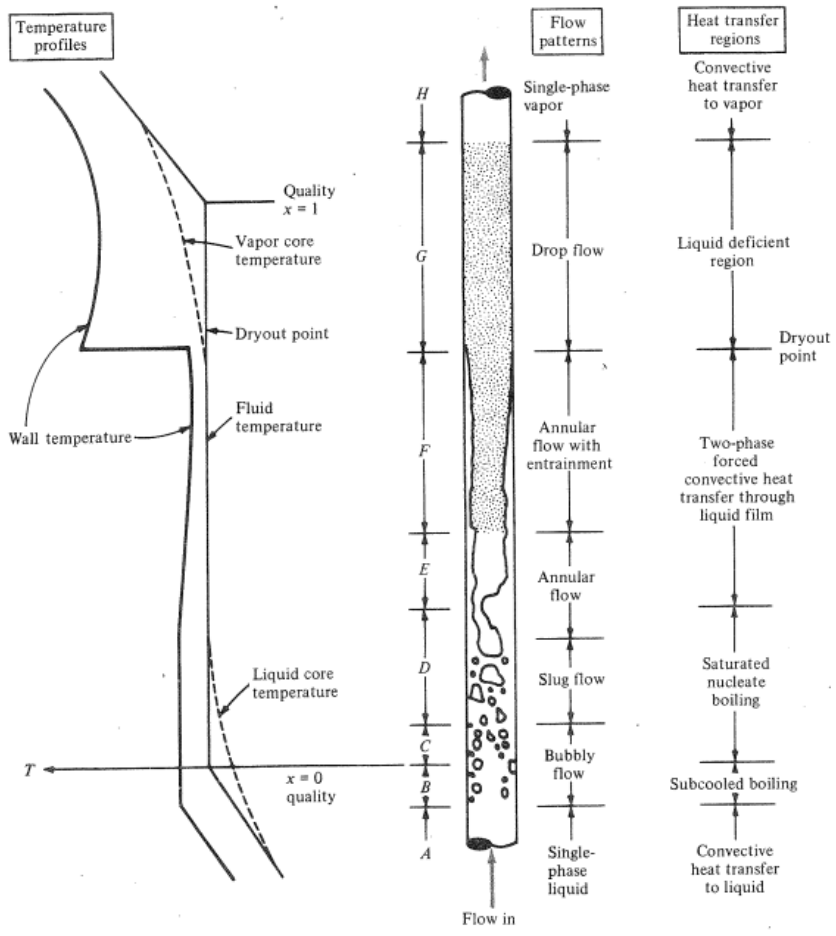
Vertical flow:

Chen[47] proposed the following correlation that covers saturated nucleate boiling region C and D of vertical flow and the two phase forced vertical convection regions E and F. It is assumed that in these regions heat transfer takes places by both nucleations and forced convection so total is addition of these two effects

$h_{TP} = h_{NB} + h_C$

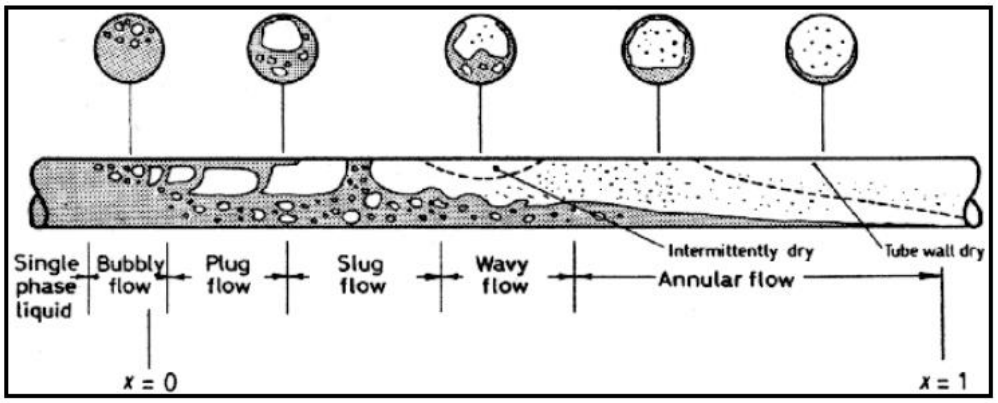


Figre 15.3 Inside pipes vertical pipe boiling zones map



Figre 15.4 Inside pipes vertical pipe boiling zones

Horizontal flow:



Figre 15.5 Inside pipes horizontal pipe zones

Chen-Forster-Zuber internal vertical pipe boiling

In order to calculate convective heat transfer, chen modified Dittus-Boelter equation as:

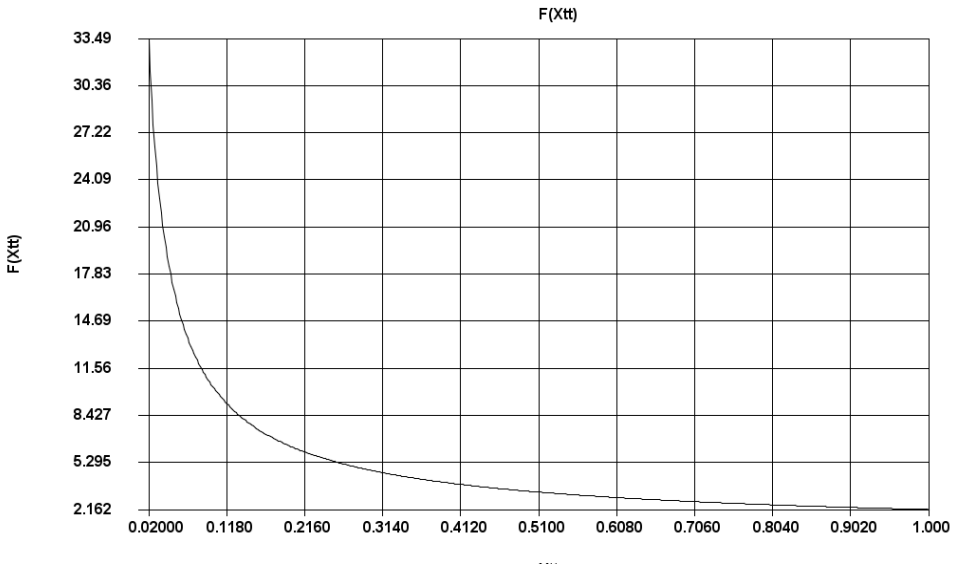
$h_C = 0.023\left(\frac{k_l}{D}\right) Re_l^{0.8} Pr_l^{0.4} F$ where F is the convective boiling factor. In order to calculate F, we should first calculate Martinelli parameter which has a form of:

$X_{tt} = \left(\frac{1-x}{x}\right)^{0.9}\left(\frac{\rho_v}{\rho_l}\right)^{0.5}\left(\frac{\mu_l}{\mu_v}\right)^{0.1}$

If $\frac{1}{X_{tt}} \leq 0.1$ F=1

Else F=2.35 $\left(\frac{1}{X_{tt}}+0.213\right)^{0.736}$

```
public class Xtt
{public static void main(String arg[])
 {if_x F=Xtt->{double F1=1.0;
                if(1.0/Xtt<=0.1) {F1=1.0;}
                else {F1=Math.pow(2.35*(1.0/Xtt+0.213),0.736);}
                return F1;
                };
  Plot p=new Plot(F,2.0e-2,1.0);
  p.setPlabel("F(Xtt)");
  p.setXlabel("Xtt");
  p.setYlabel("F(Xtt)");
  p.plot();
}}
```



In modified Dittus Boelter equation

$Re_l = \frac{m(1-x)D}{A_{cross-section}\mu_l} = \frac{G(1-x)D}{\mu_l}$ where x is vapor mass quality.

The nucleate boiling heat transfer coeeficient h_{NB} is taken as the modified Forster-Zuber[151] equation in the form:

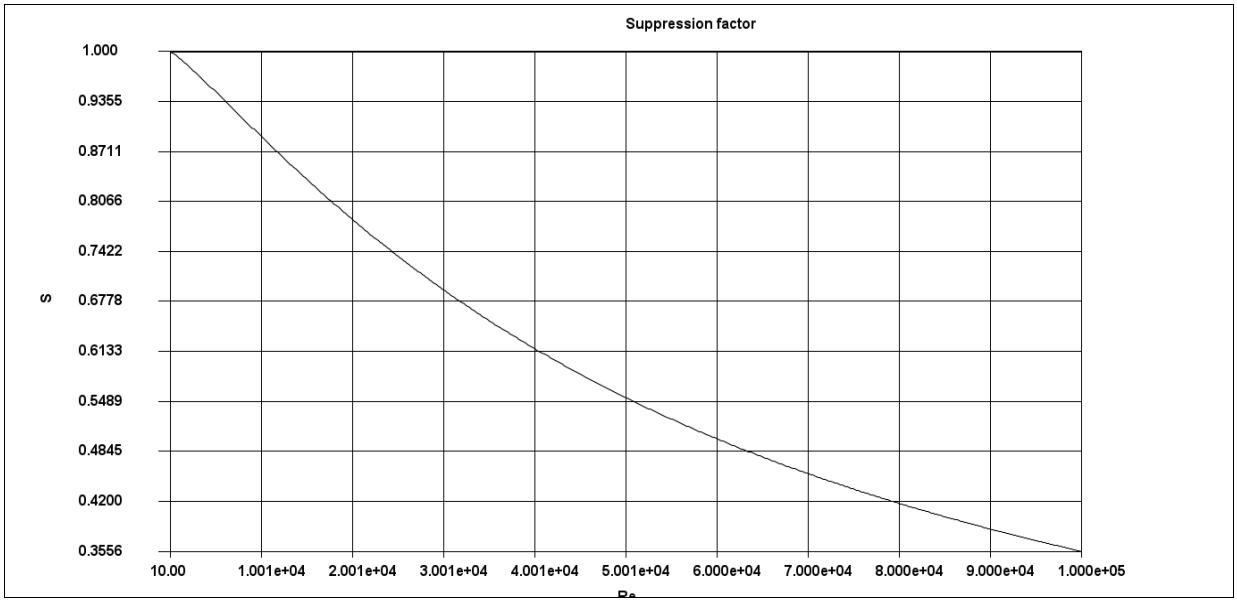
$h_{NB} = 0.00122\left(\frac{k_l^{0.79}c_{pi}^{0.45}\rho_l^{0.49}}{\sigma^{0.5}\mu_l^{0.29}h_{fg}^{0.24}\rho_v^{0.24}}\right)\Delta T_{sat}^{0.24}\Delta P_{sat}^{0.75}S$ where

$\Delta T_{sat} = T_{wall} - T_{sat} \quad \Delta P_{sat} = P(T_{wall}) - P(T_{sat})$

S is suppression factor, and defined as:

$S = \frac{1}{1+0.00000256Re_{tp}^{1.17}}$

```
public class Supression
{public static void main(String arg[])
 {if_x S=Re->1.0/(1.0+0.00000256*Math.pow(Re,1.17));
  Plot p=new Plot(S,10,1.e5);
  p.setPlabel("Suppression factor");
  p.setXlabel("Re");
  p.setYlabel("S");
  p.plot();
}}
```



Re_{tp} is the double phase Reynolds number and defined as:

$Re_{tp} = Re_l F^{1.25}$

PROBLEM : Saturated water at T_{sat}=100 C flows with a mass flow rate of m=0.1 kg/s through a $2.5x10^{-2}$ m internal diameter tube, subjected to a uniform Wall heat transfer of q_w=200 kW/m^2. Calculate the tube Wall temperature T_w and two pahase heat transfer coefficient h_{TP} at the location where the vapor mass quality is x=0.25.

Chen-Forster-Zuber internal vertical pipe boiling

```
 public class HT_internal_pipe_boiling
{ /* a[0] "P, pressure        ";
   a[1] "T, temperature     ";
   a[2] "v, specific volume ";
   a[3] "h, enthalpy        ";
   a[4] "u, internal energy ";
   a[5] "s, entropy         ";
   a[6] "x, quality         ";
   a[7] "Cp, specific heat at constant pressure ";
   a[8] "Cv, specific heat at constant volume ";
   a[9] ""+'\u03B1'+" isobaric thermal expansion coefficient ";
   a[10] ""+'\u03BA'+" isothermal compressibility" ;
   a[11] ""+'\u03B7'+" Dynamic viscosity" ;
   a[12] "k Thermal conductivity" ;
   a[13] ""+'\u03C3'+" Surface tension" ;
   a[14] "Prandtl number" ;
   a[15] ""+'\u03C1'+" density" ;
   a[16] " speed of sound" ;
*/
public static double bisection(if_x f,double a,double b)
  {double b1=2.0*b;
   double r=(a+b)/2.0;
   double eps=1.0e-8;
   int nmax=100;
   int i=0;
   while(Math.abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;
    i++;
   }
   if(i>=100) r=bisection(f,a,b1);
   return r;
  }
public static double hTP(double tsat,double tw, double D,double m,double x,boolean pr)
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // D internal diameter of the tube
  // m mass flow rate kg/s
```

```
  // x quality
  steamIAPWS_IF97 st=new steamIAPWS_IF97();
  double al[]=st.propertyC("tx",tsat,0.0);
  double av[]=st.propertyC("tx",tsat,1.0);
  double aw[]=st.propertyC("tx",tw,0.0);
  double kl=al[12];   // W/(mK)
  double mul=al[11];  //kg/(m.s)=Pa.s
  double muv=av[11];  //kg/(m.s)=Pa.s
  double Prl=al[14];
  double Cpl=al[7]*1e3;  //J/kgK
  double hfg=(av[3]-al[3])*1e3; //J/kg
  double rol=al[15];  //kg/m^3
  double rov=av[15];  //kg/m^3
  double sigma=al[13];  //N/m
  double Psat=al[0]*1e3; //Pa
  double Pw=aw[0]*1e3; //Pa
  double dP=Pw-Psat;
  double dT=tw-tsat;
  double Xtt=Math.pow(((1-x)/x),0.9)*Math.pow((rov/rol),0.5)*Math.pow((mul/muv),0.1);
  double F=1;
  if(1.0/Xtt>0.1) F=2.35*Math.pow((1.0/Xtt+0.213),0.736);
  double G=m/(Math.PI/4.0*D*D);
  double Rel=G*(1.0-x)*D/mul;
  double hc=0.023*(kl/D)*Math.pow(Rel,0.8)*Math.pow(Prl,0.4)*F;
  double ReTP=Math.pow(F,1.25)*Rel;
  double S=1.0/(1+0.00000253*Math.pow(ReTP,1.17));
  double hNB=0.00122*Math.pow(kl,0.79)*Math.pow(Cpl,0.45)*Math.pow(rol,0.49);
  hNB/=(Math.pow(sigma,0.5)*Math.pow(mul,0.29)*Math.pow(hfg,0.24)*Math.pow(rov,0.24));
  hNB*=Math.pow(dT,0.24)*Math.pow(dP,0.75)*S;
  double hTP=hNB+hc;
  if(pr)
  {
System.out.println("kl="+kl+"mul="+mul+"muv="+muv+"Prl="+Prl+"Cpl="+Cpl+"\nhfg="+hfg+"rol="+rol+"rov="+rov+"sigma
="+sigma+"Psat="+Psat);

System.out.println("kl="+kl+"mul="+mul+"muv="+muv+"Prl="+Prl+"Cpl="+Cpl+"\nhfg="+hfg+"rol="+rol+"rov="+rov+"sigma
="+sigma+"Psat="+Psat);
     System.out.println("Xtt="+Xtt);
     System.out.println("F="+F);
     System.out.println("G="+G);
     System.out.println("Rel="+Rel);
     System.out.println("hc="+hc);
     System.out.println("ReTP="+ReTP);
     System.out.println("S="+S);
     System.out.println("hNB="+hNB);
     System.out.println("hTP="+hTP);
  }
  return hTP;
}
public static void main(String arg[])
{double tsat=100.0; //degree C
 double D=2.5e-2; //m
 double m=0.1; //kg/s
 double x=0.25;
 double qw=200000; //W/m^2
 if_x f1=t->hTP(tsat,t,D,m,x,false)*(t-tsat)-qw;
 double a=tsat+2;
 double b=tsat+15;
 double tw=bisection(f1,a,b);
 System.out.println("tsat="+tsat+" degree C");
 System.out.println("tw="+tw+" degree C");
 System.out.println("q="+qw+" W/m^2");
 hTP(tsat,tw,D,m,x,true);
 }
 }
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\java.exe" HT_internal_pipe_boiling
tsat=100.0 degree C
tw=108.8432324295614 degree C
q=200000.0 W/m^2
kl=0.677757511374594mul=2.8158501936566716E-4muv=1.2232158121714269E-
5Prl=1.751871543352034Cpl=4216.645118923585
hfg=2256472.8742231308rol=958.3542772858901rov=0.5981359925257022sigma=0.058911868587664076Psat=101417.97792130
998
```

```
kl=0.677757511374594mul=2.8158501936566716E-4muv=1.2232158121714269E-
5Prl=1.751871543352034Cpl=4216.645118923585
hfg=2256472.8742231308rol=958.3542772858901rov=0.5981359925257022sigma=0.058911868587664076Psat=101417.97792130
998
Xtt=0.09188755607341963
F=13.813588692803314
G=203.71832715762602
Rel=13565.063378763023
hc=21802.53801142138
ReTP=361247.6061310052
S=0.11050709062658003
hNB=813.6265633602069
hTP=22616.164574781586

> Terminated with exit code 0.
```

PROBLEM : Saturated butane at T_{sat}=80 ℃ flows with a mass flow rate of m=0.1 kg/s through a $2.5x10^{-2}$ m internal diameter tube, subjected to a uniform Wall heat transfer of q_w=200 kW/m^2. Calculate the tube Wall temperature T_w and two pahase heat transfer coefficient h_{TP} at the location where the vapor mass quality is x=0.1,0.2,0.3,..0.9.

Chen-Forster-Zuber internal vertical pipe boiling

```
public class HT_internal_pipe_boiling_butane
{ /* a[0] "P, pressure        ";
   a[1] "T, temperature     ";
   a[2] "v, specific volume ";
   a[3] "h, enthalpy        ";
   a[4] "u, internal energy ";
   a[5] "s, entropy         ";
   a[6] "x, quality         ";
   a[7] "Cp, specific heat at constant pressure ";
   a[8] "Cv, specific heat at constant volume ";
   a[9] ""+'\u03B1'+" isobaric thermal expansion coefficient ";
   a[10] ""+'\u03BA'+" isothermal compressibility" ;
   a[11] ""+'\u03B7'+" Dynamic viscosity" ;
   a[12] "k Thermal conductivity" ;
   a[13] ""+'\u03C3'+" Surface tension" ;
   a[14] "Prandtl number" ;
   a[15] ""+'\u03C1'+" density" ;
   a[16] " speed of sound" ;
*/
public static double bisection(if_x f,double a,double b)
  {double b1=2.0*b;
   double r=(a+b)/2.0;
   double eps=1.0e-8;
   int nmax=100;
   int i=0;
   while(Math.abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;
    i++;
   }
   if(i>=100) r=bisection(f,a,b1);
   return r;
  }
public static double hTP(double tsat,double tw, double D,double m,double x,boolean pr)
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // D internal diameter of the tube
```

```
  // m mass flow rate kg/s
  // x quality
  refrigerant_coolProp st=new refrigerant_coolProp("n-Butane");
  double al[]=st.propertyC("tx",tsat,0.0);
  double av[]=st.propertyC("tx",tsat,1.0);
  double aw[]=st.propertyC("tx",tw,0.0);
  double kl=al[12];   // W/(mK)
  double mul=al[11];  //kg/(m.s)=Pa.s
  double muv=av[11];  //kg/(m.s)=Pa.s
  double Prl=al[14];
  double Cpl=al[7]*1e3;  //J/kgK
  double hfg=(av[3]-al[3])*1e3; //J/kg
  double rol=al[15];  //kg/m^3
  double rov=av[15];  //kg/m^3
  double sigma=al[13];  //N/m
  double Psat=al[0]*1e3; //Pa
  double Pw=aw[0]*1e3; //Pa
  double dP=Pw-Psat;
  double dT=tw-tsat;
  //Martinelli parameter
  double Xtt=Math.pow(((1-x)/x),0.9)*Math.pow((rov/rol),0.5)*Math.pow((mul/muv),0.1);
  //the convective boiling factor
  double F=1;
  if(1.0/Xtt>0.1) F=2.35*Math.pow((1.0/Xtt+0.213),0.736);
  double G=m/(Math.PI/4.0*D*D);
  double Rel=G*(1.0-x)*D/mul;
  double hc=0.023*(kl/D)*Math.pow(Rel,0.8)*Math.pow(Prl,0.4)*F;
  double ReTP=Math.pow(F,1.25)*Rel;
  //suppression factor
  double S=1.0/(1+0.00000253*Math.pow(ReTP,1.17));
  double hNB=0.00122*Math.pow(kl,0.79)*Math.pow(Cpl,0.45)*Math.pow(rol,0.49);
  hNB/=(Math.pow(sigma,0.5)*Math.pow(mul,0.29)*Math.pow(hfg,0.24)*Math.pow(rov,0.24));
  hNB*=Math.pow(dT,0.24)*Math.pow(dP,0.75)*S;
  double hTP=hNB+hc;
  if(pr)
  {
System.out.println("kl="+kl+"mul="+mul+"muv="+muv+"Prl="+Prl+"Cpl="+Cpl+"\nhfg="+hfg+"r
ol="+rol+"rov="+rov+"sigma="+sigma+"Psat="+Psat);

System.out.println("kl="+kl+"mul="+mul+"muv="+muv+"Prl="+Prl+"Cpl="+Cpl+"\nhfg="+hfg+"r
ol="+rol+"rov="+rov+"sigma="+sigma+"Psat="+Psat);
     System.out.println("Xtt="+Xtt);
     System.out.println("F="+F);
     System.out.println("G="+G);
     System.out.println("Rel="+Rel);
     System.out.println("hc="+hc);
     System.out.println("ReTP="+ReTP);
     System.out.println("S="+S);
     System.out.println("hNB="+hNB);
     System.out.println("hTP="+hTP);
  }
  return hTP;
}
public static void main(String arg[])
{double tsat=80.0; //degree C
 double D=2.5e-2; //m
```

```
 double L=5.0; //m
 //double x=0.0;
 double htp=0.0;
 double htp_avg=0.0;
 double m=0.1; //kg/s
  refrigerant_coolProp st=new refrigerant_coolProp("n-Butane");
  double al[]=st.propertyC("tx",tsat,0.0);
  double av[]=st.propertyC("tx",tsat,1.0);
  double hl=al[3];
  double hv=av[3];
  double hlv=hv-hl;
  double Q=m*hlv*1e3;
  double A=Math.PI*D*L; //m^2
  double qw=Q/A;
  System.out.println("Q="+Q+" W"+"A="+A+" m^2  qw="+qw+ "W/m^2");
  double tw=0.0;
 for(int i=1;i<10;i++)
 { final double x=0.1*i;
  if_x f1=t->hTP(tsat,t,D,m,x,false)*(t-tsat)-qw;
 double a=tsat+1;
 double b=tsat+20;
 tw=bisection(f1,a,b);
   htp=hTP(tsat,tw,D,m,x,false);
   System.out.println("htp="+htp+"x="+x+"tw="+tw);
 }
}
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan
Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v20230129-
1123\jre\bin\javaw.exe" HT_internal_pipe_boiling_butane
Q=29288.02803915107 WA=0.39269908169872414 m^2  qw=74581.35097351878W/m^2
htp=5940.030241969432x=0.1tw=92.5557190679865
htp=5444.82577848029x=0.2tw=93.69765608814953
htp=5215.832219043282x=0.30000000000000004tw=94.29903184025193
htp=5111.659505685699x=0.4tw=94.59043797626941
htp=5073.763079008232x=0.5tw=94.6994153672805
htp=5070.936017242009x=0.6000000000000001tw=94.70761033464612
htp=5082.757754806805x=0.7000000000000001tw=94.67340262340394
htp=5092.495970102286x=0.8tw=94.64534315027038
htp=5083.093656064891x=0.9tw=94.67243297485356

> Terminated with exit code 0.
```

PROBLEM : Saturated water at Tsat=100 C flows with a mass flow rate of m=0.1 kg/s through a 2.5x10-2 m ID vertical tube of L=5 m, subjected to a uniform wall temperature of Tw=110 C. Calculate the two phase heat transfer coefficient and total heat transfer.

Chen-Forster-Zuber internal vertical pipe boiling

```
public class HT_internal_pipe_boiling
{ /* a[0] "P, pressure        ";
   a[1] "T, temperature     ";
   a[2] "v, specific volume ";
   a[3] "h, enthalpy        ";
```

```
   a[4] "u, internal energy ";
   a[5] "s, entropy         ";
   a[6] "x, quality         ";
   a[7] "Cp, specific heat at constant pressure ";
   a[8] "Cv, specific heat at constant volume ";
   a[9] ""+'\u03B1'+" isobaric thermal expansion coefficient ";
   a[10] ""+'\u03BA'+" isothermal compressibility" ;
   a[11] ""+'\u03B7'+" Dynamic viscosity" ;
   a[12] "k Thermal conductivity" ;
   a[13] ""+'\u03C3'+" Surface tension" ;
   a[14] "Prandtl number" ;
   a[15] ""+'\u03C1'+" density" ;
   a[16] " speed of sound" ;
*/
public static double bisection(if_x f,double a,double b)
  {double b1=2.0*b;
   double r=(a+b)/2.0;
   double eps=1.0e-8;
   int nmax=100;
   int i=0;
   while(Math.abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;
    i++;
   }
   if(i>=100) r=bisection(f,a,b1);
   return r;
  }
public static double hTP(double tsat,double tw, double D,double m,double x,boolean pr)
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // D internal diameter of the tube
  // m mass flow rate kg/s
  // x quality
  steamIAPWS_IF97 st=new steamIAPWS_IF97();
  double al[]=st.propertyC("tx",tsat,0.0);
  double av[]=st.propertyC("tx",tsat,1.0);
  double aw[]=st.propertyC("tx",tw,0.0);
  double kl=al[12];   // W/(mK)
  double mul=al[11];  //kg/(m.s)=Pa.s
  double muv=av[11];  //kg/(m.s)=Pa.s
  double Prl=al[14];
  double Cpl=al[7]*1e3;  //J/kgK
  double hfg=(av[3]-al[3])*1e3; //J/kg
  double rol=al[15];  //kg/m^3
  double rov=av[15];  //kg/m^3
  double sigma=al[13];  //N/m
  double Psat=al[0]*1e3; //Pa
  double Pw=aw[0]*1e3; //Pa
  double dP=Pw-Psat;
  double dT=tw-tsat;
  double Xtt=Math.pow(((1-x)/x),0.9)*Math.pow((rov/rol),0.5)*Math.pow((mul/muv),0.1);
  double F=1;
  if(1.0/Xtt>0.1) F=2.35*Math.pow((1.0/Xtt+0.213),0.736);
  double G=m/(Math.PI/4.0*D*D);
```

```
  double Rel=G*(1.0-x)*D/mul;
  double hc=0.023*(kl/D)*Math.pow(Rel,0.8)*Math.pow(Prl,0.4)*F;
  double ReTP=Math.pow(F,1.25)*Rel;
  double S=1.0/(1+0.00000253*Math.pow(ReTP,1.17));
  double hNB=0.00122*Math.pow(kl,0.79)*Math.pow(Cpl,0.45)*Math.pow(rol,0.49);
  hNB/=(Math.pow(sigma,0.5)*Math.pow(mul,0.29)*Math.pow(hfg,0.24)*Math.pow(rov,0.24));
  hNB*=Math.pow(dT,0.24)*Math.pow(dP,0.75)*S;
  double hTP=hNB+hc;
  if(pr)
  {
System.out.println("kl="+kl+"mul="+mul+"muv="+muv+"Prl="+Prl+"Cpl="+Cpl+"\nhfg="+hfg+"r
ol="+rol+"rov="+rov+"sigma="+sigma+"Psat="+Psat);

System.out.println("kl="+kl+"mul="+mul+"muv="+muv+"Prl="+Prl+"Cpl="+Cpl+"\nhfg="+hfg+"r
ol="+rol+"rov="+rov+"sigma="+sigma+"Psat="+Psat);
     System.out.println("Xtt="+Xtt);
     System.out.println("F="+F);
     System.out.println("G="+G);
     System.out.println("Rel="+Rel);
     System.out.println("hc="+hc);
     System.out.println("ReTP="+ReTP);
     System.out.println("S="+S);
     System.out.println("hNB="+hNB);
     System.out.println("hTP="+hTP);
  }
  return hTP;
}
public static void main(String arg[])
{double tsat=100.0; //degree C
 double D=2.5e-2; //m
 double L=5.0; //m
 //double x=0.0;
 double htp=0.0;
 double htp_avg=0.0;
 double m=0.1; //kg/s
  steamIAPWS_IF97 st=new steamIAPWS_IF97();
  double al[]=st.propertyC("tx",tsat,0.0);
  double av[]=st.propertyC("tx",tsat,1.0);
  double hl=al[3];
  double hv=av[3];
  double hlv=hv-hl;
  double Q=m*hlv*1e3;
  double A=Math.PI*D*L; //m^2
  double qw=Q/A;
  System.out.println("Q="+Q+" W"+"A="+A+" m^2  qw="+qw+ "W/m^2");
  double tw=0.0;
 for(int i=1;i<10;i++)
 { final double x=0.1*i;
  if_x f1=t->hTP(tsat,t,D,m,x,false)*(t-tsat)-qw;
 double a=tsat+1;
 double b=tsat+20;
 tw=bisection(f1,a,b);
   htp=hTP(tsat,tw,D,m,x,false);
   System.out.println("htp="+htp+"x="+x+"tw="+tw);
 }
}
```

```
}
---------- Capture Output ----------
> "C:\Users\Turhan Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe" HT_internal_pipe_boiling
Q=225647.2874223131 WA=0.39269908169872414 m^2  qw=574606.0990166207W/m^2
htp=19259.923507399417 x=0.1 tw=129.83428770087613
htp=22397.3382398403 x=0.2 tw=125.65510655165767
htp=26294.67547451932 x=0.30000000000000004 tw=121.85256477393864
htp=30036.640824745533 x=0.4 tw=119.13017179148886
htp=33364.046666720686 x=0.5 tw=117.22231433004683
htp=36157.02996981076 x=0.6000000000000001tw=115.89196069191487
htp=38276.16752492072 x=0.7000000000000001tw=115.01211161338219
htp=39433.01839992653 x=0.8 tw=114.57169961449605
htp=38765.36228650972 x=0.9 tw=114.82266810173948

> Terminated with exit code 0.
```

PROBLEM : Saturated water at T_{sat}=100 C flows with a mass flow rate of m=0.1 kg/s through a $2.5x10^{-2}$ m internal diameter tube, subjected to a constant Wall temperature of tw=108 C. Calculate the heat transfer and two phase heat transfer coefficient h_{TP} at the location where the vapor mass quality is x=0.25.

Chen-Forster-Zuber internal vertical pipe boiling

```
 public class HT_internal_pipe_boiling1A
{ /* a[0] "P, pressure        ";
   a[1] "T, temperature     ";
   a[2] "v, specific volume ";
   a[3] "h, enthalpy        ";
   a[4] "u, internal energy ";
   a[5] "s, entropy         ";
   a[6] "x, quality         ";
   a[7] "Cp, specific heat at constant pressure ";
   a[8] "Cv, specific heat at constant volume ";
   a[9] ""+'\u03B1'+" isobaric thermal expansion coefficient ";
   a[10] ""+'\u03BA'+" isothermal compressibility" ;
   a[11] ""+'\u03B7'+" Dynamic viscosity" ;
   a[12] "k Thermal conductivity" ;
   a[13] ""+'\u03C3'+" Surface tension" ;
   a[14] "Prandtl number" ;
   a[15] ""+'\u03C1'+" density" ;
   a[16] " speed of sound" ;
*/
public static double hTP(double tsat,double tw, double D,double m,double x,boolean pr)
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // D internal diameter of the tube
  // m mass flow rate kg/s
  // x quality
  steamIAPWS_IF97 st=new steamIAPWS_IF97();
  double al[]=st.propertyC("tx",tsat,0.0);
  double av[]=st.propertyC("tx",tsat,1.0);
  double aw[]=st.propertyC("tx",tw,0.0);
  double kl=al[12];   // W/(mK)
  double mul=al[11];  //kg/(m.s)=Pa.s
  double muv=av[11];  //kg/(m.s)=Pa.s
  double Prl=al[14];
  double Cpl=al[7]*1e3;  //J/kgK
  double hfg=(av[3]-al[3])*1e3; //J/kg
  double rol=al[15];  //kg/m^3
  double rov=av[15];  //kg/m^3
  double sigma=al[13];  //N/m
  double Psat=al[0]*1e3; //Pa
  double Pw=aw[0]*1e3; //Pa
  double dP=Pw-Psat;
  System.out.println("dPsat="+dP);
  double dT=tw-tsat;
```

```
  double Xtt=Math.pow(((1-x)/x),0.9)*Math.pow((rov/rol),0.5)*Math.pow((mul/muv),0.1);
  double F=1;
  if(1.0/Xtt>0.1) F=2.35*Math.pow((1.0/Xtt+0.213),0.736);
  double G=m/(Math.PI/4.0*D*D);
  double Rel=G*(1.0-x)*D/mul;
  double hc=0.023*(kl/D)*Math.pow(Rel,0.8)*Math.pow(Prl,0.4)*F;
  double ReTP=Math.pow(F,1.25)*Rel;
  double S=1.0/(1+0.00000253*Math.pow(ReTP,1.17));
  double hNB=0.00122*Math.pow(kl,0.79)*Math.pow(Cpl,0.45)*Math.pow(rol,0.49);
  hNB/=(Math.pow(sigma,0.5)*Math.pow(mul,0.29)*Math.pow(hfg,0.24)*Math.pow(rov,0.24));
  hNB*=Math.pow(dT,0.24)*Math.pow(dP,0.75)*S;
  double hTP=hNB+hc;
  if(pr)
  {
System.out.println("kl="+kl+"mul="+mul+"muv="+muv+"Prl="+Prl+"Cpl="+Cpl+"\nhfg="+hfg+"rol="+rol+"rov="+rov+"sigma
="+sigma+"Psat="+Psat);

System.out.println("kl="+kl+"mul="+mul+"muv="+muv+"Prl="+Prl+"Cpl="+Cpl+"\nhfg="+hfg+"rol="+rol+"rov="+rov+"sigma
="+sigma+"Psat="+Psat);
     System.out.println("Xtt="+Xtt);
     System.out.println("F="+F);
     System.out.println("G="+G);
     System.out.println("Rel="+Rel);
     System.out.println("hc="+hc);
     System.out.println("ReTP="+ReTP);
     System.out.println("S="+S);
     System.out.println("hNB="+hNB);
     System.out.println("hTP="+hTP);
  }
  return hTP;
}
public static void main(String arg[])
{double tsat=100.0; //degree C
 double tw=108.0;   //degree C
 double D=2.5e-2; //m
 double m=0.1; //kg/s
 double x=0.25;
 System.out.println("tsat="+tsat+" degree C");
 System.out.println("tw="+tw+" degree C");
 double hTP=hTP(tsat,tw,D,m,x,true);
 double q=hTP*(tw-tsat);
 System.out.println("q="+q);
}}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\java.exe" HT_internal_pipe_boiling1A
tsat=100.0 degree C
tw=108.0 degree C
dPsat=32588.565659875792
kl=0.677757511374594mul=2.8158501936566716E-4muv=1.2232158121714269E-
5Prl=1.751871543352034Cpl=4216.645118923585
hfg=2256472.8742231308rol=958.3542772858901rov=0.5981359925257022sigma=0.058911868587664076Psat=101417.9779
2130998
kl=0.677757511374594mul=2.8158501936566716E-4muv=1.2232158121714269E-
5Prl=1.751871543352034Cpl=4216.645118923585
hfg=2256472.8742231308rol=958.3542772858901rov=0.5981359925257022sigma=0.058911868587664076Psat=101417.9779
2130998
Xtt=0.09188755607341963
F=13.813588692803314
G=203.71832715762602
Rel=13565.063378763023
hc=21802.53801142138
ReTP=361247.6061310052
S=0.11050709062658003
hNB=729.9259607453973
hTP=22532.463972166777
q=180259.71177733422

> Terminated with exit code 0.
```

The same problem solved by using excel (properties are approximated by using curve fitting functions)

	1	2.00000000E+00	3.00000000E+00	4.00000000E+00	5.00000000E+00	6.00000000E+00	7.00000000E+00	8.00000000E+00	9.00000000E+00	1.00000000E+01	1.10000000E+01	1.20000000E+01	1.30000000E+0
	Psat(T)	ρl(T)	ρv(T)	hl(t)	hv(T)	si(T)	sv(T)	CPl(T)	Cpv(t)	k(T)	al(T)	av(T)	μl(T)
	6.75659684E-04	9.99713933E+02	6.16007774E-02	-1.426565309508E-02	2.500865895875E+03	1.06231702E-05	9.15556857E+00	4.227423654412E+00	1.95962962E+00	1.53782816E+00	1.40188876E+03	4.08782579E+02	1.789184842738E+0
	5.99840321E-06	1.15544480E-01	-3.21520036E-02	4.203010603701E+00	1.854519164136E+00	1.53631412E-02	-2.66885659E-02	-7.893131793340E-03	-4.18475807E-02	-1.12710992E-01	5.12708041E+00	8.43399053E-01	-6.002611438192E+0
	5.18078571E-06	-1.33222906E-02	3.10412400E-03	-1.727725912442E-05	-2.304499311602E-03	-2.61050643E-05	1.14451110E-04	5.565834523308E-04	4.05785426E-03	1.02749722E-02	-6.44776861E-02	-1.16600260E-02	1.400180271346E+0
	-1.18947329E-07	2.52910302E-04	-1.16947093E-04	-2.822045664989E-05	7.550903751081E-05	-1.04009758E-07	-1.08423971E-07	-1.934513562758E-05	-1.52413373E-04	-3.73153410E-04	5.55039130E-04	3.97103966E-04	-2.303435685355E-0
	3.21651255E-09	-4.07501221E-06	2.28489052E-06	8.208793125318E-07	-1.629971019769E-06	4.06291175E-09	-5.84543327E-09	3.701230196663E-07	2.94968954E-06	6.97832491E-06	-5.75017668E-06	-7.46694639E-06	2.598953016126E-0
	-3.19329602E-11	4.23942224E-08	-2.57633130E-08	-1.087889717275E-08	1.902951650142E-08	-5.22137693E-11	7.96089552E-11	-4.184770734548E-09	-3.30260420E-08	-7.55097702E-08	5.16209859E-08	8.08914594E-08	-1.984144368490E-0
	2.51806444E-13	-2.79866166E-10	1.78039644E-10	8.345331909358E-11	-1.384157663502E-10	3.82922687E-13	-5.73915623E-13	2.921771053805E-11	2.26147625E-10	4.99328992E-10	-3.30568340E-10	-5.37199514E-10	1.015772106320E-0
	-1.12256819E-15	1.16387581E-12	-7.63732143E-13	-3.861427814242E-13	6.256455492842E-13	-1.70007687E-15	2.50324684E-15	-1.272113599216E-13	-9.60823969E-13	-2.04729760E-12	1.38436314E-12	2.21641143E-12	-3.421528837064E-1
	2.94559268E-18	-2.95282319E-15	1.98425740E-15	1.066750663604E-15	-1.709257205705E-15	4.51365199E-18	-6.56315463E-18	3.361969018914E-16	2.47002701E-15	5.07553489E-15	-3.57492117E-15	-5.53575870E-15	7.256067380542E-1
	-4.28269633E-21	4.17194646E-18	-2.85969899E-18	-1.622103714132E-18	2.584716685181E-18	-6.59655898E-21	9.51275606E-21	-4.933492201490E-19	-3.51755537E-18	-6.96523685E-18	5.15204393E-18	7.65847057E-18	-8.767678280160E-1
	2.68402872E-24	-2.51675519E-21	1.75598688E-21	1.047360507365E-21	-1.665076271512E-21	4.08636831E-24	-5.86375614E-24	3.084300545524E-22	2.12973116E-21	4.05992097E-21	-3.16846195E-21	-4.50288345E-21	4.597475652817E-2
Multiply with	1.00000000E+06	1.00000000E+00	1.00000000E+00	1.00000000E+03	1.00000000E+03	1.00000000E+03	1.00000000E+03	1.00000000E+03	1.00000000E+03	1.00000000E+00	1.00000000E+00	1.00000000E+00	1.00000000E-0
f(T)	1.01448940E+05	9.58310233E+02	6.21372362E-01	4.19181705E+05	2.67555435E+06	1.30728142E+03	7.35404924E+03	4.21944134E+03	2.10949176E+03	1.39316432E+00	1.54300522E+03	4.72138370E+02	2.80825398E-0
f(TA)	1.34051742E+05	9.52402105E+02	8.11314954E-01	4.52972920E+05	2.68799065E+06	1.39680824E+03	7.26058146E+03	4.23066560E+03	2.16122203E+03	1.45273074E+00	1.53504233E+03	4.76054882E+02	2.59124324E-0

T	1.00000000E+02						
TA	1.08000000E+02						
dPsat	3.26028024E+04						
D	2.50000000E-02	m		kl	6.79101938E-01	ml	2.80825398E-04
ε	4.60000000E-05	m		mv	1.22710965E-05	Prl	1.74484304E+00
ε/D	1.84000000E-03			Cpl	4.21944134E+03	hfg	2.25637264E+06
L	1.00000000E+00	m^2		rol	9.58310233E+02	rov	6.21372362E-01
x	2.50000000E-01	kg vapor/kg		sigma	5.89120890E-02	Psat	1.01448940E+05
Atube	4.90873852E-04	m^2		dT	8.00000000E+00	dP	3.26028024E+04
m	1.00000000E-01	kg/s		S	1.12125263E-01		
G	2.03718327E+02	kg/(s*m^2)		hNB	7.36047149E+02		
Xtt	9.36024753E-02	The Martinelli parameter					
Rel	1.36017563E+04						
F	1.36304587E+01						
Retp	3.56232123E+05						
S	1.12125263E-01						
hc	2.15680960E+04						
dTsat	8.00000000E+00						
hfg	2.25637264E+06						
dPsat	3.26028024E+04						
hNB	7.36047149E+02						
hTP	2.23041431E+04						
qw	1.78433145E+05	W/m^2					

Gungor & Winterton[48] correlation for flow boiling in vertical & horizontal tubes and annuli:

Gungor & Winterton equation is valid both vertical and horizontal in tubes and annuli boiling heat transfer. Basic equations are as follows:

$Re_l = \frac{m(1-x)D}{A_{cross-section}\mu_l} = \frac{G(1-x)D}{\mu_l}$ where x is vapor mass quality. Where $G = \frac{m}{A_{cross-section}}$ is the mass flux (kg/sm^2) and m is mass flow rate (kg/s)

Martinelli parameter which has a form of:

$$X_{tt} = \left(\frac{1-x}{x}\right)^{0.9}\left(\frac{\rho_v}{\rho_l}\right)^{0.5}\left(\frac{\mu_l}{\mu_v}\right)^{0.1}$$

Froude number:

$Fr_l = \frac{G^2}{\rho_l^2 gD}$ where g is gravitational constant (can be taken as g=9.806 m/s^2)

Heat flux:

$q = h_{tp}(T_w - T_{sat})$ where q is the heat flux (W/m^2) h_{tp} is total convective boiling heat transfer, T_w is Wall temperature and T_{sat} is saturation temperature

Boling number:

$Bo = \frac{q}{h_{fg}G}$ where $h_{fg} = h_g - h_f$ is latent heat (J/kg)

Cross sectional area

$$A_{cross-section} = \frac{\pi D^2}{4}$$

Enhancement factor:

If vertical or horizontal with $Fr > 0.05$

$$E = 1 + 24000Bo^{1.16} + 1.37(\frac{1}{X_{tt}})^{0.86}$$

If horizontal with $Fr < 0.05$

$$E = \left[1 + 24000Bo^{1.16} + 1.37(\frac{1}{X_{tt}})^{0.86}\right]Fr^{(0.1-2Fr)}$$

Suppression factor:
If vertical or horizontal with $Fr > 0.05$
$S = \frac{1}{1+1.15x10^{-6}E^2Re_l^{1.15}}$
If horizontal with $Fr < 0.05$
$S = \frac{\sqrt{Fr}}{1+1.15x10^{-6}E^2Re_l^{1.15}}$
Dittus-Boelter equation for liquid only flowing in the duct
$h_l = 0.023\left(\frac{k_l}{D}\right)Re_l^{0.8}Pr_l^{0.4}$
Reduced pressure:
$P_r = \left(\frac{P}{P_{critical}}\right)$ where $P_{critical}$ is critical pressure
Pool boiling convective heat transfer coefficient
$h_{pool} = 55P_r^{0.12}(-log_{10}(P_r))^{-0.55}M^{-0.5}q^{0.67}$ where M is the molecular weight (kg/kmol)
$h_{tp} = Eh_l + Sh_{pool}$
This set of equation should be solved by root solving we should enter some initial estimation for h_{tp} and use some root finding method to solve it. In the following example bisection root finding methods is used due to simplicity of the method.

PROBLEM : Saturated water at T_{sat}=100 C flows with a mass flow rate of m=0.1 kg/s through a D=2.5x10^{-2} m internal diameter horizontal tube, subjected to a constant Wall temperature of tw=108 C. Calculate the heat transfer and two phase heat transfer coefficient h_{TP} at the location where the vapor mass quality is x=0.25.

Gungor & Winterton[48] correlation for flow boiling in horizontal tubes fluid: saturated water

```
class HT_GW implements if_x
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // D internal diameter of the tube
  // m mass flow rate kg/s
  // x quality
  public double q,Bo,E,hl,S,hpool,h,G,A,P,Preduced,dT,D;
  steamIAPWS_IF97 st;
  String position;
  public double kl,mul,muv,Prl,Cpl,hfg,rol,rov,sigma,Psat,Rel,Frl,Xtt;
  public HT_GW(double tw,double tsat,double m,double x,double Di,String posi)
  {D=Di;
   A=Math.PI*D*D/4.0;
   G=m/A;
   dT=tw-tsat;
   position=posi;
   st=new steamIAPWS_IF97();
   double al[]=st.propertyC("tx",tsat,0.0);
   double av[]=st.propertyC("tx",tsat,1.0);
   double aw[]=st.propertyC("tx",tw,0.0);
   kl=al[12];   // W/(mK)
   mul=al[11];  //kg/(m.s)=Pa.s
   muv=av[11];  //kg/(m.s)=Pa.s
   Prl=al[14];
   Cpl=al[7]*1e3;  //J/kgK
   hfg=(av[3]-al[3])*1e3; //J/kg
   rol=al[15];  //kg/m^3
   rov=av[15];  //kg/m^3
   sigma=al[13];  //N/m
   Psat=al[0]*1e3; //Pa
   Preduced=Psat/(st.Pc*1e3);
   dT=tw-tsat;
   Xtt=Math.pow(((1-x)/x),0.9)*Math.pow((rov/rol),0.5)*Math.pow((mul/muv),0.1);
   Rel=G*(1.0-x)*D/mul;
   double g=9.806; // m/s^2
   Frl=G*G/(rol*rol*g*D);
  }
  public double func(double htp)
  {   q=htp*dT;
      Bo=q/(hfg*G);
      double E1=1;
      double S1=1;
      if(position.equals("horizontal") && Frl<0.05)
```

```
    {E1=Math.pow(Frl,(0.1-2.0*Frl));S1=Math.sqrt(Frl);}
    E=E1*(1+24000*Math.pow(Bo,1.16)+1.37*Math.pow((1.0/Xtt),0.86));
    S=S1/(1.0+1.15e-6*E*E*Math.pow(Rel,1.17));
    hl=0.023*(kl/D)*Math.pow(Rel,0.8)*Math.pow(Prl,0.4);
    hpool=55*Math.pow(Preduced,0.12)*Math.pow(-Math.log10(Preduced),-0.55)*Math.pow(st.M,-0.5)*Math.pow(q,0.67);
    h=E*hl+S*hpool;
            return (h-htp);
  }
}
public class HT_internal_pipe_boiling_Gungor_Winterton
{
public static double bisection(if_x f,double a,double b)
  {double b1=2.0*b;
   double r=(a+b)/2.0;
   double eps=1.0e-8;
   int nmax=100;
   int i=0;
   while(Math.abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;
    i++;
   }
   if(i>=100) r=bisection(f,a,b1);
   return r;
  }
public static void main(String arg[])
{double tsat=100.0; //degree C
 double tw=108.0 ; //degree C
 double D=2.5e-2; //m
 double m=0.1; //kg/s
 double x=0.25;
 HT_GW f=new HT_GW(tw,tsat,m,x,D,"horizontal");
 double htp=bisection(f,100.0,30000.0);
 System.out.println("htp="+htp+" W/m^2K");
 System.out.println("q="+f.q+" W/m^2");
}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_internal_pipe_boiling_Gungor_Winterton
htp=23762.079586002983 W/m^2K
q=190096.63668802386 W/m^2

> Terminated with exit code 0.
```

PROBLEM : Saturated butane at T_{sat}=80 C flows with a mass flow rate of m=0.1 kg/s through a D=2.5x10^{-2} m internal diameter vertical tube, subjected to a constant Wall temperature of tw=88 C. Calculate the heat transfer and two phase heat transfer coefficient h_{TP} at the location where the vapor mass quality is x=0.25.

Gungor & Winterton[48] correlation for flow boiling in horizontal tubes fluid: butane

```
class HT_GW1 implements if_x
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // D internal diameter of the tube
  // m mass flow rate kg/s
  // x quality
  public double q,Bo,E,hl,S,hpool,h,G,A,P,Preduced,dT,D;
  refrigerant_coolProp st;
  String position;
  public double kl,mul,muv,Prl,Cpl,hfg,rol,rov,sigma,Psat,Rel,Frl,Xtt;
  String name;
  public HT_GW1(String namei,double tw,double tsat,double m,double x,double Di,String posi)
  {name=namei;
    D=Di;
   A=Math.PI*D*D/4.0;
   G=m/A;
```

```
   dT=tw-tsat;
   position=posi;
   st=new refrigerant_coolProp(name);
   double al[]=st.propertyC("tx",tsat,0.0);
   double av[]=st.propertyC("tx",tsat,1.0);
   double aw[]=st.propertyC("tx",tw,0.0);
   kl=al[12];   // W/(mK)
   mul=al[11];  //kg/(m.s)=Pa.s
   muv=av[11];  //kg/(m.s)=Pa.s
   Prl=al[14];
   Cpl=al[7]*1e3;  //J/kgK
   hfg=(av[3]-al[3])*1e3; //J/kg
   rol=al[15];  //kg/m^3
   rov=av[15];  //kg/m^3
   sigma=al[13];  //N/m
   Psat=al[0]*1e3; //Pa
   Preduced=Psat/(st.Pcrt*1e3);
   dT=tw-tsat;
   Xtt=Math.pow(((1-x)/x),0.9)*Math.pow((rov/rol),0.5)*Math.pow((mul/muv),0.1);
   Rel=G*(1.0-x)*D/mul;
   double g=9.806; // m/s^2
   Frl=G*G/(rol*rol*g*D);
  }
  public double func(double htp)
  {   q=htp*dT;
      Bo=q/(hfg*G);
      double E1=1;
      double S1=1;
      if(position.equals("horizontal") && Frl<0.05)
      {E1=Math.pow(Frl,(0.1-2.0*Frl));S1=Math.sqrt(Frl);}
      E=E1*(1+24000*Math.pow(Bo,1.16)+1.37*Math.pow((1.0/Xtt),0.86));
      S=S1/(1.0+1.15e-6*E*E*Math.pow(Rel,1.17));
      hl=0.023*(kl/D)*Math.pow(Rel,0.8)*Math.pow(Prl,0.4);
      hpool=55*Math.pow(Preduced,0.12)*Math.pow(-Math.log10(Preduced),-0.55)*Math.pow(st.M,-
0.5)*Math.pow(q,0.67);
      h=E*hl+S*hpool;
            return (h-htp);
   }
}
public class HT_internal_pipe_boiling_Gungor_Winterton_ref
{
public static double bisection(if_x f,double a,double b)
  {double b1=2.0*b;
   double r=(a+b)/2.0;
   double eps=1.0e-8;
   int nmax=100;
   int i=0;
   while(Math.abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;
    i++;
   }
   if(i>=100) r=bisection(f,a,b1);
   return r;
  }
```

```
public static void main(String arg[])
{double tsat=80.0; //degree C
 double tw=88.0 ; //degree C
 double D=2.5e-2; //m
 double L=5.0; //m
 double m=0.1; //kg/s
 double x=0.1;
 double A=Math.PI*D*L;
 HT_GW1 f=new HT_GW1("n-Butane",tw,tsat,m,x,D,"horizontal");
 double htp=f.bisection(100.0,30000.0);
 System.out.println("htp="+htp+" W/m^2K");
 System.out.println("q="+f.q+" W/m^2");
 double Q=f.q*A;
  System.out.println("Q="+Q+" W");
}
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan
Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v20230129-
1123\jre\bin\javaw.exe" HT_internal_pipe_boiling_Gungor_Winterton_ref
htp=5928.987891446601 W/m^2K
q=47431.903131572806 W/m^2
Q=18626.46480299148 W

> Terminated with exit code 0.
```

PROBLEM : Saturated water at T_s=120 ℃ flows with a mass flow rate of m=0.1 kg/s inside a 25.4e-3 m ID horizontal pipe, subjected to a uniform heat flux q=135e3 W/m^2. Calculate the tube Wall temperature T_w and two phase heat transfer coefficient h_{TP} at the location where vapor mass quality is x=0.2

Gungor & Winterton[48] correlation for flow boiling in horizontal tubes fluid: saturated water Equation of state: steamIAPWS_IF97

```
class HT_GW1 implements if_x
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // D internal diameter of the tube
  // m mass flow rate kg/s
  // x quality
  public double q,Bo,E,hl,S,hpool,h,G,A,P,Preduced,dT,D;
  steamIAPWS_IF97 st;
  String position;
  public double kl,mul,muv,Prl,Cpl,hfg,rol,rov,sigma,Psat,Rel,Frl,Xtt;
  public HT_GW1(double tw,double tsat,double m,double x,double Di,String posi)
  {D=Di;
   A=Math.PI*D*D/4.0;
   G=m/A;
   dT=tw-tsat;
   position=posi;
   st=new steamIAPWS_IF97();
   double al[]=st.propertyC("tx",tsat,0.0);
   double av[]=st.propertyC("tx",tsat,1.0);
   double aw[]=st.propertyC("tx",tw,0.0);
   kl=al[12];   // W/(mK)
   mul=al[11];  //kg/(m.s)=Pa.s
   muv=av[11];  //kg/(m.s)=Pa.s
   Prl=al[14];
   Cpl=al[7]*1e3;  //J/kgK
   hfg=(av[3]-al[3])*1e3; //J/kg
   rol=al[15];  //kg/m^3
   rov=av[15];  //kg/m^3
   sigma=al[13];  //N/m
```

```
   Psat=al[0]*1e3; //Pa
   Preduced=Psat/(st.Pc*1e3);
   dT=tw-tsat;
   Xtt=Math.pow(((1-x)/x),0.9)*Math.pow((rov/rol),0.5)*Math.pow((mul/muv),0.1);
   Rel=G*(1.0-x)*D/mul;
   double g=9.806; // m/s^2
   Frl=G*G/(rol*rol*g*D);
  }
  public double func(double q)
  {   Bo=q/(hfg*G);
      double E1=1;
      double S1=1;
      if(position.equals("horizontal") && Frl<0.05)
      {E1=Math.pow(Frl,(0.1-2.0*Frl));S1=Math.sqrt(Frl);}
      E=E1*(1+24000*Math.pow(Bo,1.16)+1.37*Math.pow((1.0/Xtt),0.86));
      S=S1/(1.0+1.15e-6*E*E*Math.pow(Rel,1.17));
      hl=0.023*(kl/D)*Math.pow(Rel,0.8)*Math.pow(Prl,0.4);
      hpool=55*Math.pow(Preduced,0.12)*Math.pow(-Math.log10(Preduced),-0.55)*Math.pow(st.M,-0.5)*Math.pow(q,0.67);
      h=E*hl+S*hpool;
                return h;
   }
}
public class HT_internal_pipe_boiling_Gungor_Winterton1
{
public static double bisection(if_x f,double a,double b)
  {double b1=2.0*b;
   double r=(a+b)/2.0;
   double eps=1.0e-8;
   int nmax=100;
   int i=0;
   while(Math.abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;
    i++;
   }
   if(i>=100) r=bisection(f,a,b1);
   return r;
  }
public static void main(String arg[])
{double Tsat=120.0; //degree C
 //double tw=110.0 ; //degree C
 double D=2.54e-2; //m
 double m=0.1; //kg/s
 double q=135e3;
 double x=0.2;

 if_x f1=T->{ HT_GW1 f=new HT_GW1(T,Tsat,m,x,D,"horizontal");
 double ff=f.func(T)*(T-Tsat)-q;
 return ff;
 };
 double Tw=bisection(f1,Tsat,(Tsat+20.0));
 System.out.println("Tw="+Tw);
  HT_GW1 f=new HT_GW1(Tw,Tsat,m,x,D,"horizontal");
  double hTP=f.func(q);
  System.out.println("hTP="+hTP+" W/m2K");
}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_internal_pipe_boiling_Gungor_Winterton1
Tw=130.0647476686754
hTP=18260.865674793924 W/m2K

> Terminated with exit code 0.
```

Kandlikar[49] correlation for flow boiling in vertical & horizontal tubes and annuli:

Kandlikar equation is valid both vertical and horizontal in tubes and annuli boiling heat transfer. Basic equations are as follows:

$Re_l = \frac{m(1-x)D}{A_{cross-section}\mu_l} = \frac{G(1-x)D}{\mu_l}$ where x is vapor mass quality. Where $G = \frac{m}{A_{cross-section}}$ is the mass flux (kg/sm^2) and m is mass flow rate (kg/s)

Convection number:

$Co = \left(\frac{1-x}{x}\right)^{0.8} \left(\frac{\rho_v}{\rho_l}\right)^{0.5}$

Dittus-Boelter equation for liquid only flowing in the duct

$h_l = 0.023 \left(\frac{k_l}{D}\right) Re_l^{0.8} Pr_l^{0.4}$

Convective heat transfer equation:

If(Frl<0.04 and position="horizontal) $f_2 = (25Fr_l)^{C_5}$

Else $f_2 = 1$;

$\frac{h_{tp}}{h_l} = C_1 Co^{C_2} f_2 + C_3 Bo^{C_4} F_{fl}$

The coefficients of the equation. The two sets of values given in table correspond to the convective boiling and nucleate boiling regions, respectively. The heat transfer coefficient at any given condition is evaluated using the two sets of constants for the two regions, and since the transition from one region to another occurs at the intersection of the respective correlations, the higher of the two heat transfer coefficient values represents the predicted value from the proposed correlation. This method provides a continuity between the convective and nucleate boiling regions.

If (Co<0.65) convective boing

If (Co>0.65 nucleate boiling)

Constants	Convective region	Nucleate boiling region
C_1	1.1360	0.6683
C_2	-0.9	-0.2
C_3	667.2	1058
C_4	0.7	0.7
C_5	0.3	0.3

Fluid dependent parameter F_{fl} in the convective heat transfer equation

Fluid	F_{fl}
Water	1
R-11	1.30
R-12	1.50
R-13B1	1.31
R-22	2.20
R-113	1.30
R-152a	1.10
Nitrogen	4.70
Neon	3.50

PROBLEM : Saturated water at T_{sat}=100 C flows with a mass flow rate of m=0.1 kg/s through a $2.5x10^{-2}$ m internal diameter horizontal tube, subjected to a constant Wall temperature of tw=108 C. Calculate the heat transfer and two phase heat transfer coefficient h_{TP} at the location where the vapor mass quality is x=0.25.

Kandlikar[49] correlation for flow boiling in horizontal tubes working fluid: saturated water

```
class HT_K implements if_x
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // D internal diameter of the tube
  // m mass flow rate kg/s
  // x quality
  public double q,Bo,E,hl,S,hpool,h,G,A,P,Preduced,dT,D;
  steamIAPWS_IF97 st;
  double C[][];
  String position;
  public int option;
  public double f2;
  public double Ffl;
  public double kl,mul,muv,Prl,Cpl,hfg,rol,rov,sigma,Psat,Rel,Frl,Co;
```

```
  public HT_K(double tw,double tsat,double m,double x,double Di,String posi)
  { double C1[][]={{1.1360,-0.9,667.2,0.7,0.3},
                   {0.6683,-0.2,1058,0.7,0.3}};
                   C=C1;
   D=Di;
   Ffl=1.0; //case for water
   A=Math.PI*D*D/4.0;
   G=m/A;
   dT=tw-tsat;
   position=posi;
   st=new steamIAPWS_IF97();
   double al[]=st.propertyC("tx",tsat,0.0);
   double av[]=st.propertyC("tx",tsat,1.0);
   double aw[]=st.propertyC("tx",tw,0.0);
   kl=al[12];   // W/(mK)
   mul=al[11];  //kg/(m.s)=Pa.s
   muv=av[11];  //kg/(m.s)=Pa.s
   Prl=al[14];
   Cpl=al[7]*1e3;  //J/kgK
   hfg=(av[3]-al[3])*1e3; //J/kg
   rol=al[15];  //kg/m^3
   rov=av[15];  //kg/m^3
   sigma=al[13];  //N/m
   Psat=al[0]*1e3; //Pa
   Preduced=Psat/(st.Pc*1e3);
   dT=tw-tsat;
   Co=Math.pow(((1-x)/x),0.8)*Math.pow((rov/rol),0.5);
   if(Co<0.65) option=0;
   else option=1;
   Rel=G*(1.0-x)*D/mul;
   double g=9.806; // m/s^2
   Frl=G*G/(rol*rol*g*D);
   if((Frl<0.04) && position.equals("horizontal")) f2=Math.pow((25.0*Frl),C[option][4]);
   else f2=1.0;
  }
  public double func(double htp)
  {   q=htp*dT;
      Bo=q/(hfg*G);

      hl=0.023*(kl/D)*Math.pow(Rel,0.8)*Math.pow(Prl,0.4);
      h=hl*(C[option][0]*Math.pow(Co,C[option][1])*f2+C[option][2]*Math.pow(Bo,C[option][3])*Ffl);
                return (h-htp);
   }
}
public class HT_internal_pipe_boiling_Kandlikar
{
public static double bisection(if_x f,double a,double b)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   double eps=1.0e-8;
   int nmax=100;
   int i=0;
   while(Math.abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;
    i++;
   }
   if(i>=100) r=bisection(f,a,b1);
   return r;
  }
public static void main(String arg[])
{double tsat=100.0; //degree C
 double tw=108.0 ; //degree C
 double D=2.5e-2; //m
 double m=0.1; //kg/s
 double x=0.25;
 HT_K f=new HT_K(tw,tsat,m,x,D,"horizontal");
 double htp=bisection(f,100.0,50000.0);
 System.out.println("htp="+htp+" W/m^2K");
 System.out.println("q="+f.q+" W/m^2");
}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_internal_pipe_boiling_Kandlikar
```

```
htp=27492.637089892924 W/m^2K
q=219941.0967191434 W/m^2
> Terminated with exit code 0.
```

Alternative form of Kandlikar equation:
Let us use Gnielinski[33] equation to calculate liquid convective heat transfer

$h_l = \left(\frac{k_l}{D}\right)\frac{\left(\frac{f}{8}\right)(Re_l - 1000)Pr_l}{1.07+12.7\left(\frac{f}{8}\right)^{.5}(Pr_l^{2/3}-1)}$ $0.5 \le Pr \le 2000$ $2300 \le Re \le 510^6$ where f is one phase friction factor and Goudar- Sonnad equation will be used to calculate it

Goudar- Sonnad equation (2008)[21] Valid region: all values

$a = \frac{2}{\ln(10)}$

$b = \frac{(\varepsilon/D)}{3.7}$

$d = \frac{ln(10)}{5.02}Re$

$s = bd + \ln\left(\frac{d}{q}\right);$

$q = s^{\left(\frac{s}{s+1}\right)}$

$g = bd + \ln\left(\frac{d}{q}\right)$

$z = \frac{q}{g}$

$\delta_{LA} = \frac{g}{g+1}z$

$\delta_{CFA} = \delta_{LA}\left(1 + \frac{z/2}{(g+1)^2+\left(\frac{z}{3}\right)(2g-1)}\right)$

$\frac{1}{\sqrt{f}} = a\left[ln\left(\frac{d}{q}\right) + \delta_{CFA}\right]$

The remaining of the equation will be same.

PROBLEM : Saturated water at T_{sat}=100 C flows with a mass flow rate of m=0.1 kg/s through a 2.5x10^{-2} m internal diameter horizontal tube, subjected to a constant Wall temperature of tw=108 C. Calculate the heat transfer and two phase heat transfer coefficient h_{TP} at the location where the vapor mass quality is x=0.25.

Kandlikar[49] correlation for flow boiling in vertical tubes working fluid: saturated water

```
class HT_K1 implements if_x
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // D internal diameter of the tube
  // m mass flow rate kg/s
  // x quality
  public double q,Bo,E,hl,S,hpool,h,G,A,P,Preduced,dT,D;
  steamIAPWS_IF97 st;
  double C[][];
  String position;
  public int option;
  public double f2;
  public double Ffl;
  public double kl,mul,muv,Prl,Cpl,hfg,rol,rov,sigma,Psat,Rel,Frl,Co,f,eps;
  public HT_K1(double tw,double tsat,double m,double x,double Di,double epsi,String posi)
  { double C1[][]={{1.1360,-0.9,667.2,0.7,0.3},
                   {0.6683,-0.2,1058,0.7,0.3}};
                   C=C1;
   D=Di;
   eps=epsi;
   Ffl=1.0; //case for water
   A=Math.PI*D*D/4.0;
   G=m/A;
   dT=tw-tsat;
   position=posi;
   st=new steamIAPWS_IF97();
   double al[]=st.propertyC("tx",tsat,0.0);
   double av[]=st.propertyC("tx",tsat,1.0);
   double aw[]=st.propertyC("tx",tw,0.0);
   kl=al[12];   // W/(mK)
   mul=al[11];  //kg/(m.s)=Pa.s
```

```
   muv=av[11];  //kg/(m.s)=Pa.s
   Prl=al[14];
   Cpl=al[7]*1e3;  //J/kgK
   hfg=(av[3]-al[3])*1e3; //J/kg
   rol=al[15];  //kg/m^3
   rov=av[15];  //kg/m^3
   sigma=al[13];  //N/m
   Psat=al[0]*1e3; //Pa
   Preduced=Psat/(st.Pc*1e3);
   dT=tw-tsat;
   Co=Math.pow(((1-x)/x),0.8)*Math.pow((rov/rol),0.5);
   if(Co<0.65) option=0;
   else option=1;
   Rel=G*(1.0-x)*D/mul;
   double g=9.806; // m/s^2
   Frl=G*G/(rol*rol*g*D);
   if((Frl<0.04) && position.equals("horizontal")) f2=Math.pow((25.0*Frl),C[option][4]);
   else f2=1.0;
  }
 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }

  public double func(double htp)
  {   q=htp*dT;
      Bo=q/(hfg*G);
      f=f_Goudar(Rel,eps/D);
      hl=(kl/D)*(f/8.0)*(Rel-1000.0)*Prl/(1.07+12.7*Math.pow((f/8.0),0.5)*(Math.pow(Prl,(2.0/3.0)-1.0)));
      h=hl*(C[option][0]*Math.pow(Co,C[option][1])*f2+C[option][2]*Math.pow(Bo,C[option][3])*Ffl);
                return (h-htp);
   }
}
public class HT_internal_pipe_boiling_Kandlikar1
{
public static double bisection(if_x f,double a,double b)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   double eps=1.0e-8;
   int nmax=100;
   int i=0;
   while(Math.abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;
    i++;
   }
   if(i>=100) r=bisection(f,a,b1);
   return r;
  }
public static void main(String arg[])
{double tsat=100.0; //degree C
 double tw=108.0 ; //degree C
 double D=2.5e-2; //m
 double m=0.1; //kg/s
 double x=0.25;
 double eps=0.15e-3;
 HT_K1 f=new HT_K1(tw,tsat,m,x,D,eps,"horizontal");
 double htp=bisection(f,100.0,50000.0);
 System.out.println("htp="+htp+" W/m^2K");
 System.out.println("q="+f.q+" W/m^2");
}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_internal_pipe_boiling_Kandlikar1
htp=27035.595302800946 W/m^2K
q=216284.76242240757 W/m^2

> Terminated with exit code 0.
```

Shah[67] correlation for flow boiling in vertical tubes and annuli:

$Re_l = \frac{m(1-x)D}{A_{cross-section}\mu_l} = \frac{G(1-x)D}{\mu_l}$ where x is vapor mass quality. Where $G = \frac{m}{A_{cross-section}}$ is the mass flux (kg/sm^2) and m is mass flow rate (kg/s)

Boling number:

$Bo = \frac{q}{h_{fg}G}$ where $h_{fg} = h_g - h_f$ is latent heat (J/kg)

Convection number:

$Co = \left(\frac{1-x}{x}\right)^{0.8}\left(\frac{\rho_v}{\rho_l}\right)^{0.5}$

Dittus-Boelter equation for liquid only flowing in the duct

$h_l = 0.023\left(\frac{k_l}{D}\right)Re_l^{0.8}Pr_l^{0.4}$

$\varphi = \frac{h_{tp}}{h_l}$

For vertical pipes: N=Bo

$\varphi_{cb} = 1.8N^{-0.8}$

For N>1.0

For Bo>0.3x10^{-4} $\varphi_{nb} = 230Bo^{0.5}$

For Bo<0.3x10^{-4} $\varphi_{nb} = 1 + 46Bo^{0.5}$

$\varphi = \frac{h_{tp}}{h_l}$ = the larger of φ_{nb} and φ_{cb}

For $N \leq 1.0$

For $0.1 \leq N \leq 1.0$ $\varphi_{bs} = FBo^{0.5}\exp(2.74N^{-0.1})$

For $N \leq 0.1$ $\varphi_{bs} = FBo^{0.5}\exp(2.74N^{-0.15})$

For $Bo \geq 11x10^{-4}$ F=14.7

For $Bo < 11x10^{-4}$ F=15.43

Steiner and Taborek[68] correlation for flow boiling in vertical tubes and annuli:

$h_{TP} = \left[\left(h_{nb,0}F_{nb}\right)^3 + \left(h_lF_{tp}\right)^3\right]^{1/3}$ where $h_{nb,0}$ (W/m^{2}K) is nucleate boiling at the reference heat flux q_0 (W/m^2)

$h_{nb,0}$ and q_0 values for several fluids are listed in the below table

Fluid	P_{crit} bar	M kg/kmol	q_0 W/m^2	$h_{nb,0}$ W/m^{2}K
Methane	46	16.04	20000	8060
Ethane	48.8	30.07	20000	5210
Propane	42.4	44.10	20000	4000
n-Butane	38.0	58.12	20000	3300
n-pentane	33.7	72.15	20000	3070
Isopentane	33.3	72.15	20000	2940
n-Hexane	29.7	86.18	20000	2840
n-Heptane	27.3	100.2	20000	2420
Cyclehexane	40.8	84.16	20000	2420
Benzene	48.9	78.11	20000	2730
Toluene	41.1	92.14	20000	2910
Diphenyl	38.5	154.2	20000	2030
Methanol	81	32.04	20000	2770
Ethanol	63.8	46.07	20000	3690
n-Propanol	51.7	60.1	20000	3170
Isopropanol	47.6	60.1	20000	2920

n-Butanol	49.6	74.12	20000	2750
Isobutanol	47.6	74.12	20000	2940
Acetone	47	58.08	20000	3270
R-11	44	137.4	20000	2690
R-12	41.6	120.9	20000	3290
R-13	38.6	104.5	20000	3910
R13B1	39.8	148.9	20000	3380
R-22	49.9	86.47	20000	3930
R-23	48.7	70.02	20000	4870
R-113	34.1	187.4	20000	2180
R-114	32.6	170.9	20000	2460
R-115	31.2	154.5	20000	2890
R-123	36.7	152.9	20000	2600
R-134a	40.6	102.0	20000	3500
R-152a	45.2	66.05	20000	4000
R-226	30.6	182.5	20000	3700
R-227	29.3	170	20000	3800
RC318	28	200	20000	2710
R-502	40.8	111.6	20000	2900
Chloromethane	66.8	50.49	20000	4790
Tetrachloromethane	45.6	153.8	20000	2320
Tetrafluoromethane	37.4	88	20000	4500
Helium I	2.275	4	1000	1990
Hydrogen (para)	12.97	2.02	10000	12220
Neon	26.5	20.18	10000	8920
Nitrogen	34	28.02	10000	4380
Argon	49	39.95	10000	3870
Oxygen	50.8	32	10000	4120
Water	220.6	18.02	150000	25580
Ammonia	113	17.03	150000	36640
Carbondioxide	73.8	44.01	150000	18890
Sulphur Hexafluoride	37.6	146	150000	12230

At this table P_r=0.1 except for Helium I (P_r=0.3)
Gnielinski[33] equation to calculate liquid convective heat transfer

$h_l = \left(\frac{k_l}{D}\right)\frac{\left(\frac{f}{8}\right)(Re_{LO}-1000)Pr_l}{1.07+12.7\left(\frac{f}{8}\right)^{.5}(Pr_l^{2/3}-1)}$ $0.5 \le Pr \le 2000$ $2300 \le Re \le 510^6$ where f is one phase friction factor and Goudar- Sonnad equation will be used to calculate it

$Re_{LO} = \frac{mD}{A_{cross-section}\mu_l} = \frac{GD}{\mu_l}$

$h_g = \left(\frac{k_l}{D}\right)\frac{\left(\frac{f}{8}\right)(Re_{GO}-1000)Pr_g}{1.07+12.7\left(\frac{f}{8}\right)^{.5}(Pr_g^{2/3}-1)}$ $0.5 \le Pr \le 2000$ $2300 \le Re \le 510^6$ where f is one phase friction factor and The fanning friction factor for the vapor f_G is

$f_g = [0.7904\ln(Re_{GO}) - 1.64]^{-2}$

$Re_{GO} = \frac{mD}{A_{cross-section}\mu_v} = \frac{GD}{\mu_g}$

For $x<x_{crit}$ $q>q_{ONB}$ where $q_{ONB} = \frac{2\sigma T_{sat}}{r_0\rho_g h_{lg}}$ where r_0 is the critical nucleation Radius for boiling site in meters and h_{lg} is latent heat of vaporization. For $q>q_{ONB}$ nucleate boiling is present in the flow boiling but below this . The recommended value for r_0 is $0.3x10^{-6}$ m

$F_{tp} = \left[\ (1-x)^{1.5} + 1.9x^{0.6}(1-x)^{0.01}\left(\frac{\rho_l}{\rho_g}\right)^{0.35}\right]^{1.1}$

For $q>q_{ONB}$

$F_{tp} = \left[\left\{(1-x)^{1.5} + 1.9x^{0.6}(1-x)^{0.01}\left(\frac{\rho_l}{\rho_v}\right)^{0.35}\right\}^{-2.2} + \left\{\left(\frac{h_l}{h_g}\right)x^{0.01}[1+8(1-x)^{0.7}]\left(\frac{\rho_l}{\rho_g}\right)^{0.67}\right\}^{-2}\right]^{-0.5}$

The nucleate boiling correction factor F_{nb} includes the effects of the reduced pressure, heat flux, tube diameter, surface roughness and a residual molecular weight correction factor.

$F_{nb} = F_{pf}\left(\frac{q}{q_0}\right)^{nf}\left(\frac{d_i}{d_{i0}}\right)^{-0.4}\left(\frac{R_p}{R_{p0}}\right)^{0.133} F(M)$ where mean surface rougness $R_{p0} = 1\ \mu m$ q_0 referance heat flux and the value will be taken from the table above. In that table reduced pressure values are P_r=0.1 (except for Helium which P_r=0.3) . For thatreason a pressure correction coefficient F_{pf} is added to the equation

$$F_{pf} = 2.816P_r^{0.45} + \left[3.4 + \left(\frac{1.7}{1-P_r^7}\right)\right]P_r^{3.7}$$

The nucleate boiling exponent, nf, on the normalized heat flux term is:

$$n_f = 0.8 - 0.1\exp(1.75P_r)$$

This expression is for all fluids except cryogens (nitrogen, oxygen, helium, hydrogen..) where it is

$$n_f = 0.8 - 0.13\exp(1.105P_r)$$

The standard tube reference diameter $d_{i0} = 0.01\ m$. Standard value of roughness $R_{p0} = 1\ \mu m$ The residule molecular weight correction factor n terms of molecular weight is:

$$F(M) = 0.377 + 0.199\ln(M) + 0.000028427M^2$$

By combining all this, heat transfer coefficiet can be calculated. It again should be note that equations are function of heat flux, therefore curve fitting should be applied to get a solution.

PROBLEM : Saturated water at T_{sat}=100 C flows with a mass flow rate of m=0.1 kg/s through a 2.5x10^{-2} m internal diameter vertical tube, subjected to a constant Wall temperature of tw=108 C. Calculate the heat transfer and two phase heat transfer coefficient h_{TP} at the location where the vapor mass quality is x=0.25.

Steiner and Taborek[68] correlation for flow boiling in vertical tubes worlking fluid: saturated water

```
class HT_ST implements if_x
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // D internal diameter of the tube
  // m mass flow rate kg/s
  // x quality
  public double q,Bo,E,hl,S,hpool,h,hv,G,A,P,Preduced,dT,D;
  steamIAPWS_IF97 st;
  double Rel,Rev;
  double q0,hnb0,r0,Di0,q0NB;
  double Ftp, Fnb, Fpf,nf,FM,M,Rp,Rp0,x;
  public double kl,kv,mul,muv,Prl,Prv,Cpl,Cpv,hfg,rol,rov,sigma,Psat,Frl,Co,fl,fv,eps;
  public HT_ST(double tw,double tsat,double m,double xi,double Di,double epsi,double Rpi)
  { Rp0=1e-6; //m
    x=xi;
    Rp=Rpi;
    D=Di;
    Di0=0.01; //m
    eps=epsi;
    q0=150000; // W/m^2
    hnb0=36640; // W/m^2K
    A=Math.PI*D*D/4.0; //m^2
    G=m/A;  //kg/m^2s
    dT=tw-tsat; //degree K
    st=new steamIAPWS_IF97();
    M=st.M; //kg/kmol
    double al[]=st.propertyC("tx",tsat,0.0);
    double av[]=st.propertyC("tx",tsat,1.0);
    double aw[]=st.propertyC("tx",tw,0.0);
    kl=al[12];   // W/(mK)
    kv=av[12];   // W/(mK)
    mul=al[11];  //kg/(m.s)=Pa.s
    muv=av[11];  //kg/(m.s)=Pa.s
    Prl=al[14];
    Prv=av[14];
    Cpl=al[7]*1e3;  //J/kgK
    Cpv=av[7]*1e3;  //J/kgK
    hfg=(av[3]-al[3])*1e3; //J/kg
    rol=al[15];  //kg/m^3
    rov=av[15];  //kg/m^3
    sigma=al[13];  //N/m
    Psat=al[0]*1e3; //Pa
```

```
    Preduced=Psat/(st.Pc*1e3);
    dT=tw-tsat;
    Rel=G*D/mul;
    Rev=G*D/muv;
    double g=9.806; // m/s^2
    fl=f_Goudar(Rel,eps/D);
    hl=(kl/D)*(fl/8.0)*(Rel-1000.0)*Prl/(1.07+12.7*Math.pow((fl/8.0),0.5)*(Math.pow(Prl,(2.0/3.0)-1.0)));
    q0NB=2*sigma*(tsat+273.15)*hl/(r0*rov*hfg);
    fv=f_Goudar(Rev,eps/D);
    hv=(kv/D)*(fv/8.0)*(Rev-1000.0)*Prv/(1.07+12.7*Math.pow((fv/8.0),0.5)*(Math.pow(Prv,(2.0/3.0)-1.0)));
    FM=0.377+0.199*Math.log(M)+0.000028427*M*M;
    nf=0.8-0.1*Math.exp(1.75*Preduced);
    double pr7=Preduced*Preduced*Preduced*Preduced*Preduced*Preduced*Preduced;
    Fpf=2.816*Math.pow(Preduced,0.45)+(3.4+1.7/(1-pr7))*Math.pow(Preduced,3.7);
  }
 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }

  public double func(double htp)
  {   q=htp*dT;
      if(q<q0NB)
      {Ftp=Math.pow((Math.pow((1.0-x),1.5)+1.9*Math.pow(x,0.6)*Math.pow((rol/rov),0.35)),1.1);}
      else
      {double XX=(hl/hv)*Math.pow(x,0.01)*(1.0+8.0*Math.pow((1.0-x),0.7)*Math.pow((rol/rov),0.67));
       XX=1.0/(XX*XX);
       double YY=Math.pow((Math.pow((1.0-x),1.5)+1.9*Math.pow(x,0.6)*Math.pow((rol/rov),0.35)),-2.2)+XX;
       Ftp=1.0/Math.sqrt(YY);
      }
      Fnb=Fpf*FM*Math.pow((q/q0),nf)*Math.pow((D/Di0),-0.4)*Math.pow((Rp/Rp0),0.133);
      double xx1=(hnb0*Fnb);
      double xx2=(hl*Ftp);
      h=Math.pow((xx1*xx1*xx1+xx2*xx2*xx2),(1.0/3.0));
                return h-htp;
   }
}
public class HT_internal_pipe_boiling_Steiner_Taborek
{
public static double bisection(if_x f,double a,double b)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   double eps=1.0e-8;
   int nmax=100;
   int i=0;
   while(Math.abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;
    i++;
   }
   if(i>=100) r=bisection(f,a,b1);
   return r;
  }
public static void main(String arg[])
{double tsat=100.0; //degree C
 double tw=108.0 ; //degree C
 double D=2.5e-2; //m
 double m=0.1; //kg/s
 double x=0.25;
 double eps=0.15e-3;
 double Rp=1e-6;
 HT_ST f=new HT_ST(tw,tsat,m,x,D,eps,Rp);
 double htp=bisection(f,10000.0,50000.0);
```

```
 System.out.println("htp="+htp+" W/m^2K");
 System.out.println("q="+f.q+" W/m^2");
}}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_internal_pipe_boiling_Steiner_Taborek
htp=30794.52432530161 W/m^2K
q=246356.19460241287 W/m^2

> Terminated with exit code 0.
```

PROBLEM : Saturated R134a at T_{sat}=0 C flows with a mass flow rate of m=0.1 kg/s through a 2.5x10^{-2} m internal diameter vertical tube, subjected to a constant Wall temperature of tw=20 C. Calculate the heat transfer and two phase heat transfer coefficient h_{TP} at the location where the vapor mass quality is x=0.25.

Steiner and Taborek[68] correlation for flow boiling in vertical tubes working fluid R134a eos: refrigerant

```
 class HT_ST_ref implements if_x
 { // tsat : saturation temperature degree C
   // tw surface temperature degree C
   // D internal diameter of the tube
   // m mass flow rate kg/s
   // x quality
   public double q,Bo,E,hl,S,hpool,h,hv,G,A,P,Preduced,dT,D;
   refrigerant re;
   double Rel,Rev;
   double q0,hnb0,r0,Di0,q0NB;
   double Ftp, Fnb, Fpf,nf,FM,M,Rp,Rp0,x;
   public double kl,kv,mul,muv,Prl,Prv,Cpl,Cpv,hfg,rol,rov,sigma,Psat,Frl,Co,fl,fv,eps;
   public HT_ST_ref(String namei,double tw,double tsat,double m,double xi,double Di,double epsi,double Rpi)
   { Rp0=1e-6; //m
     x=xi;
     Rp=Rpi;
     D=Di;
     Di0=0.01; //m
     eps=epsi;
     String ref_name[]={"Methane",
"Ethane",
"Propane",
"n-Butane",
"n-pentane",
"Isopentane",
"n-Hexane",
"n-Heptane",
"Cyclehexane",
"Benzene",
"Toluene",
"Diphenyl",
"Methanol",
"Ethanol",
"n-Propanol",
"Isopropanol",
"n-Butanol",
"Isobutanol",
"Acetone",
"R11",
"R12",
"R13",
"R13B1",
"R22",
"R23",
"R113",
"R114",
"R115",
"R123",
"R134a",
"R152a",
"R226",
"R227",
"RC318",
"R502",
"Chloromethane",
"Tetrachloromethane",
```

```
"Tetrafluoromethane",
"Helium I",
"Hydrogen (para)",
"Neon",
"Nitrogen",
"Argon",
"Oxygen",
"Water",
"Ammonia",
"Carbondioxide",
"Sulphur Hexafluorid"};
double qq0[]={20000,
20000,
20000,
```

```
2940,
3270,
2690,
3290,
3910,
3380,
3930,
4870,
2180,
2460,
2890,
2600,
3500,
4000,
3700,
3800,
2710,
2900,
4790,
2320,
4500,
1990,
12220,
8920,
4380,
3870,
4120,
25580,
36640,
18890,
12230
};
for(int i=0;i<ref_name.length;i++)
{if(ref_name[i].equals(namei)) {q0=qq0[i];hnb0=hh[i];}}
System.out.println("q0="+q0+"hnb0="+hnb0);
   A=Math.PI*D*D/4.0; //m^2
   G=m/A;  //kg/m^2s
   dT=tw-tsat; //degree K
   re=new refrigerant(namei);
   M=re.r.M; //kg/kmol
   kl=re.r.k_l(tsat);    // W/(mK)
   kv=re.r.k_v(tsat);    // W/(mK)
   mul=re.r.viscosity_l(tsat);  //kg/(m.s)=Pa.s
   muv=re.r.viscosity_v(tsat);       //kg/(m.s)=Pa.s
   Prl=re.r.Pr_l(tsat);
   Prv=re.r.Pr_v(tsat);
   Cpl=re.r.Cpl(tsat)*1e3;  //J/kgK
   Cpv=re.r.Cpv(tsat)*1e3;  //J/kgK
   hfg=re.r.h_lv(tsat)*1e3; //J/kg
   rol=re.r.rol(tsat);;  //kg/m^3
   rov=re.r.rov(tsat);;  //kg/m^3
   sigma=re.r.yuzeygerilim(tsat);  //N/m
   Psat=re.Psb(tsat); //Pa
   Preduced=Psat/(re.r.Pc);
   System.out.println("Preduced="+Preduced);
   dT=tw-tsat;
   Rel=G*D/mul;
   Rev=G*D/muv;
   double g=9.806; // m/s^2
   fl=f_Goudar(Rel,eps/D);
   hl=(kl/D)*(fl/8.0)*(Rel-1000.0)*Prl/(1.07+12.7*Math.pow((fl/8.0),0.5)*(Math.pow(Prl,(2.0/3.0)-1.0)));
   q0NB=2*sigma*(tsat+273.15)*hl/(r0*rov*hfg);
   fv=f_Goudar(Rev,eps/D);
   hv=(kv/D)*(fv/8.0)*(Rev-1000.0)*Prv/(1.07+12.7*Math.pow((fv/8.0),0.5)*(Math.pow(Prv,(2.0/3.0)-1.0)));
   FM=0.377+0.199*Math.log(M)+0.000028427*M*M;
   nf=0.8-0.1*Math.exp(1.75*Preduced);
   double pr7=Preduced*Preduced*Preduced*Preduced*Preduced*Preduced;
   Fpf=2.816*Math.pow(Preduced,0.45)+(3.4+1.7/(1-pr7))*Math.pow(Preduced,3.7);
 }
 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
```

```
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
  public double func(double htp)
  {   q=htp*dT;
      if(q<q0NB)
      {Ftp=Math.pow((Math.pow((1.0-x),1.5)+1.9*Math.pow(x,0.6)*Math.pow((rol/rov),0.35)),1.1);}
      else
      {double XX=(hl/hv)*Math.pow(x,0.01)*(1.0+8.0*Math.pow((1.0-x),0.7)*Math.pow((rol/rov),0.67));
       XX=1.0/(XX*XX);
       double YY=Math.pow((Math.pow((1.0-x),1.5)+1.9*Math.pow(x,0.6)*Math.pow((rol/rov),0.35)),-2.2)+XX;
       Ftp=1.0/Math.sqrt(YY);
      }
      Fnb=Fpf*FM*Math.pow((q/q0),nf)*Math.pow((D/Di0),-0.4)*Math.pow((Rp/Rp0),0.133);
      double xx1=(hnb0*Fnb);
      double xx2=(hl*Ftp);
      h=Math.pow((xx1*xx1*xx1+xx2*xx2*xx2),(1.0/3.0));
                return h-htp;
   }
}
public class HT_internal_pipe_boiling_Steiner_Taborek_ref
{
public static double bisection(if_x f,double a,double b)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   double eps=1.0e-8;
   int nmax=100;
   int i=0;
   while(Math.abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;
    i++;
   }
   if(i>=100) r=bisection(f,a,b1);
   return r;
  }
public static void main(String arg[])
{double tsat=0.0; //degree C
 double tw=20.0 ; //degree C
 double D=2.5e-2; //m
 double m=0.1; //kg/s
 double x=0.25;
 double eps=0.15e-3;
 double Rp=1e-6;
 HT_ST_ref f1=new HT_ST_ref("R134a",tw,tsat,m,x,D,eps,Rp);
 double htp=bisection(f1,10.0,50000.0);
 System.out.println("htp="+htp+" W/m^2K");
 System.out.println("q="+f1.q+" W/m^2");
}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_internal_pipe_boiling_Steiner_Taborek_ref
q0=20000.0hnb0=3500.0
Preduced=0.07213066292217872
htp=46691.71296267652 W/m^2K
q=933834.2592535305 W/m^2

> Terminated with exit code 0.
```

PROBLEM 4: Saturated R134a at T_{sat}=-5 ℃ flows with a mass flow rate of m=0.1 kg/s through a $2.0x10^{-2}$ m internal diameter vertical tube, subjected to a constant Wall temperature of Tw=5 ℃. Calculate the heat transfer and two phase heat transfer coefficient h_{TP} at the location where the vapor mass quality is x=0.2.

In order to compare results equations created by using refrigerant class and refrigerant coolProp class both (They are same except EOS created properties)

Steiner and Taborek[68] correlation for flow boiling in vertical tubes working fluid R134a eos: refrigerant & refrigerant_coolProp (both is tried to see the differences)

```
public class HT_ST_ref implements if_x
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // D internal diameter of the tube
  // m mass flow rate kg/s
  // x quality
 // Steiner and Taborek[68] correlation for flow boiling:
  public double q,Bo,E,hl,S,hpool,h,hv,G,A,P,Preduced,dT,D;
  refrigerant re;
  double Rel,Rev;
  double q0,hnb0,r0,Di0,q0NB;
  double Ftp, Fnb, Fpf,nf,FM,M,Rp,Rp0,x;
  public double kl,kv,mul,muv,Prl,Prv,Cpl,Cpv,hfg,rol,rov,sigma,Psat,Frl,Co,fl,fv,eps;
  public HT_ST_ref(String namei,double tw,double tsat,double m,double xi,double Di,double
epsi,double Rpi)
  { Rp0=1e-6; //m
    x=xi;
    Rp=Rpi;
    D=Di;
    Di0=0.01; //m
    eps=epsi;
    String ref_name[]={"Methane",
"Ethane",
"n-Propane",
"n-Butane",
"n-Pentane",
"Isopentane",
"n-Hexane",
"n-Heptane",
"CycloHexane",
"Benzene",
"Toluene",
"Diphenyl",
"Methanol",
"Ethanol",
"n-Propanol",
"Isopropanol",
"n-Butanol",
"Isobutanol",
"Acetone",
"R11",
"R12",
"R13",
"R13B1",
"R22",
"R23",
"R113",
"R114",
"R115",
"R123",
"R134a",
```

```
"R152A",
"R226",
"R227EA",
"RC318",
"R502",
"Chloromethane",
"Tetrachloromethane",
"Tetrafluoromethane",
"Helium I",
"Hydrogen",
"Neon",
"Nitrogen",
"Argon",
"Oxygen",
"Water",
"Ammonia",
"CarbonDioxide",
"SulphurHexafluoride"};
double qq0[]={20000,
20000,
20000,
```

```
1000,
10000,
10000,
10000,
10000,
10000,
150000,
150000,
150000,
150000
};
double hh[]={8060,
5210,
4000,
3300,
3070,
2940,
2840,
2420,
2420,
2730,
2910,
2030,
2770,
3690,
3170,
2920,
2750,
2940,
3270,
2690,
3290,
3910,
3380,
3930,
4870,
2180,
2460,
2890,
2600,
3500,
4000,
3700,
3800,
2710,
2900,
4790,
2320,
4500,
1990,
12220,
8920,
4380,
3870,
4120,
25580,
```

```
36640,
18890,
12230
};
for(int i=0;i<ref_name.length;i++)
{if(ref_name[i].equals(namei)) {q0=qq0[i];hnb0=hh[i];}}
   A=Math.PI*D*D/4.0; //m^2
   G=m/A;  //kg/m^2s
   dT=tw-tsat; //degree K
   re=new refrigerant(namei);
   M=re.r.M; //kg/kmol
   kl=re.r.k_l(tsat);    // W/(mK)
   kv=re.r.k_v(tsat);    // W/(mK)
   mul=re.r.viscosity_l(tsat);  //kg/(m.s)=Pa.s
   muv=re.r.viscosity_v(tsat);       //kg/(m.s)=Pa.s
   Prl=re.r.Pr_l(tsat);
   Prv=re.r.Pr_v(tsat);
   Cpl=re.r.Cpl(tsat)*1e3;  //J/kgK
   Cpv=re.r.Cpv(tsat)*1e3;  //J/kgK
   hfg=re.r.h_lv(tsat)*1e3; //J/kg
   rol=re.r.rol(tsat);;  //kg/m^3
   rov=re.r.rov(tsat);;  //kg/m^3
   sigma=re.r.yuzeygerilim(tsat);  //N/m
   Psat=re.Psb(tsat)*1e3; //Pa

System.out.println("M="+M+"kl="+kl+"kv="+kv+"mul="+mul+"muv="+muv+"Prl="+Prl+"Prv="+
Prv+"Cpl="+Cpl+"Cpv="+Cpv+"hfg="+hfg+"rol="+rol+"rov="+rov+"sigma="+sigma+"Psat="+Psat
);
   Preduced=Psat/(re.r.Pc);
   dT=tw-tsat;
   Rel=G*D/mul;
   Rev=G*D/muv;
   double g=9.806; // m/s^2
   fl=f_Goudar(Rel,eps/D);
   hl=(kl/D)*(fl/8.0)*(Rel-1000.0)*Prl/(1.07+12.7*Math.pow((fl/8.0),0.5)*(Math.pow(Prl,(2.0/3.0)-
1.0)));
   q0NB=2*sigma*(tsat+273.15)*hl/(r0*rov*hfg);
   fv=f_Goudar(Rev,eps/D);
   hv=(kv/D)*(fv/8.0)*(Rev-
1000.0)*Prv/(1.07+12.7*Math.pow((fv/8.0),0.5)*(Math.pow(Prv,(2.0/3.0)-1.0)));
   FM=0.377+0.199*Math.log(M)+0.000028427*M*M;
   nf=0.8-0.1*Math.exp(1.75*Preduced);
   double pr7=Preduced*Preduced*Preduced*Preduced*Preduced*Preduced*Preduced;
   Fpf=2.816*Math.pow(Preduced,0.45)+(3.4+1.7/(1-pr7))*Math.pow(Preduced,3.7);
  }
 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
```

```
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
  public double func(double htp)
  {   q=htp*dT;
      if(q<q0NB)
      {Ftp=Math.pow((Math.pow((1.0-x),1.5)+1.9*Math.pow(x,0.6)*Math.pow((rol/rov),0.35)),1.1);}
      else
      {double XX=(hl/hv)*Math.pow(x,0.01)*(1.0+8.0*Math.pow((1.0-
x),0.7)*Math.pow((rol/rov),0.67));
       XX=1.0/(XX*XX);
       double YY=Math.pow((Math.pow((1.0-
x),1.5)+1.9*Math.pow(x,0.6)*Math.pow((rol/rov),0.35)),-2.2)+XX;
       Ftp=1.0/Math.sqrt(YY);
      }
      Fnb=Fpf*FM*Math.pow((q/q0),nf)*Math.pow((D/Di0),-0.4)*Math.pow((Rp/Rp0),0.133);
      double xx1=(hnb0*Fnb);
      double xx2=(hl*Ftp);
      h=Math.pow((xx1*xx1*xx1+xx2*xx2*xx2),(1.0/3.0));
            return h-htp;
   }
}
```

```
public class HT_ST_ref1 implements if_x
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // D internal diameter of the tube
  // m mass flow rate kg/s
  // x quality
 // Steiner and Taborek[68] correlation for flow boiling:
 public double T0=273.15; //degree K
  public double q,Bo,E,hl,S,hpool,h,hv,G,A,P,Preduced,dT,D;
  refrigerant_coolProp re;
  String name;
  double Rel,Rev;
  double q0,hnb0,r0,Di0,q0NB;
  double Ftp, Fnb, Fpf,nf,FM,M,Rp,Rp0,x;
  public double kl,kv,mul,muv,Prl,Prv,Cpl,Cpv,hfg,rol,rov,sigma,Psat,Frl,Co,fl,fv,eps;
  public HT_ST_ref1(String namei,double tw,double tsat,double m,double xi,double Di,double
epsi,double Rpi)
  {name=namei;
  Rp0=1e-6; //m
    x=xi;
    Rp=Rpi;
    D=Di;
    Di0=0.01; //m
    eps=epsi;
    String ref_name[]={"Methane",
"Ethane",
"n-Propane",
"n-Butane",
"n-Pentane",
"Isopentane",
"n-Hexane",
```

```
"n-Heptane",
"CycloHexane",
"Benzene",
"Toluene",
"Diphenyl",
"Methanol",
"Ethanol",
"n-Propanol",
"Isopropanol",
"n-Butanol",
"Isobutanol",
"Acetone",
"R11",
"R12",
"R13",
"R13B1",
"R22",
"R23",
"R113",
"R114",
"R115",
"R123",
"R134a",
"R152A",
"R226",
"R227EA",
"RC318",
"R502",
"Chloromethane",
"Tetrachloromethane",
"Tetrafluoromethane",
"Helium I",
"Hydrogen",
"Neon",
"Nitrogen",
"Argon",
"Oxygen",
"Water",
"Ammonia",
"CarbonDioxide",
"SulphurHexafluoride"};
double qq0[]={20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
```

```
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
20000,
1000,
10000,
10000,
10000,
10000,
10000,
150000,
150000,
150000,
150000
};
double hh[]={8060,
5210,
4000,
3300,
3070,
2940,
2840,
2420,
2420,
2730,
2910,
2030,
2770,
3690,
3170,
2920,
2750,
2940,
3270,
2690,
3290,
3910,
```

```
3380,
3930,
4870,
2180,
2460,
2890,
2600,
3500,
4000,
3700,
3800,
2710,
2900,
4790,
2320,
4500,
1990,
12220,
8920,
4380,
3870,
4120,
25580,
36640,
18890,
12230
};
for(int i=0;i<ref_name.length;i++)
{if(ref_name[i].equals(name)) {q0=qq0[i];hnb0=hh[i];}}
   A=Math.PI*D*D/4.0; //m^2
   G=m/A;  //kg/m^2s
   dT=tw-tsat; //degree K
   re=new refrigerant_coolProp(name);
   double al[]=re.property("tx",tsat+T0,0.0);
    double av[]=re.property("tx",tsat+T0,1.0);
   M=re.M; //kg/kmol
   kl=al[12];    // W/(mK)
   kv=av[12];    // W/(mK)
   mul=al[11];  //kg/(m.s)=Pa.s
   muv=av[11];       //kg/(m.s)=Pa.s
   Prl=al[14];
   Prv=av[14];
   Cpl=al[7]*1e3;  //J/kgK
   Cpv=av[7]*1e3;  //J/kgK
   hfg=(av[3]-al[3])*1e3; //J/kg
   rol=al[15];  //kg/m^3
   rov=av[15];  //kg/m^3
   sigma=al[13];  //N/m
   Psat=av[0]*1e3; //Pa

System.out.println("M="+M+"kl="+kl+"kv="+kv+"mul="+mul+"muv="+muv+"Prl="+Prl+"Prv="+
Prv+"Cpl="+Cpl+"Cpv="+Cpv+"hfg="+hfg+"rol="+rol+"rov="+rov+"sigma="+sigma+"Psat="+Psat
);
   Preduced=Psat/re.Pcrt;
   dT=tw-tsat;
   Rel=G*D/mul;
```

```
    Rev=G*D/muv;
    double g=9.806; // m/s^2
    fl=f_Goudar(Rel,eps/D);
    hl=(kl/D)*(fl/8.0)*(Rel-1000.0)*Prl/(1.07+12.7*Math.pow((fl/8.0),0.5)*(Math.pow(Prl,(2.0/3.0)-
1.0)));
    q0NB=2*sigma*(tsat+273.15)*hl/(r0*rov*hfg);
    fv=f_Goudar(Rev,eps/D);
    hv=(kv/D)*(fv/8.0)*(Rev-
1000.0)*Prv/(1.07+12.7*Math.pow((fv/8.0),0.5)*(Math.pow(Prv,(2.0/3.0)-1.0)));
    FM=0.377+0.199*Math.log(M)+0.000028427*M*M;
    nf=0.8-0.1*Math.exp(1.75*Preduced);
    double pr7=Preduced*Preduced*Preduced*Preduced*Preduced*Preduced*Preduced;
    Fpf=2.816*Math.pow(Preduced,0.45)+(3.4+1.7/(1-pr7))*Math.pow(Preduced,3.7);
  }
 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
  public double func(double htp)
  {   q=htp*dT;
     if(q<q0NB)
     {Ftp=Math.pow((Math.pow((1.0-x),1.5)+1.9*Math.pow(x,0.6)*Math.pow((rol/rov),0.35)),1.1);}
     else
     {double XX=(hl/hv)*Math.pow(x,0.01)*(1.0+8.0*Math.pow((1.0-
x),0.7)*Math.pow((rol/rov),0.67));
      XX=1.0/(XX*XX);
      double YY=Math.pow((Math.pow((1.0-
x),1.5)+1.9*Math.pow(x,0.6)*Math.pow((rol/rov),0.35)),-2.2)+XX;
      Ftp=1.0/Math.sqrt(YY);
     }
     Fnb=Fpf*FM*Math.pow((q/q0),nf)*Math.pow((D/Di0),-0.4)*Math.pow((Rp/Rp0),0.133);
     double xx1=(hnb0*Fnb);
     double xx2=(hl*Ftp);
     h=Math.pow((xx1*xx1*xx1+xx2*xx2*xx2),(1.0/3.0));
           return h-htp;
   }
}
```

```
public class HT_internal_pipe_boiling_Steiner_Taborek_ref
{
public static double bisection(if_x f,double a,double b)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   double eps=1.0e-8;
```

```
   int nmax=100;
   int i=0;
   while(Math.abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;
    i++;
   }
   if(i>=100) r=bisection(f,a,b1);
   return r;
  }
public static void main(String arg[])
{double tsat=-5.0; //degree C
 double tw=5.0 ; //degree C
 double D=2.0e-2; //m
 double m=0.1; //kg/s
 double x=0.2;
 double eps=0.15e-3;
 double Rp=1e-6;
 HT_ST_ref1 f1=new HT_ST_ref1("R134a",tw,tsat,m,x,D,eps,Rp);
 double htp=bisection(f1,10.0,10000.0);
 System.out.println("Equation of State : refrigerant_coolProp");
 System.out.println("htp="+htp+" W/m^2K");
 System.out.println("q="+f1.q+" W/m^2");
  System.out.println("Equation of State : refrigerant");
 HT_ST_ref f2=new HT_ST_ref("R134a",tw,tsat,m,x,D,eps,Rp);
 double htp1=bisection(f2,10.0,10000.0);
 System.out.println("htp="+htp1+" W/m^2K");
 System.out.println("q="+f2.q+" W/m^2");
}
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan
Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v20230129-
1123\jre\bin\javaw.exe" HT_internal_pipe_boiling_Steiner_Taborek_ref
M=102.032kl=0.09424213290076607kv=0.011082023038876166mul=2.8427988693524353E-
4muv=1.0543228882746463E-
5Prl=4.005538262873735Prv=0.8326451664054012Cpl=1327.8831414296967Cpv=875.196111166
0739hfg=202342.60831036032rol=1311.1176451172357rov=12.077162327075051sigma=0.012133
224451455038Psat=243342.36987785524
Equation of State : refrigerant_coolProp
htp=6033.440333903181 W/m^2K
q=60334.403339031815 W/m^2

Equation of State : refrigerant
M=102.03kl=0.09424994457991383kv=0.011084923415916161mul=2.8912840150414394E-
4muv=1.0530019369340276E-
5Prl=4.0720907346850845Prv=0.8316690638497095Cpl=1327.4182822297003Cpv=875.49581409
15844hfg=202336.2904322864rol=1311.0942306009108rov=12.076433021644615sigma=0.012285
213547332367Psat=243346.58575187082
htp=6043.948534629562 W/m^2K
q=60439.48534629562 W/m^2

> Terminated with exit code 0.
```

Turgut, Asker, Coban correlation[50] for flow boiling in horizontal tubes and annuli

This model is specificaly developed for the refrigerant R-134a.

$Re_l = \frac{m(1-x)D}{A_{cross-section}\mu_l} = \frac{G(1-x)D}{\mu_l}$ where x is vapor mass quality. Where $G = \frac{m}{A_{cross-section}}$ is the mass flux (kg/sm^2) and m is mass flow rate (kg/s)

Martinelli parameter which has a form of:

$$X_{tt} = \left(\frac{1-x}{x}\right)^{0.9}\left(\frac{\rho_v}{\rho_l}\right)^{0.5}\left(\frac{\mu_l}{\mu_v}\right)^{0.1}$$

Froude number:

$Fr_l = \frac{G^2}{\rho_l^2 gD}$ where g is gravitational constant (can be taken as g=9.806 m/s^2)

heat transfer coefficient of the convective boiling dominant regime

$$h_{cb} = \left(0.023\left(\frac{k_l}{D}\right)Re_l^{0.8}Pr_l^{0.4}\right)\left(C_1\left(\frac{1}{X_{tt}}\right)^{C_2}\right)$$

$$h_{nb} = C_3 h_{Goren}^{C_4} P_r^{C_5}(1-x)^{C_6}$$

where h_{Goren} is the fluid-specific reduced pressure correlation proposed by Gorenflo [51]. Gorenflo's correlation can be expressed as

$$h_{Goren} = h_0 F_{PF}\left(\frac{q}{q_0}\right)^{nf}\left(\frac{R_p}{R_{p0}}\right)^{0.133}$$

where h_0 is the reference heat transfer coefficient(for R134a, h_0 = 4500 J/kg); q_0 = 20,000 W/m^2 is the reference heat flux; p_{r0} = 0.1 is the reference reduced pressure. R_p is the surface roughness and R_{p0} is the reference surface roughness. F_{PF} is the pressure correction factor that is defined as:

$F_{PF} = 1.2P_r^{0.27} + \left(2.5 + \frac{1}{1-P_r}\right)P_r$ where $P_r = \frac{P}{P_{critical}}$ is eudced pressure

The exponent nf is associated with the effect of reduced pressure on the heat flux term and can be described as

$nf = 0.9 - 0.3P_r^{0.3}$

Total convective heat transfer is calculated as:

$$h_{tp} = \sqrt{(h_{cb}^2 + h_{nb}^2)}$$

Coefficients used in the correlation:

Constants	Values
C_1	1.63366
C_2	0.94494
C_3	9.86075
C_4	0.80244
C_5	0.28773
C_6	0.40317

PROBLEM : Saturated R134a at T_{sat}=0 C flows with a mass flow rate of m=0.1 kg/s through a 2.5x10^{-2} m internal diameter vertical tube, subjected to a constant Wall temperature of tw=8 C. Calculate the heat transfer and two phase heat transfer coefficient h_{TP} at the location where the vapor mass quality is x=0.25.

Turgut, Asker, Coban correlation[50] for flow boiling in horizontal tubes working fluid: R134a

```
/* ref_CS_Data property list
  0  M,
  1  BP,
  2  FP,
  3  Tc,
  4  Pc,
  5  roc,
  6  Psb(t),
  7  Psd(t),
  8  rol(t),
  9  rov(t),
  10  h_l(t),
  11  h_v(t),
  12  h_lv(t),
  13  s_l(t),
```

```
  14  s_v(t),
  15  s_lv(t),
  16  viscosity_l(t),
  17  viscosity_v(t),
  18  k_l(t),
  19  k_v(t),
  20  Cpl(t),
  21  Cpv(t),
  22  soundv_l(t),
  23  soundv_v(t),
  24  surfacetension(t),
  25  Pr_l(t),
  26  Pr_v(t)
*/
class implements if_x
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // D internal diameter of the tube
  // m mass flow rate kg/s
  // x quality
  public double q,Bo,G,A,P,Preduced,dT,D,Xtt;
  ref_CS st;
  double C[];
  String position;
  public int option;
  public double f2;
  public double Ffl;
  public double kl,mul,muv,Prl,Cpl,hfg,rol,rov,sigma,Psat,Rel,nf,FPF,Frl;
  public double hcb,hnb,hgoren,h,h0,q0,pr0,x;
  public HT_TAC(double tw,double tsat,double m,double xi,double Di)
  { double C1[]={1.63366,0.94494,9.86075,0.80244,0.28733,0.40317};
    C=C1;
    D=Di;
    x=xi;
    Ffl=1.0; //case for water
    A=Math.PI*D*D/4.0;
    dT=tw-tsat;
    st=new ref_CS("R134a");
   double a[]=st.property(tsat);
   kl=a[18];   // W/(mK)
   mul=a[16];  //kg/(m.s)=Pa.s
   muv=a[17];  //kg/(m.s)=Pa.s
   Prl=a[25];
   Cpl=a[20]*1e3;  //J/kgK
   hfg=(a[11]-a[10])*1e3; //J/kg
   rol=a[8];  //kg/m^3
   rov=a[9];  //kg/m^3
   sigma=a[24];  //N/m
   Psat=a[6]*1e3; //Pa
   Preduced=Psat/(a[4]*1e3);
   dT=tw-tsat;
   Xtt=Math.pow(((1-x)/x),0.9)*Math.pow((rov/rol),0.5)*Math.pow((mul/muv),0.1);
   Rel=G*(1.0-x)*D/mul;
   double g=9.806; // m/s^2
   Frl=G*G/(rol*rol*g*D);
   nf=0.9-0.3*Math.pow(Prl,0.3);
   FPF=1.2*Math.pow(Prl,0.27)+(2.5+1/(1-Preduced))*Preduced;
   h0=4500.0; //J/kg
   q0=20000; //W/m^2
   pr0=0.1;
  }
  public double func(double htp)
  {   q=htp*dT;
      Bo=q/(hfg*G);
      hcb=(0.023*(kl/D)*Math.pow(Rel,0.8)*Math.pow(Prl,0.4))*(C[0]*Math.pow((1.0/Xtt),C[1]));
      hgoren=h0*FPF*Math.pow((q/q0),nf);
      hnb=C[2]*Math.pow(hgoren,C[3])*Math.pow(Prl,C[4])*Math.pow((1-x),C[5]);
      h=Math.sqrt(hcb*hcb+hnb*hnb);
                return (h-htp);
   }
}
public class HT_internal_pipe_boiling_TAC
{
public static double bisection(if_x f,double a,double b)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   double eps=1.0e-8;
```

```
   int nmax=100;
   int i=0;
   while(Math.abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;
    i++;
   }
   if(i>=100) r=bisection(f,a,b1);
   return r;
  }
public static void main(String arg[])
{double tsat=0.0; //degree C
 double tw=8.0 ; //degree C
 double D=2.5e-2; //m
 double m=0.1; //kg/s
 double x=0.25;
 HT_TAC f=new HT_TAC(tw,tsat,m,x,D);
 double htp=bisection(f,100.0,50000.0);
 System.out.println("htp="+htp+" W/m^2K");
 System.out.println("q="+f.q+" W/m^2");
}}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_internal_pipe_boiling_TAC
htp=60955.50549209458 W/m^2K
q=487644.0439367566 W/m^2

> Terminated with exit code 0.
```

Model is also developed for **R-22(chlorodifloromethane)** and **R-744(carbondioxide)**. Like the previous set,this set are also based on collection of data and applying non-linear curve fitting to obtain best fit coefficients. We should note that equations can be used for other fluids as well, but aerror range will be grow. Developing of boiling heat transfer equations for each fluid indivdually ensures a better fit to the experimental data. Our basic eqautions are still the same format as the previous R134a case
Convection number:

$$Co = \left(\frac{1-x}{x}\right)^{0.8}\left(\frac{\rho_v}{\rho_l}\right)^{0.5}$$

$$h_{cb} = \left(0.023\left(\frac{k_l}{D}\right)Re_l^{0.8}Pr_l^{0.4}\right)\left(C_1\left(\frac{1}{Co}\right)^{C_2}\right)$$

Nucleate boiling mechanism of the proposed correlation is formulated as:

$$h_{nb} = C_3 h_{cooper}^{C_4} P_r^{C_5}(1-x)^{C_6}$$

where h_{cooper} is the fluid-specific reduced pressure correlation proposed by Cooper[69]. Cooper's correlation can be expressed as

$$h_{cooper} = 55P_r^{0.12-0.087\ln(\varepsilon)}[-0.4343\ln(P_r)]^{-0.55}M^{-0.5}q^{0.67}$$

In this equation ε is surface roughness (m), Pr is reduced pressure. Total heat transfer can be calculated as:
Total convective heat transfer is calculated as:

$$h_{tp} = \sqrt{(h_{cb}^2 + h_{nb}^2)}$$

Coefficients used in the correlation:

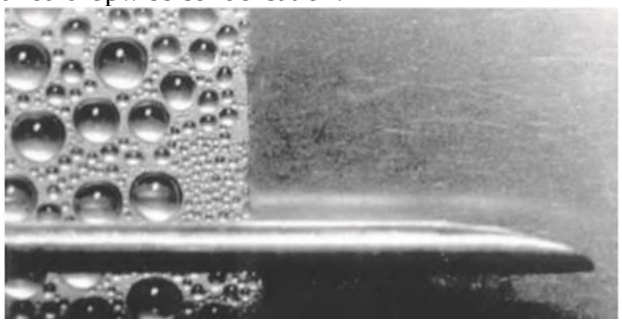
Constants	R22 Values	R744 values
C_1	1.41386119520875	1.9246533128433
C_2	0.88078846453026	0.9518395312523
C_3	3.64323982301551	0.5740708489546
C_4	1.03457978786336	1.0097850898120
C_5	0.80972363638993	-0.673980526542
C_6	-0.1058096311456	0.1260865942051

Flow boiling (in tubes) pressure drop
Total pressure drop for two-phase flow in tubes consists of change in friction, momentum and gravitational components

$$\Delta P_{total} = \Delta P_{static} + \Delta P_{momentum} + \Delta P_{frictional}$$

15.3 CONDENSATION HEAT TRANSFER

If we have a cold smooth vertical surface like a glass surface condensation will start as a smooth film on the surface. On rough surfaces though droplets happens and move downward direction as droplets. It is called dropwise condensation.



Dropwise condensation **Film condensation**

Laminar filmwise condensation on a vertical surface

Filmwise condensation has similarities to free convective cooling process. When saturated vapor contacts to a very smooth cold surface, a film of condansate occurs and moves downwards in boundary layer due to gravitational effect. Boundary layer thickness of filmwise condensation and local convective heat tansfer coefficient can be expressed as:

Boundary layer thickness:

$$\delta(x) = \left[\frac{4\mu_l k_l (T_{sat}-T_w)x}{g(\rho_l-\rho_v)\rho_l h_{fg}}\right]^{1/4}$$

$h_{fg} = h_g - h_f$ $(enthalpy\ of\ saturation\ vapor - enthalpy\ of\ saturation\ liquid)$

$$h_x(T_{sat}-T_w) = k_l \left.\frac{\partial T}{\partial x}\right|_x \cong k_l \frac{(T_{sat}-T_w)}{\delta(x)} = k_l \frac{(T_{sat}-T_w)}{\left[\frac{4\mu_l k_l (T_{sat}-T_w)x}{g(\rho_l-\rho_v)\rho_l h_{fg}}\right]^{1/4}}$$

$$h_x = \frac{k_l}{\delta(x)}$$

$$h_x = \left[\frac{g(\rho_l-\rho_v)\rho_l h_{fg} k_l^3}{4\mu_l(T_{sat}-T_w)x}\right]^{1/4}$$

Rohsenow[52] recommended that using a modified form of latent heat improve the heat transfer result from laminar film condensation equation

$h'_{fg} = h_{fg} + 0.68C_{pl}(T_{sat}-T_w)$ so eqution replaced as

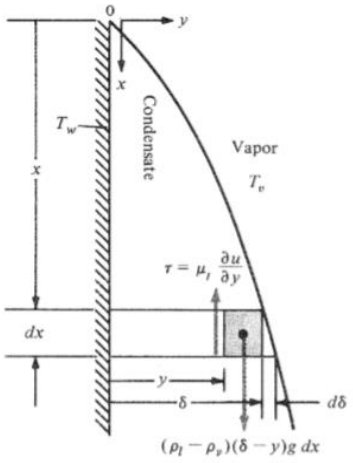
$$h_x = \left[\frac{g(\rho_l-\rho_v)\rho_l h'_{fg} k_l^3}{4\mu_l(T_{sat}-T_w)x}\right]^{1/4}$$

Average convection coefficient:

$$h_L = \frac{1}{L}\int_{x=0}^{L} h_x dx = \frac{4}{3} h_x|_{x=L}$$

$$h_L = 0.943\left[\frac{g(\rho_l-\rho_l)\rho_l h'_{fg} k_l^3}{\mu_l(T_{sat}-T_w)L}\right]^{1/4}$$

$$Nu_L = \frac{h_L L}{k_l} = 0.943\left[\frac{g(\rho_l-\rho_l)\rho_l h'_{fg} k_l^3 L^3}{\mu_l(T_{sat}-T_w)}\right]^{1/4}$$



$q = h_L A(T_{sat} - T_w)$=m$h'_{fg} = m\left(h_g - h_f\right)$ where m is condensation rate

Turbulent filmwise condensation on a vertical surface

Condensation rate:

$m = \rho_l UW\delta$ where W is the width of the plate

Reynolds number in the film condensation boundary layer can be given as

$Re_\delta = \frac{4m}{\mu_l W} = \frac{4\rho_l U\delta}{\mu_l}$ Flow is laminar when $Re_\delta < 30$ $30 \le Re_\delta \le 1800$ is transation region and $Re_\delta > 1800$ is fully turbulent region. Considering that $\frac{m}{W} = \frac{g(\rho_l - \rho_v)\rho_l\delta^3}{3\mu_l}$ $Re_\delta = \frac{4g(\rho_l - \rho_v)\rho_l\delta^3}{3\mu_l^2}$ so for this case

Laminar flow convective film heat transfer coefficient

$$Nu_L = \frac{h_L\left(\nu_l^2/g\right)^{1/3}}{k_l} = 1.47Re_\delta^{-1/3} \quad Re_\delta < 30$$

Transation region (laminar wavy flow) convective film heat transfer coefficient (Kutateladze[53])

$$Nu_L = \frac{h_L\left(\nu_l^2/g\right)^{1/3}}{k_l} = \frac{Re_\delta}{1.08Re_\delta^{1.22} - 5.2} \quad 30 \le Re_\delta < 1800$$

Turbulent region convective film heat trasnfer coefficient (Labuntsov[54])

$$Nu_L = \frac{h_L\left(\nu_l^2/g\right)^{1/3}}{k_l} = \frac{Re_\delta}{8750 + 58Pr_l^{-0.5}\left(Re_\delta^{0.75} - 253\right)} \quad Re_\delta \ge 1800 \;\; Pr_l \ge 1$$

These equations can also be written as a function of dimensionless parameter P instead of Re_δ as:

$$P = \frac{4k_l(T_{sat} - T_w)}{\mu_l h'_{fg}\left(\nu_l^2/g\right)^{1/3}}$$

Laminar region :

$$Nu_L = \frac{h_L\left(\nu_l^2/g\right)^{1/3}}{k_l} = 0.943P^{-1/4} \quad P < 15.8$$

Transation region (laminar wavy flow):

$$Nu_L = \frac{h_L\left(\nu_l^2/g\right)^{1/3}}{k_l} = \frac{1}{P}(0.68P + 0.89)^{0.82} \quad 15.8 \le P < 2530$$

Turbulent region convective film heat trasnfer coefficient

$$Nu_L = \frac{h_L\left(\nu_l^2/g\right)^{1/3}}{k_l} = \frac{1}{P}\left[(0.024P - 53)Pr_l^{1/2} + 89\right]^{4/3} \quad P \ge 2530 \quad Pr_l \ge 1$$

PROBLEM : The outer surface of a vertical tube, which is 1 m long has an outer diameter of 0.08 m, is exposed to saturated steam at atmospheric pressure and is maintained at 50 C by the flow of water through the tube. Calculate rate of heat transfer and rate of condensation.

Note that liquid properties are taken at the average temperature $T_f = \frac{(T_{sat} + T_w)}{2}$, but vapor properties, including h_{fg} are taken at saturation temperature T_{sat}

```
public class HT_vertical_film_condensation
{ /* a[0] "P, pressure       ";
  a[1] "T, temperature     ";
```

```
   a[2] "v, specific volume ";
   a[3] "h, enthalpy        ";
   a[4] "u, internal energy ";
   a[5] "s, entropy         ";
   a[6] "x, quality         ";
   a[7] "Cp, specific heat at constant pressure ";
   a[8] "Cv, specific heat at constant volume ";
   a[9] ""+'\u03B1'+" isobaric thermal expansion coefficient ";
   a[10] ""+'\u03BA'+" isothermal compressibility" ;
   a[11] ""+'\u03B7'+" Dynamic viscosity" ;
   a[12] "k Thermal conductivity" ;
   a[13] ""+'\u03C3'+" Surface tension" ;
   a[14] "Prandtl number" ;
   a[15] ""+'\u03C1'+" density" ;
   a[16] " speed of sound" ;
*/
public static double h(double tsat,double tw, double L,double W,boolean pr)
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // L vertical length of vertical plate
  // W width of the plate
  steamIAPWS_IF97 st=new steamIAPWS_IF97();
  double tf=(tw+tsat)/2.0;
  double al[]=st.propertyC("tx",tf,0.0);
  double av[]=st.propertyC("tx",tsat,1.0);
  double al1[]=st.propertyC("tx",tsat,0.0);
  double kl=al[12];   // W/(mK)
  double mul=al[11];  //kg/(m.s)=Pa.s
  double muv=av[11];  //kg/(m.s)=Pa.s
  double Prl=al[14];
  double Cpl=al[7]*1e3;  //J/kgK
  double hfg=(av[3]-al1[3])*1e3; //J/kg
  double dT=tsat-tw;
  double hfgp=hfg+0.68*Cpl*dT;
  double rol=al[15];  //kg/m^3
  double rov=av[15];  //kg/m^3
  double Psat=al[0]*1e3; //Pa
  double g=9.806; //m/s^2
  double hL=0.943*Math.pow((g*rol*(rol-rov)*kl*kl*kl*hfgp/(mul*dT*L)),0.25);
  double q=hL*(L*W)*dT;
  double m=q/hfgp;
  if(pr)
  {  System.out.println("kl="+kl+"mul="+mul+"muv="+muv+"Prl="+Prl+"Cpl="+Cpl+"\nhfg="+hfg+" hfgp="+hfgp+"
rol="+rol+"rov="+rov+"Psat="+Psat);
     System.out.println("hL="+hL+" W/m^2K");
     System.out.println("q="+q+" W");
     System.out.println("m="+m+" kg/s");
  }
  return L;
}
public static void main(String arg[])
{double tsat=100.0; //degree C
 double tw=50.0;   //degree C
 double D=0.08; //m
 double W=Math.PI*D;
 double L=1.0;
 System.out.println("tsat="+tsat+" degree C");
 System.out.println("tw="+tw+" degree C");
 double hL=h(tsat,tw,L,W,true);
}
}
```

```
 ---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_vertical_film_condensation
tsat=100.0 degree C
tw=50.0 degree C
kl=0.6634381991038097mul=3.7740743463307E-4muv=1.2232158121714269E-5Prl=2.3845056244104246Cpl=4191.682441947153
hfg=2256472.8742231308 hfgp=2398990.077249334 rol=974.8288462197903rov=0.5981359925257022Psat=38595.36268655684
hL=4066.2545848657896 W/m^2K
q=51098.06212576072 W
m=0.02129982220866433 kg/s

> Terminated with exit code 0.
```

```
public class HT_vertical_film_condensation1
{ /* a[0] "P, pressure        ";
```

```
   a[1] "T, temperature     ";
   a[2] "v, specific volume ";
   a[3] "h, enthalpy        ";
   a[4] "u, internal energy ";
   a[5] "s, entropy         ";
   a[6] "x, quality         ";
   a[7] "Cp, specific heat at constant pressure ";
   a[8] "Cv, specific heat at constant volume ";
   a[9] ""+'\u03B1'+" isobaric thermal expansion coefficient ";
   a[10] ""+'\u03BA'+" isothermal compressibility" ;
   a[11] ""+'\u03B7'+" Dynamic viscosity" ;
   a[12] "k Thermal conductivity" ;
   a[13] ""+'\u03C3'+" Surface tension" ;
   a[14] "Prandtl number" ;
   a[15] ""+'\u03C1'+" density" ;
   a[16] " speed of sound" ;
*/
public static double h(double tsat,double tw, double L,double W,boolean pr)
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // L vertical length of vertical plate
  // W width of the plate
  steamIAPWS_IF97 st=new steamIAPWS_IF97();
  double tf=(tw+tsat)/2.0;
  double al[]=st.propertyC("tx",tf,0.0);
  double av[]=st.propertyC("tx",tsat,1.0);
  double al1[]=st.propertyC("tx",tsat,0.0);
  double kl=al[12];   // W/(mK)
  double mul=al[11];  //kg/(m.s)=Pa.s
  double muv=av[11];  //kg/(m.s)=Pa.s
  double Prl=al[14];
  double Cpl=al[7]*1e3;  //J/kgK
  double hfg=(av[3]-al1[3])*1e3; //J/kg
  double dT=tsat-tw;
  double hfgp=hfg+0.68*Cpl*dT;
  double rol=al[15];  //kg/m^3
  double rov=av[15];  //kg/m^3
  double nul=mul/rol;
  double Psat=al[0]*1e3; //Pa
  double g=9.806; //m/s^2
  double delta=Math.pow(4.0*kl*mul*dT*L/(g*rol*(rol-rov)*hfgp),0.25);
  double Red=4*g*(rol-rov)*rol*delta*delta*delta/(3.0*mul*mul);
  double hL=0;
  if(Red<30)
  {hL=kl/Math.pow((nul*nul/g),(1.0/3.0))*(1.47*Math.pow(Red,(-1.0/3.0)));}
  else if(Red>=30 && Red<1800)
  {hL=kl/Math.pow((nul*nul/g),(1.0/3.0))*(Red/(1.08*Math.pow(Red,1.22)-5.2));}
  else
  {hL=kl/Math.pow((nul*nul/g),(1.0/3.0))*(Red/(8750+58*Math.pow(Prl,-0.5)*(Math.pow(Red,0.75)-253)));}
  double q=hL*(L*W)*dT;
  double m=q/hfgp;
  if(pr)
  {  System.out.println("delta="+delta);
     System.out.println("Red="+Red);
              System.out.println("kl="+kl+"mul="+mul+"muv="+muv+"Prl="+Prl+"Cpl="+Cpl+"\nhfg="+hfg+" hfgp="+hfgp+"
rol="+rol+"rov="+rov+"Psat="+Psat);
     System.out.println("hL="+hL+" W/m^2K");
     System.out.println("q="+q+" W");
     System.out.println("m="+m+" kg/s");
  }
  return L;
}
public static void main(String arg[])
{double tsat=100.0; //degree C
 double tw=50.0;   //degree C
 double D=0.08; //m
 double W=Math.PI*D;
 double L=1.0;
 System.out.println("tsat="+tsat+" degree C");
 System.out.println("tw="+tw+" degree C");
 double hL=h(tsat,tw,L,W,true);
}}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_vertical_film_condensation1
tsat=100.0 degree C
```

```
tw=50.0 degree C
delta=2.17586826516166E-4
Red=898.0441845705835
kl=0.6634381991038097mul=3.7740743463307E-4muv=1.2232158121714269E-5Prl=2.3845056244104246Cpl=4191.682441947153
hfg=2256472.8742231308 hfgp=2398990.077249334 rol=974.8288462197903rov=0.5981359925257022Psat=38595.36268655684
hL=5551.568680075589 W/m^2K
q=69763.06952489863 W
m=0.029080182609545636 kg/s

> Terminated with exit code 0.
```

```
public class HT_vertical_film_condensation2
{ /* a[0] "P, pressure        ";
   a[1] "T, temperature     ";
   a[2] "v, specific volume ";
   a[3] "h, enthalpy        ";
   a[4] "u, internal energy ";
   a[5] "s, entropy         ";
   a[6] "x, quality         ";
   a[7] "Cp, specific heat at constant pressure ";
   a[8] "Cv, specific heat at constant volume ";
   a[9] ""+'\u03B1'+" isobaric thermal expansion coefficient ";
   a[10] ""+'\u03BA'+" isothermal compressibility" ;
   a[11] ""+'\u03B7'+" Dynamic viscosity" ;
   a[12] "k Thermal conductivity" ;
   a[13] ""+'\u03C3'+" Surface tension" ;
   a[14] "Prandtl number" ;
   a[15] ""+'\u03C1'+" density" ;
   a[16] " speed of sound" ;
*/
public static double h(double tsat,double tw, double L,double W,boolean pr)
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // L vertical length of vertical plate
  // W width of the plate
  steamIAPWS_IF97 st=new steamIAPWS_IF97();
  double tf=(tw+tsat)/2.0;
  double al[]=st.propertyC("tx",tf,0.0);
  double av[]=st.propertyC("tx",tsat,1.0);
  double al1[]=st.propertyC("tx",tsat,0.0);
  double kl=al[12];   // W/(mK)
  double mul=al[11];  //kg/(m.s)=Pa.s
  double muv=av[11];  //kg/(m.s)=Pa.s
  double Prl=al[14];
  double Cpl=al[7]*1e3;  //J/kgK
  double hfg=(av[3]-al1[3])*1e3; //J/kg
  double dT=tsat-tw;
  double hfgp=hfg+0.68*Cpl*dT;
  double rol=al[15];  //kg/m^3
  double rov=av[15];  //kg/m^3
  double nul=mul/rol;
  double Psat=al[0]*1e3; //Pa
  double g=9.806; //m/s^2
  double P=kl*dT*L/(mul*hfgp*Math.pow((nul*nul/g),(1.0/3.0)));
  double hL=0;
  if(P<15.8)
  {hL=kl/Math.pow((nul*nul/g),(1.0/3.0))*(0.943*Math.pow(P,(-1.0/4.0)));}
  else if(P>=15.8 && P<2530)
  {hL=kl/Math.pow((nul*nul/g),(1.0/3.0))/P*Math.pow((0.68*P+0.89),0.82);}
  else
  {hL=kl/Math.pow((nul*nul/g),(1.0/3.0))/P*Math.pow(((0.024*P-53)*Math.sqrt(Prl)+89),(4.0/3.0));}
  double q=hL*(L*W)*dT;
  double m=q/hfgp;
  if(pr)
  {  System.out.println("P="+P);
            System.out.println("kl="+kl+"mul="+mul+"muv="+muv+"Prl="+Prl+"Cpl="+Cpl+"\nhfg="+hfg+" hfgp="+hfgp+"
rol="+rol+"rov="+rov+"Psat="+Psat);
     System.out.println("hL="+hL+" W/m^2K");
     System.out.println("q="+q+" W");
     System.out.println("m="+m+" kg/s");
  }
  return L;
}
public static void main(String arg[])
{double tsat=100.0; //degree C
 double tw=50.0;   //degree C
```

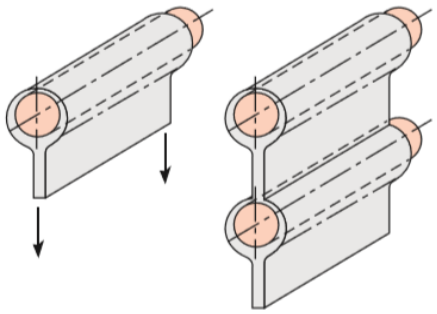
```
 double D=0.08; //m
 double W=Math.PI*D;
 double L=1.0;
 System.out.println("tsat="+tsat+" degree C");
 System.out.println("tw="+tw+" degree C");
 double hL=h(tsat,tw,L,W,true);
 }
 }
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_vertical_film_condensation2
tsat=100.0 degree C
tw=50.0 degree C
P=1476.2984093623422
kl=0.6634381991038097mul=3.7740743463307E-4muv=1.2232158121714269E-5Prl=2.3845056244104246Cpl=4191.682441947153
hfg=2256472.8742231308 hfgp=2398990.077249334 rol=974.8288462197903rov=0.5981359925257022Psat=38595.36268655684
hL=5242.815729240598 W/m^2K
q=65883.16551642912 W
m=0.027462875374611933 kg/s

> Terminated with exit code 0.
```

Film condensation on cylinders

Cylindrical surfaces as one or row of cylinders is one of the most use geometry in condensers



For a single cylinder

all liquid properties should be evaluated at the film temperature, ρ_v and h_{fg} should be evaluated at saturation temperature

For N cylinder in a row:

$$h_{D,N} = h_D N^{-1/6} = \frac{h_D}{N^{1/6}}$$

PROBLEM: Saturated steam at T_g=50 C condenses on the outer surface of a 2.5x10^{-2} m outside diameter horizontal tube maintained at a uniform temperature T_w=30 C. Calculate the length required to condansate m=0.01388 kg/s of steam.

```
 public class HT_horizontal_film_condensation
{ /* a[0] "P, pressure        ";
   a[1] "T, temperature     ";
   a[2] "v, specific volume ";
   a[3] "h, enthalpy        ";
   a[4] "u, internal energy ";
   a[5] "s, entropy         ";
   a[6] "x, quality         ";
   a[7] "Cp, specific heat at constant pressure ";
   a[8] "Cv, specific heat at constant volume ";
   a[9] ""+'\u03B1'+" isobaric thermal expansion coefficient ";
   a[10] ""+'\u03BA'+" isothermal compressibility" ;
   a[11] ""+'\u03B7'+" Dynamic viscosity" ;
   a[12] "k Thermal conductivity" ;
   a[13] ""+'\u03C3'+" Surface tension" ;
   a[14] "Prandtl number" ;
   a[15] ""+'\u03C1'+" density" ;
   a[16] " speed of sound" ;
*/
public double Q;
public double h(double m,double tsat,double tw, double D)
{ // tsat : saturation temperature degree C
```

```
  // tw surface temperature degree C
  // L vertical length of vertical plate
  // W width of the plate
  steamIAPWS_IF97 st=new steamIAPWS_IF97();
  double tf=(tw+tsat)/2.0;
  double al[]=st.propertyC("tx",tf,0.0);
  double av[]=st.propertyC("tx",tsat,1.0);
  double al1[]=st.propertyC("tx",tsat,0.0);
  double kl=al[12];   // W/(mK)
  double mul=al[11];  //kg/(m.s)=Pa.s
  double muv=av[11];  //kg/(m.s)=Pa.s
  double Prl=al[14];
  double Cpl=al[7]*1e3;  //J/kgK
  System.out.println("kl="+kl+" W/mK"+"mul="+mul+" Pas");
  System.out.println("Cpl="+Cpl+" J/kgK"+"Prl="+Prl);
  double hfg=(av[3]-al1[3])*1e3; //J/kg
  double dT=tsat-tw;
  double hfgp=hfg+0.68*Cpl*dT;
  double rol=al[15];  //kg/m^3
  double rov=av[15];  //kg/m^3
  System.out.println("rol="+rol+" kg/m^3");
  System.out.println("rov="+rov+" kg/m^3");
  double Psat=al[0]*1e3; //Pa
  System.out.println("Psat="+Psat+" kPa");
  double g=9.806; //m/s^2
  Q=m*hfg;
  double hL=kl/D*0.729*Math.pow((g*(rol-rov)*rol*hfgp*D*D*D/(mul*kl*dT)),0.25);
  return hL;
}
public static void main(String arg[])
{double tsat=50.0; //degree C
 double tw=30.0;   //degree C
 double D=0.025; //m
 double L=1.0;
  double m=0.01388;
 HT_horizontal_film_condensation hc=new  HT_horizontal_film_condensation();
 System.out.println("tsat="+tsat+" degree C");
 System.out.println("tw="+tw+" degree C");
 System.out.println("m = "+m+" kg/s");
 double h=hc.h(m,tsat,tw,D);
 System.out.println("h="+h+" W/m^2K");
 System.out.println("Q="+hc.Q+" W");
 L=hc.Q/(h*(tsat-tw))/Math.PI/D;
  System.out.println("L="+L+" m ");

}
}
```

```
 ---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_horizontal_film_condensation
tsat=50.0 degree C
tw=30.0 degree C
m = 0.01388 kg/s
kl=0.6285626394434158 W/mK mul=6.52719243154035E-4 Pas
Cpl=4178.784391773394 J/kgK  Prl=4.3393813350368
rol=992.1831489305185 kg/m^3
rov=0.08314028058584058 kg/m^3
Psat=7384.427487069553 kPa
h=8433.715295719887 W/m^2K
Q=33061.80000029381 W
L=2.495673004391233 m

> Terminated with exit code 0.
```

PROBLEM:

Air-free saturated steam at T=60 C condenses on the outer surface of 100 horizontal tubes with $2.5x10^{-2}$ m OD and 2 m long arranged in 10 by 10 square arrays. The surface of the tubes is maintained at a uniform temperature T_w=40 C. Calculate the average condensation heat transfer coefficient for the entire tube bundle, the total rate of heat transfer, and the rate of condensation at the surface of the tubes in the bundle.

```
public class HT_horizontal_film_condensation
{ /* a[0] "P, pressure        ";
  a[1] "T, temperature     ";
```

```
   a[2] "v, specific volume ";
   a[3] "h, enthalpy        ";
   a[4] "u, internal energy ";
   a[5] "s, entropy         ";
   a[6] "x, quality         ";
   a[7] "Cp, specific heat at constant pressure ";
   a[8] "Cv, specific heat at constant volume ";
   a[9] ""+'\u03B1'+" isobaric thermal expansion coefficient ";
   a[10] ""+'\u03BA'+" isothermal compressibility" ;
   a[11] ""+'\u03B7'+" Dynamic viscosity" ;
   a[12] "k Thermal conductivity" ;
   a[13] ""+'\u03C3'+" Surface tension" ;
   a[14] "Prandtl number" ;
   a[15] ""+'\u03C1'+" density" ;
   a[16] " speed of sound" ;
*/
public double Q,m;
public double h(double tsat,double tw, double D,double L,int N)
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // L vertical length of vertical plate
  // W width of the plate
  steamIAPWS_IF97 st=new steamIAPWS_IF97();
  double tf=(tw+tsat)/2.0;
  double al[]=st.propertyC("tx",tf,0.0);
  double av[]=st.propertyC("tx",tsat,1.0);
  double al1[]=st.propertyC("tx",tsat,0.0);
  double kl=al[12];   // W/(mK)
  double mul=al[11];  //kg/(m.s)=Pa.s
  double muv=av[11];  //kg/(m.s)=Pa.s
  double Prl=al[14];
  double Cpl=al[7]*1e3;  //J/kgK
  System.out.println("kl="+kl+" W/mK"+"mul="+mul+" Pas");
  System.out.println("Cpl="+Cpl+" J/kgK"+"Prl="+Prl);
  double hfg=(av[3]-al1[3])*1e3; //J/kg
  double dT=tsat-tw;
  double hfgp=hfg+0.68*Cpl*dT;
  System.out.println("hfg="+hfg+"hfgp="+hfgp);
  double rol=al[15];  //kg/m^3
  double rov=av[15];  //kg/m^3
  System.out.println("rol="+rol+" kg/m^3");
  System.out.println("rov="+rov+" kg/m^3");
  double Psat=al[0]; //Pa
  System.out.println("Psat="+Psat+" kPa");
  double g=9.806; //m/s^2
  double A=Math.PI*D*L;
  double hL=kl/D*0.729*Math.pow((g*(rol-
rov)*rol*hfgp*D*D*D/(mul*kl*dT)),0.25)*Math.pow(N,(-1.0/6.0));
  Q=A*hL*dT;
  m=Q/hfg;
  return hL;
}
public static void main(String arg[])
{double tsat=60.0; //degree C
 double tw=40.0;   //degree C
 double D=2.5e-2; //m inlet diameter
```

```
 double L=2; //m length of horizontal pipe
 int N=10;
 //double m=0.01388;
 HT_horizontal_film_condensation hc=new  HT_horizontal_film_condensation();
 System.out.println("tsat="+tsat+" degree C");
 System.out.println("tw="+tw+" degree C");

 double h=hc.h(tsat,tw,D,L,N);
 System.out.println("h="+h+" W/m^2K");
 System.out.println("Q total(100 pipe)="+hc.Q*100+" W");
  System.out.println("m total (100 pipe)= "+hc.m*100+" kg/s");
 System.out.println("L="+L+" m ");
 System.out.println("L total (100 pipe) ="+L*100+" m ");

}
}
```

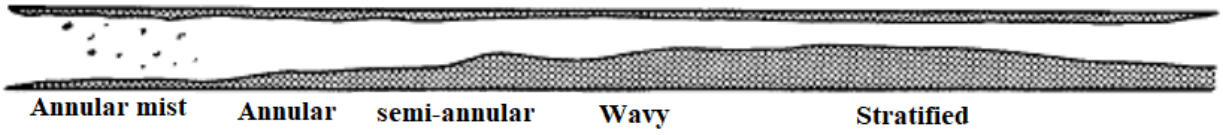
```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" HT_horizontal_film_condensation
tsat=60.0 degree C
tw=40.0 degree C
kl=0.6404605091774686 W/mKmul=5.465041540815809E-4 Pas
Cpl=4179.760087560044 J/kgKPrl=3.5665840722163864
hfg=2357691.011563888hfgp=2414535.7487547044
rol=988.0087757351531 kg/m^3
rov=0.13041794752318994 kg/m^3
Psat=12.351270434023366 kPa
h=6063.686105635067 W/m^2K
Q total(100 pipe)=1904963.172313763 W
m total (100 pipe)= 0.8079782986703485 kg/s
L=2.0 m
L total (100 pipe) =200.0 m

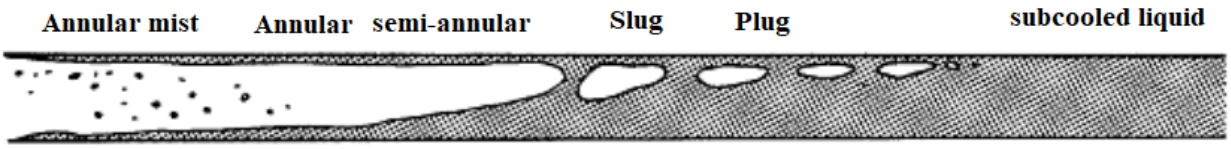
> Terminated with exit code 0.
```

Film condensation inside tubes

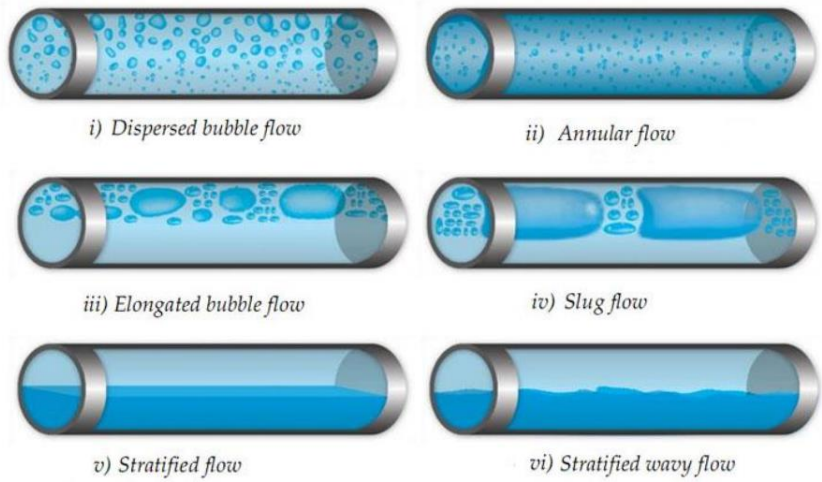
Inside tube condensation involves condensing of a saturated vapor flow inside of a tube first to a saturated vapor-liquid mixture and then to liquid. This type of heat transfer storngly depends on vapor quality, and therefore should be treated as local zones instead of an average total heat transfer coefficient. The condensing heat transfer coefficient is also a strong function of mass velocity, increasing as the mass velocity increases. Two-phase flow patterns of the horizontal tube condensation processes is given in the figure below. When condensation starts, a mist starts and a film on the tube surface started, and the annular type of flow patterns (without a mist in between) occurs. Annular flow with high stratification follows. In low mass flow rates wave flow and stratified flow region follows, in high flow region slug elongated buble region comes next.

LOW FLOW





HIGH FLOW



In order to determine which flow region condensation heat transfer falls into, the following equations can be utilised:

Reynolds number:

$$Re_l = \frac{mD(1-x)}{A\mu_l} = \frac{GD(1-x)}{\mu_l}$$

Vapor void fraction : Two different definition is given for the void fracttion First definition

$$\varepsilon = \frac{V_g}{V} = \left[1 + \frac{1-x}{x}\left(\frac{\rho_g}{\rho_l}\right)^{0.795} Re_l^{-0.016}\right]^{-1} \qquad x = \frac{m_g}{m}$$

Second definition:

$$\varepsilon = \frac{V_g}{V} = \frac{\varepsilon_h - \varepsilon_{ra}}{\ln\left(\frac{\varepsilon_h}{\varepsilon_{ra}}\right)}$$

Homogeneous void fraction: $\varepsilon_h = \left[1 + \frac{1-x}{x}\left(\frac{\rho_g}{\rho_l}\right)\right]^{-1}$

Non-homogeneous void fraction drift model of Rouhani & Axelson[104]

$$\varepsilon_{ra} = \frac{x}{\rho_g}\left([1 + 0.12(1-x)]\left[\frac{x}{\rho_g} + \frac{1-x}{\rho_l}\right] + \frac{1.18(1-x)\left[g\sigma(\rho_l - \rho_g)\right]^{0.25}}{G\rho_l^{0.5}}\right)^{-1}$$

Dobson and Chato[57,108] equations inside tubes

Annular flow correlation (original 1962 corrolation)

Martinelli parameter which has a form of:

$$X_{tt} = \left(\frac{1-x}{x}\right)^{0.9}\left(\frac{\rho_v}{\rho_l}\right)^{0.5}\left(\frac{\mu_l}{\mu_v}\right)^{0.1}$$

$Re_l = \frac{m(1-x)D}{A_{cross-section}\mu_l} = \frac{G(1-x)D}{\mu_l}$ where x is vapor mass quality. Where $G = \frac{m}{A_{cross-section}}$ is the mass flux (kg/sm^2)

$$Nu_{annular} = \frac{h_l D}{k_l} = 0.023 Re_l^{0.8} Pr_l^{0.4}\left[1 + \frac{2.22}{X_{tt}^{0.89}}\right]$$

$$h_l = 0.023\left(\frac{k_l}{D}\right) Re_l^{0.8} Pr_l^{0.4}\left[1 + \frac{2.22}{X_{tt}^{0.89}}\right]$$

Stratified-wavy flow correlation
Zivi void fraction[109]:

$$\varepsilon = \frac{V_g}{V} = \left[1 + \frac{1-x}{x}\left(\frac{\rho_v}{\rho_l}\right)^{2/3}\right]^{-1}$$

Stratified angle is determined as

$$\theta_{strat} = 1 - \text{acos}\,(2\varepsilon - 1)$$

The liquid Galileo number Ga_l for the tube is

$$Ga_l = \frac{g\rho_l(\rho_l - \rho_v)D^3}{\mu_l^2}$$

The Vapor only Reynolds number

$$Re_v = \frac{mD}{\mu_v}$$

The liquid Jacob number

$$Ja_l = \frac{C_{pl}(T_{sat} - T_w)}{h_{lg}}$$

The liquid Froude number

$$Fr_l = \frac{m}{\rho_l^2 gD}$$

$$Nu_{stratified-wavy} = \frac{h_D D}{k_l} = \frac{0.23Re_v^{0.12}}{1 + 1.11X_{tt}^{0.58}}\left[\frac{Ga_l Pr_l}{Ja_l}\right]^{0.25} + \left(1 - \frac{\theta_{strat}}{\pi}\right)Nu_{strat}$$

$$Nu_{strat} = 0.0195Re_l^{0.8}Pr_l^{0.4}\left[1.376 + \frac{C_1}{X_{tt}^{C_2}}\right]$$

For $0 < Fr_l \leq 0.7$ $\quad C_1 = 4.172 + 5.48Fr_l - 1.564Fr_l^2 \quad C_2 = 1.773 - 0.169Fr_l$
For $Fr_l > 0.7$ $\quad C_1 = 7.242 \quad C_2 = 1.655$

Transition criteria from annular flow to stratified-wavy flow is given by Soliman[110]
For $Re_l \leq 1250$

$$Fr_{so} = 0.025Re_l^{1.59}\left(\frac{1 + 1.09X_{tt}^{0.039}}{X_{tt}}\right)^{1.5}\frac{1}{Ga_l^{0.5}}$$

For $Re_l > 1250$

$$Fr_{so} = 1.26Re_l^{1.04}\left(\frac{1 + 1.09X_{tt}^{0.039}}{X_{tt}}\right)^{1.5}\frac{1}{Ga_l^{0.5}}$$

While Soliman set the transition from annular flow to wavy flow at Fr_{so}=7 Dobson and Chato noted that a transition value of 20 fit their heat transfer data better. The method can be implemented as:

- For mass velocities greater than 500 kg/m^{2}s, the annular flow correlation, $Nu_{annular}$, is always utilised
- For mass velocities less than 500 kg/m^{2}s, and for $Fr_{so} < 20$ the strafied-wavy correlation, Nu_{strat}, is used
- For mass velocities less than 500 kg/m^{2}s, and for $7 \leq Fr_{so} \leq 20$ linear interpolation between annular and the strafied-wavy correlation is used

$$Nu = Nu_{annular} + \frac{Frso - 7}{13}(Nu_{stratified-wavy} - Nu_{annular})$$

Rohsenow equation inside tubes[56]

$Re_l = \frac{m(1-x)D}{A_{cross-section}\mu_l} = \frac{G(1-x)D}{\mu_l}$ where x is vapor mass quality. Where $G = \frac{m}{A_{cross-section}}$ is the mass flux (kg/sm^2) and m is mass flow rate (kg/s)

$$Re_v = \frac{mxD}{A_{cross-section}\mu_l} = \frac{GxD}{\mu_l}$$

Martinelli parameter:

$$X_{tt} = \left(\frac{1-x}{x}\right)^{0.9}\left(\frac{\rho_v}{\rho_l}\right)^{0.5}\left(\frac{\mu_l}{\mu_v}\right)^{0.1}$$

Thermal resistance of the annular film:

$F = 5Pr_l + 5\ln(1 + 5Pr_l) + 2.5\ln(0.00313Re_l^{0.812})$ $1125 < Re_l$

$F = 5Pr_l + 5\ln(1 + Pr_l(0.09636Re_l^{0.585} - 1))$ $50 < Re_l < 1125$

$F = 0.707Pr_l Re_l^{0.5}$ $Re_l < 50$

$$F(X_{tt}) = 0.15\left(\frac{1}{X_{tt}} + \frac{2.85}{X_{tt}^{0.476}}\right)$$

$0.1 \le F(X_{tt}) \le 1$ $Nu_D = \frac{h_D D}{k_l} = \frac{Pr_l Re_l}{F} F(X_{tt})$

$1 < F(X_{tt}) \le 15$ $Nu_D = \frac{h_D D}{k_l} = \frac{Pr_l Re_l}{F} \lfloor F(X_{tt}) \rfloor^{1.15}$

Shah correlation[102] inside tube condensation

With the Dittus-Boelter correlation as a starting point, Shah [102] proposed an alternative equation acting on the liquid Reynolds number as

$Re_{ls} = \frac{mD(1-x)}{A_{cross-section}\mu_l} = \frac{GD(1-x)}{\mu_l}$ where x is vapor mass quality. Where $G = \frac{m}{A_{cross-section}}$ is the mass flux

(kg/sm^2) and m is mass flow rate (kg/s)

$Re_{lt} = \frac{mD}{A_{cross-section}\mu_l} = \frac{GD}{\mu_l}$

$Re_{gt} = \frac{mD}{A_{cross-section}\mu_g} = \frac{GD}{\mu_g}$

$h_{lt} = 0.023\left(\frac{k_l}{D}\right)(Re_{lt}^{0.8} Pr_l^{0.4}$

$h_{tp} = h_{lt}\left[(1-x)^{0.8} + \frac{3.8x^{0.76}(1-x)^{0.04}}{P_r^{0.38}}\right]$ where $P_r = \frac{P_{saturation}}{P_{critical}}$

This corrolation is valid for $Re_{lt} > 350,\ Re_g > 35000\ , V_g > 3\ m/s$

Shah correlation[106] inside tube condensation

In this correlation, for vertical tubes three heat transfer regimes have been identified and defined as follows:

$Z = \left(\frac{1}{x} - 1\right)^{0.8} P_r^{0.4}$ Where Z is Sha's correlating parameter

$J_g = \frac{xG}{[gD\rho_g(\rho_l - \rho_g)]^{0.5}}$ Where J_g is dimensionless vapor velocity

Regime I occurs when

$$J_g \ge \frac{1}{2.4Z + 0.73}$$

Regime III occurs when

$J_g \le 0.89 - 0.93\exp(-0.087Z^{-1.17})$

For horizontal tubes only two regimes have been identified as: Regime I occurs when

$J_g \ge 0.98\exp(Z + 0.263)^{-0.62}$

Heat Transfer equations:

$h_{lt} = 0.023\left(\frac{k_l}{D}\right)(Re_{lt}^{0.8} Pr_l^{0.4}$

$$h_I = h_{lt}\left(\frac{\mu_l}{14\mu_g}\right)^{(0.0058+0.557P_r)}\left[(1-x)^{0.8} + \frac{3.8x^{0.76}(1-x)^{0.04}}{P_r^{0.38}}\right]$$

$$h_{Nu} = 1.32(Re_{ls})^{-1/3}\left[\frac{\rho_l(\rho_l - \rho_g)}{\mu_f^2}\right]^{1/3}$$

In regime I

$h_{tp} = h_I$

In regime II

$h_{tp} = h_I + h_{Nu}$

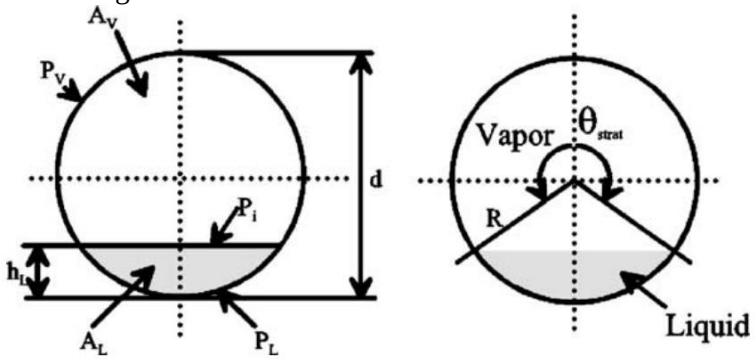
In regime III

$h_{tp} = h_{Nu}$

Shah correlation[107] inside tube condensation

Regime I occurs when
$J_g \geq 0.98(Z + 0.263)^{-0.62}$
Regime III occurs when
$J_g \leq 0.95(1.254 + 2.27Z^{1.249})^{-1}$
In regime I
$h_{tp} = h_I$
In regime II
$h_{tp} = h_I + h_{Nu}$
In regime III
$h_{tp} = h_{Nu}$

El Hajal, Thome & Cavallini correlation[103]
This corrolations required flow pattern map. Therefore we will define flow pattern map for the horizontal pipe internal flow. Different flow regions are given in the Pictures above. Furthermore calculation for vapor void fraction, ε, was also given above. Geometrical parameters of stratified flow is given in the figure below.



In this figure P_L is the strafied perimeter around the bottom of the tube. P_v is the non-strafied perimeter around the top of the tube. H_L is the height of the strafied liquid, P_i is the length of the interface, and A_L & A_v are corresponding cross-sectional areas occupied by liquid and vapor. Normalised form of these dimensions are given as:
$h_{LD} = \frac{h_L}{D}$ $P_{iD} = \frac{P_i}{D}$ $A_{LD} = \frac{A_L}{D^2}$ $A_{vD} = \frac{A_v}{D^2}$ $A_L = A(1-\varepsilon)$ $A_v = A\varepsilon$
The stratified angle θ_{strat} should also be known to solve pattern map. It can be solve either iteratively by using equation
$A_{LD} = \frac{1}{8}[(2\pi - \theta_{strat}) - \sin(2\pi - \theta_{strat})]$ or as an approximation equation of Biberg[104]
$$\theta_{strat} = 2\pi - 2\left\{\pi(1-\varepsilon) + \left(\frac{3\pi}{2}\right)^{1/3}\left[1 - 2(1-\varepsilon) + (1-\varepsilon)^{\frac{1}{3}} - \varepsilon^{\frac{1}{3}}\right] - \frac{1}{200}(1-\varepsilon)\varepsilon[1-2(1-\varepsilon)][1+4((1-\varepsilon)^2+\varepsilon^2]\right\}$$
Authors claim that error in this equation is less than 0.00005 radian in the range of $0 \leq \theta_{strat} \leq 2\pi$. G_{wavy} for condensation Zürcher at al.[105] equation can be used:
$$G_{wavy} = \left\{\frac{16A_{vD}^3 g D \rho_l \rho_v}{x^2\pi^2[1-(2h_{Ld}-1)^2]^{0.5}}\left[\frac{\pi^2}{25h_{LD}^2}\left(\frac{gD^2\rho_l}{\sigma}\right)^{-1.023} + 1\right]\right\}^{0.5} + 50 - 75e^{(-x^2-0.97)/(x(1-x)))}$$
Transition curve from strafied-wavy flow to fully stratified flow is determined using the other expression of Zürcher at. al.[105]
$$G_{strat} = \left\{\frac{51211.69A_{LD}A_{vD}^2\rho_v(\rho_l-\rho_v)\mu_l g}{x^2(1-x)\pi^3}\right\}^{1/3} + 20x$$
Transition between intermittent flow an annular flow is given as:
$$x_{IA} = \left\{\left[0.2914\left(\frac{\rho_v}{\rho_l}\right)^{-1/1.75}\left(\frac{\mu_v}{\mu_l}\right)^{-1/7}\right] + 1\right\}^{-1}$$
Transition curve from annular and intemittent flow to mist flow is given as:

$$G_{mist} = \left\{ \frac{7680 A_{vD}^2 g D \rho_l \left[1.138 + 2\log\left(\frac{\pi}{1.5 A_{LD}}\right)\right]^2}{x^2 \pi^2 \left(\frac{g D^2 \rho_l}{\sigma}\right)} \right\}^{0.5}$$

The last transition zone is bubly flow which occurs at very high mass velocities

$$G_{bubly} = \left\{ \frac{256 A_{vD} A_{LD}^2 D^{1.25} \rho_l (\rho_l - \rho_v) g}{0.3164 (1-x)^{1.75} \pi^2 P_{iD} \mu_l^{0.25}} \right\}^{1/1.75}$$

The local flow pattern is determined by the following procedure:

-+1. Input: values of d, G and x ($0 < x < 1$) and saturation physical properties ;

2.Find ε_h ε_{ra} and ε

3. Find $h_{LD} = \frac{h_L}{D}$ $P_{iD} = \frac{P_i}{D}$ $A_{LD} = \frac{A_L}{D^2}$ $A_{vD} = \frac{A_v}{D^2}$ $A_L = A(1-\varepsilon)$ $A_v = A\varepsilon$

4. Solve $A_{LD} = \frac{1}{8}[(2\pi - \theta_{strat}) - \sin(2\pi - \theta_{strat})]$ iteratively

or use

$$\theta_{strat} = 2\pi - 2\left\{\pi(1-\varepsilon) + \left(\frac{3\pi}{2}\right)^{1/3}\left[1 - 2(1-\varepsilon) + (1-\varepsilon)^{\frac{1}{3}} - \varepsilon^{\frac{1}{3}}\right] - \frac{1}{200}(1-\varepsilon)\varepsilon[1-2(1-\varepsilon)][1+4((1-\varepsilon)^2+\varepsilon^2]\right\}$$

to find θ_{strat}

5. Evaluate Eqs. $h_{LD} = 0.5\left(1 - cos\left(\frac{2\pi - \theta_{strat}}{2}\right)\right)$ and $P_{iD} = sin\left(\frac{2\pi - \theta_{strat}}{2}\right)$

6. Evaluate Eq.

$$G_{wavy} = \left\{ \frac{16 A_{vD}^3 g D \rho_l \rho_v}{x^2 \pi^2 [1 - (2h_{Ld} - 1)^2]^{0.5}} \left[\frac{\pi^2}{25 h_{LD}^2} \left(\frac{g D^2 \rho_l}{\sigma} \right)^{-1.023} + 1 \right] \right\}^{0.5} + 50 - 75 e^{\left(-x^2 - 0.97\right)/(x(1-x))}$$

over a range of x to find G_{wavy} and x_{min}

7. Evaluate Eqs. $x_{IA} = \left\{ \left[0.2914 \left(\frac{\rho_v}{\rho_l}\right)^{-1/1.75} \left(\frac{\mu_v}{\mu_l}\right)^{-1/7} \right] + 1 \right\}^{-1}$ to find x_{IA}

8. Evaluate Eqs.

$$G_{strat} = \left\{ \frac{51211.69 A_{LD} A_{vD}^2 \rho_v (\rho_l - \rho_v) \mu_l g}{x^2 (1-x) \pi^3} \right\}^{1/3} + 20x$$ to find Gstrat

9. Evaluate Eqs.

$$G_{strat} = \left\{ \frac{51211.69 A_{LD} A_{vD}^2 \rho_v (\rho_l - \rho_v) \mu_l g}{x^2 (1-x) \pi^3} \right\}^{1/3} + 20x$$

10. Evaluate Eqs.

$$G_{mist} = \left\{ \frac{7680 A_{vD}^2 g D \rho_l \left[1.138 + 2\log\left(\frac{\pi}{1.5 A_{LD}}\right)\right]^2}{x^2 \pi^2 \left(\frac{g D^2 \rho_l}{\sigma}\right)} \right\}^{0.5}$$

11. Evaluate Eqs.

$$G_{bubly} = \left\{ \frac{2564 A_{lD}^2 A_{vd} D^{1.25} \rho_l (\rho_l - \rho_v)}{0.3164 (1-x)^{1.75} \pi^2 P_{id} \mu_l^{0.25}} \right\}^{1/1.75}$$

To identify the flow pattern at any value of x the following logic applies

If $G > G_{wavy}$ and $G < G_{mist}$ and $x > x_{IA}$ Flow is annular

If $G > G_{wavy}$ and $G < G_{mist}$ or $G < G_{bubly}$ and $x < x_{IA}$ Flow is intermittent

If $G_{strat} < G < G_{wavy}$ flow is stratified-wavy

If $G < G_{strat}$ flow is fully stratified

If $G > G_{mist}$ mist flow exists

For convective heat transfer coefficient:

1. Determine the local flow pattern according to recipe given above
2. If the flow is annular or intermittent or mist, then $\theta = 0$ and

$h_{tp} = h_c = 0.003 Re_l^{0.74} Pr_l^{0.5} \frac{k_l}{\delta} f_i$ where

$Re_l = \frac{4G(1-x)\delta}{(1-\varepsilon)\mu_l}$ $Pr_l = \frac{C_{pl}\mu_l}{k_l}$

Solve δ by using the following equation

$$A_L = (1-\varepsilon)A = \frac{(2\pi-\theta)}{8}[D^2-(D-2\delta)^2]$$

$$u_L = \frac{G(1-x)}{\rho_l(1-\varepsilon)} \qquad u_v = \frac{Gx}{\rho_v\varepsilon}$$

$$f_i = 1+\left(\frac{u_v}{u_L}\right)^{1/2}\left(\frac{(\rho_l-\rho_v)g\delta^2}{\sigma}\right)^{1/4}$$

3. If the flow is strafied-wavy, then calculate θ_{strat} and θ

$A_L = (1-\varepsilon)A = \frac{D^2}{8}[(2\pi-\theta_{strat}) - \sin(2\pi-\theta_{strat})]$ or as an approximation equation of Biberg[104]

$$\theta_{strat} = 2\pi - 2\left\{\pi(1-\varepsilon)+\left(\frac{3\pi}{2}\right)^{1/3}\left[1-2(1-\varepsilon)+(1-\varepsilon)^{\frac{1}{3}}-\varepsilon^{\frac{1}{3}}\right]-\frac{1}{200}(1-\varepsilon)\varepsilon[1-2(1-\varepsilon)][1+4((1-\varepsilon)^2+\varepsilon^2]\right\}$$

$$\theta = \theta_{strat}\left[\frac{(G_{wavy}-G)}{(G_{wavy}-G_{strat})}\right]^{0.5}$$

$$h_{tp} = \frac{h_f r\theta + (2\pi-\theta)rh_c}{2\pi r}$$

$$h_c = 0.003Re_l^{0.74}Pr_l^{0.5}\frac{k_l}{\delta}f_i$$

$$h_f = 0.728\left[\frac{\rho_l(\rho_l-\rho_v)gh_{LV}k_l^3}{\mu_l D(T_{sat}-T_w)}\right]^{1/4}$$

4. If the flow is fully stratified, then $\theta = \theta_{strat}$

$$f_i = 1+\left(\frac{u_v}{u_L}\right)^{1/2}\left(\frac{(\rho_l-\rho_v)g\delta^2}{\sigma}\right)^{1/4}\left(\frac{G}{G_{strat}}\right)$$

PROBLEM: R134a with a mass flow rate of 0.01388 kg/s at T_g=50 C condenses in the surface of a $2.5x10^{-2}$ m inside diameter horizontal tube maintained at a uniform temperature T_w=30 C. Find the heat transfer coefficient when x=0.3 by using **Dobson and Chato** equation

```
 public class HT_condensation_inside_ref_chato
{ refrigerant re;
public HT_condensation_inside_ref_chato(String namei)
{re=new refrigerant(namei);}
public double h(double m,double x, double tsat,double tw, double D)
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // L vertical length of vertical plate
  // Dobson & Chato
    double tf=(tsat+tw);
    double muv=re.r.viscosity_v(tsat);
    double mul=re.r.viscosity_l(tf);
    double rov=re.r.rov(tsat);
    double rol=re.r.rol(tf);
    double A=Math.PI*D*D/4.0;
    double G=m/A;
    double Rev=m*D/(A*muv);
    double Rel=m*(1.0-x)*D/(A*mul);
    double Prl=re.r.Pr_l(tf);
    double hfg=(re.r.h_v(tsat)-re.r.h_l(tsat))*1e3; //J/kg
    double dT=tsat-tw;
    double kl=re.r.k_l(tf);
    double Cpl=re.r.Cpl(tf)*1e3;  //J/kgK
    double hfgp=hfg+0.375*Cpl*dT;
    double Nuanuular=0.0;
    double Nustratified_wavy=0.0;
    double hh=0;
    double g=9.81; //m/s^2
    double Xtt=Math.pow(((1.0-x)/x),0.9)*Math.sqrt(rov/rol)*Math.pow((mul/muv),0.1);
    Nuanuular=0.023*Math.pow(Rel,0.8)*Math.pow(Prl,0.4)*(1.0+2.22/Math.pow(Xtt,0.89));
    if(G>500) hh=Nuanuular*(kl/D);
```

```
    else
    {double eps=1.0/(1+(1.0-x)/x*Math.pow((rov/rol),2.0/3.0));
     double teta_strat=1.0-Math.acos(2.0*eps-1.0);
     double Gal=g*rol*(rol-rov)*D*D*D/(mul*mul);
     double Jal=Cpl*(tsat-tw)/hfg;
     double Frl=m*m/(rol*rol*g*D);
     double Frso=0;
     double C1=0,C2=0;
     if(Frl<=0.7)
     {C1=4.172+5.48*Frl-1.564*Frl*Frl;
      C2=1.773-0.169*Frl;
     }
     else
     {C1=7.242;C2=1.655;}
     double Nustrat=0.0195*Math.pow(Rel,0.8)*Math.pow(Prl,0.4)*(1.376+C1/Math.pow(Xtt,C2));
     Nustratified_wavy=0.23*Math.pow(Rev,0.12)/(1.0+1.11*Math.pow(Xtt,0.58)*Math.pow((Gal*Prl/Jal),0.25))+(1.0-
teta_strat/Math.PI)*Nustrat;
     if(Rel<=1250)
     {Frso=0.025*Math.pow(Rel,1.59)*Math.pow((1.0+1.09*Math.pow(Xtt,0.039)/Xtt),1.5)/Math.sqrt(Gal);}
     else
     {Frso=1.26*Math.pow(Rel,1.04)*Math.pow((1.0+1.09*Math.pow(Xtt,0.039)/Xtt),1.5)/Math.sqrt(Gal);}
     if(G<=500 && Frso<20) {hh=Nustratified_wavy*(kl/D);}
     else if(G<=500 && (Frso<20 && Frso>=7)) {hh=(kl/D)*(Nuanuular+(Frso-7.0)/13.0*(Nustratified_wavy-Nuanuular));}
     }
    return hh;
}
public static void main(String arg[])
{double tsat=50.0; //degree C
 double tw=30.0;   //degree C
 double D=0.025; //m inlet diameter
 double L=5.0; //m length of horizontal pipe
 double m=0.01388;
 double x=0.3;
 HT_condensation_inside_ref_chato hc=new  HT_condensation_inside_ref_chato("R134a");
 System.out.println("m="+m+" kg/s");
 System.out.println("x="+x+" kg vapor/kg total");
 System.out.println("tsat="+tsat+" degree C");
 System.out.println("tw="+tw+" degree C");
 double h=hc.h(m,x,tsat,tw,D);
 System.out.println("h="+h+" W/m^2K");
}}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_condensation_inside_ref_chato
m=0.01388 kg/s
x=0.3 kg vapor/kg total
tsat=50.0 degree C
tw=30.0 degree C
h=686.9680334641049 W/m^2K

> Terminated with exit code 0.
```

PROBLEM: Saturated R134a with a mass flow rate of 0.01388 kg/s at T_g=50 C condenses in the o surface of a 2.5x10^{-2} m inside diameter horizontal tube maintained at a uniform temperature T_w=30 C. Find the heat transfer coefficient when x=0.3 by using **Rohsenow** equation

```
 public class HT_condensation_inside_ref_rohsenow
{ refrigerant re;
public HT_condensation_inside_ref_rohsenow(String namei)
{re=new refrigerant(namei);}
public double h(double m,double x, double tsat,double tw, double D)
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // m mass flow rate
  // x vapor quality
  // Reference: Traviss, D.P., Rohseno
  // Rohsenow equation
w, W.M. and Baron A.B.,
  // Forced convection condensation inside tubes:a heat trasnfer equation for
  // condenser design, ASHRAE Transactions, 79, part1, 157-165 (1973)
    double tf=(tsat+tw);
    double muv=re.r.viscosity_v(tf);
    double mul=re.r.viscosity_l(tf);
    double rov=re.r.rov(tf);
```

```
    double rol=re.r.rol(tf);
    double A=Math.PI*D*D/4.0;
    double Rev=m*x*D/(A*muv);
    double Rel=m*(1.0-x)*D/(A*mul);
    double Prl=re.r.Pr_l(tf);
    double hfg=(re.r.h_v(tsat)-re.r.h_l(tsat))*1e3; //J/kg
    double dT=tsat-tw;
    double kl=re.r.k_l(tf);
    double Cpl=re.r.Cpl(tf)*1e3;  //J/kgK
    double hfgp=hfg+0.375*Cpl*dT;
    double hh=0.0;
    double g=9.81; //m/s^2
    double F=0;
    double hL=0;
    double Xtt=Math.pow(((1.0-x)/x),0.9)*Math.sqrt(rov/rol)*Math.pow((mul/muv),0.1);
    double FXtt=0.15*(1.0/Xtt+2.85/Math.pow(Xtt,0.476));
    if(Rel>=1125.0) F=5.0*Prl+5.0*Math.log(1.0+5.0*Prl)+2.5*Math.log(0.00313*Math.pow(Rel,0.812));
    else if(Rel>50 && Rel<1125) F=5.0*Prl+5.0*Math.log(1.0+Prl*(0.09636*Math.pow(Rel,0.585)-1.0));
    else F=0.707*Prl*Math.sqrt(Rel);
    if(FXtt>=0.1 && FXtt<=1.0)
      {hL=(kl/D)*Rel*Prl/F*FXtt;}
    else if(FXtt>1 && FXtt<=15.0)
      {hL=(kl/D)*Rel*Prl/F*Math.pow(FXtt,1.15);}
    return hL;
}

public static void main(String arg[])
{double tsat=50.0; //degree C
 double tw=30.0;   //degree C
 double D=0.025; //m inlet diameter
 double L=5.0; //m length of horizontal pipe
 double m=0.01388;
 double x=0.3;
 HT_condensation_inside_ref_rohsenow hc=new  HT_condensation_inside_ref_rohsenow("R134a");
 System.out.println("m="+m+" kg/s");
 System.out.println("x="+x+" kg vapor/kg total");
 System.out.println("tsat="+tsat+" degree C");
 System.out.println("tw="+tw+" degree C");
 double h=hc.h(m,x,tsat,tw,D);
 System.out.println("h="+h+" W/m^2K");
}}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_condensation_inside_ref_rohsenow
m=0.01388 kg/s
x=0.3 kg vapor/kg total
tsat=50.0 degree C
tw=30.0 degree C
h=690.7483637879325 W/m^2K

> Terminated with exit code 0.
```

PROBLEM: Saturated R134a with a mass flow rate of 0.01388 kg/s at T_g=50 C condenses in the o surface of a 2.5x10^{-2} m inside diameter horizontal tube maintained at a uniform temperature T_w=30 C. Find the heat transfer coefficient when x=0.3 by using **Shah (2009) correlation**

```
 public class HT_condensation_inside_ref_Shah_2009
{ refrigerant re;
public HT_condensation_inside_ref_Shah_2009(String namei)
{re=new refrigerant(namei);}
public double h(double m,double x, double tsat,double tw, double D)
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // m mass flow rate
  // x vapor quality
  // Shah, M.M. (1979). A General Correlation for Heat Transfer during Film
  // Condensation inside of Pipes, Int. J. Heat Mass Transfer, Vol. 22, pp. 547-556.
    double tf=(tsat+tw);
    double Ps=re.Psb(tsat);
    double Pc=re.r.Pc;
    Double Pr=Ps/Pc;
    double muf=re.r.viscosity_l(tf);
    double mug=re.r.viscosity_l(tsat);
    double rol=re.r.rol(tf);
```

```
    double rog=re.r.rov(tsat);
    double g=9.806; //m/s^2
    double A=Math.PI*D*D/4.0;
    double G=m/A;
    double Relt=m*D/(A*muf);
    double Rels=m*D*(1.0-x)/(A*muf);
    double Prl=re.r.Pr_l(tf);
    double kl=re.r.k_l(tf);
    double Z=Math.pow((1.0/x-1),0.8)*Math.pow(Pr,0.4);
    double hlt=0.023*(kl/D)*Math.pow(Relt,0.8)*Math.pow(Prl,0.4);
    double hI=hlt*Math.pow((muf/(14.0*mug)),(0.0058+0.557*Pr))*(1.0+3.8/Math.pow(Z,1.8));
    double hNu=1.32*Math.pow(Rels,(-1.0/3.0))*Math.pow((rol*(rol-rog)*g*kl*kl*kl/muf/muf),(1.0/3.0));

    double Jg=x*G/Math.sqrt(g*D*rog*(rol-rog));
    double hTP=0;
    if(Jg>=1.0/(2.4*Z+0.73)) hTP=hI;
    else if(Jg<= (0.89-0.93*Math.exp(-0.087*Math.pow(Z,-1.17)))) hTP=hNu;
    else hTP=hI+hNu;
    return hTP;
}

public static void main(String arg[])
{double tsat=50.0; //degree C
 double tw=30.0;   //degree C
 double D=0.025; //m inlet diameter
 double L=5.0; //m length of horizontal pipe
 double m=0.01388;
 double x=0.3;
 HT_condensation_inside_ref_Shah_2009 hc=new  HT_condensation_inside_ref_Shah_2009("R134a");
 System.out.println("m="+m+" kg/s");
 System.out.println("x="+x+" kg vapor/kg total");
 System.out.println("tsat="+tsat+" degree C");
 System.out.println("tw="+tw+" degree C");
 double h=hc.h(m,x,tsat,tw,D);
 System.out.println("h="+h+" W/m^2K");
}}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_condensation_inside_ref_Shah_2009
m=0.01388 kg/s
x=0.3 kg vapor/kg total
tsat=50.0 degree C
tw=30.0 degree C
h=642.7362323178916 W/m^2K

> Terminated with exit code 0.
```

PROBLEM: Saturated R134a with a mass flow rate of 0.01388 kg/s at T_g=50 C condenses in the o surface of a 2.5x10^{-2} m inside diameter horizontal tube maintained at a uniform temperature T_w=30 C. Find the heat transfer coefficient when x=0.3 by using **El Hajal, Thome & Cavallini correlation**

```
import static java.lang.Math.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
  // single function single independent variable
  // example f=x*x
  // includes full set of derivatives
  // Reference :"Generation of Finite Difference Formulas on Arbitrary Spaced Grids",
  // Bength Fornberg, Mathematics of Computation, Volume 51, Number 184, October 1988
  // pages 699-706
  //interface version
@FunctionalInterface
interface if_x
{  public double func(double x);
//first order derivative
default double dfunc(double x)
{double h=1.0e-3;
 int n=1;
 int M=10;
 return dfunc(x,n,M,h);
 }
```

```
//second order derivative
default double dfunc2(double x)
{double h=1.0e-3;
 int n=2;
 int M=10;
 return dfunc(x,n,M,h);
}
default double dfunc(double x,int N,int Mi,double hi)
{// order of the maximum derivative
 // N order of derivative
 // M degree of difference formula
double c[][][];
double alpha[];
double h;
int M;
double a[]=new double[0];
h=0.01;
double x0=0;
M=20;
double alphai[]={0,1,-1,2,-2,3,-3,4,-4,5,-5,6,-6,7,-7,8,-8,9,-9,10,-10,11,-11,12,-12,13,-13,14,-14,15,
-15,16,-16,17,-17,18,-18,19,-19,20,-20,21,-21,22,-22,23,-23,24,-24,25,-25,26,-26,27,-27,28,-28,29,-29,30,-30,
31,-31,32,-32,33,-33,34,-34,35,-35,36,-36,37,-37,38,-38,39,-39,40,-40,41,-41,42,-42,43,-43,44,-44,45,-45,46,
-46,47,-47,48,-48,49,-49,50,-50,51,-51,52,-52,53,-53,54,-54,55,-55,56,-56,57,-57,58,-58,59,-59,60,-60,
-61,61,-62,62,-63,63,-64,64,-65,65,-66,66,-67,67,-68,68,-70,70,-71,71,-72,72,-73,73,-74,74,-75,75,
-76,76,-77,77,-78,78,-79,79,-80,80,-81,81,-82,82,-83,83,-84,84,-85,85,-86,86,-87,87,
-88,88,-89,89,-90,90,-91,91,-92,92,-93,93,-94,94,-95,95,-96,96,-97,97,-98,98,-99,99,-100,100};
alpha=alphai;
int N1=alpha.length-1;
// M   degree of highest derivative
// N+1 number of coefficients
double delta[][][]=new double[N1+1][N1+1][M+1];
double c1,c2,c3;
delta[0][0][0]=1.0;
c1=1.0;
for(int n=1;n<=N1;n++)
{  c2=1;
   for(int nu=0;nu<=(n-1);nu++)
   {c3=alpha[n]-alpha[nu];
    c2=c2*c3;
    if(n<=M) delta[n-1][nu][n]=0.0;
    for(int m=0;m<=Math.min(n,M);m++)
    {
       if(m==0)
       { delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m])/c3;}
       else
       {delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m]-m*delta[n-1][nu][m-1])/c3;}
    }//next m
   }//next nu
   for(int m=0;m<=Math.min(n,M);m++)
   {   if(m==0)
                {delta[n][n][m]=c1/c2*(-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
               else
                {delta[n][n][m]=c1/c2*(m*delta[n-1][n-1][m-1]-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
   }//next m
   c1=c2;
}//next n
c=delta;
  if(Mi<N) M=N;
  else M=Mi;
  h=hi;
  double deriv=0;
  double h1=1/h;
  double h2=1;
  for(int j=0;j<N;j++)
  {h2*=h1;}
  for(int i=0;i<c[0].length;i++)
  {   deriv+=c[M][i][N]*func(x+alpha[i]*h);}
  return deriv*h2;}
default double dfunc(double x,int N)
{int M=30;double h=0.05*N;return dfunc(x,N,M,h);}
}
```

```
 public class HT_condensation_inside_ref_ElHajal_Thome_cavallini
{ refrigerant re;
String flow_pattern;
```

```
public HT_condensation_inside_ref_ElHajal_Thome_cavallini(String namei)
{re=new refrigerant(namei);}
public double h(double m,double x, double tsat,double tw, double D)
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // m mass flow rate of refrigerant kg(s
  // x vapor quality of refrigerant kg vapor/kg total
  // L vertical length of vertical plate
  // El Hajal, J., Thome, J.R. and Cavallini, A.(2003).Condensation in Horizontal Tubes, Part 1: Two-
  // Phase Flow Pattern Map, Int. J. Heat Mass Transfer, Vol. 46, pp. 3349-3363.
  // Condensation in Horizontal Tubes, Part 2: Two Phase Flow Pattern Map, Int. J. Heat Mass Transfer,
  // Vol. 46, pp. 3365-3387.

     double tf=(tsat+tw);
    double muv=re.r.viscosity_v(tsat);
    double mul=re.r.viscosity_l(tf);
    double rov=re.r.rov(tsat);
    double rol=re.r.rol(tf);
    double A=Math.PI*D*D/4.0;
    double G=m/A;
    double Rev=m*D/(A*muv);
    double Rel=m*(1.0-x)*D/(A*mul);
    double Prl=re.r.Pr_l(tf);
    double hfg=(re.r.h_v(tsat)-re.r.h_l(tsat))*1e3; //J/kg
    double dT=tsat-tw;
    double g=9.806; //m/s^2
    double kl=re.r.k_l(tf);
    double Cpl=re.r.Cpl(tf)*1e3;  //J/kgK
    double hfgp=hfg+0.375*Cpl*dT;
    double sigma=re.sigma(tsat);
    //Homegeneous void fraction
    double epsh=1.0/(1+(1.0-x)/x*(rov/rol));
    //Non-homegeneous void fraction
    double epsra=x/rov/((1.0+0.12*(1.0-x))*(x/rov+(1.0-x)/rol)+
    1.18*(1.0-x)*Math.pow((g*sigma*(rol-rov)),0.25)/G/Math.sqrt(rol));
    double eps=(epsh-epsra)/Math.log(epsh/epsra);
    double Al=A*(1.0-eps);
    double Av=A*eps;
    double Ald=Al/(D*D);
    double Avd=Av/(D*D);
    double ul=G*(1.0-x)/(rol*(1.0-eps));
    double uv=G*x/(rov*eps);
    double teta=0;
    double teta_strat=2.0*Math.PI-2.0*(Math.PI*(1.0-eps)+Math.pow(1.5*Math.PI,(1.0/3.0))*(1.0-2.0*(1.0-eps)+Math.pow((1.0-
eps),(1.0/3.0))-Math.pow(eps,1.0/3.0))-
    1.0/200.0*(1.0-eps)*eps*(1.0-2.0*(1.0-eps))*(1.0+4.0*((1.0-eps)*(1.0-eps)+eps*eps)));
    if_x f2=tet->Ald-1.0/8.0*((2.0*Math.PI-tet)-Math.sin(2.0*Math.PI-tet));
    System.out.println("teta_strat="+teta_strat);
    teta_strat=bisection(f2,0.0,1.5*Math.PI);
    System.out.println("teta_strat="+teta_strat);
    double hld=0.5*(1.0-Math.cos((2.0*Math.PI-teta)/2.0));
    double Pid=Math.sin((2.0*Math.PI-teta)/2.0);
    double Gwavy=(16.0*Avd*Avd*Avd*g*D*rol*rov/(x*x*Math.PI*Math.PI*Math.sqrt(1.0-(2.0*hld-1.0)*(2.0*hld-1.0))*
    (Math.PI*Math.PI/(25.0*hld*hld)*Math.pow((g*D*D*rol/sigma),-1.023)+1.0)+50.0-75.0*Math.exp((-x*x-0.97)/(x*(1.0-x)))));
    double Gstrat=Math.pow((51211.69*Ald*rov*(rol-rov)*mul*g)/(x*x*(1.0-x)*Math.PI*Math.PI*Math.PI),(1.0/3.0))+20.0*x;
    double xIA=1.0/((0.2914*Math.pow((rov/rol),(-1.0/1.75))*Math.pow((muv/mul),(-1.0/7.0)))+1.0);
    double Gmist=Math.sqrt(7680.0*Avd*Avd*g*D*rol*Math.pow((1.138+2.0*Math.log(Math.PI/(1.5*Ald))),2.0)/
    (x*x*Math.PI*Math.PI*(g*D*D*rol/sigma)));
    if(G>Gwavy & G<Gmist && x>xIA)
    {flow_pattern="annular";}
    else if(G>Gwavy & G<Gmist && x<xIA)
    {flow_pattern="intermittent";}
    else if(G<Gwavy & G>Gstrat)
    {flow_pattern="Stratified_wavy";}
    else if(G<Gstrat)
    {flow_pattern="Fully_Stratified";}
    else
    {flow_pattern="Mist";}
    System.out.println("flow pattern = "+flow_pattern);
    double htp=0;
    double hc=0,hf=0;
    double delta=0;
    if(flow_pattern.equals("annular") || flow_pattern.equals("intermittent") || flow_pattern.equals("Mist"))
    {   teta=0.0;
        final double teta1=teta;
        if_x f3=delt->Al-(2.0*Math.PI-teta1)/8.0*(D*D-(D-2.0*delt)*(D-2.0*delt));
        delta=bisection(f3,0.0,0.6*D);
```

```
        System.out.println("Al="+Al+"delta="+delta+"  "+f3.func(teta_strat));
        double fi=1.0+Math.sqrt(uv/ul)*Math.pow((rol-rov)*g*delta*delta/sigma,0.25);
        hc=0.003*Math.pow(Rel,0.74)*Math.sqrt(Prl)*kl/delta*fi;
                htp=hc;
              }
    else if(flow_pattern.equals("Stratified_wavy"))
    {teta=teta_strat*Math.sqrt((Gwavy-G)/(Gwavy-Gstrat));
     final double teta1=teta;
     if_x f3=delt->Al-(2.0*Math.PI-teta1)/8.0*(D*D-(D-2.0*delt)*(D-2.0*delt));
     delta=bisection(f3,0.0,0.6*D);
     System.out.println("Al="+Al+"delta="+delta+"  "+f3.func(teta_strat));
     double fi=1.0+Math.sqrt(uv/ul)*Math.pow((rol-rov)*g*delta*delta/sigma,0.25);
     hc=0.003*Math.pow(Rel,0.74)*Math.sqrt(Prl)*kl/delta*fi;
     hf=0.728*Math.pow((rol*(rol-rov)*g*hfg*kl*kl*kl/(mul*D*dT)),0.25);
     htp=(hf/2.0*teta+(2*.0*Math.PI-teta)/2.0*hc)/(Math.PI);
    }
    else if(flow_pattern.equals("Fully_Stratified"))
    {teta=teta_strat;
     final double teta1=teta;
     if_x f3=delt->Al-(2.0*Math.PI-teta1)/8.0*(D*D-(D-2.0*delt)*(D-2.0*delt));
     delta=bisection(f3,0.0,0.6*D);
     double fi=1.0+Math.sqrt(uv/ul)*Math.pow((rol-rov)*g*delta*delta/sigma,0.25);
     hc=0.003*Math.pow(Rel,0.74)*Math.sqrt(Prl)*kl/delta*fi;
     hf=0.728*Math.pow((rol*(rol-rov)*g*hfg*kl*kl*kl/(mul*D*dT)),0.25);
     htp=(hf/2.0*teta+(2*.0*Math.PI-teta)/2.0*hc)/(Math.PI);
    }
    return htp;
}
public static double bisection(if_x f,double a,double b)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   double eps=1.0e-8;
   int nmax=100;
   int i=0;
   while(Math.abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;;
    i++;
   }
   if(i>=nmax) r=bisection(f,a,b1);
   return r;
  }
public static void main(String arg[])
{double tsat=50.0; //degree C
 double tw=30.0;   //degree C
 double D=0.025; //m inlet diameter
 double L=5.0; //m length of horizontal pipe
 double m=0.01388;
 double x=0.3;
 HT_condensation_inside_ref_ElHajal_Thome_cavallini hc=new  HT_condensation_inside_ref_ElHajal_Thome_cavallini("R134a");
 System.out.println("m="+m+" kg/s");
 System.out.println("x="+x+" kg vapor/kg total");
 System.out.println("tsat="+tsat+" degree C");
 System.out.println("tw="+tw+" degree C");
 double h=hc.h(m,x,tsat,tw,D);
 System.out.println("h="+h+" W/m^2K");
}}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_condensation_inside_ref_ElHajal_Thome_cavallini
m=0.01388 kg/s
x=0.3 kg vapor/kg total
tsat=50.0 degree C
tw=30.0 degree C
teta_strat=3.7756579709472127
teta_strat=3.7757249003375026
flow pattern = Fully_Stratified
Al=1.496079432042635E-4delta=0.0064233398437500006  17.755107400526267
h=557.3485181044167 W/m^2K
> Terminated with exit code 0.
```

Dropwise condensation

Dropwise condensation occurs when a vapor condenses on a surface not wetted by the condensate. For nonmetal vapors, dropwise condensation gives much higher heat transfer coefficients than

those found with film condensation. For instance, the heat transfer coefficient for dropwise condensation of steam is around 10 times that for film condensation at power station condenser pressures and more than 20 times that for film condensation at atmospheric pressure. In circumstances where the filmwise coefficient is of similar magnitude to that on the cooling side, a change of mode to dropwise condensation offers a potential improvement in overall coefficient by a factor of up to around 2.

15.4 MULTIPHASE PRESSURE DROP

Pressure drop is an integral part of heat transfer and should be calculatedalmost in every heat transfer and thermal design problem. In two pahase internal flow, presssure drop caused by change of momentum, statik pressure drop and frictional pressure drop. They are due to kinetic and potential energy changes and energy losses through friction. Some of the suggested pressure drop correlations are given here. Caution should be taken while utilising these equations due to widely changing results in different equations.

Internal pipe homogeneous model

In this approach fluid assume to behave like a single phase fluid. Gas and liquid phases has the same velocity, therefore interfacial surface friction and surface tension effects are ignored. Total pressure drop:

$$\Delta p_{toplam} = \Delta p_{statik} + \Delta p_{momentum} + \Delta p_{sürtünme}$$

Static pressure drop can be given as: $\Delta P_{static} = \rho_H g H sin\theta$ where H is the height, θ is the angle form horizontal plane. ρ_H is called homogen density and calculated as $\rho_H = \rho_l(1-\varepsilon_H) + \rho_g \varepsilon_H$ where ε_H is homogeneous phase volume ratio $\varepsilon_H = \frac{1}{1+\left(\frac{u_g(1-x)\rho_g}{u_l x \rho_l}\right)}$ u_g/u_l in the equation is called slip ratio and in homogeneous assumption it is equal to 1.Momentum pressure gradient can be given as:

$\left(\frac{\partial P}{\partial z}\right)_{momentum} = \frac{d(m/\rho_H)}{dz}$

Friction pressure drop is:

$\Delta P_{friction} = \frac{2 f_{tp} L G^2}{\rho_H d}$ where m is total mass flow rate (kg/s), $f_{tp} = \frac{0.079}{Re^{0.25}}$ $Re = \frac{Gd}{\mu_{tp}}$

$\mu_{tp} = x\mu_g + (1-x)\mu_l$

This correlation is usefull when mass flow rate and frictional pressure drop is high. Assumption of constant vapor quality is made in the equation, therefore should be used with caution.

PROBLEM: Using the homogeneous flow pressure drop method, calculate the two-phase pressure drop for upflow in a vertical tube of 10 mm internal diameter that is 2 m long. The flow is adiabatic, the mass flow rate is 0.02 kg/s and the vapor quality is 0.05. The fluid is R-123 at a saturation temperature of 3°C

```
/* ref_CS_Data property list
  0  M,
  1  BP,
  2  FP,
  3  Tc,
  4  Pc,
  5  roc,
  6  Psb(t),
  7  Psd(t),
  8  rol(t),
  9  rov(t),
  10  h_l(t),
  11  h_v(t),
  12  h_lv(t),
  13  s_l(t),
  14  s_v(t),
  15  s_lv(t),
  16  viscosity_l(t),
  17  viscosity_v(t),
  18  k_l(t),
  19  k_v(t),
  20  Cpl(t),
```

```
  21  Cpv(t),
  22  soundv_l(t),
  23  soundv_v(t),
  24  surfacetension(t),
  25  Pr_l(t),
  26  Pr_v(t)
*/
class HT_internal_pipe_two_phase_homogeneous_pressure_drop
{
public static void main(String arg[])
{  ref_CS st=new ref_CS("R123");
   double tsat=3.0; //degree C
   double m=0.02; //kg/s
   double d=10e-3; //m
   double H=2.0; //m
   double x=0.05; //
   double a[]=st.property(tsat);
   double mul=a[16];  //kg/(m.s)=Pa.s
   double muv=a[17];  //kg/(m.s)=Pa.s
   double rol=a[8];  //kg/m^3
   double rov=a[9];  //kg/m^3
   double Psat=a[6]*1e3; //Pa
   System.out.println("mul="+mul+"muv="+muv+"rol="+rol+"rov="+rov+"Psat="+Psat);
   double eH=1.0/(1.0+(1.0-x)/x*rov/rol);
   System.out.println("eH="+eH);
   double roH=rol*(1.0-eH)+rov*eH;
   System.out.println("roH="+roH);
   double mutp=x*muv+(1.0-x)*mul;
   double g=9.806; //m/s^2
   double dP_static=roH*g*H;
   System.out.println("dP_static="+dP_static+" Pa");
   double A=Math.PI*d*d/4.0;
   double G=m/A;
   System.out.println("G="+G);
   double Re=G*d/mutp;
   double ftp=0.079/Math.pow(Re,.25);
   System.out.println("Re="+Re);
   System.out.println("ftp="+ftp);
   double dP_friction=2.0*ftp*H*G*G/d/roH;
   System.out.println("dP_friction="+dP_friction+" Pa");
   double dP_total=dP_static+dP_friction;
   System.out.println("dP_total="+dP_total+" Pa");
}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_internal_pipe_two_phase_homogeneous_pressure_drop
mul=5.437531752955245E-4muv=9.950042086177276E-
6rol=1518.8583822113003rov=2.5435853055350255Psat=37375.00022326565
eH=0.9691624955216591
roH=49.30294964569111
dP_static=966.9294484512939 Pa
G=254.6479089470325
Re=4924.891159674347
ftp=0.009430352616649046
dP_friction=4961.297265466007 Pa
dP_total=5928.226713917301 Pa

> Terminated with exit code 0.
```

Internal pipe seperated phase models

Here, the phases are considered to be flowing separately in the channel, each with a given velocity and each occupying a given fraction of the channel cross section. The separated flow momentum pressure drop equation reduces, for a duct of constant cross sectional area and steady flow, to:

$$\Delta P_{momentum} = G^2 \left\{ \left[\frac{(1-x)^2}{\rho_l (1-\varepsilon)} + \frac{x^2}{\rho_g \varepsilon} \right]_{out} - \left[\frac{(1-x)^2}{\rho_l (1-\varepsilon)} + \frac{x^2}{\rho_g \varepsilon} \right]_{in} \right\}$$

The separated flow model considers the two phases to be artificially separated into two streams, each flowing in its own pipe. The areas of the two pipes are proportional to the void fraction ε. It is

recommended here to use the Steiner (1993)[82] version of the drift flux model of Rouhani and Axelsson (1970)[83] for horizontal flows:

$$\varepsilon = \frac{x}{\rho_g}\left[[1+0.12(1-x)]\left(\frac{x}{\rho_g}+\frac{1-x}{\rho_l}\right)+\frac{1.18(1-x)\left[g\sigma(\rho_l-\rho_g)\right]^{0.25}}{G^2\rho_l^{0.5}}\right]^{-1}$$

For vertical flows, the Rouhani and Axelsson (1970) [83] expression can be used for void fractions larger than 0.1:

$$\varepsilon = \frac{x}{\rho_g}\left[\left[1+0.2(1-x)\left(\frac{gd\rho_l}{G^2}\right)^{1/4}\right]\left(\frac{x}{\rho_g}+\frac{1-x}{\rho_l}\right)+\frac{1.18(1-x)\left[g\sigma(\rho_l-\rho_g)\right]^{0.25}}{G^2\rho_l^{0.5}}\right]^{-1}$$

The two-phase density is obtained from:

$\rho_H = \rho_l(1-\varepsilon_H)+\rho_g\varepsilon_H$

The momentum pressure drop depends on the inlet and outlet vapor qualities and void fractions. The correlation method of Friedel (1979)[85] utilizes a two-phase multiplier:

$\Delta P_{friction} = \Delta P_L \emptyset_{fr}^2$ where ΔpL is calculated for the liquid-phase flow as

$\Delta P_L = \frac{2f_L L G^2}{\rho_l d}$ The liquid friction factor fL and liquid Reynolds number (and vapor friction factor fG and vapor Reynolds number with the vapor viscosity) are obtained from

$f = \frac{0.079}{Re^{0.25}}$ $Re = \frac{Gd}{\mu_l}$ where $G = \frac{m}{A}$ is mass flux(kg/sm^2).

two-phase multiplier is

$\emptyset_{fr}^2 = E + \frac{3.24FH}{Fr_H^{0.045}We_L^{0.035}}$ where The dimensionless factors FrH, E, F and H are as follows:

$Fr_H = \frac{G^2}{gd\rho_H^2}$ $F = x^{0.78}(1-x)^{0.224}$ $E = (1-x)^2 + x^2\frac{\rho_l f_g}{\rho_g f_l}$

$Re_l = \frac{G(1-x)d}{\mu_l}$ $Re_g = \frac{Gxd}{\mu_g}$ $f_l = \frac{0.079}{Re_l^{0.25}}$ $f_g = \frac{0.079}{Re_g^{0.25}}$

$H = \left(\frac{\rho_l}{\rho_g}\right)^{0.91}\left(\frac{\rho_l}{\rho_g}\right)^{0.91}\left(\frac{\mu_g}{\mu_l}\right)^{0.19}\left(1-\frac{\mu_g}{\mu_l}\right)^{0.7}$

The liquid Weber number defined as:

$We_l = \frac{G^2 d}{\sigma\rho_H}$ where $\rho_H = \left(\frac{x}{\rho_g}+\frac{1-x}{\rho_l}\right)^{-1}$

The method of Lockhart and Martinelli (1949)[84] is the orginal method that predicted the two phase frictional pressure drop based on a two-phase multiplier for the liquid-phase, or the vapor-phase, respectively, as:

$\Delta P_{Lfriction} = \Delta P_L \emptyset_{Ltt}^2$ $\Delta P_L = \frac{2f_L L(1-x)^2 G^2}{\rho_l d}$ $\Delta P_{Gfriction} = \Delta P_G \emptyset_{Gtt}^2$ $\Delta P_G = \frac{2f_G L x^2 G^2}{\rho_g d}$

The liquid friction factor fL and liquid Reynolds number (and vapor friction factor fG and vapor Reynolds number with the vapor viscosity) are obtained from

$Re_l = \frac{G(1-x)d}{\mu_l}$ $Re_g = \frac{Gxd}{\mu_g}$ $f_l = \frac{0.079}{Re_l^{0.25}}$ $f_g = \frac{0.079}{Re_g^{0.25}}$

$$\emptyset_{Ltt}^2 = 1 + \frac{C}{X_{tt}} + \frac{1}{X_{tt}^2}$$

$$\emptyset_{Gtt}^2 = 1 + CX_{tt} + X_{tt}^2$$

Martinelli parameter:

$$X_{tt} = \left(\frac{1-x}{x}\right)^{0.9} \left(\frac{\rho_v}{\rho_l}\right)^{0.5} \left(\frac{\mu_l}{\mu_v}\right)^{0.1}$$

liquid	vapor	C
Turbulent	Turbulent	20
Laminar	Turbulent	12
Turbulent	Laminar	10
Laminar	Laminar	5

Müller-Steinhagen and Heck (1986)[86] proposed a two-phase frictional pressure gradient correlation that is an empirical interpolation between all liquid flow and all vapor flow:

$\Delta P_{friction} = [\Delta P_L + 2(\Delta P_G - \Delta P_L)x](1-x)^{1/3} + \Delta P_G x^3$ where

$\Delta P_L = \frac{2 f_L L G^2}{\rho_l d}$ $\Delta P_G = \frac{2 f_g L G^2}{\rho_g d}$

$Re_l = \frac{G(1-x)d}{\mu_l}$ $Re_g = \frac{Gxd}{\mu_g}$ $f_l = \frac{0.079}{Re_l^{0.25}}$ $f_g = \frac{0.079}{Re_g^{0.25}}$

PROBLEMS TO SOLVE (HOMEWORK PROBLEMS)

PROBLEM 1 : Saturated steam at T_g=100 C condenses on the outer surface of a $1x10^{-2}$ m outside diameter horizontal tube maintained at a uniform temperature T_w=25 C. We have N=10 pipes in each row in heat eaxhanger. Calculate for L=1xN=10 m of pipes (1 m of pipe 10 vertical layers) total heat transfer and condensation rate.

PROBLEM 2: saturated steam at T=110 C condenses inside of a $5x10^{-2}$ m inside diameter horizontal tube maintained at a uniform temperature T_w=25 C. Total pipe length is L=100 m. Calculate total heat transfer and condensation rate.

PROBLEM 3: saturated R134a at T=40 C condenses inside of a $1x10^{-2}$ m inside diameter horizontal tube maintained at a uniform temperature T_w=25 C. Total pipe length is L=100 m. Calculate total heat transfer and condensation rate.

PROBLEM 4: Using the homogeneous flow pressure drop method, calculate the two-phase pressure drop for upflow in a vertical tube of 10 mm internal diameter that is 2 m long. The flow is adiabatic, the mass flow rate is 0.02 kg/s and the vapor quality is 0.05. The fluid is R-134a at a saturation temperature of 3°C

Try to solve problems
a) by hand
b) by computer program (for properties of steam use steamIAPWS_IF97.java)

16. HEAT EXCHANGERS

16.1 HEAT EXCHANGER CLASSIFICATION

Heat exchangers are basic devices utilised to transfer heat from one or more fluids to another set of fluids of different temperatures. Heat exchangers are devices that you can find in every home, factory or building. Classifications of heat exchangers can be done considering fluid type, phase of fluids , compactness of the device etc. Let us look at few of these classifications more closely. According to contact type of the fluids, indirect contact type and direct contact type of heat exchangers are existed. Acording to phase of liquids single phase or multiphase heat exchangers exists. According surface compactness of the heat exchanger if area to volume ratio of a gas to fluid heat exchanger is bigger than 700 m^2/m^3 it is classified as compact heat exchanger. Fort he liquid to liquid heat exchanger compactness limit is bigger than 400 m^2/m^3. Accordingto type of construction tubular (double pipe, Shell & tube, spiral tube, pipe coils), plate type (plate heat exchangers, spiral heat exchangers), extended surface type(plate fin, tube fin) and reganarative heat exchangers existed. According to flow arrangement counter flow, parallel flow, cross flow, divided flow heat exchangers existed. According to heat transfer mechanisms we can group them as single phase convection in both side, single phase convection in one side and two pahse convection on the other side and two pahese convection in both sides (for example evaporator-condenser). In the figure below some of the heat exchangers are shown



Closed body welded evaporator



Internal pipes of the evaporator in the left side



Double pipe heat exchanger



Plate heat exchanger



Finned air-refrigerant heat exchanger with fans



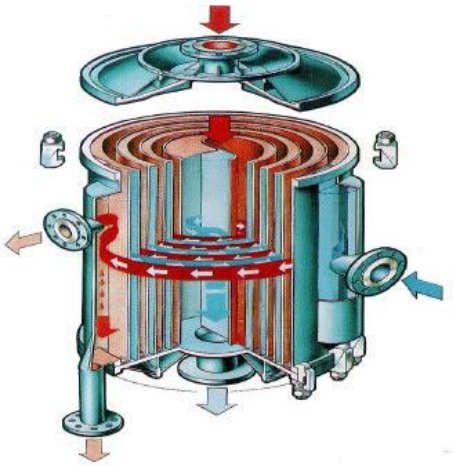
Finned heat exchanger



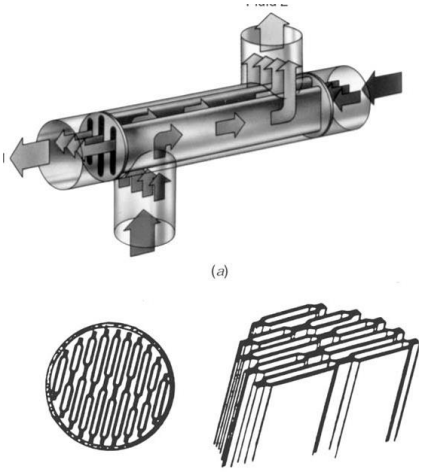
Shell and tube heat exchanger



Internal pipes of shell & tube heat exchanger



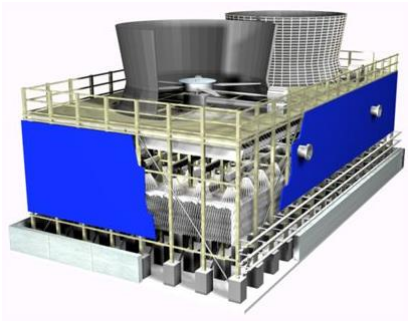
Spiral heat exchanger



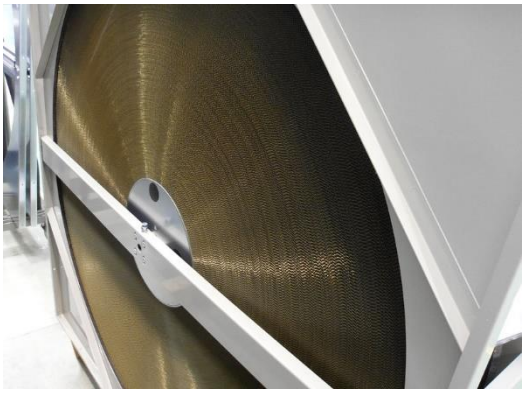
Lamella heat exchanger



Closed circuit cooling tower heat exchanger



Cooling tower (direct contact water-air heat exchanger)

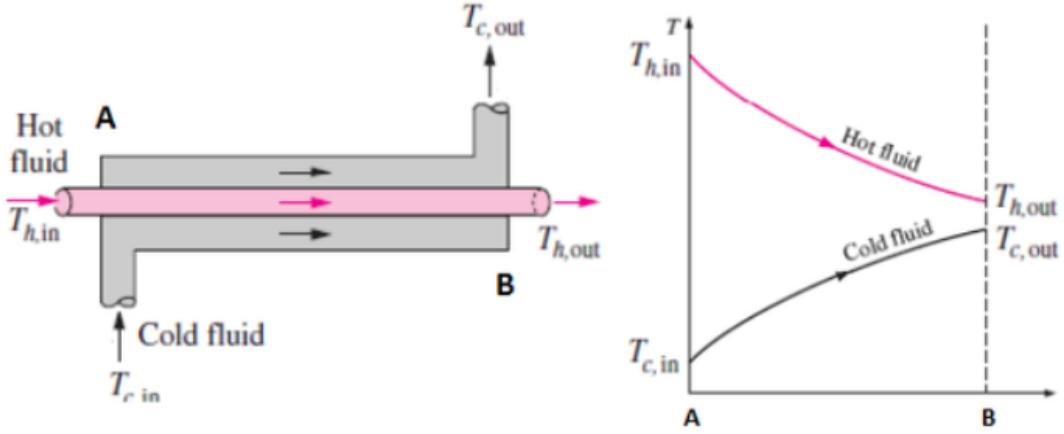


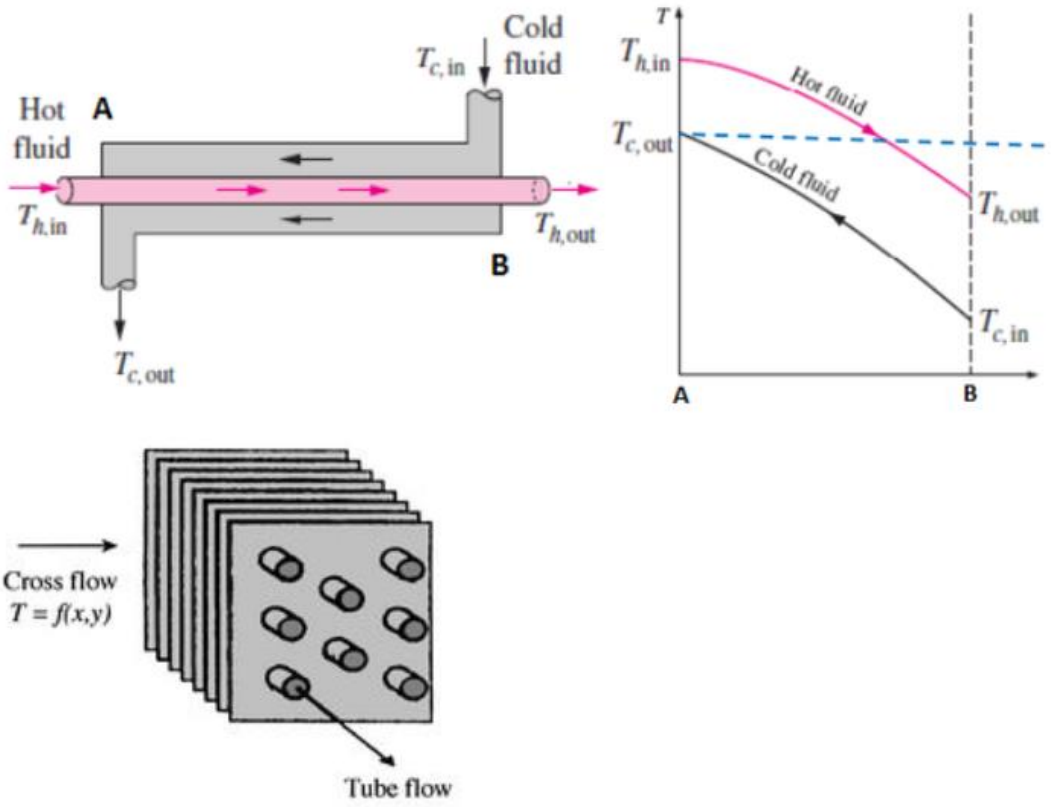
Turning wheel recuperator



Plate coil heat exchanger

Let us now consider a very simple double pipe heat exchanger and consider the flow paths of both fluids If both fluid flow in the same direction, it is called parallel flow arrangment. If they are flowing in opposite direction, they are called counter flow heat exchangers.





If flow direction is perpendicular to each other, it is called cross flow.

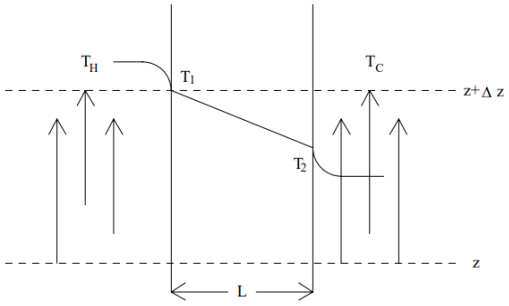
16.2 OVERALL HEAT TRANSFER COEFFICIENT

Heat exchangers are usually buıld from a sıngle material, But sometimes surface can be covered with a protective thin layer for the purpose of avoiding corrosion. After a period of time, the heat transfer surfaces of a heat exchanger may become coated with deposits from the heat transfer fluids or corrosion. In either case, the additional resistance to heat transfer due to these materials decreases the performance of the heat exchanger and must be accounted for. This is done through the use of experimentally determined fouling factors. Fouling factors causes extra resistances that should be taken into account. Furtheremore fins can be added one or both surfaces of the heat exchanger to improve surface area and overall heat transfer coefficient. Fouling factors for varius applications and operationg fluids are listed in the table below:

	Rf min	**Rf max**
Oils	**m^2K/W**	**m^2K/W**
Fuel oil #2	0.00035222	
Fuel oil #6	0.000880551	
Transformer oil	0.00017611	
Engine Lube oil	0.00017611	
Gases and vapors	0	
Acid gas	0.00035222	0.000528331
Ammonia vapor	0.00017611	
Chlorinated hydrocarbon vapors	0.00017611	0.000264165
Chlorine vapor	0.00035222	
CO2 vapor	0.00017611	
Compressed air	0.00017611	
Hydrogen	8.80551E-05	
Hydrogen(saturated with vapor)	0.00035222	
Light hydrocarbon vapors (clean)	0.00017611	
Natural gas	0.00017611	0.00035222
Natural gas flue gas	0.000880551	

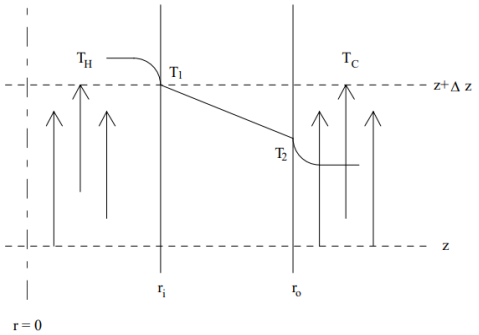
Nitrogen	8.80551E-05	
Polymerizable vapors (with inhibitor)	0.000528331	0.005283306
Refrigerant vapors (oil bearing)	0.00035222	
Solvent vapor	0.00017611	
Steble overhear products	0.00017611	
Steam (non-oil bearing)	8.80551E-05	
Steam (exhoust - oil bearing)	0.000264165	0.00035222
	Rf min	**Rf max**
Liquids	**m^2K/W**	**m^2K/W**
Ammonia liquid	0.00017611	
Ammonia liquid (oil bearing)	0.000528331	
CO2 liquid	0.00017611	
Chlorinated hydrocarbons liquid	0.00017611	0.00035222
Cholorine liquid	0.00035222	
DEG & TEG solutions	0.00035222	
Gasoline	0.00035222	
Heavy fuel oils	0.000880551	0.001232771
Heavy gas oil	0.000528331	0.000880551
Hydrolic fluid	0.00017611	
Kerosene	0.00035222	0.000528331
Light gas oil	0.00035222	0.000528331
light hydrocarbon liquid	0.00017611	
MEA & DEA solutions	0.00035222	
Naphta and light distilates	0.00035222	0.000528331
Organic heat transfer liquids	0.00035222	
Refrigerant liquids	0.00017611	
	Rf	**Rf**
	m^2K/W	**m^2K/W**
Water at 50 C or less	**velocity <1 m/s**	**velocity > 1 m/s**
Boiler blowdown water	0.00035222	0.00035222
Brackish water	0.00035222	0.00035222
condansate	8.80551E-05	8.80551E-05
cooling tower water (treated)	0.00017611	0.00017611
cooling tower water (untreated)	0.000528331	0.000528331
city or well water	0.00017611	0.00017611
river water (minimum)	0.00035222	0.00017611
river water (average)	0.000528331	0.00035222
River water (muddy or silty)	0.000528331	0.00035222
sea water	8.80551E-05	8.80551E-05
treated boiler feed water	0.00017611	8.80551E-05
Water at more than 50 C	**velocity <1 m/s**	**velocity > 1 m/s**
Boiler blowdown water	0.00035222	0.00035222
Brackish water	0.000528331	0.00035222
condansate	8.80551E-05	8.80551E-05
cooling tower water (treated)	0.00035222	0.00035222
cooling tower water (untreated)	0.000880551	0.000704441
city or well water	0.00035222	0.00035222
river water (minimum)	0.000528331	0.00035222
river water (average)	0.000704441	0.000528331
River water (muddy or silty)	0.000704441	0.000528331
sea water	0.00017611	0.00017611
treated boiler feed water	0.00017611	0.00017611

In a planar wall without any fin and fouling factor overall heat transfer coefficient :



$$Q_x = \frac{T_H - T_C}{\left(\frac{1}{h_H A}\right)+\left(\frac{L}{kA}\right)+\left(\frac{\Delta x_B}{k_B A}\right)+\left(\frac{\Delta x_C}{k_C A}\right)+\left(\frac{1}{h_C A}\right)} = UA(T_H - T_C)$$

Where U is called overall heat transfer coefficient. In heat exchangers it is common to use pipes as transfer medium, in this case cylindrical coordinate equivalent of heat transfer should apply



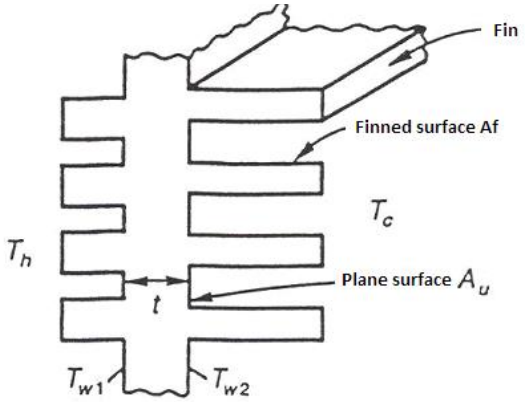
$$U_i A_i = U_o A_o = \frac{1}{\left(\frac{1}{h_H A_i}\right)+\left(\frac{\ln(\frac{r_o}{r_i})}{2\pi kL}\right)+\left(\frac{1}{h_C A_o}\right)}$$

If fouling factor is taken into account

$\frac{1}{U_i A_i} = \frac{1}{U_o A_o} = \left(\frac{1}{h_H A_i}\right) + \left(\frac{\ln(\frac{r_o}{r_i})}{2\pi kL}\right) + \left(\frac{1}{h_C A_o}\right) + \frac{R_{fi}}{A_i} + \frac{R_{fo}}{A_o}$ where R_{fi} is the internal fouling factor and R_{fo} is external fouling factor.

$Q = U_i A_i (T_H - T_C) = U_o A_o (T_H - T_C)$

Now lets investigate when the finned surfaces are existed. Let us consider finned surface diven below



If the fin efficiency is h_f, Plane surface is A_u and finned surface is A_f

$Q = (\eta_f A_f h_f + A_u h_u)\Delta T$ where $\Delta T = T_h - T_{w1}$ or $\Delta T = T_{w2} - T_c$ depends on which side of the finned surface we considered. Total area: $A = A_f + A_u$ and if we assume $h = h_f = h_u$

$Q = hA\left[1 - \frac{A_f}{A}(1 - \eta_f)\right]\Delta T$ or overall fin efficiency: $\eta = \left[1 - \frac{A_f}{A}(1 - \eta_f)\right]$

Now overall heat transfer coefficents should include overall fin efficiences of the surfaces, so

$$\frac{1}{U_i A_i} = \frac{1}{U_o A_o} = \left(\frac{1}{h_H \eta_i A_i}\right) + \left(\frac{\ln\left(\frac{r_o}{r_i}\right)}{2\pi k L}\right) + \left(\frac{1}{h_C \eta_o A_o}\right) + \frac{R_{fi}}{\eta_i A_i} + \frac{R_{fo}}{\eta_o A_o}$$

PROBLEM: Engine oil at a mean temperature T_i=80 C and mean velocity V_m=0.2 m/s flows inside a thin-walled, horizontal copper tube with an ID of D_i=1.9x10^{-2} m and OD of D_0=2.29x10^{-2} m. Tube is made of stainless steel. At the outer surface, atmospheric air at T_∞ =15 C and velocity of V_∞=5 m/s flows across the tube. Calculate the overall heat transfer coefficient and rate of heat loss per meter length of tube.

```
public class HT_pipe_air_oil
{
  public static double mu(double T)
  {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double k(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pr(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;

return y;
}
  public static double ro(double T,double P)
  {   double M=2.8965000E+01;
      double R=8314.5/M;
              return P*1e3/R/T;
  }
   public static double Nu2(double U,double Tinf,double Ts,double P,double D)
  {//Nusselt number Churchill & Bernstein equation
```

```
   double Tf=(Tinf+Ts)/2.0;
   double Re=Re(U,Tf,P,D);
   double Pr=Pr(Tf);
   double Nu=0.3+0.62*Math.pow(Re,0.5)*Math.pow(Pr,(1.0/3.0))/Math.pow((1+Math.pow((0.4/Pr),(2.0/3.0))),0.25);
   Nu*=Math.pow((1+Math.pow((Re/282000),(5.0/8.0))),(4.0/5.0));
   return Nu;
   }

  public static double Re(double U,double T,double P,double D)
  {   double mu=mu(T);
      double ro=ro(T,P);
      double Re=ro*U*D/mu;
                return Re;
  }
  public static void main(String arg[])
  { double Tinf=15.85+273.15;  //air temp. degree K;
    double tinf=15.0;
    double Ti=80+273.15;
    double ti=80.0;
    double T=(Tinf+Ti)/2.0;
    double U=5.0; //m/s
    double Ui=0.2; //m/s
    double L=1; // m
    double P=101.325; //kPa
    double Di=1.9e-2; //m copper tube
    double Do=2.29e-2; //m copper tube
    liquid_CS_EN l=new liquid_CS_EN("engineoil");
    double nu_oil=l.vis(ti);
    double ro_oil=l.ro(ti);
    double k_oil=l.k(ti);
    double k_steel=46.0; // W/mK
    double Pr_oil=l.Pr(ti);
    double Cp_oil=l.Cp(ti);
    System.out.println("nu="+nu_oil+"k="+k_oil+"Pr="+Pr_oil);
    double Re=Ui*Di/nu_oil;
    System.out.println("Re oil="+Re);
    double Nui=3.66;
    double hi=k_oil/Di*Nui;
    System.out.println("hi="+hi);
    double Nu2=Nu2(U,T,Ti,P,Do);
    double k2=k(T);
    double h2=Nu2*k2/Do;
    System.out.println(" h2="+h2);
    System.out.println(" Nu2="+Nu2);
    double Ai=Math.PI*Di*L;
    double Ao=Math.PI*Do*L;
    double Um=1.0/(1.0/(Ai*hi+Math.log(Do/Di)/(2.0*Math.PI*k_steel)+1.0/(Ao*h2)));
    System.out.println("Um="+Um);
    double Q=Um*(Ti-Tinf);
    System.out.println("Q="+Q);

    double mi=ro_oil*Ai*Ui;
    double To=Ti-Q/(mi*Cp_oil*1e3);
    double to=To-273.15;
    System.out.println("to="+to);

}}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_pipe_air_oil
nu=3.7500000000000003E-5k=0.138Pr=0.5793478260869567
Re oil=101.33333333333333
hi=26.583157894736843
 h2=48.95423817331114
 Nu2=40.13529381119457
Um=1.8713401141654942
Q=120.04646832371641
to=79.99446421864053

> Terminated with exit code 0.
```

PROBLEM: Engine oil at a mean temperature T_i=80 C and mean velocity V_m=5 m/s flows inside a thin-walled, horizontal copper tube with an ID of $D_i=1.9\text{x}10^{-2}$ m and OD of $D_0=2.29\text{x}10^{-2}$ m. Tube has round fins of $D=6\text{x}10^{-2}$ m and a thickness of $t=10^{-3}$ m. Distance between fins are $5\text{x}10^{-3}$ m. Tube and

fins are made of stainless steel. At the outer surface, atmospheric air at T_∞ =15 C and velocity of V_∞=5 m/s flows across the tube. Calculate the overall heat transfer coefficient and rate of heat loss per meter length of tube.

```
public class HT_pipe_air_oil1
{
  public static double mu(double T)
  {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double k(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pr(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;

return y;
}
  public static double ro(double T,double P)
  {   double M=2.8965000E+01;
      double R=8314.5/M;
                return P*1e3/R/T;
  }
   public static double Nu2(double U,double Tinf,double Ts,double P,double D)
  {///Nusselt number Churchill & Bernstein equation
   double Tf=(Tinf+Ts)/2.0;
   double Re=Re(U,Tf,P,D);
   double Pr=Pr(Tf);
   double Nu=0.3+0.62*Math.pow(Re,0.5)*Math.pow(Pr,(1.0/3.0))/Math.pow((1+Math.pow((0.4/Pr),(2.0/3.0))),0.25);
   Nu*=Math.pow((1+Math.pow((Re/282000),(5.0/8.0))),(4.0/5.0));
   return Nu;
   }
 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
```

```
double a=2/Math.log(10);
double b=eod/3.7;
double d=Math.log(10.0)/5.02*Re;
double s=b*d+Math.log(d);
double q=Math.pow(s,(s/(s+1)));
double g=b*d+Math.log(d/q);
double z=Math.log(q/g);
double dLA=(g/(g+1))*z;
double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
double f1=a*(Math.log(d/q)+dCFA);
f1=1.0/(f1*f1);
return f1;
}
public static double Nu(double Re,double eod,double Pr)
{ //constant surface temperature internal flow
  double Nu=0;
  double x=Re/1000;
  double f=0;
  if(Re<2300) Nu=3.66;
  else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
  else {f=f_Goudar(Re,eod);
  Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
  }
return Nu;
  }

 public static double Re(double U,double T,double P,double D)
 {   double mu=mu(T);
     double ro=ro(T,P);
     double Re=ro*U*D/mu;
                return Re;
 }
 public static void main(String arg[])
 { double Tinf=15.85+273.15;  //air temp. degree K;
   double tinf=15.0;
   double Ti=80+273.15;
   double ti=80.0;
   double T=(Tinf+Ti)/2.0;
   double U=5.0; //m/s
   double Ui=5.0; //m/s
   double L=1; // m
   double P=101.325; //kPa
   double Di=1.9e-2; //m copper tube
   double Do=2.29e-2; //m copper tube
   double D=6e-2; //m fin diameter
   double rb=Do/2.0; //base radius of fin
   double rt=D/2; //m fin radius
   double d=1e-3; //m fin thickness
   double dx=5e-3; //m distance between fins
   double Rb=rb/rt;
   double Rt=1.0;
   System.out.println("Rb="+Rb);
   liquid_CS_EN l=new liquid_CS_EN("engineoil");
   double nu_oil=l.vis(ti);
   double ro_oil=l.ro(ti);
   double k_oil=l.k(ti);
   double k_steel=46.0; // W/mK
   double Pr_oil=l.Pr(ti);
   double Cp_oil=l.Cp(ti);
   System.out.println("oil nu="+nu_oil+"k="+k_oil+"Pr="+Pr_oil);
   double Re=Ui*Di/nu_oil;
   double eps=1.0e-6;
   double eod=eps/Do;
   System.out.println("Re oil="+Re);
   double Nui=Nu(Re,eod,Pr_oil);
   System.out.println("Nui="+Nui);
   double hi=k_oil/Di*Nui;
   System.out.println("hi="+hi);
   double Nu2=Nu2(U,T,Ti,P,Do);
   double k2=k(T);
   double h2=Nu2*k2/Do;
   double M1=Math.sqrt(2.0*h2/(k_steel*d))*rt;
   System.out.println("M1="+M1);
   double M0=Math.sqrt(2.0*h2/(k_steel*d))*rb;
             System.out.println("M0="+M0);
```

```
    if_x eta=M->2.0/M*Rb/(1-
Rb*Rb)*(Mathd.I(1.0,M)*Mathd.K(1.0,M*Rb)+Mathd.K(1.0,M)*Mathd.I(1.0,M*Rb))/(Mathd.I(1.0,M)*Mathd.K(0.0,M*Rb)+Mathd.
K(1.0,M)*Mathd.I(0.0,M*Rb));
    double eta_fin=eta.func(M1);
    System.out.println("eta_fin="+eta_fin);
    System.out.println(" h2="+h2);
    System.out.println(" Nu2="+Nu2);
    double Ai=Math.PI*Di*L;
    double Ao=Math.PI*Do*L;
    double A_fin= 2.0*Math.PI/4.0*(D*D-Do*Do);
    double N_fin=L/(d+dx);
    double A_no_fin=Math.PI*Do*dx;
    System.out.println("A_fin="+A_fin+"A_no_fin="+A_no_fin);
    double A_total=A_fin+A_no_fin;
    double eta_total=(1.0-A_fin/A_total*(1.0-eta_fin));
    System.out.println("eta="+eta_total);
    double Rm=1.0/(Ai*hi)+Math.log(Do/Di)/(2.0*Math.PI*k_steel*L)+1.0/(A_total*N_fin*eta_total*h2) ;
    double Um=1.0/Rm;
    System.out.println("Um="+Um);
    double Q=Um*(Ti-Tinf);
    System.out.println("Q="+Q);

    double mi=ro_oil*Ai*Ui;
    double To=Ti-Q/(mi*Cp_oil*1e3);
    double to=To-273.15;
    System.out.println("to="+to);

}}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_pipe_air_oil1
Rb=0.3816666666666667
oil nu=3.7500000000000003E-5k=0.138Pr=0.5793478260869567
Re oil=2533.333333333333
Nui=4.916796349629749
hi=35.711468223626596
M1=1.3840522476312245
M0=0.5282466078459174
eta_fin=0.82832744771129
 h2=48.95423817331114
 Nu2=40.13529381119457
A_fin=0.004831125474727116A_no_fin=3.5971235883603135E-4
eta=0.8402239354674457
Um=2.0085457665028703
Q=128.84821092115908
to=79.99976233352493

> Terminated with exit code 0.
```

PROBLEM: water at a mean temperature T_i=80 C and mean velocity V_m=3 m/s flows inside a thin-walled, horizontal copper tube with an ID of $D_i=1.9\text{x}10^{-2}$ m and OD of $D_0=2.29\text{x}10^{-2}$ m. Tube has round fins of $D=6\text{x}10^{-2}$ m and a thickness of $t=10^{-3}$ m. Distance between fins are $5\text{x}10^{-3}$ m. Tube and fins are made of stainless steel. At the outer surface, atmospheric air at T_∞ =15 C and velocity of V_∞=5 m/s flows across the tube. Calculate the overall heat transfer coefficient and rate of heat loss per meter length of tube.

```
public class HT_pipe_water
{
  public static double mu(double T)
  {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
```

```
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double k(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pr(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;

return y;
}
  public static double ro(double T,double P)
  {   double M=2.8965000E+01;
      double R=8314.5/M;
                return P*1e3/R/T;
  }
   public static double Nu2(double U,double Tinf,double Ts,double P,double D)
  {//Nusselt number Churchill & Bernstein equation
   double Tf=(Tinf+Ts)/2.0;
   double Re=Re(U,Tf,P,D);
   double Pr=Pr(Tf);
   double Nu=0.3+0.62*Math.pow(Re,0.5)*Math.pow(Pr,(1.0/3.0))/Math.pow((1+Math.pow((0.4/Pr),(2.0/3.0))),0.25);
   Nu*=Math.pow((1+Math.pow((Re/282000),(5.0/8.0))),(4.0/5.0));
   return Nu;
   }
 public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 public static double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
   double Nu=0;
   double x=Re/1000;
   double f=0;
   if(Re<2300) Nu=3.66;
   else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
```

```
   else {f=f_Goudar(Re,eod);
   Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
   }
 return Nu;
   }

  public static double Re(double U,double T,double P,double D)
  {   double mu=mu(T);
     double ro=ro(T,P);
     double Re=ro*U*D/mu;
                return Re;
  }
  public static void main(String arg[])
  { double Tinf=15.85+273.15;  //air temp. degree K;
    double tinf=15.0;
    double Ti=80+273.15;
    double ti=80.0;
    double T=(Tinf+Ti)/2.0;
    double U=5.0; //m/s
    double Ui=3.0; //m/s
    double L=1; // m
    double P=101.325; //kPa
    double Di=1.9e-2; //m copper tube
    double Do=2.29e-2; //m copper tube
    double D=6e-2; //m fin diameter
    double rb=Do/2.0; //base radius of fin
    double rt=D/2; //m fin radius
    double d=1e-3; //m fin thickness
    double dx=5e-3; //m distance between fins
    double Rb=rb/rt;
    double Rt=1.0;
    double k_steel=46.0;
    System.out.println("Rb="+Rb);
    water w=new water();
                    double a[]=w.property_TP(T,P);
                    double ro=a[2];
                    double Cp=a[5];
                    double k=a[6];
                    double mu=a[7];
                    double Pr=a[9];
    double Re=ro*Ui*Di/mu;
    double eps=1.0e-6;
    double eod=eps/Do;
    System.out.println("Re water="+Re);
    double Nui=Nu(Re,eod,Pr);
    System.out.println("Nui="+Nui);
    double hi=k/Di*Nui;
    System.out.println("hi="+hi);
    double Nu2=Nu2(U,T,Ti,P,Do);
    double k2=k(T);
    double h2=Nu2*k2/Do;
    double M1=Math.sqrt(2.0*h2/(k_steel*d))*rt;
    System.out.println("M1="+M1);
    double M0=Math.sqrt(2.0*h2/(k_steel*d))*rb;
              System.out.println("M0="+M0);
    if_x eta=M->2.0/M*Rb/(1-
Rb*Rb)*(Mathd.I(1.0,M)*Mathd.K(1.0,M*Rb)+Mathd.K(1.0,M)*Mathd.I(1.0,M*Rb))/(Mathd.I(1.0,M)*Mathd.K(0.0,M*Rb)+Mathd.
K(1.0,M)*Mathd.I(0.0,M*Rb));
    double eta_fin=eta.func(M1);
    System.out.println("eta_fin="+eta_fin);
    System.out.println(" h2="+h2);
    System.out.println(" Nu2="+Nu2);
    double Ai=Math.PI*Di*L;
    double Ao=Math.PI*Do*L;
    double A_fin= 2.0*Math.PI/4.0*(D*D-Do*Do);
    double N_fin=L/(d+dx);
    double A_no_fin=Math.PI*Do*dx;
    System.out.println("A_fin="+A_fin+"A_no_fin="+A_no_fin);
    double A_total=A_fin+A_no_fin;
    double eta_total=(1.0-A_fin/A_total*(1.0-eta_fin));
    System.out.println("eta="+eta_total);
    double Rm=1.0/(Ai*hi)+Math.log(Do/Di)/(2.0*Math.PI*k_steel*L)+1.0/(A_total*N_fin*eta_total*h2) ;
    double Um=1.0/Rm;
    System.out.println("Um="+Um);
    double Q=Um*(Ti-Tinf);
    System.out.println("Q="+Q);
```

```
    double mi=ro*Ai*Ui;
    double To=Ti-Q/(mi*Cp*1e3);
    double to=To-273.15;
    System.out.println("to="+to);

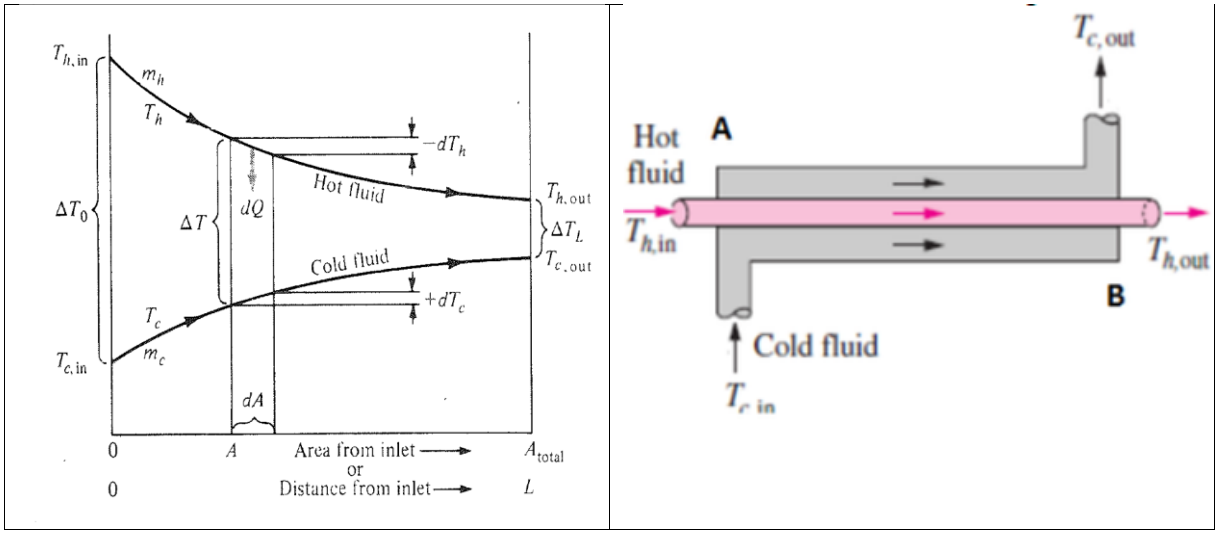
}}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_pipe_water
Rb=0.3816666666666667
Re water=99574.98625178046
Nui=434.8059684791874
hi=14604.286438448667
M1=1.3840522476312245
M0=0.5282466078459174
eta_fin=0.82832744771129
 h2=48.95423817331114
 Nu2=40.13529381119457
A_fin=0.004831125474727116A_no_fin=3.597123588360313E-4
eta=0.8402239354674457
Um=33.45094107268375
Q=2145.8778698126616
to=79.99710060269558

> Terminated with exit code 0.
```

16.3 LOGARITHMIC MEAN TEMPERATURE DIFFERENCE

Let us consider simple double pipe parallel flow heat exchanger. And further assume that $C_p(T) = C_p$=constant. And now let us consider a small differential part inside of the heat exchanger with area dA and heat transfer dQ, mass flow rates of cold and hot fluids are m_C , m_h and local heat transfer coefficient is U. Because of the differential nature, temperature in both sides can be assumed constant and $\Delta T = T_h - T_c$



The rate of heat transfer from hot to cold fluid can be written as

$$dQ = UdA\Delta T$$

in the same time dQ should be equal to

$dQ = -m_h C_{ph} dT_h$ (hot fluid) and $dQ = m_c C_{pc} dT_c$ (cold fluid)

Where C_{pc} and C_{ph} are specific heat and values are considered not to be function of temperature (constant). dT_h and dT_c are changes in thetemperatures in hot and cold fluids. Consider also relation $\Delta T = T_h - T_c$ and taken derivative of this relation gives $d(\Delta T) = dT_h - dT_c$. Combining equations gives

$$d(\Delta T) = -\frac{dQ}{m_h C_{ph}} - \frac{dQ}{m_c C_{pc}} = -dQ\left[\frac{1}{m_h C_{ph}} + \frac{1}{m_c C_{pc}}\right]$$

or by defining

$$B = \left[\frac{1}{m_h C_{ph}} + \frac{1}{m_c C_{pc}}\right]$$

$$d(\Delta T) = -B dQ$$

remembering that

$$dQ = U dA \Delta T$$

and substituting iy into the previous equation gives

$$d(\Delta T) = -BU dA \Delta T$$

by arranging

$$\frac{d(\Delta T)}{\Delta T} = -BU dA$$

integrating the equation

$$\int_{\Delta T_0}^{\Delta T_L} \frac{d(\Delta T)}{\Delta T} = -B \int_0^{A_t} U dA$$

Average overall heat transfer coefficient

$$U_m = \frac{1}{A_t}\int_0^{A_t} U dA \text{ then } \int_{\Delta T_0}^{\Delta T_L} \frac{d(\Delta T)}{\Delta T} = -B U_m A_t$$

$$ln\left(\frac{\Delta T_0}{\Delta T_L}\right) = B U_m A_t$$

We can also integrate $d(\Delta T) = -B dQ$ relation as

$$\int_{\Delta T_0}^{\Delta T_L} d(\Delta T) = -B \int_0^{Q} dQ \text{ so}$$

$$\Delta T_0 - \Delta T_L = BQ$$

so by eliminating B we get

$$Q = U_m A_t \frac{\Delta T_0 - \Delta T_L}{ln\left(\frac{\Delta T_0}{\Delta T_L}\right)} = U_m A_t \Delta T_{ln}$$

This mean temperature ΔT_{ln} defined by the equation is called logarithmic mean temperature difference. Caution shold be observed using this equation due to the **assumption that C_p is constant**. Heat exchangers such as combustion boilers temperature difference could be very large and this assumption can cause a big error. But when the temperature difference is relatively small, we can use this equation for simple and fast evaluation of heat exchanger performance. It should be note that when $\Delta T_0 = \Delta T_L$ equation will not be valid, so equation wil be converted to

$$Q = U_m A_t \Delta T_0 = U_m A_t \Delta T_L$$

PROBLEM : A double tube heat exchanger is used to heat water at a rate of m=0.8 kg/s from T_{ci}=30 ℃ to T_{co}=80 ℃, with hot oil entering at T_{hi}=120 ℃ and leaving at T_{ho}=85 ℃. The overall heat transfer coefficient U=125 W/(m^2K). Calculate the heat transfer area required. Pressure is 2 bar in both sides.
$Q = m[h(T_0) - h(T_i)]$ using steamTableIF97.java

water and steam IAPWS Industrial formulation 1997

Properties of water and steam
International Association for the Properties of Water and Steam (IAPWS) Industrial formulation 1997
Dr. M. Turhan Çoban, Ege University, School of Engineering, Mechanical Eng. Dept., Izmir, Turkey
email : turhan_coban@yahoo.com web : www.turhancoban.com
Reference: International Steam Tables, Wolfgang Wagner, Hans-Joachim Kretzschamar ISBN 978-3-540-21419-9, Springer 2008
273.15 K <= T <= 1073.15 K && 0< P <= 100 MPa
1073.15 K <= T <= 2273.15 K && 0< p <= 50 MPa

select known property pair :	tp	tp
Temperature	30	degree C
Pressure	200	kPa

ref no	Property	Value	Units
0	P, pressure	200.0	kPA
1	T, temperature	303.15	deg K
2	v, specific volume	0.0010043226783249158	m³/kg
3	h, enthalpy	125.92371781004759	KJ/kg
4	u, internal energy	125.72285327438262	KJ/kg
5	s, entropy	0.4367333918378573	KJ/kgK
6	x, quality	-1.0	kg vap/kg mix
7	Cp, specific heat at constant ...	4.179752552491479	KJ/kgK
8	Cv, specific heat at constant ...	4.117164390601918	KJ/kgK
9	α isobaric thermal expansion ...	3.029625501780504E-4	1/K
10	κ isothermal compressibility	4.464949388513845E-7	1/Pa
11	η Dynamic viscosity	7.9721968712262E-4	Pa.s
12	k Thermal conductivity	0.615055222436942	W/mK
13	σ Surface tension	0.07119415109567316	N/m
14	Prandtl number	5.417694055087645	
15	ρ density	995.6959267990189	kg/m^3
16	speed of sound	1511.1394602627677	m/s

WS Industrial formulation 1997

Properties of water and steam
International Association for the Properties of Water and Steam (IAPWS) Industrial formulation 1997
Dr. M. Turhan Çoban, Ege University, School of Engineering, Mechanical Eng. Dept., Izmir, Turkey
email : turhan_coban@yahoo.com web : www.turhancoban.com
Reference: International Steam Tables, Wolfgang Wagner, Hans-Joachim Kretzschamar ISBN 978-3-540-21419-9, Springer 2008
273.15 K <= T <= 1073.15 K && 0< P <= 100 MPa
1073.15 K <= T <= 2273.15 K && 0< p <= 50 MPa

select known property pair :	tp	tp
Temperature	80	degree C
Pressure	200	kPa

ref no	Property	Value	Units
0	P, pressure	200.0	kPA
1	T, temperature	353.15	deg K
2	v, specific volume	0.0010289685369951104	m³/kg
3	h, enthalpy	335.07012856607645	KJ/kg
4	u, internal energy	334.8643348586774	KJ/kg
5	s, entropy	1.0752906663117836	KJ/kgK
6	x, quality	-1.0	kg vap/kg mix
7	Cp, specific heat at constant ...	4.195298991380056	KJ/kgK
8	Cv, specific heat at constant ...	3.870141065784111	KJ/kgK
9	α isobaric thermal expansion ...	6.415453729987036E-4	1/K
10	κ isothermal compressibility	4.5996174819744045E-7	1/Pa
11	η Dynamic viscosity	3.5408459955547645E-4	Pa.s
12	k Thermal conductivity	0.6670418360574104	W/mK
13	σ Surface tension	0.06267285496652644	N/m
14	Prandtl number	2.2269829013400133	
15	ρ density	971.8470138263829	kg/m^3
16	speed of sound	1557.249682324861	m/s

$Q = 0.8[335.07012856607645 - 125.92371781004759\ ]$ = 167317.1 W Another approach:

$C_{pavg} = \frac{C_p(T_0)+C_p(T_i)}{2} = \frac{4.179752552491479+4.195298991380056}{2}$ =4.187526

$Q = mC_{pavg}[T_0 - T_i] = 0.8 * 4.187526 * (80 - 30) = 167501$ W

$\Delta T_{ln} = \frac{50-40}{ln\frac{55}{40}} = 47.1076 \quad A = \frac{Q}{U\Delta T_{ln}} = \frac{167317.1}{125*47.1076} = 28.4144\ m^2$

PROBLEM : Engine oil thermophysical properties are given in the followin table:

T	ρ	Cp	ν	k	α	Pr
℃	kg/m^3	kJ/(kgK)	m^2/s	W/(mK)	m^2/s	
0	889.12	1.796	0.00428	0.147	0.911	47100
20	888.23	1.88	0.0009	0.145	0.872	10400
40	876.05	1.964	0.00024	0.144	0.834	2870
60	864.04	2.047	8.39E-05	0.14	0.8	1050
80	852.02	2.131	3.75E-05	0.138	0.769	490
100	840.01	2.219	2.03E-05	0.137	0.738	276
120	828.96	2.307	1.24E-05	0.135	0.71	175
140	816.94	2.395	8.00E-06	0.133	0.686	116
160	805.89	2.483	5.60E-06	0.132	0.663	84

Engine oil entering into a copper tube with internal diameter of D_i=1.9x10^{-2} m and D_o=2x10^{-2} m Oil inlet velocity V=0.1 m/s and inlet temperature T_i=80 ℃. At the outer surface air at a pressure P=101.325 kPa and $T_\infty = 15$ ℃ and a velocity of V=5 m/s flows across the tube. Calculate the overall heat transfer and the rate of heat loss to air for L=10 m of pipe length. What is the oil exit temperature?

```
 public class HT_HE_LMTD_1
{
            //Air properties
public static double mua(double T)
  {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double ka(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pra(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
```

```
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;

return y;
}
public static double Cpa(double T)
{double a[]={1.89668613E+00,
-2.23937197E-02,
2.40885259E-04,
-1.45254290E-06,
5.43552393E-09,
-1.32310554E-11,
2.13239465E-14,
-2.25744420E-17,
1.50864395E-20,
-5.76865768E-24,
9.61540784E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
return y*1e3;
}

  public static double roa(double T,double P)
  {   double M=2.8965000E+01;
      double R=8314.5/M;
                return P*1e3/R/T;
  }
//Outside of pipe heat transfer coefficients
  public static double Nu1(double U,double Tinf,double Ts,double P,double D)
  {//Nusselt number Hilpert equation
   double Re=Re(U,Tinf,P,D);
   double Pr=Pra(Tinf);
   double Nu=0.0;
   double C=0;
   double m=0;
   if(Re>0.4 && Re<=4) {C=0.989;m=0.330;}
   else if(Re>4.0 && Re<=40) {C=0.911;m=0.385;}
   else if(Re>40.0 && Re<=4000) {C=0.683;m=0.466;}
   else if(Re>4000.0 && Re<=40000) {C=0.193;m=0.618;}
   else if(Re>40000.0 && Re<=400000) {C=0.027;m=0.805;}
   Nu=C*Math.pow(Re,m)*Math.pow(Pr,(1.0/3.0));
   return Nu;
   }
   public static double Nu2(double U,double Tinf,double Ts,double P,double D)
  {//Nusselt number Churchill & Bernstein equation
   double Tf=(Tinf+Ts)/2.0;
   double Re=Re(U,Tf,P,D);
   double Pr=Pra(Tf);
   double Nu=0.3+0.62*Math.pow(Re,0.5)*Math.pow(Pr,(1.0/3.0))/Math.pow((1+Math.pow((0.4/Pr),(2.0/3.0))),0.25);
   Nu*=Math.pow((1+Math.pow((Re/282000),(5.0/8.0))),(4.0/5.0));
   return Nu;
   }
  public static double Nu3(double U,double Tinf,double Ts,double P,double D)
  {//Nusselt number Zukauskas equation
   double Re=Re(U,Tinf,P,D);
   double Pr=Pra(Tinf);
   double Prs=Pra(Ts);
   double Nu=0.0;
   double C=0;
   double m=0;
   double n=0;
   if(Re>1 && Re<=40) {C=0.75;m=0.4;}
   else if(Re>40.0 && Re<=1000) {C=0.51;m=0.5;}
   else if(Re>40.0 && Re<=4000) {C=0.683;m=0.466;}
   else if(Re>1000.0 && Re<=2e5) {C=0.26;m=0.6;}
   else if(Re>2e5 && Re<=1e6) {C=0.076;m=0.7;}
   if(Pr<=10) n=0.37;
   else n=0.36;
   System.out.println("Re="+Re+"C="+C+"n="+n);
  Nu=C*Math.pow(Re,m)*Math.pow(Pr,n)*Math.pow((Pr/Prs),0.25);
   return Nu;
   }
   //Outside of pipe Reynolds Number
  public static double Re(double U,double T,double P,double D)
  {   double mu=mua(T);
      double ro=roa(T,P);
      double Re=ro*U*D/mu;
                return Re;
```

```
  }

 //inside of the pipe
  public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 public static double Nu(double Re,double eod,double Pr)
 { //Gnilensky equation constant surface temperature internal flow
   double Nu=0;
   double x=Re/1000;
   double f=0;
   if(Re<2300) Nu=3.66;
   else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
   else {f=f_Goudar(Re,eod);
   Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
   }
 return Nu;
   }
public static void main(String arg[])
{
if_x ro=(T)->{
              double c[]=new double[9];
c[0] = 889.1199999952275;
c[1] = 0.7788495031013842;
c[2] = -0.06351544019571408;
c[3] = 0.0013855048184076356;
c[4] = -1.4433484802821403E-5;
c[5] = 4.487425847930485E-8;
c[6] = 3.9556119555151096E-10;
c[7] = -3.6094028331533092E-12;
c[8] = 8.263961649139699E-15;
double ro1=c[0]+c[1]*T+c[2]*T*T+c[3]*T*T*T+c[4]*T*T*T*T+c[5]*T*T*T*T*T+c[6]*T*T*T*T*T*T+
c[7]*T*T*T*T*T*T*T+c[8]*T*T*T*T*T*T*T*T;
return ro1;};
if_x Cp=(T)->{
double c[]=new double[9];
c[0] = 1.7959999998943406;
c[1] = 0.004723169829219559;
c[2] = -6.862011661913738E-5;
c[3] = 3.4558988531768852E-6;
c[4] = -8.729900756439303E-8;
c[5] = 1.2014135033860537E-9;
c[6] = -9.120186836609464E-12;
c[7] = 3.588547572026437E-14;
c[8] = -5.716096780813412E-17;
double Cp1=c[0]+c[1]*T+c[2]*T*T+c[3]*T*T*T+c[4]*T*T*T*T+c[5]*T*T*T*T*T+c[6]*T*T*T*T*T*T+
c[7]*T*T*T*T*T*T*T+c[8]*T*T*T*T*T*T*T*T;
return Cp1;};

if_x nu=(T)->{
              double c[]=new double[9];
c[0] = 0.004280000003261192;
c[1] = -3.3523085818949495E-4;
c[2] = 1.2652236211123124E-5;
c[3] = -2.8511979544733536E-7;
c[4] = 4.063643181328594E-9;
c[5] = -3.682322901476503E-11;
c[6] = 2.051382329611078E-13;
c[7] = -6.392170741711082E-16;
c[8] = 8.51226550326221E-19;
double nu1=c[0]+c[1]*T+c[2]*T*T+c[3]*T*T*T+c[4]*T*T*T*T+c[5]*T*T*T*T*T+c[6]*T*T*T*T*T*T+
c[7]*T*T*T*T*T*T*T+c[8]*T*T*T*T*T*T*T*T;
return nu1;};
```

```
if_x k=(T)->{
double c[]=new double[9];
c[0] = 0.14700000005528546;
c[1] = -8.645905018173647E-4;
c[2] = 8.174515733992113E-5;
c[3] = -3.149258103502365E-6;
c[4] = 5.979366047785655E-8;
c[5] = -6.250116681333913E-10;
c[6] = 3.6730444648733335E-12;
c[7] = -1.1393550677860256E-14;
c[8] = 1.45326814566017E-17;
double k1=c[0]+c[1]*T+c[2]*T*T+c[3]*T*T*T+c[4]*T*T*T*T+c[5]*T*T*T*T*T+c[6]*T*T*T*T*T*T+
c[7]*T*T*T*T*T*T*T+c[8]*T*T*T*T*T*T*T*T;
return k1;};

if_x alpha=(T)->{
double c[]=new double[9];
c[0] = 0.911000000081612;
c[1] = -0.0023371545636132474;
c[2] = 5.144486108645957E-5;
c[3] = -2.6514532888268044E-6;
c[4] = 6.856485653279935E-8;
c[5] = -9.40991349596015E-10;
c[6] = 7.053089834391705E-12;
c[7] = -2.7282273770986304E-14;
c[8] = 4.2628551613634344E-17;
double alpha1=c[0]+c[1]*T+c[2]*T*T+c[3]*T*T*T+c[4]*T*T*T*T+c[5]*T*T*T*T*T+c[6]*T*T*T*T*T*T+
c[7]*T*T*T*T*T*T*T+c[8]*T*T*T*T*T*T*T*T;
return alpha1;};
if_x Pr=(T)->{
double c[]=new double[9];
c[0] = 47100.00003196616;
c[1] = -3578.477288756393;
c[2] = 131.46605758119622;
c[3] = -2.8944472149120264;
c[4] = 0.040465648321428946;
c[5] = -3.610324689955692E-4;
c[6] = 1.9865482107834683E-6;
c[7] = -6.129734294732644E-9;
c[8] = 8.099544328479861E-12;
double Pr1=c[0]+c[1]*T+c[2]*T*T+c[3]*T*T*T+c[4]*T*T*T*T+c[5]*T*T*T*T*T+c[6]*T*T*T*T*T*T+
c[7]*T*T*T*T*T*T*T+c[8]*T*T*T*T*T*T*T*T;
return Pr1;};

double Ti=80.0;
double To=60.0; // estimation
double Tf=(Ti+To)/2.0; //average temperature for oil
double ro1=ro.func(Ti);
double nu1=nu.func(Ti);
double k1=k.func(Ti);
double Pr1=Pr.func(Ti);
double Cp1=Cp.func(Ti);
System.out.println("Ti="+Ti+"nu1="+nu1+"k1="+k1+"Pr1="+Pr1);
double Tinf=15.0;
double P=101.325;
double T=0;
for(int i=0;i<5;i++)
{
Tf=(Ti+To)/2.0;
T=(Tf+Tinf)/2.0;
double roa=roa(T+273.15,P);
double mua=mua(T+273.15);
double nua=mua/roa;
double ka=ka(T+273.15);
double Pra=Pra(T+273.15);
System.out.println("i="+i+" air side :T="+T+"roa="+roa+"mua="+mua+"nua="+nua+"ka="+ka+"Pra="+Pra);
//Oil:
double V=0.1;  //m/s
double Di=0.019; //m
double Do=0.02; //m
double Re=V*Di/nu1;
double eod=0.0002;
double Nu=Nu(Re,eod,Pr1);
double h1=Nu*k1/Di;
double V1=5.0; //m/s
System.out.println("inside pipe(oil) Re="+Re);
```

```
System.out.println("inside pipe(oil) Nu="+Nu+" h1 = "+h1+" W/m2K");
double Nu2=Nu1(V1,(Tinf+273.15),(Ti+273.15),P,Do);
double h2=Nu2*ka/Do;
double Nu3=Nu2(V1,(Tinf+273.15),(Ti+273.15),P,Do);
double h3=Nu3*ka/Do;
double Nu4=Nu3(V1,(Tinf+273.15),(Ti+273.15),P,Do);
double h4=Nu4*ka/Do;
System.out.println("outside pipe(air) Nu2="+Nu2+" h2 = "+h2+" W/m2K");
System.out.println("outside pipe(air) Nu3="+Nu3+" h3 = "+h3+" W/m2K");
System.out.println("outside pipe(air) Nu4="+Nu4+" h4 = "+h4+" W/m2K");
double km=320.0;
double L=10.0; //m
double A1=Math.PI*Di*L;
double A2=Math.PI*Do*L;
double R1=1.0/(h1*A1);
double R3=1.0/(h2*A2);
double R2=Math.log(Do/Di)/(2.0*Math.PI*km*L);
double R=R1+R2+R3;
double U=1/R;
double dT1=Ti-Tinf;
double dT2=To-Tinf;
double lmtd=(dT1-dT2)/Math.log(dT1/dT2);
double Q=U*lmtd;
double A=Math.PI*Di*Di/4.0;
double m=ro1*A*V;
To=Ti-Q/(m*Cp1*1e3);
System.out.println("i="+i+"  U="+U+"Q="+Q+"Ti="+Ti+"To="+To);
System.out.println("============================");
} //end of iteration
}
}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_HE_LMTD_1
Ti=80.0nu1=3.750015790262087E-5k1=0.1380000279934416Pr1=490.00153819951265
i=0 air side :T=42.5roa=1.1182739647710274mua=1.9283336828076815E-5nua=1.72438395559224E-5ka=0.02753670618667034Pra=0.7046866710239903
inside pipe(oil) Re=50.66645332357946
inside pipe(oil) Nu=3.66 h1 = 26.58315843397893 W/m2K
Re=6820.111861657432C=0.26n=0.37
outside pipe(air) Nu2=40.2683449544062 h2 = 55.44287918164863 W/m2K
outside pipe(air) Nu3=39.17388618092624 h3 = 53.93598969771156 W/m2K
outside pipe(air) Nu4=45.81804188312344 h4 = 63.083897869206304 W/m2K
i=0  U=10.90151880084154Q=592.9172789022485Ti=80.0To=68.48235242114575
============================
i=1 air side :T=44.62058810528644roa=1.1108113531986836mua=1.9383197703123384E-5nua=1.744958551900616E-5ka=0.027691417545822935Pra=0.7044315254317708
inside pipe(oil) Re=50.66645332357946
inside pipe(oil) Nu=3.66 h1 = 26.58315843397893 W/m2K
Re=6820.111861657432C=0.26n=0.37
outside pipe(air) Nu2=40.2683449544062 h2 = 55.75437770058472 W/m2K
outside pipe(air) Nu3=39.17388618092624 h3 = 54.239021956428566 W/m2K
outside pipe(air) Nu4=45.81804188312344 h4 = 63.43832644587874 W/m2K
i=1  U=10.920612292973326Q=644.9069184545257Ti=80.0To=67.47243355485224
============================
i=2 air side :T=44.36810838871306roa=1.111694633012378mua=1.937132310355375E-5nua=1.74250396901377E-5ka=0.02767301514981229Pra=0.7044615767567972
inside pipe(oil) Re=50.66645332357946
inside pipe(oil) Nu=3.66 h1 = 26.58315843397893 W/m2K
Re=6820.111861657432C=0.26n=0.37
outside pipe(air) Nu2=40.2683449544062 h2 = 55.71732599905751 W/m2K
outside pipe(air) Nu3=39.17388618092624 h3 = 54.202977288089706 W/m2K
outside pipe(air) Nu4=45.81804188312344 h4 = 63.396168358320445 W/m2K
i=2  U=10.918348884138389Q=638.8639878429013Ti=80.0To=67.58981982036474
============================
i=3 air side :T=44.397454955091185roa=1.1115918942883516mua=1.9372703535319835E-5nua=1.7427892048207465E-5ka=0.027675154367635343Pra=0.7044580792553955
inside pipe(oil) Re=50.66645332357946
inside pipe(oil) Nu=3.66 h1 = 26.58315843397893 W/m2K
Re=6820.111861657432C=0.26n=0.37
outside pipe(air) Nu2=40.2683449544062 h2 = 55.72163313711908 W/m2K
outside pipe(air) Nu3=39.17388618092624 h3 = 54.20716736186552 W/m2K
outside pipe(air) Nu4=45.81804188312344 h4 = 63.401069096911144 W/m2K
i=3  U=10.918612104346138Q=639.5682628646691Ti=80.0To=67.57613900553955
============================
i=4 air side :T=44.394034751384886roa=1.1116038670237856mua=1.9372542655323007E-5nua=1.74275596100535E-5ka=0.027674905055226524Pra=0.7044584868112539
inside pipe(oil) Re=50.66645332357946
```

```
inside pipe(oil) Nu=3.66 h1 = 26.58315843397893 W/m2K
Re=6820.111861657432C=0.26n=0.37
outside pipe(air) Nu2=40.2683449544062 h2 = 55.721131167215084 W/m2K
outside pipe(air) Nu3=39.17388618092624 h3 = 54.20667903506921 W/m2K
outside pipe(air) Nu4=45.81804188312344 h4 = 63.400497946591685 W/m2K
i=4  U=10.918581429124801Q=639.4862086906131Ti=80.0To=67.5777329396224
============================

> Terminated with exit code 0.
```

PROBLEM :

water entering into a copper tube with internal diameter of $D_i=1.9x10^{-2}$ m and external diameter $D_o = 2x10^{-2}$ m water inlet velocity V=0.1 m/s , inlet pressure P_i=200 kPa and inlet temperature T_i=80 ℃. At the outer surface air at a pressure P=101.325 kPa and $T_\infty = 15$ ℃ and a velocity of V=5 m/s flows across the tube. Calculate the overall heat transfer and the rate of heat loss to air for L=10 m of pipe length. What is the water exit temperature?

```
 public class HT_HE_LMTD_2
{
            //Air properties
public static double mua(double T)
  {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double ka(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pra(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;

return y;
}
public static double Cpa(double T)
{double a[]={1.89668613E+00,
```

```
-2.23937197E-02,
2.40885259E-04,
-1.45254290E-06,
5.43552393E-09,
-1.32310554E-11,
2.13239465E-14,
-2.25744420E-17,
1.50864395E-20,
-5.76865768E-24,
9.61540784E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
return y*1e3;
}

  public static double roa(double T,double P)
  {   double M=2.8965000E+01;
      double R=8314.5/M;
                return P*1e3/R/T;
  }
//Outside of pipe heat transfer coefficients
  public static double Nu1(double U,double Tinf,double Ts,double P,double D)
  {//Nusselt number Hilpert equation
   double Re=Re(U,Tinf,P,D);
   double Pr=Pra(Tinf);
   double Nu=0.0;
   double C=0;
   double m=0;
   if(Re>0.4 && Re<=4) {C=0.989;m=0.330;}
   else if(Re>4.0 && Re<=40) {C=0.911;m=0.385;}
   else if(Re>40.0 && Re<=4000) {C=0.683;m=0.466;}
   else if(Re>4000.0 && Re<=40000) {C=0.193;m=0.618;}
   else if(Re>40000.0 && Re<=400000) {C=0.027;m=0.805;}
   Nu=C*Math.pow(Re,m)*Math.pow(Pr,(1.0/3.0));
   return Nu;
   }
   public static double Nu2(double U,double Tinf,double Ts,double P,double D)
  {//Nusselt number Churchill & Bernstein equation
   double Tf=(Tinf+Ts)/2.0;
   double Re=Re(U,Tf,P,D);
   double Pr=Pra(Tf);
   double Nu=0.3+0.62*Math.pow(Re,0.5)*Math.pow(Pr,(1.0/3.0))/Math.pow((1+Math.pow((0.4/Pr),(2.0/3.0))),0.25);
   Nu*=Math.pow((1+Math.pow((Re/282000),(5.0/8.0))),(4.0/5.0));
   return Nu;
   }
  public static double Nu3(double U,double Tinf,double Ts,double P,double D)
  {//Nusselt number Zukauskas equation
   double Re=Re(U,Tinf,P,D);
   double Pr=Pra(Tinf);
   double Prs=Pra(Ts);
   double Nu=0.0;
   double C=0;
   double m=0;
   double n=0;
   if(Re>1 && Re<=40) {C=0.75;m=0.4;}
   else if(Re>40.0 && Re<=1000) {C=0.51;m=0.5;}
   else if(Re>40.0 && Re<=4000) {C=0.683;m=0.466;}
   else if(Re>1000.0 && Re<=2e5) {C=0.26;m=0.6;}
   else if(Re>2e5 && Re<=1e6) {C=0.076;m=0.7;}
   if(Pr<=10) n=0.37;
   else n=0.36;
   System.out.println("Re="+Re+"C="+C+"n="+n);
  Nu=C*Math.pow(Re,m)*Math.pow(Pr,n)*Math.pow((Pr/Prs),0.25);
   return Nu;
   }
   //Outside of pipe Reynolds Number
  public static double Re(double U,double T,double P,double D)
  {   double mu=mua(T);
      double ro=roa(T,P);
      double Re=ro*U*D/mu;
                return Re;
  }

 //inside of the pipe
  public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
```

```
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 public static double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
  double Nu=0;
  double x=Re/1000;
  double f=0;
  if(Re<2300) Nu=3.66;
  else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
  else {f=f_Goudar(Re,eod);
  Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
  }
 return Nu;
  }
public static void main(String arg[])
{
double Pi=200.0;
double Ti=80.0;
double To=60.0; // estimation
double Tf=(Ti+To)/2.0;
double Tinf=15.0;
double P=101.325;
double V=0.1;  //m/s
double V1=5.0; //m/s
double Di=0.019; //m
double Do=0.02; //m
double A=Math.PI*Di*Di/4.0;
double L=10.0;
double km=320.0;
//properties of water is calculated from steam EOS: steam_IAPWS95
steam_IAPWS95 ap=new steam_IAPWS95();
double T=0;
for(int i=0;i<5;i++)
{
Tf=(Ti+To)/2.0;
T=(Tf+Tinf)/2.0;
double a1[]=ap.propertyC("tp",Tf,Pi);
double ro1=1.0/a1[2];
double nu1=a1[11]/ro1;
double k1=a1[12];
double Pr1=a1[14];
double Cp1=a1[7];
System.out.println("ro1="+ro1+"nu1="+nu1+"k1="+k1+"Pr1="+Pr1+"Cp1="+Cp1);
System.out.println("i="+i+" water side Ti="+Ti+"Tf="+Tf+"Nu1="+nu1+"k1="+k1+"Pr1="+Pr1);
double roa=roa(T+273.15,P);
double mua=mua(T+273.15);
double nua=mua/roa;
double ka=ka(T+273.15);
double Pra=Pra(T+273.15);
System.out.println("T="+T+"roa="+roa+"mua="+mua+"nua="+nua+"ka="+ka+"Pra="+Pra);
//water:

double Re=V*Di/nu1;
double eod=0.0002;
double Nu=Nu(Re,eod,Pr1);
double h1=Nu*k1/Di;

System.out.println("inside pipe(water) Re="+Re);
System.out.println("inside pipe(water) Nu="+Nu+" h1 = "+h1+" W/m2K");
double Nu2=Nu1(V1,(Tinf+273.15),(Ti+273.15),P,Do);
double h2=Nu2*ka/Do;
double Nu3=Nu2(V1,(Tinf+273.15),(Ti+273.15),P,Do);
double h3=Nu3*ka/Do;
double Nu4=Nu3(V1,(Tinf+273.15),(Ti+273.15),P,Do);
double h4=Nu4*ka/Do;
```

```
System.out.println("outside pipe(air) Nu2="+Nu2+" h2 = "+h2+" W/m2K");
System.out.println("outside pipe(air) Nu3="+Nu3+" h3 = "+h3+" W/m2K");
System.out.println("outside pipe(air) Nu4="+Nu4+" h4 = "+h4+" W/m2K");


double A1=Math.PI*Di*L;
double A2=Math.PI*Do*L;
double R1=1.0/(h1*A1);
double R3=1.0/(h2*A2);
double R2=Math.log(Do/Di)/(2.0*Math.PI*km*L);
double R=R1+R2+R3;
double U=1/R;
double dT1=Ti-Tinf;
double dT2=To-Tinf;
double lmtd=(dT1-dT2)/Math.log(dT1/dT2);
double Q=U*lmtd;

double m=ro1*A*V;
To=Ti-Q/(m*Cp1*1e3);
System.out.println("i="+i+"  U="+U+"Q="+Q+"Ti="+Ti+"To="+To+" LMTD="+lmtd);
System.out.println("============================");
} //end of iteration
}
}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_HE_LMTD_2
ro1=977.8081987574116nu1=4.1273313978230366E-7k1=0.6596421102911764Pr1=2.562218988173881Cp1=4.189851573066506
i=0 water side Ti=80.0Tf=70.0Nu1=4.1273313978230366E-7k1=0.6596421102911764Pr1=2.562218988173881
T=42.5roa=1.1182739647710274mua=1.9283336828076815E-5nua=1.72438395559224E-
5ka=0.02753670618667034Pra=0.7046866710239903
inside pipe(water) Re=4603.458789381817
inside pipe(water) Nu=24.172035866755703 h1 = 839.2048815358282 W/m2K
Re=6820.111861657432C=0.26n=0.37
outside pipe(air) Nu2=40.2683449544062 h2 = 55.44287918164863 W/m2K
outside pipe(air) Nu3=39.17388618092624 h3 = 53.93598969771156 W/m2K
outside pipe(air) Nu4=45.81804188312344 h4 = 63.083897869206304 W/m2K
i=0  U=32.56801284599537Q=1771.3254371868265Ti=80.0To=64.75073317902294 LMTD=54.38850216508166
============================
ro1=976.4368704197756nu1=4.0021785269029327E-7k1=0.6615219088637725Pr1=2.474919137688163Cp1=4.191291761486483
i=1 water side Ti=80.0Tf=72.37536658951147Nu1=4.0021785269029327E-7k1=0.6615219088637725Pr1=2.474919137688163
T=43.687683294755736roa=1.1140820539695484mua=1.933930132556162E-5nua=1.7358955973354387E-
5ka=0.027623397442086965Pra=0.7045430037495769
inside pipe(water) Re=4747.414407498473
inside pipe(water) Nu=24.61597955179493 h1 = 857.0531464028934 W/m2K
Re=6820.111861657432C=0.26n=0.37
outside pipe(air) Nu2=40.2683449544062 h2 = 55.617424850530995 W/m2K
outside pipe(air) Nu3=39.17388618092624 h3 = 54.105791366340185 W/m2K
outside pipe(air) Nu4=45.81804188312344 h4 = 63.28249904778528 W/m2K
i=1  U=32.70826562427765Q=1865.5491086440968Ti=80.0To=63.92253625890966 LMTD=57.036014384675795
============================
ro1=976.6781104770996nu1=4.0235014821955534E-7k1=0.6611999093426022Pr1=2.4897639048358826Cp1=4.1910342677998305
i=2 water side Ti=80.0Tf=71.96126812945482Nu1=4.0235014821955534E-7k1=0.6611999093426022Pr1=2.4897639048358826
T=43.48063406472741roa=1.1148105679118085mua=1.9329551467290247E-5nua=1.733886637215605E-
5ka=0.027608292171269623Pra=0.7045679087345669
inside pipe(water) Re=4722.2550020366925
inside pipe(water) Nu=24.53957694787551 h1 = 853.9771606969 W/m2K
Re=6820.111861657432C=0.26n=0.37
outside pipe(air) Nu2=40.2683449544062 h2 = 55.58701163773587 W/m2K
outside pipe(air) Nu3=39.17388618092624 h3 = 54.07620475835366 W/m2K
outside pipe(air) Nu4=45.81804188312344 h4 = 63.24789435123703 W/m2K
i=2  U=32.6840011507320Q=1849.2960736265325Ti=80.0To=64.06556392889544 LMTD=56.58107968782503
============================
ro1=976.6365141914225nu1=4.019804435282003E-7k1=0.661255689459987Pr1=2.4871892059215224Cp1=4.1910785411065286
i=3 water side Ti=80.0Tf=72.03278196444772Nu1=4.019804435282003E-7k1=0.661255689459987Pr1=2.4871892059215224
T=43.51639098222386roa=1.1146846872037948mua=1.9331235439373254E-5nua=1.7342335156559818E-
5ka=0.027610901046528892Pra=0.7045636034607876
inside pipe(water) Re=4726.5980984637345
inside pipe(water) Nu=24.55280107967959 h1 = 854.5094424272338 W/m2K
Re=6820.111861657432C=0.26n=0.37
outside pipe(air) Nu2=40.2683449544062 h2 = 55.592264392180034 W/m2K
outside pipe(air) Nu3=39.17388618092624 h3 = 54.081314747477 W/m2K
outside pipe(air) Nu4=45.81804188312344 h4 = 63.253871029031885 W/m2K
i=3  U=32.68819704233824Q=1852.107907555797Ti=80.0To=64.04082468605982 LMTD=56.65983673424751
============================
ro1=976.6437109791911nu1=4.0204434687124306E-7k1=0.6612460463977132Pr1=2.4876342160308504Cp1=4.191070877408291
i=4 water side Ti=80.0Tf=72.0204123430299Nu1=4.0204434687124306E-7k1=0.6612460463977132Pr1=2.4876342160308504
```

```
T=43.51020617151495roa=1.1147064585336182mua=1.933094417150925E-5nua=1.73417351478692E-
5ka=0.027610449800874216Pra=0.7045643480097253
inside pipe(water) Re=4725.846824575513
inside pipe(water) Nu=24.55051460984809 h1 = 854.4174064627024 W/m2K
Re=6820.111861657432C=0.26n=0.37
outside pipe(air) Nu2=40.2683449544062 h2 = 55.59135584639595 W/m2K
outside pipe(air) Nu3=39.17388618092624 h3 = 54.08043089518121 W/m2K
outside pipe(air) Nu4=45.81804188312344 h4 = 63.2528372694166 W/m2K
i=4  U=32.687471450523915Q=1851.621699329746Ti=80.0To=64.04510262317021 LMTD=56.646220009167095
============================

> Terminated with exit code 0.
```

PROBLEM :

water entering into a copper tube with internal diameter of $D_i=1.9x10^{-2}$ m and external diameter $D_o = 2x10^{-2}$ m water inlet velocity U=0.1 m/s , inlet pressure P_i=200 kPa and inlet temperature T_i=80 ℃. At the outer surface has air at a pressure P=101.325 kPa and $T_\infty = 15$ ℃ . Outside air is still (not moving) Calculate the overall heat transfer and the rate of heat loss to air for L=10 m of pipe length. What is the water exit temperature ?

```
 public class HT_HE_LMTD_4
{
            //Air properties
public static double mua(double T)
  {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
public static double ka(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
public static double Pra(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
```

```
return y;
}
public static double Cpa(double T)
{double a[]={1.89668613E+00,
-2.23937197E-02,
2.40885259E-04,
-1.45254290E-06,
5.43552393E-09,
-1.32310554E-11,
2.13239465E-14,
-2.25744420E-17,
1.50864395E-20,
-5.76865768E-24,
9.61540784E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
return y*1e3;
}

  public static double roa(double T,double P)
  {   double M=2.8965000E+01;
      double R=8314.5/M;
                return P*1e3/R/T;
  }
//Outside of pipe heat transfer coefficients
  public static double Nu(double Ts,double Tinf,double P,double D,int select)
  {//Nusselt number
   double T=(Tinf+Ts)/2.0;
   double Ra=Ra(Ts,Tinf,P,D);
   double Pr=Pra(T);
   double Gr=Ra/Pr;
   double Nu=0.0;
   double Racritic=1e9;
   double C=0,n=0;
   double Pr_05=Math.sqrt(Pr);
   if(select==1) {Nu=(0.60+0.387*Math.pow(Ra,(1.0/6.0))/Math.pow((1+Math.pow((0.559/Pr),(9.0/16.0))),(8.0/27.0)));
    Nu=Nu*Nu;
   }
   else if(select==2)
   {  if(Ra>=1e-10 && Ra<1e-2)    {C=0.675;n=0.058;}
      else if(Ra>=1e-2 && Ra<1e2) {C=1.02;n=0.148;}
      else if(Ra>=1e2 && Ra<1e4) {C=0.85;n=0.188;}
      else if(Ra>=1e4 && Ra<1e7)  {C=0.480;n=0.250;}
      else if(Ra>=1e7 && Ra<=1e12){C=0.125;n=0.333;}
                Nu=C*Math.pow(Ra,n);
                System.out.println("C="+C+"n="+n);
   }
   return Nu;
   }

  public static double Ra(double Ts,double Tinf,double P,double D)
  {   double T=(Ts+Tinf)/2.0;
                double mu=mua(T);
      double ro=roa(T,P);
      double Cp=Cpa(T);
      double k=ka(T);
      double beta=1/T;
      double nu=mu/ro;
      double alpha=k/(ro*Cp);
      double g=9.806; //m/s^2
      double Ra=g*beta*(Ts-Tinf)*D*D*D/alpha/nu;
      System.out.println("Ra="+Ra);
                return Ra;
  }

 //inside of the pipe
  public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
```

```
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 public static double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
  double Nu=0;
  double x=Re/1000;
  double f=0;
  if(Re<2300) Nu=3.66;
  else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
  else {f=f_Goudar(Re,eod);
  Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
  }
 return Nu;
  }
public static void main(String arg[])
{
double Pi=200.0;
double Ti=80.0;
double To=60.0;
double Tf=(Ti+To)/2.0;
double Tinf=15.0;
double P=101.325;
double V=0.1;  //m/s
double Di=0.019; //m
double Do=0.02; //m
double eod=0.0002;
double km=320.0;
double L=10.0; //m
steam_IAPWS95 ap=new steam_IAPWS95();
double T=0;
for(int i=0;i<5;i++)
{
Tf=(Ti+To)/2.0;
T=(Tf+Tinf)/2.0;
double a1[]=ap.propertyC("tp",Tf,Pi);
double ro1=1.0/a1[2];
double nu1=a1[11]/ro1;
double k1=a1[12];
double Pr1=a1[14];
double Cp1=a1[7];
double A=Math.PI*Di*Di/4.0;
System.out.println("Ti="+Ti+"nu1="+nu1+"k1="+k1+"Pr1="+Pr1);
T=(Tf+Tinf)/2.0;
double roa=roa(T+273.15,P);
double mua=mua(T+273.15);
double nua=mua/roa;
double ka=ka(T+273.15);
double Pra=Pra(T+273.15);
System.out.println("i = "+i+" T="+T+"roa="+roa+"mua="+mua+"nua="+nua+"ka="+ka+"Pra="+Pra);
//water:
double Re=V*Di/nu1;
double Nu=Nu(Re,eod,Pr1);
double h1=Nu*k1/Di;
System.out.println("inside pipe(water) Re="+Re);
System.out.println("inside pipe(water) Nu="+Nu+" h1 = "+h1+" W/m2K");
double Nu2=Nu((Tf+273.15),(Tinf+273.15),P,Do,1);
double h2=Nu2*ka/Do;
double Nu3=Nu((Tf+273.15),(Tinf+273.15),P,Do,2);
double h3=Nu3*ka/Do;
System.out.println("outside pipe(air) Nu2="+Nu2+" h2 = "+h2+" W/m2K");
System.out.println("outside pipe(air) Nu3="+Nu3+" h3 = "+h3+" W/m2K");
double A1=Math.PI*Di*L;
double A2=Math.PI*Do*L;
double R1=1.0/(h1*A1);
double R3=1.0/(h2*A2);
double R2=Math.log(Do/Di)/(2.0*Math.PI*km*L);
double R=R1+R2+R3;
double U=1/R;
double dT1=Ti-Tinf;
double dT2=To-Tinf;
double lmtd=(dT1-dT2)/Math.log(dT1/dT2);
double Q=U*lmtd;
double m=ro1*A*V;
```

```
To=Ti-Q/(m*Cp1*1e3);
System.out.println("i="+i+"  U="+U+"Q="+Q+"Ti="+Ti+"To="+To+" LMTD="+lmtd);
System.out.println("============================");
} //end of iteration
}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_HE_LMTD_4
Ti=80.0nu1=4.1273313978230366E-7k1=0.6596421102911764Pr1=2.562218988173881
i = 0 T=42.5roa=1.1182739647710274mua=1.9283336828076815E-5nua=1.72438395559224E-
5ka=0.02753670618667034Pra=0.7046866710239903
inside pipe(water) Re=4603.458789381817
inside pipe(water) Nu=24.172035866755703 h1 = 839.2048815358282 W/m2K
Ra=32395.05967421249
Ra=32395.05967421249
C=0.48n=0.25
outside pipe(air) Nu2=5.8220607155556845 h2 = 8.016018766260627 W/m2K
outside pipe(air) Nu3=6.439630274399245 h3 = 8.866310340845965 W/m2K
i=0  U=4.9864124128528 55Q=271.20350231243754Ti=80.0To=77.66522035831316 LMTD=54.38850216508166
============================
Ti=80.0nu1=3.6944375791468264E-7k1=0.6662357398839044Pr1=2.262063735856217
i = 1 T=46.91630508957829roa=1.1028439150481146mua=1.9490985140696055E-5nua=1.7673385031866187E-
5ka=0.027858525997821373Pra=0.704162329347211
inside pipe(water) Re=5142.866699723144
inside pipe(water) Nu=25.755491434323048 h1 = 903.1173100957777 W/m2K
Ra=35272.10578204864
Ra=35272.10578204864
C=0.48n=0.25
outside pipe(air) Nu2=5.947160123283945 h2 = 8.283955745385617 W/m2K
outside pipe(air) Nu3=6.5780789935096 h3 = 9.162779232820492 W/m2K
i=1  U=5.1551197383585 99Q=329.0280588772013Ti=80.0To=77.1560714622631 LMTD=63.825493019870095
============================
Ti=80.0nu1=3.70574516126896E-7k1=0.6660607183269764Pr1=2.269837149545824
i = 2 T=46.789017865565775roa=1.1032826797270903mua=1.9485017489012165E-5nua=1.7660947504253405E-
5ka=0.02784927086805778Pra=0.7041770639660997
inside pipe(water) Re=5127.173934835233
inside pipe(water) Nu=25.712355212756655 h1 = 901.3678833098461 W/m2K
Ra=35195.66386807389
Ra=35195.66386807389
C=0.48n=0.25
outside pipe(air) Nu2=5.943933034927489 h2 = 8.27671005556462 W/m2K
outside pipe(air) Nu3=6.57451208054307 h3 = 9.154768387818104 W/m2K
i=2  U=5.150558361423512Q=327.40777491029473Ti=80.0To=77.17040757409515 LMTD=63.5674332636444
============================
Ti=80.0nu1=3.7054259025563836E-7k1=0.6660656584438004Pr1=2.269617622977284
i = 3 T=46.79260189352379roa=1.103270320647848mua=1.9485185534159593E-5nua=1.7661297661590094E-
5ka=0.02784953148134067Pra=0.7041766487763781
inside pipe(water) Re=5127.6156910577665
inside pipe(water) Nu=25.713571749667246 h1 = 901.4172157044221 W/m2K
Ra=35197.82129160465
Ra=35197.82129160465
C=0.48n=0.25
outside pipe(air) Nu2=5.944024182398165 h2 = 8.276914429677396 W/m2K
outside pipe(air) Nu3=6.574612829324842 h3 = 9.154994348395421 W/m2K
i=3  U=5.150687021833394Q=327.4534292885616Ti=80.0To=77.17000368456361 LMTD=63.57470914084081
============================
Ti=80.0nu1=3.7054348963006285E-7k1=0.6660655192760391Pr1=2.2696238071552655
i = 4 T=46.7925009211409roa=1.1032706688348908mua=1.948518079985517E-5nua=1.766128779661341E-
5ka=0.027849524139127484Pra=0.7041766604732272
inside pipe(water) Re=5127.6032454298165
inside pipe(water) Nu=25.713537477869597 h1 = 901.4158259274267 W/m2K
Ra=35197.76051478287
Ra=35197.76051478287
C=0.48n=0.25
outside pipe(air) Nu2=5.944021614736029 h2 = 8.27690867215433 W/m2K
outside pipe(air) Nu3=6.57460999119228 h3 = 9.154987982752907 W/m2K
i=4  U=5.150683397278735Q=327.4521430988321Ti=80.0To=77.17001506311283 LMTD=63.57450416615301
============================

> Terminated with exit code 0.
```

PROBLEM :

water entering into a copper tube with internal diameter of $D_i=1.9x10^{-2}$ m and external diameter $D_o = 2x10^{-2}$ m water inlet velocity V=1 m/s , inlet pressure P_i=200 kPa and inlet temperature T_i=20 ℃. At the

outer surface has saturated steam at a pressure P=101.325 kPa ($T_{sat} = 100$ ℃). Steam is condensing on the surface of the pipe. Calculate the overall heat transfer and the rate of condensation for L=5 m of pipe length. What is the water exit temperature?

```
 public class HT_HE_LMTD_5
{
public static double h( steam_IAPWS95 st,double tsat,double tw, double D)
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // L pipe length
  // D pipe diameter
  // W width of the plate
  double tf=(tw+tsat)/2.0;
  double al[]=st.propertyC("tx",tf,0.0);
  double av[]=st.propertyC("tx",tsat,1.0);
  double al1[]=st.propertyC("tx",tsat,0.0);
  double kl=al[12];   // W/(mK)
  double mul=al[11];  //kg/(m.s)=Pa.s
  double muv=av[11];  //kg/(m.s)=Pa.s
  double Prl=al[14];
  double Cpl=al[7]*1e3;  //J/kgK
  double hfg=(av[3]-al1[3])*1e3; //J/kg
  double dT=tsat-tw;
  double hfgp=hfg+0.68*Cpl*dT;
  double rol=al[15];  //kg/m^3
  double rov=av[15];  //kg/m^3
  double Psat=al[0]*1e3; //Pa
  double g=9.806; //m/s^2
  double hL=kl/D*0.729*Math.pow((g*(rol-rov)*rol*hfgp*D*D*D/(mul*kl*dT)),0.25);
  return hL;
}

 //inside of the pipe
  public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 public static double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
   double Nu=0;
   double x=Re/1000;
   double f=0;
   if(Re<2300) Nu=3.66;
   else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
   else {f=f_Goudar(Re,eod);
   Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
   }
 return Nu;
   }
public static void main(String arg[])
{
double Pi=200.0;
double Ti=20.0;
double To=60.0;
double Tsat=100.0;
double P=101.325;
double V=1.0;  //m/s
double Di=0.019; //m
double Do=0.02; //m
double L=5.0; //m
double km=320.0;
double h1=0,h2=0,Q=0,lmtd=0;
steam_IAPWS95 ap=new steam_IAPWS95();
```

```
//Saturated steam
// iteration
for(int i =0;i<5;i++)
{
double Tf=(Ti+To)/2.0;
double Tw=(Tsat+Tf)/2.0;
//saturated steam
h2=h(ap,Tsat,Tw,Do);
//water:
double a1[]=ap.propertyC("tp",Tf,Pi);
double ro1=1.0/a1[2];
double nu1=a1[11]/ro1;
double k1=a1[12];
double Pr1=a1[14];
double Cp1=a1[7];
double Re=V*Di/nu1;
double eod=0.0002;
double Nu=Nu(Re,eod,Pr1);
h1=Nu*k1/Di;

double A1=Math.PI*Di*L;
double A2=Math.PI*Do*L;
double R1=1.0/(h1*A1);
double R3=1.0/(h2*A2);
double R2=Math.log(Do/Di)/(2.0*Math.PI*km*L);
double R=R1+R2+R3;
double U=1.0/R;
double dT1=Tsat-Ti;
double dT2=Tsat-To;
lmtd=(dT1-dT2)/Math.log(dT1/dT2);
Q=U*lmtd;
double A=Math.PI*Di*Di/4.0;
double m=ro1*A*V;
To=Ti+Q/(m*Cp1*1e3);
} //enf for (iteration)
System.out.println("h_saturation = "+h2+" W/m2K");
System.out.println("h_water = "+h1+" W/m2K");
System.out.println("Q="+Q+"W");
System.out.println("Ti = "+Ti+" degree C\nTo = "+To+" degree C");
System.out.println("Tsat = "+Tsat+" degree C\n");
System.out.println("LMTD = "+lmtd+" degree C\n");
  double av[]=ap.propertyC("tx",Tsat,1.0);
  double al1[]=ap.propertyC("tx",Tsat,0.0);
  double hfg=(av[3]-al1[3])*1e3; //J/kg
  double msat=Q/hfg;
  System.out.println("m saturation = "+msat+" kg/s");
  System.out.println("============================");

}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_HE_LMTD_5
h_saturation = 9979.005603306858 W/m2K
h_water = 5621.294800792641 W/m2K
Q=56952.555118449636W
Ti = 20.0 degree C
To = 68.51066118855188 degree C
Tsat = 100.0 degree C

LMTD = 52.40444094890017 degree C

m saturation = 0.02524041002286076 kg/s

> Terminated with exit code 0.
```

PROBLEM: Water entering into a double pipe heat exchanger internal pipe with internal diameter of $D_{ci}=1.9x10^{-2}$ m and external diameter of $D_{co}=2.0x10^{-2}$ m with an inlet velocity of U=0.1 m/s and temperature of $T_{ci}=20$ ℃. Inlet pressure id $P_{ci}=200$ kPa. In the outside pipe water is enetring as counter flow (from opposite direction) with an internal diameter of $D_{hi}=3.5x10^{-2}$ m, with an inlet velocity of U=0.1 m/s. Inlet pressure is $P_{hi}=250$ kPa. Inlet temperature is $T_{ci}=80$ ℃. The total length of the heat exchager is L=10.0 m. Find exit temperatures of water and heat transfer.

```
 public class HT_HE_LMTD_3
{


 //inside of the pipe
  public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 public static double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
   double Nu=0;
   double x=Re/1000;
   double f=0;
   if(Re<2300) Nu=3.66;
   else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
   else {f=f_Goudar(Re,eod);
   Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
   }
 return Nu;
   }
public static void main(String arg[])
{
double Pci=200.0;
double Phi=250.0;
double Tci=20.0;
double Thi=90.0;
//estimations
double Tco=60.0;
double Tho=70.0;

double Dci=1.9e-2;
double Dco=2.0e-2;
double Dhi=3.5e-2;
//inside cross sectional area
double Ac=Math.PI*Dci*Dci/4.0;
//Hydraulic diameter
double Dh=Dhi-Dco;
//outside cross sectional area
double Ah=Math.PI*(Dhi*Dhi-Dco*Dco)/4.0;
steam_IAPWS95 ap=new steam_IAPWS95();
for(int i=0;i<10;i++)
{
double Tcf=(Tci+Tco)/2.0;
double Thf=(Thi+Tho)/2.0;
double ac[]=ap.propertyC("tp",Tcf,Pci);
double roc=1.0/ac[2];
double nuc=ac[11]/roc;
double kc=ac[12];
double Prc=ac[14];
double Cpc=ac[7];
double ah[]=ap.propertyC("tp",Thf,Phi);
double roh=1.0/ah[2];
double nuh=ah[11]/roc;
double kh=ah[12];
double Prh=ah[14];
double Cph=ah[7];
//water:
double Uc=0.1;  //m/s
double Uh=0.1; //m/s
double Rec=Uc*Dci/nuc;
double Reh=Uh*Dh/nuh;
```

```
double eod=0.0002;
double Nuc=Nu(Rec,eod,Prc);
double hc=Nuc*kc/Dci;
double Nuh=Nu(Reh,eod,Prh);
double hh=Nuh*kh/Dh;
System.out.println("inside pipe(water) Rec="+Rec);
System.out.println("inside pipe(water) Nuc="+Nuc+" hc = "+hc+" W/m2K");
System.out.println("outside pipe(water) Reh="+Reh);
System.out.println("outside pipe(water) Nuh="+Nuh+" hh = "+hh+" W/m2K");

double km=320.0;
double L=10.0; //m
double A1=Math.PI*Dci*L;
double A2=Math.PI*Dco*L;
double R1=1.0/(hc*A1);
double R3=1.0/(hh*A2);
double R2=Math.log(Dco/Dci)/(2.0*Math.PI*km*L);
double R=R1+R2+R3;
double dT1=Thi-Tco;
double dT2=Tho-Tci;
double lmtd=(dT1-dT2)/Math.log(dT1/dT2);
System.out.println("LMTD="+lmtd);
double Q=lmtd/R;
double mc=roc*Ac*Uc;
double mh=roh*Ah*Uh;
Tco=Tci+Q/(mc*Cpc*1e3);
Tho=Thi-Q/(mh*Cph*1e3);
System.out.println("i="+i+"R="+R+"\nQ="+Q+"  Tco="+Tco+"degree C"+"  Tho="+Tho+"degree C");
}
}
}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" HT_HE_LMTD_3
inside pipe(water) Rec=2888.2712134515223
inside pipe(water) Nuc=7.455990560211944 hc = 246.69660354771727 W/m2K
outside pipe(water) Reh=4203.41399113451
outside pipe(water) Nuh=20.74142272051443 hh = 922.3763475240984 W/m2K
LMTD=39.15230377942435
i=0R=0.008519033385997792
Q=4595.862230540917  Tco=59.08898259334578degree C  Tho=72.60834042013472degree C
inside pipe(water) Rec=2864.08192605879
inside pipe(water) Nuc=7.278606957763714 hc = 240.60481348673915 W/m2K
outside pipe(water) Reh=4272.780559424517
outside pipe(water) Nuh=20.96245388521363 hh = 933.4108646320947 W/m2K
LMTD=40.80269562574076
i=1R=0.008670573989549367
Q=4705.881718432967  Tco=60.01811937922233degree C  Tho=72.18125414576684degree C
inside pipe(water) Rec=2888.7529825625256
inside pipe(water) Nuc=7.4595110421682875 hc = 246.81761097556392 W/m2K
outside pipe(water) Reh=4260.762565080699
outside pipe(water) Nuh=20.923113614748406 hh = 931.4643450557759 W/m2K
LMTD=40.06166375798099
i=2R=0.00849886892924804
Q=4713.764159853393  Tco=60.09190045733399degree C  Tho=72.15317399300913degree C
inside pipe(water) Rec=2890.714985776805
inside pipe(water) Nuc=7.473842723857956 hc = 247.31026825039643 W/m2K
outside pipe(water) Reh=4259.962151749991
outside pipe(water) Nuh=20.920468366028363 hh = 931.3337476193213 W/m2K
LMTD=40.00511821594785
i=3R=0.008485587081394925
Q=4714.478542523131  Tco=60.09851344807033degree C  Tho=72.15058538437845degree C
inside pipe(water) Rec=2890.890860852026
inside pipe(water) Nuc=7.475126991374054 hc = 247.35441882761324 W/m2K
outside pipe(water) Reh=4259.888520158891
outside pipe(water) Nuh=20.92022529712247 hh = 931.3217433270271 W/m2K
LMTD=40.0000187716094
i=4R=0.008484399983512166
Q=4714.5371327780285  Tco=60.09905991644612degree C  Tho=72.15037426145636degree C
inside pipe(water) Rec=2890.905394546189
inside pipe(water) Nuc=7.47523311549412 hc = 247.35806718978935 W/m2K
outside pipe(water) Reh=4259.882509030763
outside pipe(water) Nuh=20.920205441833144 hh = 931.3207628968377 W/m2K
LMTD=39.999598535237055
i=5R=0.008484301886396888
Q=4714.542112105829  Tco=60.09910624537395degree C  Tho=72.15035628237234degree C
inside pipe(water) Rec=2890.9066266962855
inside pipe(water) Nuc=7.47524211255427 hc = 247.3583764932017 W/m2K
```

```
outside pipe(water) Reh=4259.881997298155
outside pipe(water) Nuh=20.920203751867053 hh = 931.3206794439914 W/m2K
LMTD=39.99956287439251
i=6R=0.008484293570597877
Q=4714.542529858947  Tco=60.099110135759425degree C  Tho=72.15035477507386degree C
inside pipe(water) Rec=2890.9067301637915
inside pipe(water) Nuc=7.475242868065379 hc = 247.35840246637056 W/m2K
outside pipe(water) Reh=4259.881954391198
outside pipe(water) Nuh=20.920203610159174 hh = 931.3206724463963 W/m2K
LMTD=39.999559880870166
i=7R=0.008484292872276637
Q=4714.54256507023  Tco=60.09911046356329degree C  Tho=72.15035464799405degree C
inside pipe(water) Rec=2890.906738881979
inside pipe(water) Nuc=7.475242931724779 hc = 247.3584046548723 W/m2K
outside pipe(water) Reh=4259.88195077387
outside pipe(water) Nuh=20.920203598212616 hh = 931.3206718564679 W/m2K
LMTD=39.99955962860462
i=8R=0.008484292813436599
Q=4714.542568033155  Tco=60.099110491150306degree C  Tho=72.15035463730226degree C
inside pipe(water) Rec=2890.9067396156656
inside pipe(water) Nuc=7.475242937082271 hc = 247.35840483905133 W/m2K
outside pipe(water) Reh=4259.881950469523
outside pipe(water) Nuh=20.92020359720746 hh = 931.320671806831 W/m2K
LMTD=39.9995596073759
i=9R=0.008484292808484745
Q=4714.542568282675  Tco=60.09911049347354degree C  Tho=72.15035463640123degree C

> Terminated with exit code 0.
```

16.4 ε-NUMBER OF TRANSFER UNITS (NTU) METHOD

When the inlet or outlet temperatures of the fluid straems are not known, a trai and error procedure or numerical curve fitting methods should be applied to get a solution. To avoid this trial and error procedure the method of number of transfer units(NTU) based on the concept of heat exchanger effectiveness may be used. Defining

$C_h = m_h C_{ph}$ and $C_c = m_c C_{pc}$ and then $\boldsymbol{C = \frac{C_{min}}{C_{max}}}$ where C_{min} and C_{max} are smaller and larger of the two magnitudes of C_h and C_c respectively. Heat exchanger effectiveness is defined as

$\varepsilon = \frac{Q}{Q_{max}}$ where Q_{max} is defined as thermodynamically limited maximum possible heat transfer rate if an infinite heat transfer surface is existed for **counterflow heat exchanger**.

The actual heat transfer is found from energy balance equations (first law of thermodynamics)

$Q = m_h C_{ph}(T_{h1} - T_{h2}) = m_c C_{pc}(T_{c2} - T_{c1})$

$Q = C_h(T_{h1} - T_{h2}) = C_c(T_{c2} - T_{c1})$

If $C_h > C_c \quad C_{max} = C_h \quad C_{min} = C_c \quad$ then $(T_{h1} - T_{h2}) < (T_{c2} - T_{c1})$

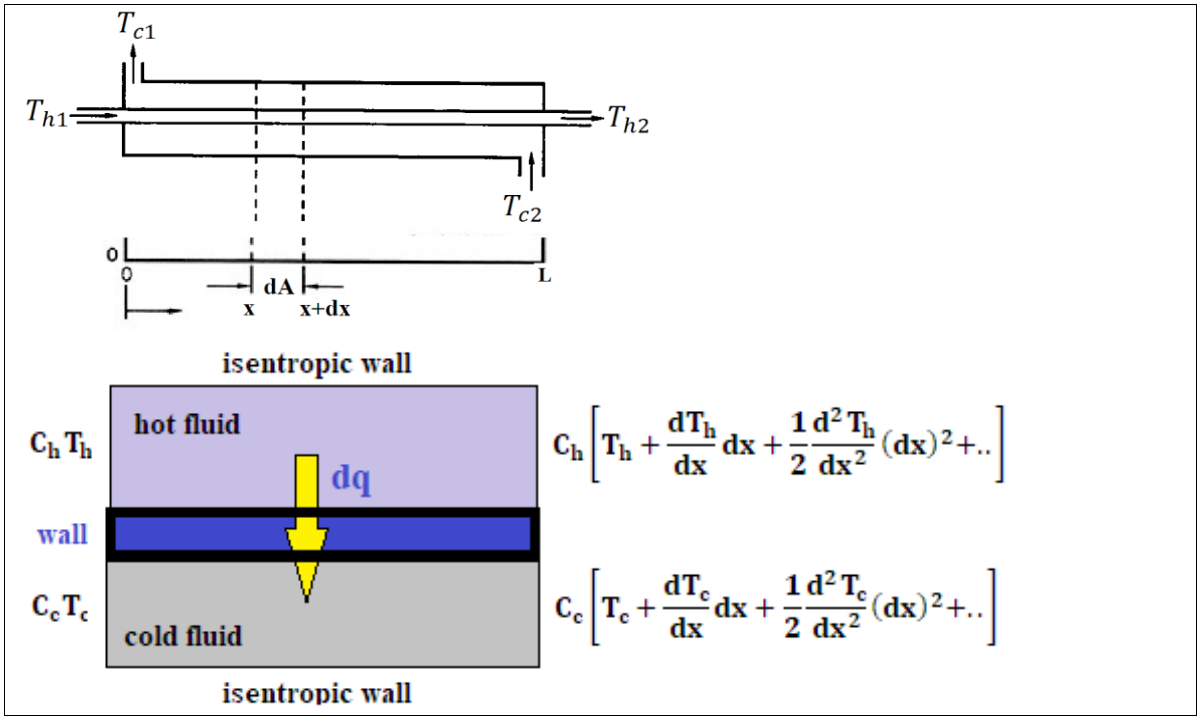
If $C_h < C_c \quad C_{max} = C_c \quad C_{min} = C_h \quad$ then $(T_{h1} - T_{h2}) > (T_{c2} - T_{c1})$

$Q_{max} = C_{min}(T_{h1} - T_{c1})$

$\varepsilon = \frac{C_h(T_{h1} - T_{h2})}{C_{min}(T_{h1} - T_{c1})}$ (where $C_{min} = C_c$) or $\varepsilon = \frac{C_c(T_{c2} - T_{c1})}{C_{min}(T_{h1} - T_{c1})}$ (where $C_{min} = C_h$)

Equation is valid for alll heat exchanger flow arrangments and therefore heat transfer in general can be defined as: $Q = \varepsilon C_{min}(T_{h1} - T_{c1})$ Number of transfer units can be defined as:

$NTU = \frac{AU}{C_{min}} = \frac{1}{C_{min}} \int_A U dA$ for simplicity of further relations symbol N will replace NTU (N=NTU)



Now energy balance of a simple counter flow heat exchanger can be investigated. According to first low of thermodynamics

$dq = dq_{h-} - dq_{ho}$ $dq = dq_{ci} - dq_{co}$

$dq_{hi} = C_h T_{hi}$ and through taylos expansion

$dq_{ho} = C_h\left[T_h + \frac{dT_h}{dx}dx + \frac{1}{2}\frac{d^2T_h}{dx^2}(dx)^2+..\right]$ ignoring higher derivatives

$dq_{ho} = C_h\left[T_h + \frac{dT_h}{dx}dx\right]$

$dq_{ci} = C_c T_{ci}$

Similarly by using taylor expansion formula

$dq_{co} = C_c\left[T_c + \frac{dT_c}{dx}dx + \frac{1}{2}\frac{d^2T_c}{dx^2}(dx)^2+..\right]$ andignoring higher derivatives

$dq_{co} = C_c\left[T_c + \frac{dT_c}{dx}dx\right]$ so

$dq = -C_h dT_h = -C_c dT_c$

$d(T_h - T_c) = -\left(\frac{1}{C_h} - \frac{1}{C_c}\right)dq = -\left(1 - \frac{C_h}{C_c}\right)\frac{dq}{C_h}$

and from heat transfer

$dq = U(T_h, T_c, A_h, A_c, m_h, m_c)dA(T_h - T_c)$ and assuming that

$U = U(T_h, T_c, A_h, A_c, m_h, m_c) = U_{average} = constant$ and $dA = A\frac{dx}{L}$ where L is length of the heat exchanger and

$$\frac{1}{U_i A_i} = \frac{1}{U_o A_o} = \left(\frac{1}{h_H \eta_i A_i}\right) + \left(\frac{\ln(\frac{r_o}{r_i})}{2\pi k L}\right) + \left(\frac{1}{h_C \eta_o A_o}\right) + \frac{R_{fi}}{\eta_i A_i} + \frac{R_{fo}}{\eta_o A_o}$$

$dq = UA\frac{dx}{L}(T_h - T_c)$

so

$$\frac{d(T_h - T_c)}{(T_h - T_c)} = -\left(1 - \frac{C_h}{C_c}\right)\frac{UA}{C_h}\frac{dx}{L}$$

integration of the equation gives

$$\frac{T_{h2}-T_{c2}}{T_{h1}-T_{c1}} = exp\left[-\frac{UA}{C_h}\left(1-\frac{C_h}{C_c}\right)\right]$$

by algebraic manupulation

$$\frac{T_{h2}-T_{c2}}{T_{h1}-T_{c1}} = \frac{1-\frac{T_{h1}-T_{h2}}{T_{h1}-T_{c2}}}{1-\frac{T_{c1}-T_{c2}}{T_{h1}-T_{c2}}}$$

and

$$\varepsilon_h = \frac{T_{h1}-T_{h2}}{T_{h1}-T_{c2}} \quad \varepsilon_c = \frac{T_{c1}-T_{c2}}{T_{h1}-T_{c2}}$$

$$\frac{T_{h2}-T_{c2}}{T_{h1}-T_{c1}} = \frac{1-\varepsilon_h}{1-\varepsilon_c} = \frac{1-\varepsilon_h}{1-\frac{C_h}{C_c}\varepsilon_h}$$

so

$$\varepsilon_h = \frac{1-exp\left[-\left(\frac{UA}{C_h}\right)\left(1-\frac{C_h}{C_c}\right)\right]}{1-\left(\frac{C_h}{C_c}\right)exp\left[-\left(\frac{UA}{C_h}\right)\left(1-\frac{C_h}{C_c}\right)\right]}$$

$$\varepsilon_c = \left(\frac{C_h}{C_c}\right)\varepsilon_h = \frac{1-exp\left[-\left(\frac{UA}{C_c}\right)\left(1-\frac{C_c}{C_h}\right)\right]}{\left(\frac{C_c}{C_h}\right)-exp\left[-\left(\frac{UA}{C_c}\right)\left(1-\frac{C_c}{C_h}\right)\right]}$$

$$\varepsilon_c = \frac{1-exp\left[-\left(\frac{UA}{C_c}\right)\left(1-\frac{C_c}{C_h}\right)\right]}{1-\left(\frac{C_c}{C_h}\right)exp\left[-\left(\frac{UA}{C_c}\right)\left(1-\frac{C_c}{C_h}\right)\right]}$$

and when we specify $C_{min} = C_c$ C=$\frac{C_c}{C_h}$ or $C_{min} = C_h$ C=$\frac{C_h}{C_c}$ $NTU = N = \left(\frac{UA}{C_{min}}\right)$

$$\varepsilon = \frac{1-exp[-N(1-C)]}{1-Cexp[-N(1-C)]}$$

Some of the $\varepsilon - NTU$ relations are as follows (N=NTU , C=C_{min}/C_{max})

Parallel flow heat exchanger:	$\varepsilon = \frac{1-\exp[-N(1+C)]}{1+C}$
Parallel flow heat exchanger: C=1	$\varepsilon = \frac{1}{2}[1-\exp[-N]]$
Counter flow heat exchanger: C<1	$\varepsilon = \frac{1-\exp[-N(1-C)]}{1-Cexp[-N(1-C)]}$
Counter flow heat exchanger: C=1	$\varepsilon = \frac{N}{1+N}$
Shell & Tube (one Shell pass (2,4,.. tube passes)	$\varepsilon_1 = 2\left\{1+C+(1+C^2)^{1/2}\frac{1+exp[-N_1(1+C^2)^{1/2}]}{1-exp[-N_1(1+C^2)^{1/2}]}\right\}^{-1}$
Shell & Tube (n Shell pass (2,4,.. tube passes)	$\varepsilon = \left[\left(\frac{1-\varepsilon_1 C}{1-\varepsilon_1}\right)^n - 1\right]\left[\left(\frac{1-\varepsilon_1 C}{1-\varepsilon_1}\right)^n - C\right]^{-1}$
Cross flow single pass Both fluids unmixed (approximate eqn?)	$\varepsilon = 1-exp\left[\frac{1}{C}N^{0.22}\{\exp(-CN^{0.78})-1\}\right]$
Cross flow single pass Both fluids unmixed	$\varepsilon = 1-exp[-N]-exp[-(1+C)N]\sum_{j=1}^{n} C^n P_n(N)$ $P_n(y) = \frac{1}{(n+1)!}\sum_{j=1}^{n}\frac{(n+1-j)}{j!}y^{n+j}$

Cross flow single pass C_{max}(mixed) , C_{min}(unmixed)	$\varepsilon = \left(\frac{1}{C}\right)(1 - exp\{-C[1 - \exp(-N)]\})$
Cross flow single pass C_{max}(unmixed) , C_{min}(mixed)	$\varepsilon = (1 - exp\{-[1 - \exp(-CN)]/C\})$
Cross flow single pass Both fluids mixed	$\varepsilon = \left[\frac{1}{[1 - \exp[-N]]} + \frac{C}{[1 - \exp(-CN)]} - \frac{1}{N}\right]^{-1}$

Some of the $NTU - \varepsilon$ relations are as follows (N=NTU , C=C_{min}/C_{max})

Parallel flow heat exchanger:	$N = \frac{ln[1-\varepsilon(1+C)]}{(1+C)}$
Counter flow heat exchanger: C<1	$N = \frac{1}{(C-1)} ln\left[\frac{\varepsilon-1}{\varepsilon C-1}\right]$
Counter flow heat exchanger: C=1	$N = \frac{\varepsilon}{1-\varepsilon}$
Shell & Tube (one Shell pass (2,4,.. tube passes)	$E = \frac{2/\varepsilon_1 - (1+C)}{(1+C^2)^{1/2}}$ $N_1 = -(1 + C^2)^{-1/2} ln\left(\frac{E-1}{E+1}\right)$
Shell & Tube (n Shell pass (2,4,.. tube passes)	$\varepsilon_1 = \frac{F-1}{F+C}$ $F = \left(\frac{\varepsilon C-1}{\varepsilon-1}\right)^{1/n}$ N=nN_1
Cross flow single pass Both fluids unmixed	
Cross flow single pass C_{max}(mixed) , C_{min}(unmixed)	$N = -ln\left[1 + \left(\frac{1}{C}\right)\ln(1 - \varepsilon C)\right]$
Cross flow single pass C_{max}(unmixed) , C_{min}(mixed)	$N = -\left(\frac{1}{C}\right) ln[C\ln(1 - \varepsilon) + 1]$

PROBLEM: A counter flow heat exchanger of heat transfer area A=12.5 $m^2$is to cool oil [Cph=2000 J/kgK] with water[C_{pc}=4170 J/kgK]. The oil enters at $T_{h,in}$=100 C and m_h=2 kg/s, while the water enters at $T_{c,in}$=0.48 kg/s. The overall heat transfer coefficient is U_m =400 W/(m^2K). Calculate the exit temperature of water and the total heat transfer.

A		12.5	m^2
U		400	W/m^2K
Tcin		20	C
Thin		100	C
Cpc		4170	J/(kgK)
Cph		2000	J/(kgK)
Ch	mh*Cph	4000	Cmax
Cc	mc*Cpc	2001.6	Cmin
C=Cmin/Cmax		0.5004	
N	AU/Cmin	2.498002	
ε		0.832516	
Tc,out		86.60129	
Q	CminΔT	133309.2	W

PROBLEM: A cross flow heat exchanger of heat transfer area A=12.5 $m^2$is to cool oil [Cph=2000 J/kgK] with water[C_{pc}=4170 J/kgK]. The oil enters at $T_{h,in}$=100 C and m_h=2 kg/s, while the water enters at $T_{c,in}$=0.48 kg/s. The overall heat transfer coefficient is U_m =400 W/(m^2K). Calculate the exit temperature of water and the total heat transfer.

A		12.5	m^2
U		400	W/m^2K
Tcin		20	C
Thin		100	C
Cpc		4170	J/(kgK)

Cph		2000	J/(kgK)
Ch	mh*Cph	4000	Cmax
Cc	mc*Cpc	2001.6	Cmin
C=Cmin/Cmax		0.5004	
N	AU/Cmin	2.498002	

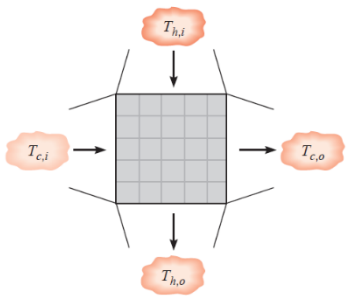
```
import java.util.*;
import java.lang.*;
import java.io.*;
import javax.swing.*;

class eps_NTU{
                public static double P(int n,double y)
                {long jf=1;
                double pow=pow(y,n);
                double total=0;
                for(int j=1;j<=n;j++)
                {jf*=j;pow*=y;total+=(n+1-j)/jf*pow;}
                 return total/fact(n+1);
                }
                public static long fact(int n)
                {long f=1;
                 for(int i=1;i<=n;i++) {f*=i;}
                 return f;
                }
                public static double pow(double y,int n)
                {double pow=1.0;
                 for(int i=1;i<=n;i++) {pow*=y;}
                 return pow;
                }
                public static double eps_cross_both_unmixed(double N,double C)
                {double total=0;
                 double powC=1;
                 for(int i=1;i<=40;i++)
                 {powC*=C;total+=powC*P(i,N);}
                 double eps=1-Math.exp(-N)-Math.exp(-(1+C)*N)*total;
                 return eps;
                }
                public static double eps_cross_both_unmixed1(double N,double C)
                {double eps=1-Math.exp(1.0/C*Math.pow(N,0.22)*(Math.exp(-C*Math.pow(N,0.78))-1.0));
                 return eps;
                }
    public static void main (String args[]) throws IOException
    { double C=0.5004;
     double N=2.498002;
     double eps1=eps_cross_both_unmixed(N,C);
     double eps2=eps_cross_both_unmixed1(N,C);
     System.out.println("eps1="+eps1+"eps2="+eps2);
    }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" eps_NTU
eps1=0.8130722886886853eps2=0.7908347275793921

> Terminated with exit code 0.
```

PROBLEM



Hot exhaust gases, which enter a finned-tube, cross-flow heat exchanger at 300 C and leave at 100 C, are used to heat pressurized water at a flow rate of 1 kg/s from 35 to 125 C. The overall heat transfer coefficieant based on the gas-side surface area is U_h=100 W/m^{2}K. Determine the required surface area A_h
Note: for this case reverse equation(NTU) is undefined therefore

bisection root finding is used to calculate it.

```
 import java.util.*;
import java.lang.*;
import java.io.*;
import javax.swing.*;

class eps_NTU{
                public static double P(int n,double y)
                {long jf=1;
                double pow=pow(y,n);
                double total=0;
                for(int j=1;j<=n;j++)
                {jf*=j;pow*=y;total+=(n+1-j)/jf*pow;}
                 return total/fact(n+1);
                }
                public static long fact(int n)
                {long f=1;
                 for(int i=1;i<=n;i++) {f*=i;}
                 return f;
                }
                public static double pow(double y,int n)
                {double pow=1.0;
                 for(int i=1;i<=n;i++) {pow*=y;}
                 return pow;
                }
                public static double eps_cross_both_unmixed(double N,double C)
                {double total=0;
                 double powC=1;
                 for(int i=1;i<=40;i++)
                 {powC*=C;total+=powC*P(i,N);}
                 double eps=1-Math.exp(-N)-Math.exp(-(1+C)*N)*total;
                 return eps;
                }
  public static double NTU(double C,double eps)
  {double a=0.01;
   double b=10;
   double r=(a+b)/2.0;
   double epsilon=1.0e-8;
   int nmax=100;
   int i=0;
   double fa=0,fr=eps_cross_both_unmixed(r,C)-eps;
   while((Math.abs(fr))>epsilon && i<nmax)
   { fa=eps_cross_both_unmixed(a,C)-eps;
     fr=eps_cross_both_unmixed(r,C)-eps;
                  if(fa*fr<0) b=r;
    else a=r;
    r=(a+b)/2.0;
    i++;
   }
   return r;
  }
                public static double eps_cross_both_unmixed1(double N,double C)
                {double eps=1-Math.exp(1.0/C*Math.pow(N,0.22)*(Math.exp(-C*Math.pow(N,0.78))-1.0));
                 return eps;
                }
    public static void main (String args[]) throws IOException
    { double m=1; //kg/s
      double Tc0=125.0;//degree C
      double Tc1=35.0;//degree C
      double Tc=(Tc0+Tc1)/2;//degree C
      double Pc=300.0; //kPa
      double U=100; //W/m^2K
      System.out.println("Tc average = "+Tc);
      steamIAPWS_IF97 st=new steamIAPWS_IF97();
      double a[]=st.propertyC("tp",Tc,Pc);
      double Cpc=a[7]*1e3;  //J/kgK
      System.out.println("Cpc = "+Cpc+" J/kgK");
      double Cc=m*Cpc;
      double Th0=300.0; //degree C;
      double Th1=100.0; //degree C
      double q=Cc*(Tc0-Tc1);
      System.out.println("q="+q+" Watt");
      double Ch=q/(Th0-Th1);
```

```
        double C=0,Cmin=0;
        if(Ch<Cc) {C=Ch/Cc;Cmin=Ch;}
        else      {C=Cc/Ch;Cmin=Cc;}
        System.out.println("C="+C+"Cc="+Cc+"Ch="+Ch+"Cmin="+Cmin);
        double qmax=Cmin*(Th0-Tc1);
        System.out.println("qmax="+qmax+" Watt");
        double eps=q/qmax;
        System.out.println("eps="+eps);
        double NTU=NTU(C,eps);
        System.out.println("NTU="+NTU);
        double A=NTU*Ch/U;
        System.out.println("A="+A+" m^2");
        double eps1=eps_cross_both_unmixed(NTU,C);
        System.out.println("eps1="+eps1);
      }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" eps_NTU
Tc average = 80.0
Cpc = 4195.079645877253 J/kgK
q=377557.16812895273 Watt
C=0.45Cc=4195.079645877253Ch=1887.7858406447638Cmin=1887.7858406447638
qmax=500263.2477708624 Watt
eps=0.7547169811320754
NTU=1.9328542635589838
A=36.48814910776512 m^2
eps1=0.7547169856306125

> Terminated with exit code 0.
```

PROBLEM: Saturated steam at 1 atm condenses on the outer surface of the tubes of a shell-and-tube heat exchanger (1 shell, 2-tube passes). There are total 206 tubes with a diameter of 14 mm. Cooling water enters the tubes at 15°C with a mean velocity of U_m=3.5 m/s. The convective heat transfer coefficient on the outer surface of the tubes is h_o= 21800W/m^{2}K and the condensation rate is m_h=2.3 kg/s. (**hint:** you can assume T_{co} between 20°C-40°C)

a) What is the heat transfer coefficient for the cooling water, h_i?

b) Find the overall heat transfer coefficient, U ?

c) What is the required heat exchanger surface area, A? and the required tube length, L?

d) Find the outlet water temperature, T_{co}?

e) When $q=q_{max}$ maximum condensation rate occurs. Find the value of $m_{h,\,max}$?

```
/* steam property vector
  a[0]="P, pressure       ";
  a[1]="T, temperature    ";
  a[2]="v, specific volume ";
  a[3]="h, enthalpy       ";
  a[4]="u, internal energy ";
  a[5]="s, entropy        ";
  a[6]="x, quality        ";
  a[7]="Cp, specific heat at constant pressure ";
  a[8]="Cv, specific heat at constant volume ";
  a[9]="isobaric thermal expansion coefficient ";
  a[10]="" isothermal compressibility" ;
  a[11]=" Dynamic viscosity" ;
```

```
   a[12]="k Thermal conductivity" ;
   a[13]=" Surface tension" ;
   a[14]="Prandtl number" ;
   a[15]=" density" ;
   a[16]=" speed of sound" ;
*/
public class HT_2023_Q2
{
public static double f(double Re,double eod)
 {
 //Goudar equation
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
public static double Nu(double Re,double Pr,double f)
{double xx=(f/8.0)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8.0)*(Math.pow(Pr,(2.0/3.0))-1));
   return xx;
}
public static double Nu1(double Re,double Pr,double f)
{double xx=0.023*Math.pow(Re,(4.0/5.0))*Math.pow(Pr,0.4);
   return xx;
}

public static void main(String arg[])
{ steamIAPWS_IF97 water=new steamIAPWS_IF97();
  double Tci=15.0;//Degree C
  double Tco=30.0;//Degree C
  double Vm=3.5; //m/s
  double D=14e-3;//m pipe diameter
  double Th=100.0;//degree C
  double ho=21800;// W/m^2K
  double a1[]=water.propertyC("tx",Th,0.0);
  double hl=a1[3];
   double a2[]=water.propertyC("tx",Th,1.0);
  double hv=a2[3];
  double hlv=hv-hl; //kJ/kg
  double mh=2.3; //kg/s condensation rate
  double Q=mh*hlv*1e3;
  System.out.println("hlv="+hlv+"kj/kg \nQ="+Q+" W");
  double Tcm=(Tci+Tco)/2.0;
  double Pc=101.325; //kPa
  double a3[]=water.propertyC("tp",Tcm,Pc);
```

```
    double ro=a3[15];
  double Cp=a3[7];
  double mu=a3[11];
  double k=a3[12];
  double Pr=a3[14];
   System.out.println("Tcm="+Tcm+"degree
C\nro="+ro+"kg/m^3\nCp="+Cp+"kJ/kgK\nmu="+mu+"Pas\nk="+k+"W/mK\nPr="+Pr);
   double Re=ro*Vm*D/mu;
   System.out.println("Re="+Re);
   double eps=46e-6;//m commercial steel
    double eod=eps/D;
  double f=f(Re,eod);
  double Nu=Nu(Re,Pr,f);
  double hi=Nu*k/D;
  System.out.println("f="+f+"\nNu="+Nu+"\nhi="+hi+" W/m2K");
  double Nu1=Nu1(Re,Pr,f); //assumed smooth tube
   double hi1=Nu1*k/D;
   System.out.println("Nu1="+Nu1+"\nhi1="+hi1+" W/m2K");
   double U=1.0/(1.0/hi+1.0/ho);
   double U1=1.0/(1.0/hi1+1.0/ho);
   System.out.println("U="+U+" U1="+U1+" W/m^2K");
   double A=Math.PI*D*D/4.0;
   double mc1=ro*Vm*A;
   double N=206;
   double mcN=mc1*N;
   System.out.println("A="+A+"m^2\nmc1="+mc1+"kg/s\nmcN="+mcN+"kg/s");
   double Cc=mcN*Cp*1e3;
   System.out.println("Cc="+Cc);
   double Cmin=Cc;
   double Qmax=Cmin*(Th-Tci);
   System.out.println("Qmax) = "+Qmax+" W");
   double epsilon=Q/Qmax;
   System.out.println("epsilon = "+epsilon);
   double NTU=-Math.log(1-epsilon);
   System.out.println("NTU="+NTU);
   double Ap=NTU*Cmin/U;
   double Ap1=NTU*Cmin/U1;
   System.out.println("Ap = "+Ap+"Ap1="+Ap1);
   double L=Ap/(Math.PI*N*D);
   double L1=Ap1/(Math.PI*N*D);
   System.out.println("L = "+L+"L1 = "+L1+" m two pass total length");
    System.out.println("L/2 = "+L/2+" L1/2 = "+L1/2+" m  one pass length");
    Tco=Q/Cc+Tci;
    System.out.println("Tco="+Tco+" degree C");
    double mhmax=Qmax/(hlv*1e3);
    System.out.println("mhmax="+mhmax+" kg/s");
}
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan
Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-
0933\jre\bin\javaw.exe" HT_2023_Q2
```

```
hlv=2256.472874223131kj/kg
Q=5189887.610713201 W
Tcm=22.5degree C
ro=997.6582439143023kg/m^3
Cp=4.183198469552712kJ/kgK
mu=9.431552089290104E-4Pas
k=0.6035822527735999W/mK
Pr=6.5366491615889215
Re=51831.611052980275
f=0.029098963907471086
Nu=405.32136786793694
hi=17474.627451071883 W/m2K
Nu1=288.10160171492606
hi1=12420.929556484107 W/m2K
U=9699.566950901028 U1=7912.592318230801 W/m^2K
A=1.5393804002589986E-4m^2
mc1=0.5375214413934709kg/s
mcN=110.72941692705501kg/s
Cc=463203.1274237207
Qmax) = 3.9372265831016265E7 W
epsilon = 0.13181582266532313
NTU=0.14135140096058116
Ap = 6.750240636731041Ap1=8.27471053194677
L = 0.7450306271813649L1 = 0.9132878528529094 m two pass total length
L/2 = 0.37251531359068246 L1/2 = 0.4566439264264547 m  one pass length
Tco=26.204344926552466 degree C
mhmax=17.448588139829305 kg/s

> Terminated with exit code 0.
```

Hand solution:

```
P=101.325 kPa
```

select known property pair :	tx	tx
Temperature	100	degree K
Quality	0	kg vapor/kg mix.

ref no	Property	Value	Units
0	P, pressure	101.41797792130998	kPA
1	T, temperature	373.15	deg K
2	v, specific volume	0.001043455456610527	m³/kg
3	h, enthalpy	419.09915499770307	KJ/kg
4	u, internal energy	418.9933298552427	KJ/kg
5	s, entropy	1.3070143278413395	KJ/kgK
6	x, quality	0.0	kg vap/kg mix
7	Cp, specific heat at co...	4.2166451189235845	KJ/kgK
8	Cv, specific heat at co...	3.7677002201427507	KJ/kgK
9	α isobaric thermal exp...	7.510100333728113E-4	1/K
10	κ isothermal compres...	4.89165475398767E-7	1/Pa
11	η Dynamic viscosity	2.8158501936566716E-4	Pa.s
12	k Thermal conductivity	0.677757511374594	W/mK
13	σ Surface tension	0.058911868587664076	N/m
14	Prandtl number	1.751871543352034	
15	ρ density	958.3542772858901	kg/m³
16	speed of sound	1545.0922491938707	m/s

select known property pair :	tx	tx
Temperature	100	degree K
Quality	1	kg vapor/kg mix.

ref no	Property	Value	Units
0	P, pressure	101.41797792130998	kPA
1	T, temperature	373.15	deg K
2	v, specific volume	1.6718606010940391	m³/kg
3	h, enthalpy	2675.572029220834	KJ/kg
4	u, internal energy	2506.015307691571	KJ/kg
5	s, entropy	7.354077050958405	KJ/kgK
6	x, quality	1.0	kg vap/kg mix
7	Cp, specific heat at co...	2.077491868482266	KJ/kgK
8	Cv, specific heat at co...	1.5536986960345107	KJ/kgK
9	α isobaric thermal exp...	0.0027023117539003...	1/K
10	κ isothermal compres...	0.010023461333813512	1/Pa
11	η Dynamic viscosity	1.2232158121714269E-5	Pa.s
12	k Thermal conductivity	0.02479378675595491	W/mK
13	σ Surface tension	0.058911868587664076	N/m
14	Prandtl number	1.024942631070555	
15	ρ density	0.5981359925257022	kg/m³
16	speed of sound	471.8157238382752	m/s

T=100 °C x=0 h_l= 419.09915499770307 kJ/kg

T=100 °C x=1 h_v= 2675.572029220834 kJ/kg

$h_{fg} = h_v - h_l$ =2675.572029220834-419.09915499770307=2256.472874223 kJ/kg

$$\dot{m}_h = 2.3\ kg/s$$

$Q = \dot{m}_h h_{fg} = 2.3x2675.572029220834$=5189.887610713 kW

Cold water properties at P=101.325 kPa $T_{ci} = 15$ °C

Assume that $T_{co} = 30$ °C

$$T_c = \frac{T_{ci} + T_{co}}{2} = \frac{15 + 30}{2} = 22.5\ ℃ = 295.65\ K$$

select known property pair :	tp	tp
Temperature	22.5	degree C
Pressure	101.325	kPa

ref no	Property	Value	Units
0	P, pressure	101.325	kPA
1	T, temperature	295.65	deg K
2	v, specific volume	0.001002347252779178	m³/kg
3	h, enthalpy	94.47298275865998	KJ/kg
4	u, internal energy	94.37141992327211	KJ/kg
5	s, entropy	0.3320125048621994	KJ/kgK
6	x, quality	-1.0	kg vap/kg mix
7	Cp, specific heat at co...	4.183198469552712	KJ/kgK
8	Cv, specific heat at co...	4.148007697561816	KJ/kgK
9	α isobaric thermal exp...	2.323640394007918E-4	1/K
10	κ isothermal compres...	4.5467924354136274E-7	1/Pa
11	η Dynamic viscosity	9.431552089290104E-4	Pa.s
12	k Thermal conductivity	0.6035822527735999	W/mK
13	σ Surface tension	0.07235594732800321	N/m
14	Prandtl number	6.5366491615889215	
15	ρ density	997.6582439143023	kg/m³
16	speed of sound	1491.0458419044974	m/s

$$V_\infty = 3.5\ m/s$$
$$D = 14x10^{-3}\ m$$

$A = \frac{\pi D^2}{4} = \frac{\pi 14x10^{-3^2}}{4}$ =1.539380400x$10^{-4}\ m^2$

$$Re = \frac{\rho VD}{\mu} = \frac{997.6582439143023x3.5x14x10^{-3}}{9.431552089290104E-4} = 51831.611$$

$\varepsilon = 46x10^{-6}\ m$ (commersial steel)

$$\frac{\varepsilon}{D} = \frac{46x10^{-6}}{14x10^{-3}} = 0.003285714$$

f=0.029098963907471086

Gnielinski[33] equation

$$Nu = \frac{\left(\frac{f}{8}\right)(Re-1000)Pr}{1.07+12.7\left(\frac{f}{8}\right)^{.5}\left(Pr^{\frac{2}{3}}-1\right)} \quad 0.5 \le Pr \le 2000 \quad 2300 \le Re \le 510^6$$

Nu=405.32136786793694

hi=17474.627451071883 W/m^2K

Alternatively

Dittus-Boelter equation[24] (This equation assumes smooth tubes)

$Nu = 0.023Re^{4/5}Pr^n$ where n=0.4 for heating and n=0.3 for cooling $0.7 \le Pr \le 160$

Nu1=288.10160171492606

hi1=12420.929556484107 W/m^{2}K

$$\frac{1}{U} = \frac{1}{hi} + \frac{1}{ho}$$

U=9699.566950901028 W/m^{2}K (Gnilenski eqn)

U1=7912.592318230801 W/m^2K (Dittus-Boelter eqn)

$$A = \frac{\pi D^2}{4} = 1.5393804002589986x10^{-4}\ m^2$$

$$m_{c1} = \rho V_c A = 997.658x3.5 * 1.5393804002589986x10^{-4} = 0.5375214413934709\ m/s$$

$$m_{cN} = N\rho V_c A = Nm_{c1} = 110.72941692705501\ kg/s$$
$$C_c = m_{cN} C_p = 110.72941692705501x.4.183198469552712x10^3 = 463203.1274237207$$
$$C_{min} = C_c$$
$$Q_{max} = C_{min}(T_h - T_{ci}) = 463203.1274237207(100 - 15) = 3.9372265831016265x10^7\ W$$

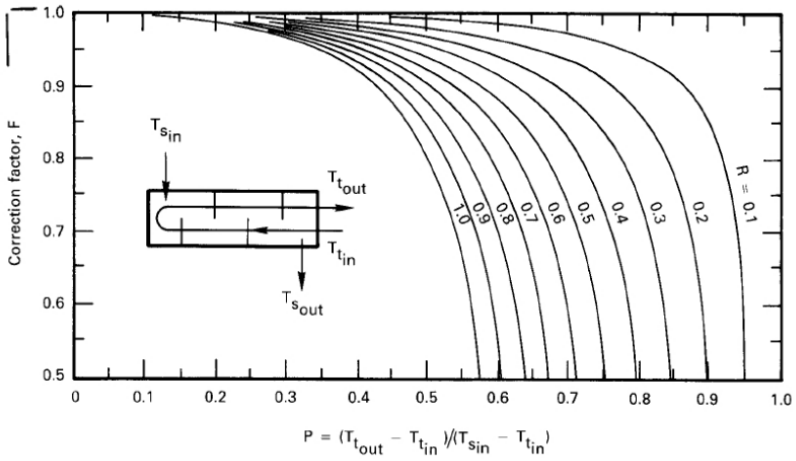
$$\varepsilon = \frac{Q}{Q_{max}} = \frac{5189.887610713x10^3}{3.9372265831016265x10^7} = 0.13181582266532313$$

$$NTU = ln(1 - \varepsilon) = ln(1 - 0.13181582266532313) = 0.14135140096058116$$
$$NTU = \frac{A_p U}{C_{min}}$$
$$A_p = \frac{NTU\ C_{min}}{U} = N\pi DL$$
$$A_p = 6.750240636731041\ (Gnilenski)$$
$A_{p1} = 8.27471053194677$ (Dittus-Boelter eqn)
Pipe length
L = 0.7450306271813649L1 = 0.9132878528529094 m two pass total length
L/2 = 0.37251531359068246 L1/2 = 0.4566439264264547 m one pass length
$$Q = C_c(T_{co} - T_{ci})$$
$$T_{co} = \frac{Q}{C_c} + T_{ci} = \frac{5189.887610713x10^3}{463203.1274237207} + 15 = 26.204344926552466\ ℃$$
$$m_{hmax} = \frac{Q_{max}}{h_{fg}} = \frac{3.9372265831016265x10^7}{2256.472874223x10^3} = 17.448588139829305\ kg/s$$

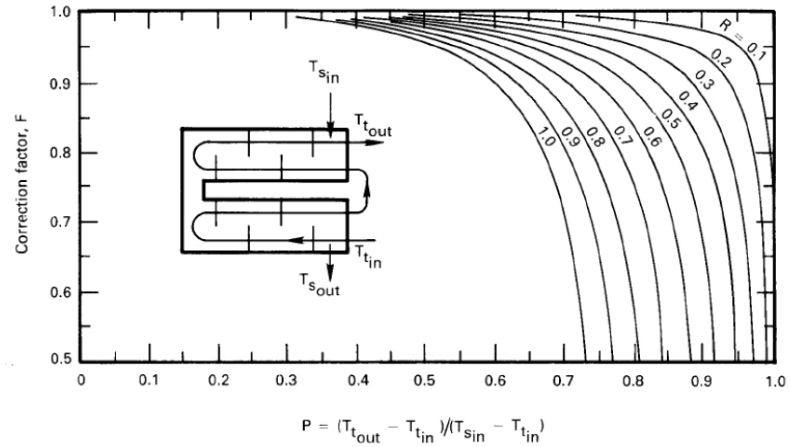
F CORRECTION FACTOR FOR LMTD METHOD

In previous section logarithmic mean temperature difference approach was given as:
$Q = U_m A_t \frac{\Delta T_0 - \Delta T_L}{ln\left(\frac{\Delta T_0}{\Delta T_L}\right)} = U_m A_t \Delta T_{ln}$
For different heat exchanger configurations a correction factor can be employed. **ε-NTU** method can be used to calculate this correction factor.
$Q = FU_m A_t \frac{\Delta T_0 - \Delta T_L}{ln\left(\frac{\Delta T_0}{\Delta T_L}\right)} = FU_m A_t \Delta T_{ln}$
$F = \frac{NTU_{counter\ flow}}{NTU}$

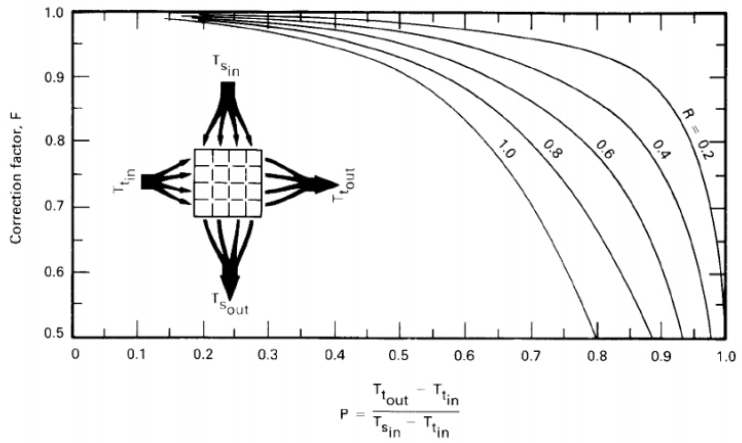
Some of the correction factors are also given below as graphic format:



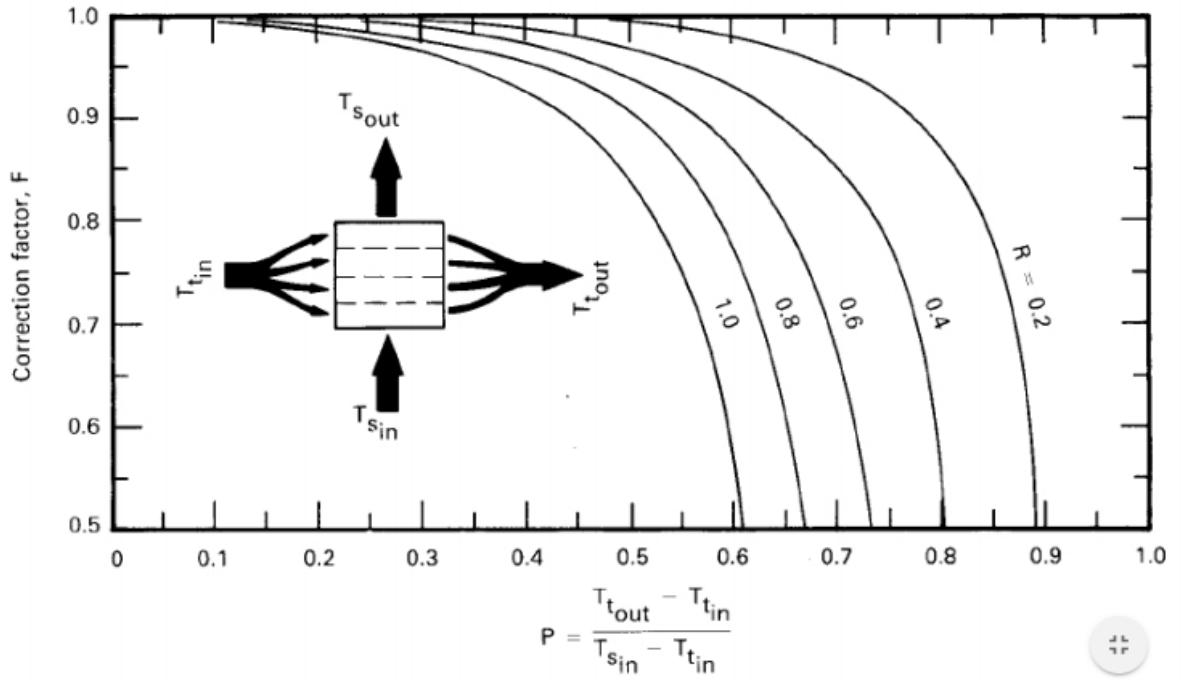
One Shell pass two tube pass



Two pass & two tube pass Shell and tube heat exchanger



Cross flow both fluid unmixed



Cross flow one fluid unmixed one fluid mixed

PROBLEM: A cross flow heat exchanger is to be designed to cool m_h=1 kg/s water from T_{1h}=75 C to T_{2h}=50 C with cooling water entering the tube side at T_{1c}=20 C at a rate m_c=1 kg/s. The overall heat transfer coefficient based on the outer surface is U_o=500 W/m^{2}K. Calculate the heat transfer area by using LMTD method. Pressure is P_h=300kPa and P_c=300kPa

```
 import java.util.*;
import java.lang.*;
import java.io.*;
import javax.swing.*;

class HT_LMTD_F_cross{
                        public static double P(int n,double y)
                {long jf=1;
                double pow=pow(y,n);
                double total=0;
                for(int j=1;j<=n;j++)
                {jf*=j;pow*=y;total+=(n+1-j)/jf*pow;}
                 return total/fact(n+1);
                }
                public static long fact(int n)
                {long f=1;
                 for(int i=1;i<=n;i++) {f*=i;}
                 return f;
                }
                public static double pow(double y,int n)
                {double pow=1.0;
                 for(int i=1;i<=n;i++) {pow*=y;}
                 return pow;
                }
                public static double eps_cross_both_unmixed(double N,double C)
                {double total=0;
                 double powC=1;
                 for(int i=1;i<=40;i++)
                 {powC*=C;total+=powC*P(i,N);}
                 double eps=1-Math.exp(-N)-Math.exp(-(1+C)*N)*total;
                 return eps;
                }
  public static double NTU(double C,double eps) //cross flow both unmixed
  {double a=0.01;
   double b=10;
   double r=(a+b)/2.0;
   double epsilon=1.0e-8;
   int nmax=100;
```

```
  int i=0;
  double fa=0,fr=eps_cross_both_unmixed(r,C)-eps;
  while((Math.abs(fr))>epsilon && i<nmax)
  { fa=eps_cross_both_unmixed(a,C)-eps;
    fr=eps_cross_both_unmixed(r,C)-eps;
                if(fa*fr<0) b=r;
   else a=r;
   r=(a+b)/2.0;
   i++;
  }
  return r;
 }
 public static double NTU_counter(double C,double eps)
 { return 1.0/(C-1.0)*Math.log((eps-1.0)/(eps*C-1.0));

 }
 public static double F(double C,double eps)
 {return NTU_counter(C,eps)/NTU(C,eps);}

   public static void main (String args[]) throws IOException
   { double mc=1; //kg/s
     double mh=1;  //kg/s
     double Tc1=20.0;//degree C
     double Tc2;//degree C
     double Th1=75.0;//degree C
     double Th2=50;//degree C
     double Pc=300.0; //kPa
     double Ph=300.0; //kPa
     double U=500; //W/m^2K
     double Th=(Th1+Th2)/2.0;
     System.out.println("Th average = "+Th);
     steamIAPWS_IF97 st=new steamIAPWS_IF97();
     double a[]=st.propertyC("tp",Th,Ph);
     double Cph=a[7]*1e3;  //J/kgK
     double Q=mh*Cph*(Th1-Th2);
     //Assuming Cph=Cpc
     Tc2=Tc1+Q/(mc*Cph);
     double Tc=(Tc1+Tc2)/2.0;
     a=st.propertyC("tp",Tc,Pc);
     double Cpc=a[7]*1e3;  //J/kgK
     Tc2=Tc1+Q/(mc*Cpc);
     System.out.println("Cph = "+Cph+" J/kgK");
     System.out.println("Cpc = "+Cpc+" J/kgK");
     double Cc=mc*Cpc;
     double Ch=mh*Cph;
     double C=0,Cmin=0;
     if(Ch<Cc) {C=Ch/Cc;Cmin=Ch;}
     else      {C=Cc/Ch;Cmin=Cc;}
     System.out.println("C="+C+"Cc="+Cc+"Ch="+Ch+"Cmin="+Cmin);
     double Qmax=Cmin*(Th1-Tc1);
     System.out.println("Qmax="+Qmax+" Watt");
     double eps=Q/Qmax;
     System.out.println("eps="+eps);
     double NTU=NTU(C,eps);
     System.out.println("NTU="+NTU);
     double NTU_counter=NTU_counter(C,eps);
     System.out.println("NTU_counter="+NTU_counter);
     double F=F(C,eps);
     System.out.println("F="+F);
     double dT1=Th1-Tc2;
     double dT2=Th2-Tc1;
     double dTln=F*(dT1-dT2)/Math.log(dT1/dT2);
     System.out.println("dTln="+dTln);
     double A=Q/(U*dTln);
     System.out.println("A="+A+" m^2");
    }
}
```

```
 ---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_LMTD_F_cross
Th average = 62.5
Cph = 4183.459304365295 J/kgK
Cpc = 4178.868474471499 J/kgK
C=0.9989026235084909Cc=4178.868474471499Ch=4183.459304365295Cmin=4178.868474471499
Qmax=229837.76609593246 Watt
eps=0.4550448100225566
```

```
NTU=0.86796401463449
NTU_counter=0.8346309227567256
F=0.9615962282816513
dTln=28.83467992826911
A=7.254214915463464 m^2

> Terminated with exit code 0.
```

PROBLEM: A One shell two tube pass shell & tube heat exchanger is to be designed to cool m_h=10 kg/s water from T_{1h}=75 C to T_{2h}=50 C with cooling water entering the tube side at T_{1c}=20 C at a rate m_c=10 kg/s. The overall heat transfer coefficient based on the outer surface is U_o=500 W/m^2K. Calculate the heat transfer area by using LMTD method. Pressure is P_h=300kPa and P_c=300kPa

```
import java.util.*;
import java.lang.*;
import java.io.*;
import javax.swing.*;

class HT_LMTD_F_shell_and_tube{
  public static double NTU(double C,double eps) //Shell & tube
  {double E=(2.0/eps-(1.0+C))/Math.sqrt(1.0+C*C);
   double N=-1.0/Math.sqrt(1.0+C*C)*Math.log((E-1.0)/(E+1.0));
   return N;
  }
  public static double NTU_counter(double C,double eps)
  { return 1.0/(C-1.0)*Math.log((eps-1.0)/(eps*C-1.0));

  }
  public static double F(double C,double eps)
  {return NTU_counter(C,eps)/NTU(C,eps);}

    public static void main (String args[]) throws IOException
    { double mc=1; //kg/s
      double mh=1;  //kg/s
      double Tc1=20.0;//degree C
      double Tc2;//degree C
      double Th1=75.0;//degree C
      double Th2=50;//degree C
      double Pc=300.0; //kPa
      double Ph=300.0; //kPa
      double U=500; //W/m^2K
      double Th=(Th1+Th2)/2.0;
      System.out.println("Th average = "+Th);
      steamIAPWS_IF97 st=new steamIAPWS_IF97();
      double a[]=st.propertyC("tp",Th,Ph);
      double Cph=a[7]*1e3;  //J/kgK
      double Q=mh*Cph*(Th1-Th2);
      //Assuming Cph=Cpc
      Tc2=Tc1+Q/(mc*Cph);
      double Tc=(Tc1+Tc2)/2.0;
      a=st.propertyC("tp",Tc,Pc);
      double Cpc=a[7]*1e3;  //J/kgK
      Tc2=Tc1+Q/(mc*Cpc);
      System.out.println("Cph = "+Cph+" J/kgK");
      System.out.println("Cpc = "+Cpc+" J/kgK");
      double Cc=mc*Cpc;
      double Ch=mh*Cph;
      double C=0,Cmin=0;
      if(Ch<Cc) {C=Ch/Cc;Cmin=Ch;}
      else      {C=Cc/Ch;Cmin=Cc;}
      System.out.println("C="+C+"Cc="+Cc+"Ch="+Ch+"Cmin="+Cmin);
      double Qmax=Cmin*(Th1-Tc1);
      System.out.println("Qmax="+Qmax+" Watt");
      double eps=Q/Qmax;
      System.out.println("eps="+eps);
      double NTU=NTU(C,eps);
      System.out.println("NTU="+NTU);
      double NTU_counter=NTU_counter(C,eps);
      System.out.println("NTU_counter="+NTU_counter);
      double F=F(C,eps);
      System.out.println("F="+F);
      double dT1=Th1-Tc2;
      double dT2=Th2-Tc1;
      double dTln=F*(dT1-dT2)/Math.log(dT1/dT2);
```

```
        System.out.println("dTln="+dTln);
        double A=Q/(U*dTln);
        System.out.println("A="+A+" m^2");
    }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_LMTD_F_shell_and_tube
Th average = 62.5
Cph = 4183.459304365295 J/kgK
Cpc = 4178.868474471499 J/kgK
C=0.9989026235084909Cc=4178.868474471499Ch=4183.459304365295Cmin=4178.868474471499
Qmax=229837.76609593246 Watt
eps=0.4550448100225566
NTU=0.9585637148222919
NTU_counter=0.8346309227567256
F=0.870709906760301
dTln=26.109338549165557
A=8.011423377286203 m^2

> Terminated with exit code 0.
```

16.5 FINITE DIFFERENCE APPROACH

As a simple first example let us consider a parallel flow double pipe heat exchanger. Assume that heat transfer area for a final difference step is ΔA_i and temperature profiles for this finite element piece is T_{hi} and T_{ci}. If heat transfer of finite difference piece is defined as a function of physical properties:
$U_i = U_i(T_{hi}, T_{ci}, m_h, m_c, D_h, D_c, \varepsilon_h, \varepsilon_c)$ where $m_h \; and \; m_c$ are mass flow rates of fluids, D_h and D_c are dimeter or hydrolic diameter of pipes, $\varepsilon_h, \varepsilon_c$ are surface roughness. From conservation of energy for surface ΔA_i heat transfer will be

$$\Delta Q_i = -\left(m_h C_{ph}\right)_i [T_{hi+1} - T_{hi}] = \left(m_c C_{pc}\right)_i [T_{ci+1} - T_{ci}]$$

$$\Delta Q_i = U_i(T_{hi}, T_{ci}, m_h, m_c, A_h, A)\Delta A_i [T_{hi} - T_{ci}]$$

Note that U is changing in every step and is not constant throughout.
For parallel flow heat exchanger starting from i=0 (entering temperatures of T_{h0} and T_{c0} will be known) ΔQ_i is calculated from heat transfer equation. In order to do that properties are calculated from the given temperatures and flow conditions and remain constant only for this single calculation step. Related heat transfer and overall heat transfer equations are taken according to given geometry and conditions. Once ΔQ_i is known for the finite difference step T_{hi+1} and T_{ci+1} is calculated from the conservation of energy equations.

$$T_{hi+1} = T_{hi} - \frac{\Delta Q_i}{\left(m_h C_{ph}\right)_i} \quad \text{and} \quad T_{ci+1} = T_{ci} + \frac{\Delta Q_i}{\left(m_c C_{pc}\right)_i}$$

Pressure drop is also an important parameter for heat exchangers, therefore it should also be calculated in each step

$$\Delta P_{hi} = f_{hi}\rho_{hi}\frac{\Delta x_i}{D}\frac{U_{hi}^2}{2} \quad \Delta P_{ci} = f_{ci}\rho_{ci}\frac{\Delta x_i}{D}\frac{U_{ci}^2}{2}$$

Calculation continues stepwise to the end of heat exchanger. So total heat transfer can be obtained as:

$$Q = \sum_{i=0}^{N} \Delta Q_i$$

$$P_h = \sum_{i=0}^{N} \Delta P_{hi} \quad P_c = \sum_{i=0}^{N} \Delta P_{ci}$$

In counter heat exchanger profile usually inlet temperatures of two sides are given, but this temperatures exists in the opposite directions of the heat exchanger in order to solve this problem shooting method is used. A guest value is assigned to the unknown initil value, problem is solved and final value is observed, then this procesure is repeated and enough values are obtained initial and final value relationship, then actual value corresponding to the given final value is solved by using curve fitting or root finding methods.

Due to repeated solution requirements, this method is involved more computer time. In counterflow case both pressure and temperature changes and , the problem is not and initial value problem, but boundary value problem. Therefore, a system of equation for temperature and pressure should be solve to obtain initial temperature and pressure values. Nelder-Mead optimisation method is used in this example to solve non-linear system of equation.

Double pipe heat exchanger algorithm with pressure drop in each side, parallel and counter flow cases both investigated

double_pipe_HE_with_PressureD

```
 //This class adapting nonlinear system of equation
// function to optimisation problem function
class f2 implements if_xj
{//adaptation function
ifi_xj ff1;
public f2(ifi_xj ffi) {ff1=ffi;}

public double func(double x[])
{
double g=0.0;
double fa[]=ff1.func(x);
for(int i=0;i<fa.length;i++)  g+=fa[i]*fa[i];
return g;
}
}
// Double pipe heat exchanger with pressure drop in each
// finite difference step
public class double_pipe_HE_with_PressureD
{  public double din,dout; //iç ve dış çap
   public double mh,mc; //sıcak ve soğuk sıvı iç sıcak dış soğuk
   public double L;
   public double dL;
   public int n;
   // both side of the heat exchanger utilises water as warking fluid
   steamIAPWS_IF97 water;  //water and steam properties
   double Th[];
   double Tc[];
   double Ph[];
   double Pc[];
   double x[];
   double Ah;
   double Ac;
   double dh;
   double eps;
   double dA;
   public double Q;
   public double dPh;
   public double dPc;
   public double_pipe_HE_with_PressureD(double dini,double douti,double mhi,double mci,double Li)
   {din=dini;dout=douti;mh=mhi;mc=mci;L=Li;n=100;dL=L/(n-1);
     water=new steamIAPWS_IF97();
     Th=new double[n];
      Tc=new double[n];
      Ph=new double[n];
      Pc=new double[n];
      x=new double[n];
      Ah=Math.PI*din*din/4.0;
      Ac=Math.PI*(dout*dout-din*din)/4.0;
      dh=dout-din;
      eps=1.2e-4; //m
      dA=dL*Math.PI*din;
     }
    // eps     : pipe equivalent roughness m
 //          ----------------------------------------
 //          value of equivalent roughness for some pipes:
 //          Pipe             equivalent rougness m
 //          -------------    --------------------
 //          Riveted steel    0.9e-3 - 9e-3
 //          Concrete         0.3e-3 - 3e-3
 //          Wood stave       0.18e-3 - 0.9 e-3
 //          Cast iron        0.26e-3
```

```
//          Galvanised iron  0.15e-3
//          Commercial steel 0.045e-3
//          wrought iron     0.045e-3
//          drawn tubing     0.0015e-3
//          Plastic,glass    0.0 (smooth)
//          ------------------------------------------
public double f(double Re,double eod)
{
//Goudar equation
double a=2/Math.log(10);
double b=eod/3.7;
double d=Math.log(10.0)/5.02*Re;
double s=b*d+Math.log(d);
double q=Math.pow(s,(s/(s+1)));
double g=b*d+Math.log(d/q);
double z=Math.log(q/g);
double dLA=(g/(g+1))*z;
double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
double f1=a*(Math.log(d/q)+dCFA);
f1=1.0/(f1*f1);
return f1;
}
 public static double f_Haaland(double Re,double eod)
 {double xx=-1.8*Math.log10(Math.pow((eod/3.7) ,1.11)+6.9/Re);
   double f=1.0/(xx*xx);
   return f;
  }
  public static double Nu(double Re,double Pr,double f)
  {double xx=(f/8.0)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8.0)*(Math.pow(Pr,(2.0/3.0))-1));
   return xx;
  }
  //Parallel flow
  public void calculate_parallel(double Thi,double Tci,double Phi,double Pci)
  { Th[0]=Thi;
     Tc[0]=Tci;
     Ph[0]=Phi;
     Pc[0]=Pci;
      x[0]=0.0;
     Q=0.0;
     dPc=0.0;
     dPh=0.0;
     for(int i=1;i<n;i++)
     { x[i]=dL*i;
                double a[]=water.propertyC("tp",Th[i-1],Ph[i-1]);
     double roh=1.0/a[2];
     double vish=a[11];
     double kh=a[12];
     double Cph=a[7];
      double Prh=a[14];//Cph*1e3*vish/kh;
     double b[]=water.propertyC("tp",Tc[i-1],Pc[i-1]);
     double roc=1.0/b[2];
     double visc=b[11];
     double kc=b[12];
      double Cpc=b[7];
      double Prc=b[14];//Cpc*1e3*visc/kc;
     double Vc=mc/(roc*Ac);  //cold site velocity m/s
     double Vh=mh/(roh*Ah); //hot side velocity m/s
     double Reh=roh*Vh*din/vish;
      double Rec=roc*Vc*dh/visc;
      double eodh=eps/din;
      double eodc=eps/dh;
      double fh=f(Reh,eodh);
      double fc=f(Rec,eodc);
      double Nuh=Nu(Reh,Prh,fh);
      double Nuc=Nu(Rec,Prc,fc);
      double hh=Nuh*kh/din;
       double hc=Nuc*kc/dh;
       double U=1.0/(1.0/hh+1.0/hc);
       double dQ=U*dA*(Th[i-1]-Tc[i-1]);
       double dPc1=fc*L/dh*roc*Vc*Vc/2.0;
       double dPh1=fh*L/din*roh*Vh*Vh/2.0;
       dPc+=dPc1*1e-3;
       dPh+=dPh1*1e-3;
       Pc[i]=Pc[i-1]-dPc1*1e-3;
       Ph[i]=Ph[i-1]-dPh1*1e-3;
       Q+=dQ;
       Th[i]=Th[i-1]-dQ/(mh*Cph*1e3);
```

```
      Tc[i]=Tc[i-1]+dQ/(mc*Cpc*1e3);
    }
  }
    //Counter flow Thi (hot side inlet) and Tco (cold site outlet) is given
  public double[] calculate1(double Thi,double Tco,double Phi,double Pco)
  { Th[0]=Thi;
    Tc[0]=Tco;
    Ph[0]=Phi;
    Pc[0]=Pco;
     x[0]=0.0;
    Q=0.0;
    for(int i=1;i<n;i++)
    { x[i]=dL*i;
                double a[]=water.propertyC("tp",Th[i-1],Ph[i-1]);
    double roh=1.0/a[2];
    double vish=a[11];
    double kh=a[12];
    double Cph=a[7];
     double Prh=a[14];//Cph*1e3*vish/kh;
    double b[]=water.propertyC("tp",Tc[i-1],Pc[i-1]);
    double roc=1.0/b[2];
    double visc=b[11];
    double kc=b[12];
     double Cpc=b[7];
     double Prc=b[14];//Cpc*1e3*visc/kc;
    double Vc=mc/(roc*Ac);  //cold site velocity m/s
    double Vh=mh/(roh*Ah); //hot side velocity m/s
    double Reh=roh*Vh*din/vish;
     double Rec=roc*Vc*dh/visc;
     double eodh=eps/din;
     double eodc=eps/dh;
     double fh=f(Reh,eodh);
     double fc=f_Haaland(Rec,eodc);
     double Nuh=Nu(Reh,Prh,fh);
     double Nuc=Nu(Rec,Prc,fc);
     double hh=Nuh*kh/din;
      double hc=Nuc*kc/dh;
      double U=1.0/(1.0/hh+1.0/hc);
      double dQ=U*dA*(Th[i-1]-Tc[i-1]);
      Q+=dQ;
      double dPc1=fc*L/dh*roc*Vc*Vc/2.0;
      double dPh1=fh*L/din*roh*Vh*Vh/2.0;
      dPc+=dPc1*1e-3;
      dPh+=dPh1*1e-3;
      Pc[i]=Pc[i-1]-dPc1*1e-3;
      Ph[i]=Ph[i-1]-dPh1*1e-3;
      Th[i]=Th[i-1]-dQ/(mh*Cph*1e3);
      Tc[i]=Tc[i-1]-dQ/(mc*Cpc*1e3);
    }
    double d[]={Tc[n-1],Pc[n-1]};
    return d;
  }
 //Counter flow Thi (hot side inlet) and Tci (cold site inlet) is given
 //bisection root finding is applied in limits of Tci and Thi to find Tco
  public void calculate_counter(double Thi,double Tci,double Phi,double Pci)
  { ifi_xj f1=(double x[])->{double a[]=calculate1(Thi,x[0],Phi,x[1]);
                                double ff[]=new double[2];
                                ff[0]=a[0]-Tci;
                                ff[1]=a[1]-Pci;
                               return ff;};
   f2 f=new f2(f1);
    double [] x0={(Thi+Tci)/2.0,Pci};
    double [] dx0=new double[x0.length];
    for(int i=0;i<x0.length;i++) dx0[i]=0.2*x0[i];
    double a[]=nelderi.nelder(f,x0,dx0);
    double Tco=a[0];
    double Pco=a[1];
    calculate1(Thi,Tco,Phi,Pci);
  }
  public static void main(String arg[])
  {  double dini=12.7e-3; //m
     double douti=30e-3; //m
     double mhi=0.02; //kg/s
     double mci=0.05; //kg/s
     double Li=10.0; //m
               double_pipe_HE_with_PressureD ib=new double_pipe_HE_with_PressureD(dini,douti,mhi,mci,Li);
               double Thi=70.0; //degree C
```

```
                double Tci=20.0; //kPa
                double Phi=200.0; //kPa
                double Pci=300.0; //kPa
                //Parallel flow heat exchanger
                ib.calculate_parallel(Thi,Tci,Phi,Pci);
                System.out.println("Q_parallel ="+ib.Q);
                double cc[][]={ib.x,ib.Tc,ib.Th,ib.Pc,ib.Ph};
                String s[]={"x m","cold water temp.","hot water temp.","cold water pressure","hot water pressure"};
                Text.printT(cc,s,"Parallel flow heat exchanger");
                Plot p=new Plot(ib.x,ib.Th);
                p.addData(ib.x,ib.Tc);
                p.setPlabel("Parallel Flow double pipe heat exchanger");
    p.setXlabel("pipe length x m");
    p.setYlabel("Temperature degree C");
                p.plot();
                Plot p1=new Plot(ib.x,ib.Ph);
                p1.addData(ib.x,ib.Pc);
                p1.setPlabel("Parallel Flow double pipe heat exchanger");
    p1.setXlabel("pipe length x m");
    p1.setYlabel("Pressure kPa");
                p1.plot();
                // counter flow heat exchanger
                ib.calculate_counter(Thi,Tci,Phi,Pci);
                System.out.println("Q_counter ="+ib.Q);
                double cc1[][]={ib.x,ib.Tc,ib.Th};
                String s1[]={"x m","cold water temp.","hot water temp."};
                Text.printT(cc1,s1,"Counter flow heat exchanger");
                Plot p2=new Plot(ib.x,ib.Th);
                p2.addData(ib.x,ib.Tc);
                p2.setPlabel("CounterFlow double pipe heat exchanger");
    p2.setXlabel("pipe length x m");
    p2.setYlabel("Temperature degree C");
                p2.plot();

                Plot p3=new Plot(ib.x,ib.Ph);
                p3.addData(ib.x,ib.Pc);
                p3.setPlabel("Counter Flow double pipe heat exchanger");
    p3.setXlabel("pipe length x m");
    p3.setYlabel("Pressure kPa");
                p3.plot();
  }
}
```

Nelder & Mead optimisation algorithm

```
 import java.io.*;
import javax.swing.*;

    class yy implements if_xj
    {
              double xi[]; // independent variable data set
    double yi[]; //dependent variable data set
    double a[];  //fit function coefficient set
    int nn;
              public yy(String filename,double ia[])
    {
     //read the data to curvefit
     //get the data file and initial fit coefficient when class is defined
                double[] Xi=new double[500];
     double[] Yi=new double[500];
     int n=ia.length;
     a=new double[n];
     seta(ia);
     int i=-1;
     try{
                BufferedReader fin=new BufferedReader(new FileReader(filename));
     try {
     while(fin != null)
     {
     i++;
     Xi[i]=Text.readDouble(fin);
     Yi[i]=Text.readDouble(fin);
     }
     } catch(EOFException e_eof) {System.out.println("end of file"); }
     }catch(IOException e_io) {System.out.println("dosya bulunamadı"); }
    nn=i;
    xi=new double[nn];
```

```
    yi=new double[nn];
    for(int j=0;j<nn;j++){xi[j]=Xi[j];yi[j]=Yi[j];}
    a=ia;
    }

    public void seta(double ia[])
    {
                //assign new fit coefficient set
                for(int ii=0;ii<nn;ii++)
          a[ii]=ia[ii];
              }
              public double[] geta()
    {
                // return fit coefficient set
                return a;
              }

              double Cp(double T,double ai[])
    {a=ai;
    double A1=ai[2]/T/Math.sinh(ai[2]/T);
    double A2=a[3]/T/Math.cosh(ai[4]/T);
    double cp=ai[0]+ai[1]*A1*A1+ai[3]*A2*A2;
    return cp; //kJ/kgK
    }

    public double func(double ai[])
    {
    double ff=0;
    double w;
    double yy;
     for(int i=0;i<nn;i++)
                {w= Cp(xi[i],ai)-yi[i];
      ff+=w*w;
                }
    return ff;
              }
}
public class nelderi
{//2. Nelder ve Mead method(simpleks method)_____________________________________________________________________
public static double[] nelder(if_xj fnelder,double a[],double da[],int maxiteration,double tolerance,int printlist)
{
double x[][]=new double[a.length+1][a.length];
for(int i=0;i<x.length;i++)
  {for(int j=0;j<x[0].length;j++)
    {if(i==j){x[i][j]=a[j]+da[j];}
     else    {x[i][j]=a[j]; }
    }
  }
  // Nelder mead multidimensional simplex minimisation method
  //  Nelder & Mead 1965 Computer J, v.7, 308-313.

  // Giriş değişkenleri tanımlaması
  // fnelder : abstract çok boyutlu fonksiyon f(x)
  // x : for n boyutlu n+1 simlex noktasını içerir bağımsız değişken seti
  // maxiteration : maximum iterasyon sayısı
  // tolerance :
       int NDIMS = x.length-1;
       int NPTS = x.length;
       int FUNC = NDIMS;
       int ncalls = 0;
         ////// başlangıç simplexini oluştur ///////////////////
         double p[][]=new double[NPTS][NPTS]; // [row][col] = [whichvx][coord,FUNC]
         double z[]=new double[NDIMS];
         double best = 1E99;
         ///////////////// iilk fonksiyon değerlerini hesapla ////////////////
         for (int i=0; i<NPTS; i++)
         {
           for (int j=0; j<NDIMS; j++)
               {p[i][j] = x[i][j];}
           p[i][NDIMS] = fnelder.func(p[i]);
         }
         int iter=0;
         for (iter=1; iter<maxiteration; iter++)
         {
           ///////////  lo, nhi, hi noktalarını tanımla ///////////////
           int  ilo=0, ihi=0, inhi = -1; // -1 means missing
           double flo = p[0][FUNC];
```

```
double fhi = flo;
double pavg,sterr;
for (int i=1; i<NPTS; i++)
{
  if (p[i][FUNC] < flo)
    {flo=p[i][FUNC]; ilo=i;}
  if (p[i][FUNC] > fhi)
    {fhi=p[i][FUNC]; ihi=i;}
}
double fnhi = flo;
inhi = ilo;
for (int i=0; i<NPTS; i++)
 if ((i != ihi) && (p[i][FUNC] > fnhi))
   {fnhi=p[i][FUNC]; inhi=i;}
////////// çıkış kriteri //////////////
if ((iter % 4*NDIMS) == 0)
{
   //yi nin standart hata kriteri set değerinden (tolerance)
   // küçük olmalı
   // ortalama değeri hesapla (en büyük değer de dahil olmak üzere)
   pavg=0;
   for(int i=0;i<NPTS;i++)
     pavg+=p[i][FUNC];
   pavg/=NPTS;
   double tot=0;
   if(printlist!=0)
   {  System.out.print(iter);
      for (int j=0; j<=NDIMS; j++)
        { System.out.print(p[ilo][j]+" ");}
      System.out.println("");
   }
   for(int i=0;i<NPTS;i++)
   { tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg);}
   sterr=Math.sqrt(tot/NPTS);
   //if(sterr < tolerance)
     { for (int j=0; j<NDIMS; j++)
         { z[j]=p[ilo][j];}
       //break;
     }
   best = p[ilo][FUNC];
}

///// ave[] vektorünü en büyük değeri hariç tutarak hesapla //////

double ave[] = new double[NDIMS];
for (int j=0; j<NDIMS; j++)
 ave[j] = 0;
for (int i=0; i<NPTS; i++)
 if (i != ihi)
   for (int j=0; j<NDIMS; j++)
     ave[j] += p[i][j];
for (int j=0; j<NDIMS; j++)
  ave[j] /= (NPTS-1);


///////// yansıt ////////////////

double r[] = new double[NDIMS];
for (int j=0; j<NDIMS; j++)
 r[j] = 2*ave[j] - p[ihi][j];
double fr = fnelder.func(r);

if ((flo <= fr) && (fr < fnhi))  // in zone: accept
{
   for (int j=0; j<NDIMS; j++)
     p[ihi][j] = r[j];
   p[ihi][FUNC] = fr;
   continue;
}

if (fr < flo)  //// genişlet, else kabul et
{
   double e[] = new double[NDIMS];
   for (int j=0; j<NDIMS; j++)
     e[j] = 3*ave[j] - 2*p[ihi][j];
   double fe = fnelder.func(e);
   if (fe < fr)
```

```
                {
                  for (int j=0; j<NDIMS; j++)
                    p[ihi][j] = e[j];
                  p[ihi][FUNC] = fe;
                  continue;
                }
                else
                {
                  for (int j=0; j<NDIMS; j++)
                    p[ihi][j] = r[j];
                  p[ihi][FUNC] = fr;
                  continue;
                }
              }

              ////////////// daralt:

              if (fr < fhi)
              {
                 double c[] = new double[NDIMS];
                 for (int j=0; j<NDIMS; j++)
                   c[j] = 1.5*ave[j] - 0.5*p[ihi][j];
                 double fc = fnelder.func(c);
                 if (fc <= fr)
                 {
                    for (int j=0; j<NDIMS; j++)
                      p[ihi][j] = c[j];
                    p[ihi][FUNC] = fc;
                    continue;
                 }
                 else   /////// daralt
                 {
                    for (int i=0; i<NPTS; i++)
                      if (i != ilo)
                      {
                         for (int j=0; j<NDIMS; j++)
                           p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                         p[i][FUNC] = fnelder.func(p[i]);
                      }
                    continue;
                 }
              }

              if (fr >= fhi)   ///
              {
                 double cc[] = new double[NDIMS];
                 for (int j=0; j<NDIMS; j++)
                   cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
                 double fcc = fnelder.func(cc);
                 if (fcc < fhi)
                 {
                    for (int j=0; j<NDIMS; j++)
                      p[ihi][j] = cc[j];
                    p[ihi][FUNC] = fcc;
                    continue;
                 }
                 else    /////////
                 {
                    for (int i=0; i<NPTS; i++)
                      if (i != ilo)
                      {
                         for (int j=0; j<NDIMS; j++)
                           p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                         p[i][FUNC] = fnelder.func(p[i]);
                      }
                 }
              }
           }
       return z;
}
    public static double[] nelder(if_xj fnelder,double a[],double da[],double tolerance)
    {return nelder(fnelder,a,da,300,tolerance,0);}

    public static double[] nelder(if_xj fnelder,double a[],double da[])
    {return nelder(fnelder,a,da,500,1.0e-20,0);}
  // input data
  //_____________________________________________________________
```

```
    public static double[] inputdata(String s)
    { String s2[]=JOptionPane.showInputDialog(s).split("\\s+");
      int m=s2.length;
      double a[]=new double[m];
      for(int i=0;i<m;i++)
      { a[i]=Double.parseDouble(s2[i]);}
      return a;
    }
   public static void main(String args[]) throws IOException
   {
   String in_name=JOptionPane.showInputDialog(" veri dosyasının ismini giriniz : ");
   //String in_name="D1.txt";
   double a[];
   a=new double[5];
   a[0] = 1.0;
   a[1] = 1.0;
   a[2] = 1.0;
   a[3] = 1.0;
   a[4] = 1.0;
   double da[];
   da=new double[5];
   da[0] =0.3;
   da[1] =0.3;
   da[2] =0.3;
   da[3] =0.3;
   da[4] =0.3;
   yy f=new yy(in_name,a);
   int i=0;
   double p[]=nelder(f,a,da,1000,1e-15,0);
   String s1=" optimisation value  : \n"+Matrix.toStringT(p)+"\n";
   s1+="original function value = "+f.func(a)+"\n";
   s1+="function value = "+f.func(p)+"\n";
   s1+="standard division= "+Math.sqrt(f.func(p)/(f.nn-1));
   System.out.println(s1);
   Plot pp=new Plot(f.xi,f.yi);
   pp.setPlotType(0,20);
   double y[]=new double[f.xi.length];
   for(i=0;i<f.xi.length;i++)
   {y[i]=f.Cp(f.xi[i],f.a);}
   pp.addData(f.xi,y);
   pp.plot();
   }
}
```

Interface if_xj

```
@FunctionalInterface
interface if_xj extends iMathd
{
  // single function  multi independent variable
  // a single value is returned indiced to equation_ref
public double func(double x[]);
default double func(double x1,double y1)
{
double x[]=new double[2];
x[0]=x1;
x[1]=y1;
return func(x);
}
default double[] dfunc(double x[])
{int n=x.length;
 double c[]=new double[n];
 for(int i=0;i<n;i++)
 {c[i]=dfunc(x,i);}
 return c;
}

default double[][] d2func(double x[])
{int n=x.length;
 double c[][]=new double[n][n];
 for(int i=0;i<n;i++)
 {for(int j=0;j<n;j++)
   {   int xref[]={i,j};
                c[i][j]=d2func(x,xref);
   }
 }
```

```
 return c;
}

default double d2func(double x[],int x_ref[])
{// derivative of the function with respect to x_ref
double h0=0.0256;
int i,m;
int n=7;
double f1,f2;
double x1[];
x1=new double[x.length];
double x2[];
x2=new double[x.length];
for(i=0;i<x.length;i++)
{
x1[i]=x[i];
x2[i]=x[i];
}
//derivative of a simple function
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
//vector<double> h(n,0);
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
h[i]=h0*Math.pow(r,i);
}

for(i=0;i<n;i++)
{
x1[x_ref[1]]+=h[i];
x2[x_ref[1]]-=h[i];
f1=dfunc(x1,x_ref[0]);
f2=dfunc(x2,x_ref[0]);
T[i][0]=( f1 - f2)/(2.0*h[i]);
x1[x_ref[1]]=x[x_ref[1]];
x2[x_ref[1]]=x[x_ref[1]];
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i]
  - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}
default double dfunc(double x[],int x_ref)
{ // derivative of the function with respect to x_ref
double h0=0.0256;
int i,m;
int n=7;
double f1,f2;
double x1[];
x1=new double[x.length];
double x2[];
x2=new double[x.length];
for(i=0;i<x.length;i++)
{
x1[i]=x[i];
x2[i]=x[i];
}
//derivative of a simple function
double T[][];
T=new double[n][n];
double h[];
```

```
h=new double[n];
//vector<double> h(n,0);
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
h[i]=h0*Math.pow(r,i);
}

for(i=0;i<n;i++)
{
x1[x_ref]+=h[i];
x2[x_ref]-=h[i];
f1=func(x1);
f2=func(x2);
T[i][0]=( f1 - f2)/(2.0*h[i]);
x1[x_ref]=x[x_ref];
x2[x_ref]=x[x_ref];
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i]
  - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}
}
```

Interface ifi_xj

```
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import static java.lang.Math.*;
@FunctionalInterface
interface  ifi_xj
{ // multifunction multi independent variable
  // vector of dependent variables are returned
  // example f[0]=x[0]+sin(x[1])
  //         f[1]=x[0]*x[1]-x[1]
  // func(x) returns the value of f[0] and f[1]
  // as a two dimensional vector
public double[] func(double x[]);
//special case: time dependent functions
default double[] func(double x,double y)
{double xx[]={x,y};
 return func(xx);
}
default  double[] xp(double x[],double j,double m,double h)
{
//h = dx
//j=-4..4 multiplication factor
//m derivative taken value
int k=x.length;
double xx[]=new double[k];
for(int i=0;i<k;i++)
{               if(i==m) xx[i]=x[i]+j*h;
    else     xx[i]=x[i];
}
return xx;
}
default double[][] dfunc(double x[])
{
double h=1.0e-12;
```

```
return dfunc(x,1,h);
}
default double[][] dfunc(double x[],int n)
{
double h=1.0e-12;
return dfunc(x,n,h);
}

default double[][] dfunc(double x[],int n,double h)
{
int k=x.length; //number of x values (dimension)
double df[][]=new double[k][k];
for(int i=0;i<k;i++)
{ for(int j=0;j<k;j++)
  {df[i][j]=dfunc(x,n,i,j,h);}
}
return df;
}

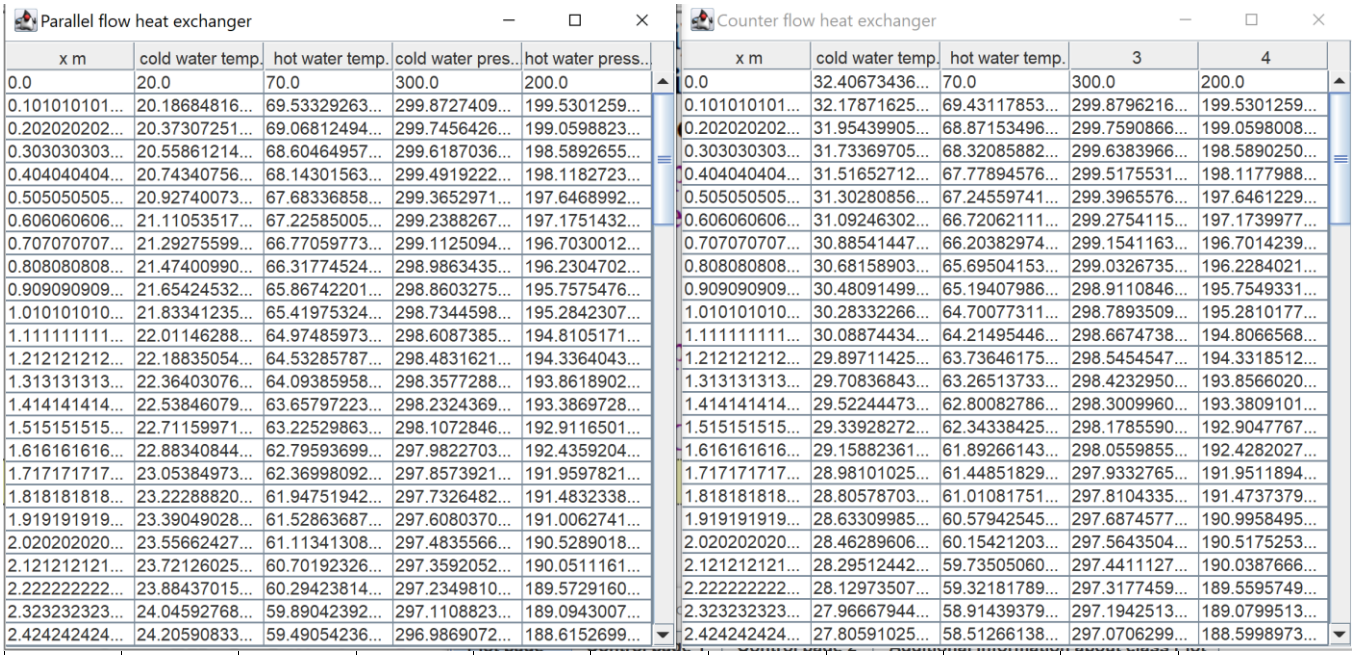
default double dfunc(double x[],int n,int j,int m)
{
double h=1.0e-12;
return dfunc(x,n,j,m,h);
}

default double dfunc(double x[],int n,int j,int m,double h)
{ // derivative of x[m]
  //m derivative taken value
  //jth equation
  //nth derivative of the equation (n=0) function value
  //n=1 first derivative n=2 second derivative..
int k=x.length; //number of x values (dimension)
double hh=1/h;
double ff[][]=new double[9][3];
double xp[][]=new double[9][3];
double df;
int j1;
double x1=0;
x1=x[m];
    for(int jj=-4;jj<=4;jj++)
    {  j1=jj+4;
                xp[j1]=xp(x,jj,m,h);
       ff[j1]=func(xp[j1]);
    }
if(n==0)
{df=ff[4][j];}
else if(n==1)
{ df=(3.0*ff[0][j]-32.0*ff[1][j]+168.0*ff[2][j]-672.0*ff[3][j]+672.0*ff[5][j]-168.0*ff[6][j]+32.0*ff[7][j]-3.0*ff[8][j])/840.0*hh;}
else if(n==2)
{df=(-14350.0*ff[4][j]-9.0*ff[0][j]+128*ff[1][j]-1008*ff[2][j]+8064*ff[3][j]+8064.0*ff[5][j]-1008.0*ff[6][j]+128.0*ff[7][j]-
9.0*ff[8][j])/5040.0*hh*hh;}
else if(n==3)
{df=(-7.0*ff[0][j]+72.0*ff[1][j]-338.0*ff[2][j]+488.0*ff[3][j]-488.0*ff[5][j]+338.0*ff[6][j]-72.0*ff[7][j]+7.0*ff[8][j])/240.0*hh*hh*hh;}
else if(n==4)
{df=(2730.0*ff[4][j]+7.0*ff[0][j]-96.0*ff[1][j]+676.0*ff[2][j]-1952.0*ff[3][j]-1952.0*ff[5][j]+676.0*ff[6][j]-
96.0*ff[7][j]+7.0*ff[8][j])/240.0*hh*hh*hh*hh;}
else if(n==5)
{df=(ff[0][j]-9.0*ff[1][j]+26.0*ff[2][j]-29.0*ff[3][j]+29.0*ff[5][j]-26.0*ff[6][j]+9.0*ff[7][j]-ff[8][j])/6.0*hh*hh*hh*hh*hh;}
else if(n==6)
{df=(-150.0*ff[4][j]-ff[0][j]+12.0*ff[1][j]-52.0*ff[2][j]+116.0*ff[3][j]+116.0*ff[5][j]-52.0*ff[6][j]+12.0*ff[7][j]-
ff[8][j])/4.0*hh*hh*hh*hh*hh*hh;}
else if(n==7)
{df=(-ff[0][j]+6.0*ff[1][j]-14.0*ff[2][j]+14.0*ff[3][j]-14.0*ff[5][j]+14.0*ff[6][j]-6.0*ff[7][j]+ff[8][j])/2.0*hh*hh*hh*hh*hh*hh*hh;}
else if(n==8)
{df=(70.0*ff[4][j]+ff[0][j]-8.0*ff[1][j]+28.0*ff[2][j]-56.0*ff[3][j]-56.0*ff[5][j]+28.0*ff[6][j]-
8.0*ff[7][j]+ff[8][j])*hh*hh*hh*hh*hh*hh*hh*hh;}
else df=0;
return df;
}
default double[] secant2(double x[])
{
// Roots of nonlinear system of equations secant method
//ti : weight function
//x vector of independent variables
//y vector of dependent variables
//dy derivative vector of dependent variables
double ti=1.0;
```

```
int i;
int nmax=20;
double tolerance=1.0e-5;
int n=x.length;
double b[];
b=new double[n];
for(i=0;i<n;i++)
{
b[i]=1.0;
}
i=0;
while( i++ < nmax && Matrix.abs(b) > tolerance )
{ b=Matrix.multiply(Matrix.divide(func(x),dfunc(x)),-ti);
  x=Matrix.add(x,b);
}
if(i >= nmax) System.out.println("Warning : Maximum number of iterations are exceeded \nResults may not be valid");
return x;
}

}
```

Results

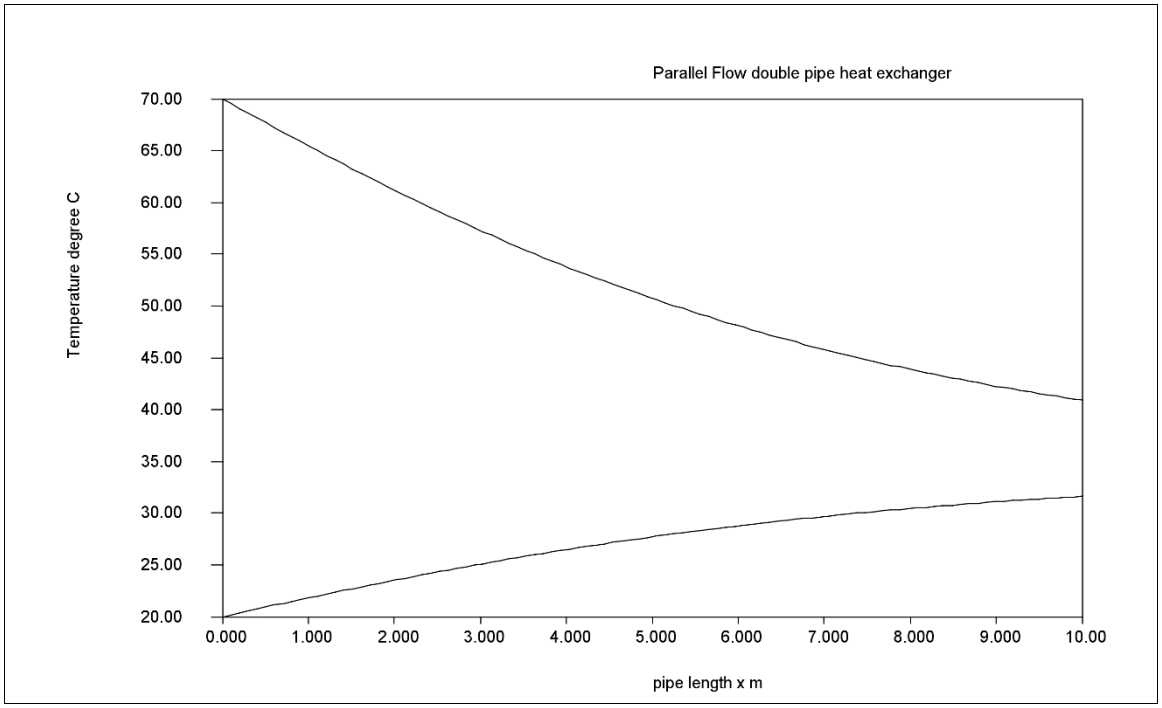


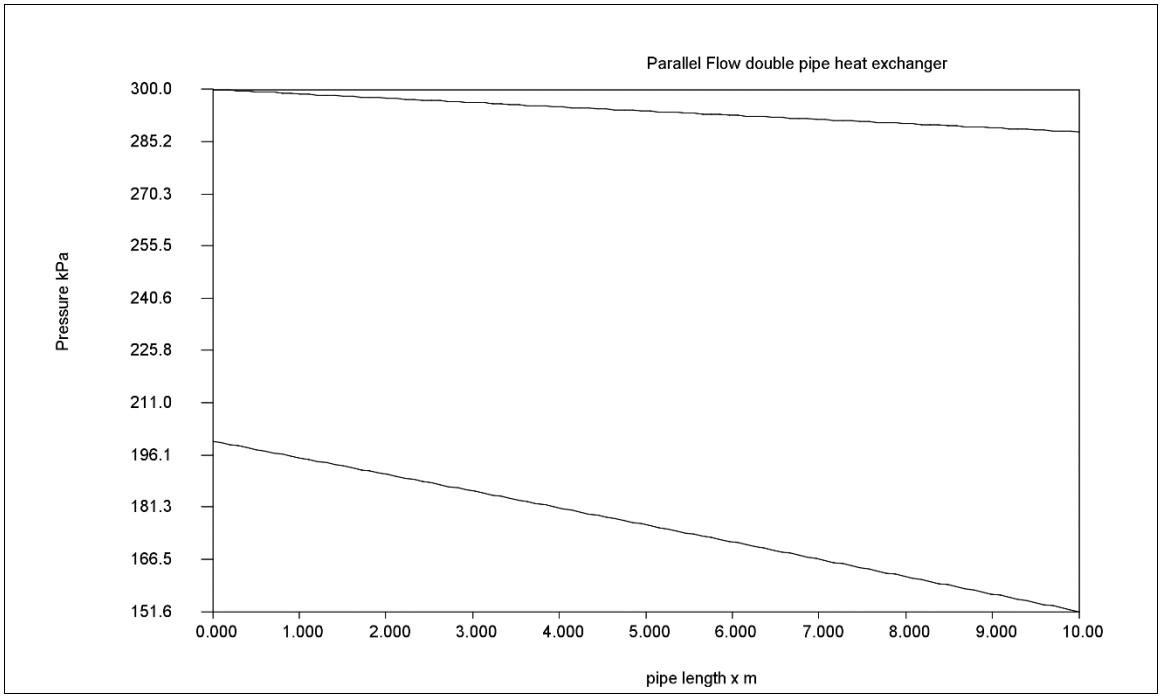
Parallel Flow					Counter flow				
x m	**Tc degree C**	**Th degree C**	**Pc kPa**	**Ph kPa**	**x m**	**Tc degree C**	**Th degree C**	**Pc kPa**	**Ph kPa**
0.0000	20.0000	70.0000	300.0000	200.0000	0.0000	32.4067	70.0000	300.0000	200.0000
0.1010	20.1868	69.5333	299.8727	199.5301	0.1010	32.1787	69.4312	299.8796	199.5301
0.2020	20.3731	69.0681	299.7456	199.0599	0.2020	31.9544	68.8715	299.7591	199.0598
0.3030	20.5586	68.6046	299.6187	198.5893	0.3030	31.7337	68.3209	299.6384	198.5890
0.4040	20.7434	68.1430	299.4919	198.1183	0.4040	31.5165	67.7789	299.5176	198.1178
0.5051	20.9274	67.6834	299.3653	197.6469	0.5051	31.3028	67.2456	299.3966	197.6461
0.6061	21.1105	67.2259	299.2388	197.1751	0.6061	31.0925	66.7206	299.2754	197.1740
0.7071	21.2928	66.7706	299.1125	196.7030	0.7071	30.8854	66.2038	299.1541	196.7014
0.8081	21.4740	66.3177	298.9863	196.2305	0.8081	30.6816	65.6950	299.0327	196.2284
0.9091	21.6542	65.8674	298.8603	195.7575	0.9091	30.4809	65.1941	298.9111	195.7549
1.0101	21.8334	65.4198	298.7345	195.2842	1.0101	30.2833	64.7008	298.7894	195.2810
1.1111	22.0115	64.9749	298.6087	194.8105	1.1111	30.0887	64.2150	298.6675	194.8067
1.2121	22.1884	64.5329	298.4832	194.3364	1.2121	29.8971	63.7365	298.5455	194.3319

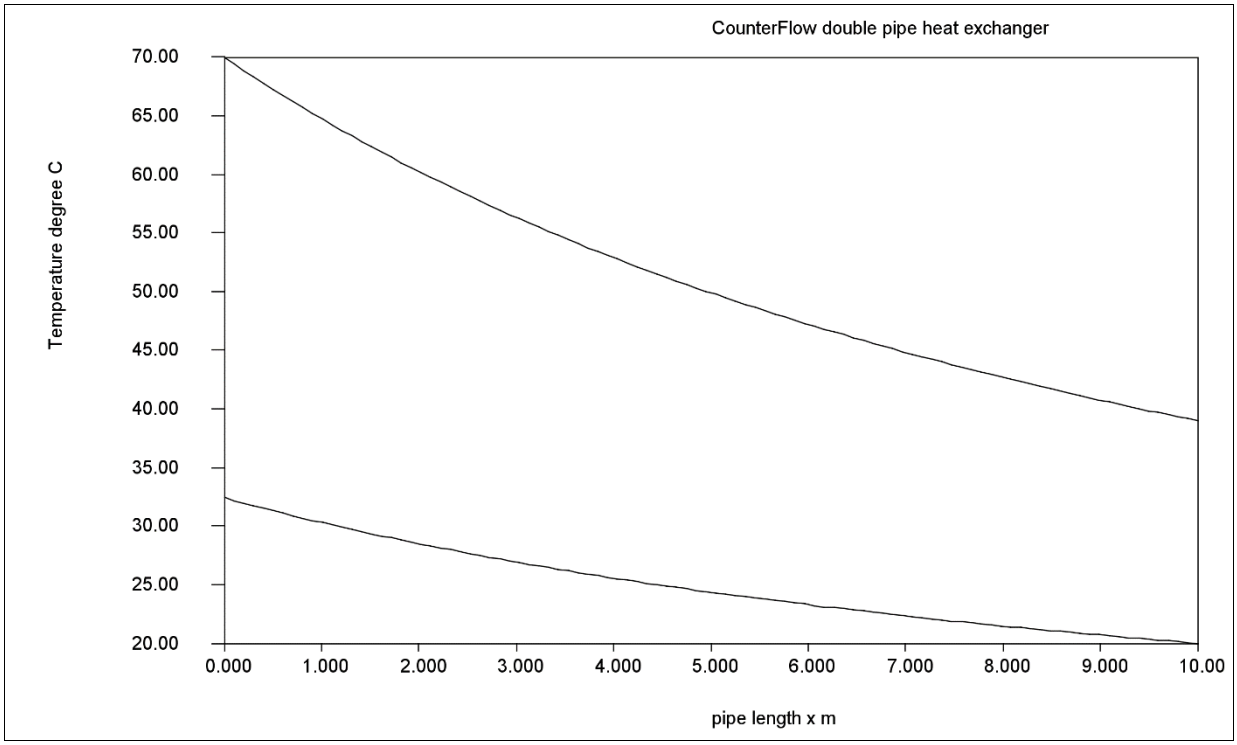
1.3131	22.3640	64.0939	298.3577	193.8619	1.3131	29.7084	63.2651	298.4233	193.8566
1.4141	22.5385	63.6580	298.2324	193.3870	1.4141	29.5224	62.8008	298.3010	193.3809
1.5152	22.7116	63.2253	298.1073	192.9117	1.5152	29.3393	62.3434	298.1786	192.9048
1.6162	22.8834	62.7959	297.9823	192.4359	1.6162	29.1588	61.8927	298.0560	192.4282
1.7172	23.0538	62.3700	297.8574	191.9598	1.7172	28.9810	61.4485	297.9333	191.9512
1.8182	23.2229	61.9475	297.7326	191.4832	1.8182	28.8058	61.0108	297.8104	191.4737
1.9192	23.3905	61.5286	297.6080	191.0063	1.9192	28.6331	60.5794	297.6875	190.9958
2.0202	23.5566	61.1134	297.4836	190.5289	2.0202	28.4629	60.1542	297.5644	190.5175
2.1212	23.7213	60.7019	297.3592	190.0511	2.1212	28.2951	59.7351	297.4411	190.0388
2.2222	23.8844	60.2942	297.2350	189.5729	2.2222	28.1297	59.3218	297.3177	189.5596
2.3232	24.0459	59.8904	297.1109	189.0943	2.3232	27.9667	58.9144	297.1943	189.0800
2.4242	24.2059	59.4905	296.9869	188.6153	2.4242	27.8059	58.5127	297.0706	188.5999
2.5253	24.3643	59.0947	296.8631	188.1358	2.5253	27.6474	58.1165	296.9469	188.1194
2.6263	24.5210	58.7028	296.7393	187.6560	2.6263	27.4910	57.7258	296.8230	187.6385
2.7273	24.6762	58.3150	296.6157	187.1757	2.7273	27.3369	57.3405	296.6990	187.1572
2.8283	24.8296	57.9314	296.4922	186.6950	2.8283	27.1848	56.9604	296.5749	186.6754
2.9293	24.9814	57.5520	296.3688	186.2139	2.9293	27.0348	56.5855	296.4507	186.1932
3.0303	25.1315	57.1768	296.2456	185.7323	3.0303	26.8868	56.2156	296.3263	185.7106
3.1313	25.2799	56.8058	296.1224	185.2504	3.1313	26.7408	55.8507	296.2018	185.2276
3.2323	25.4266	56.4391	295.9993	184.7680	3.2323	26.5968	55.4906	296.0772	184.7441
3.3333	25.5716	56.0767	295.8764	184.2853	3.3333	26.4547	55.1353	295.9525	184.2603
3.4343	25.7148	55.7186	295.7536	183.8021	3.4343	26.3144	54.7847	295.8277	183.7760
3.5354	25.8563	55.3648	295.6308	183.3185	3.5354	26.1760	54.4387	295.7028	183.2913
3.6364	25.9961	55.0154	295.5082	182.8345	3.6364	26.0394	54.0971	295.5777	182.8062
3.7374	26.1341	54.6704	295.3857	182.3501	3.7374	25.9046	53.7600	295.4525	182.3207
3.8384	26.2703	54.3297	295.2633	181.8652	3.8384	25.7715	53.4272	295.3273	181.8348
3.9394	26.4048	53.9935	295.1409	181.3800	3.9394	25.6401	53.0987	295.2019	181.3484
4.0404	26.5376	53.6616	295.0187	180.8944	4.0404	25.5103	52.7743	295.0764	180.8617
4.1414	26.6686	53.3340	294.8965	180.4083	4.1414	25.3823	52.4540	294.9508	180.3746
4.2424	26.7978	53.0109	294.7745	179.9219	4.2424	25.2558	52.1378	294.8250	179.8870
4.3434	26.9253	52.6921	294.6525	179.4350	4.3434	25.1309	51.8255	294.6992	179.3991
4.4444	27.0511	52.3777	294.5307	178.9478	4.4444	25.0076	51.5171	294.5733	178.9107
4.5455	27.1751	52.0676	294.4089	178.4601	4.5455	24.8858	51.2125	294.4473	178.4220
4.6465	27.2974	51.7619	294.2872	177.9721	4.6465	24.7655	50.9117	294.3211	177.9328
4.7475	27.4179	51.4605	294.1656	177.4837	4.7475	24.6467	50.6145	294.1949	177.4433
4.8485	27.5367	51.1633	294.0441	176.9948	4.8485	24.5294	50.3209	294.0686	176.9534
4.9495	27.6539	50.8705	293.9226	176.5056	4.9495	24.4134	50.0309	293.9421	176.4631
5.0505	27.7693	50.5819	293.8013	176.0161	5.0505	24.2989	49.7444	293.8156	175.9724
5.1515	27.8830	50.2975	293.6800	175.5261	5.1515	24.1857	49.4613	293.6889	175.4813
5.2525	27.9950	50.0174	293.5588	175.0357	5.2525	24.0739	49.1815	293.5622	174.9898
5.3535	28.1054	49.7414	293.4377	174.5450	5.3535	23.9634	48.9051	293.4353	174.4979
5.4545	28.2141	49.4696	293.3166	174.0539	5.4545	23.8542	48.6320	293.3084	174.0057
5.5556	28.3212	49.2019	293.1957	173.5624	5.5556	23.7462	48.3620	293.1814	173.5130
5.6566	28.4266	48.9383	293.0747	173.0706	5.6566	23.6396	48.0952	293.0542	173.0200
5.7576	28.5304	48.6787	292.9539	172.5784	5.7576	23.5342	47.8314	292.9270	172.5267
5.8586	28.6326	48.4232	292.8331	172.0859	5.8586	23.4300	47.5707	292.7997	172.0329
5.9596	28.7332	48.1717	292.7125	171.5929	5.9596	23.3270	47.3130	292.6723	171.5388

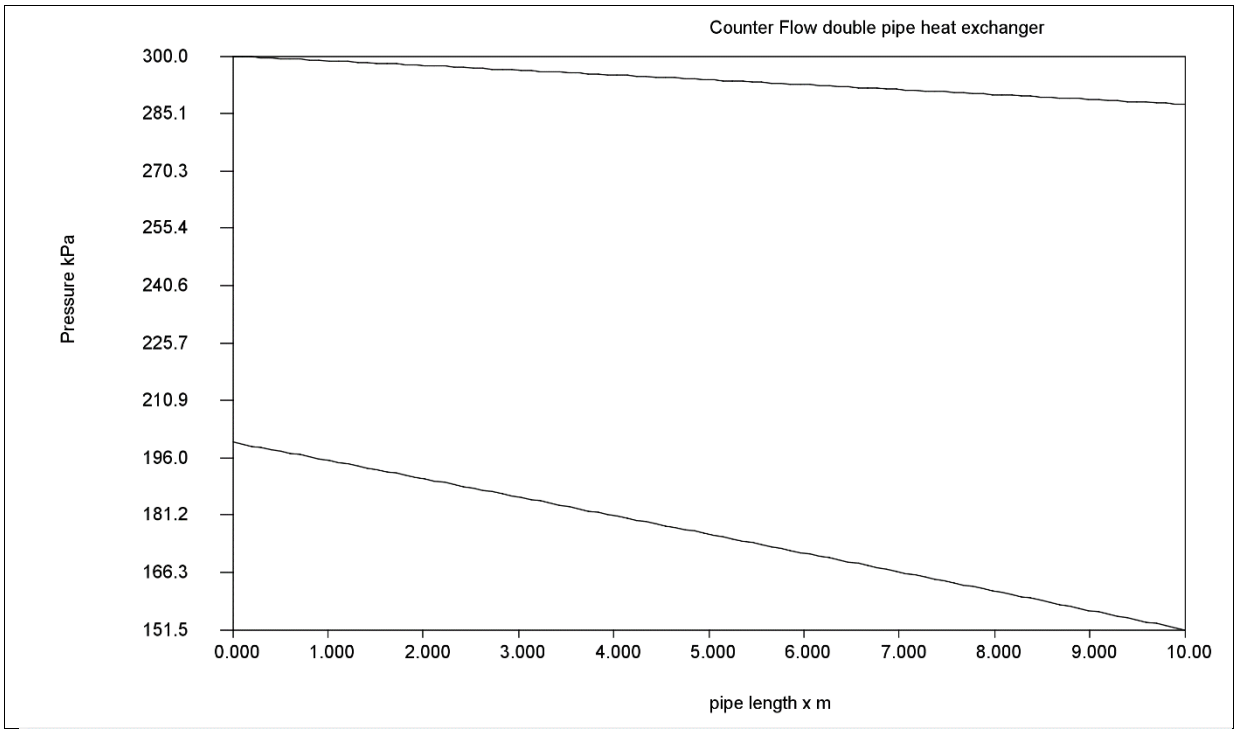
6.0606	28.8322	47.9241	292.5918	171.0997	6.0606	23.2251	47.0583	292.5448	171.0443
6.1616	28.9297	47.6804	292.4713	170.6060	6.1616	23.1245	46.8064	292.4172	170.5494
6.2626	29.0256	47.4406	292.3508	170.1121	6.2626	23.0249	46.5573	292.2895	170.0541
6.3636	29.1199	47.2046	292.2303	169.6177	6.3636	22.9265	46.3111	292.1617	169.5585
6.4646	29.2128	46.9724	292.1100	169.1231	6.4646	22.8292	46.0676	292.0339	169.0626
6.5657	29.3041	46.7440	291.9897	168.6281	6.5657	22.7330	45.8269	291.9059	168.5662
6.6667	29.3940	46.5193	291.8694	168.1327	6.6667	22.6378	45.5887	291.7779	168.0695
6.7677	29.4824	46.2982	291.7492	167.6371	6.7677	22.5437	45.3533	291.6498	167.5724
6.8687	29.5693	46.0808	291.6291	167.1411	6.8687	22.4506	45.1204	291.5215	167.0750
6.9697	29.6549	45.8670	291.5090	166.6448	6.9697	22.3586	44.8900	291.3932	166.5772
7.0707	29.7389	45.6568	291.3890	166.1481	7.0707	22.2675	44.6621	291.2649	166.0791
7.1717	29.8216	45.4500	291.2690	165.6511	7.1717	22.1775	44.4367	291.1364	165.5806
7.2727	29.9029	45.2467	291.1491	165.1539	7.2727	22.0884	44.2137	291.0078	165.0818
7.3737	29.9829	45.0468	291.0292	164.6563	7.3737	22.0002	43.9931	290.8792	164.5826
7.4747	30.0615	44.8503	290.9094	164.1584	7.4747	21.9130	43.7749	290.7505	164.0830
7.5758	30.1387	44.6571	290.7897	163.6602	7.5758	21.8267	43.5589	290.6217	163.5832
7.6768	30.2147	44.4672	290.6700	163.1617	7.6768	21.7413	43.3452	290.4928	163.0829
7.7778	30.2893	44.2805	290.5503	162.6629	7.7778	21.6568	43.1338	290.3638	162.5824
7.8788	30.3627	44.0971	290.4307	162.1638	7.8788	21.5733	42.9246	290.2348	162.0814
7.9798	30.4348	43.9168	290.3111	161.6644	7.9798	21.4905	42.7175	290.1057	161.5802
8.0808	30.5056	43.7396	290.1916	161.1648	8.0808	21.4087	42.5126	289.9765	161.0786
8.1818	30.5753	43.5655	290.0721	160.6648	8.1818	21.3276	42.3098	289.8472	160.5766
8.2828	30.6437	43.3943	289.9527	160.1646	8.2828	21.2474	42.1091	289.7179	160.0744
8.3838	30.7110	43.2262	289.8333	159.6641	8.3838	21.1681	41.9104	289.5884	159.5718
8.4848	30.7770	43.0610	289.7140	159.1633	8.4848	21.0895	41.7138	289.4589	159.0688
8.5859	30.8419	42.8987	289.5947	158.6623	8.5859	21.0117	41.5191	289.3293	158.5655
8.6869	30.9057	42.7393	289.4754	158.1610	8.6869	20.9347	41.3264	289.1997	158.0619
8.7879	30.9683	42.5826	289.3562	157.6594	8.7879	20.8585	41.1356	289.0699	157.5580
8.8889	31.0299	42.4288	289.2370	157.1576	8.8889	20.7830	40.9467	288.9401	157.0538
8.9899	31.0903	42.2776	289.1179	156.6555	8.9899	20.7083	40.7597	288.8103	156.5492
9.0909	31.1497	42.1291	288.9988	156.1532	9.0909	20.6344	40.5745	288.6803	156.0443
9.1919	31.2080	41.9833	288.8797	155.6506	9.1919	20.5611	40.3912	288.5503	155.5391
9.2929	31.2653	41.8401	288.7607	155.1478	9.2929	20.4886	40.2096	288.4202	155.0335
9.3939	31.3216	41.6994	288.6417	154.6448	9.3939	20.4168	40.0298	288.2900	154.5276
9.4949	31.3768	41.5612	288.5227	154.1415	9.4949	20.3456	39.8518	288.1598	154.0215
9.5960	31.4311	41.4255	288.4038	153.6379	9.5960	20.2752	39.6755	288.0295	153.5150
9.6970	31.4844	41.2923	288.2849	153.1342	9.6970	20.2054	39.5008	287.8991	153.0082
9.7980	31.5367	41.1614	288.1660	152.6302	9.7980	20.1364	39.3279	287.7686	152.5010
9.8990	31.5881	41.0330	288.0472	152.1260	9.8990	20.0679	39.1566	287.6381	151.9936
10.0000	31.6386	40.9068	287.9284	151.6215	10.0000	20.0002	38.9869	287.5075	151.4858

---------- Capture Output ----------
> "C:\Users\Turhan Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe" double_pipe_HE_with_PressureD
Q_parallel =2433.168966973114
Q_counter =2593.6128222553075


Parallel Flow double pipe heat exchanger
Temperature degree C
70.00
65.00
60.00
55.00
50.00
45.00
40.00
35.00
30.00
25.00
20.00
0.000
1.000
2.000
3.000
4.000
5.000
6.000
7.000
8.000
9.000
10.00
pipe length x m


Parallel Flow double pipe heat exchanger
Pressure kPa
300.0
285.2
270.3
255.5
240.6
225.8
211.0
196.1
181.3
166.5
151.6
0.000
1.000
2.000
3.000
4.000
5.000
6.000
7.000
8.000
9.000
10.00
pipe length x m

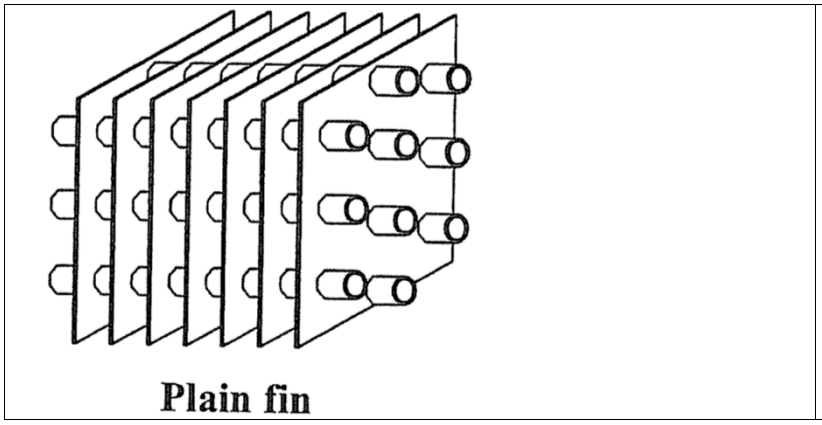


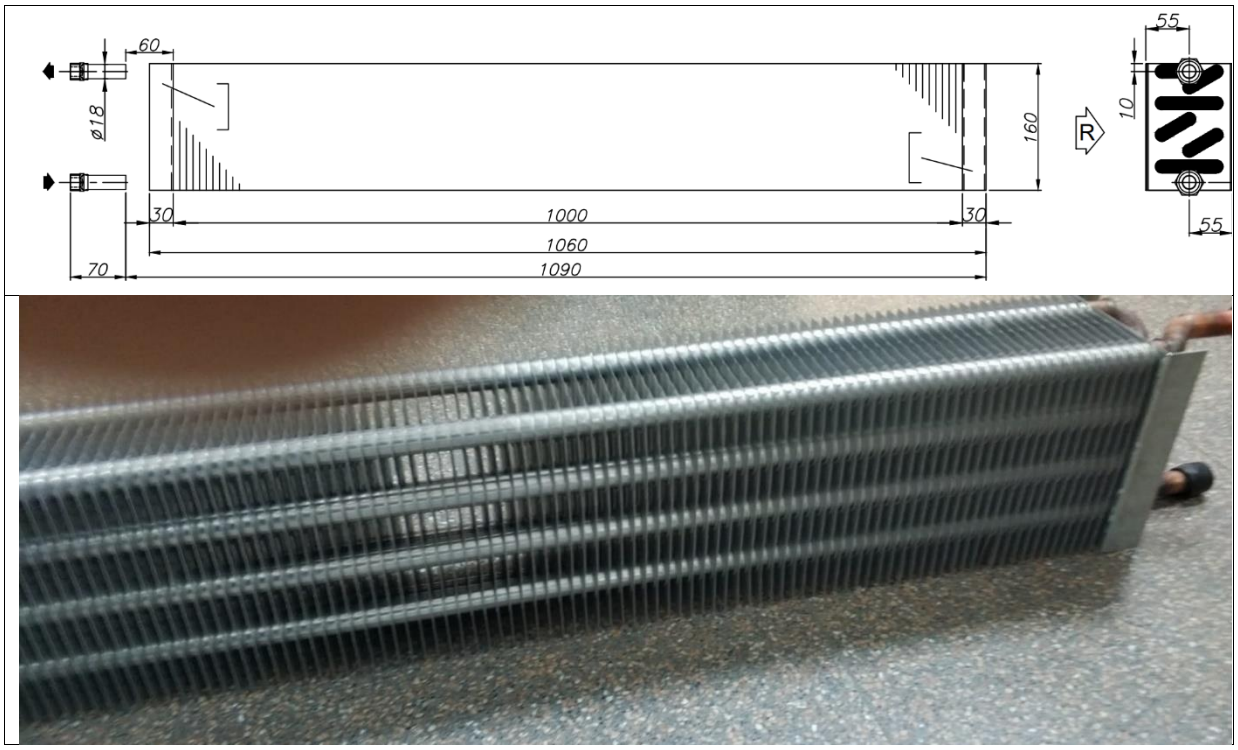


When results of parallel flow and counter flow cases are compared, Pressure drop values are relatively similar, But heat transfer of counter flow is better in counter flow heat exchanger (**2433.168966973114 W vs. 2593.6128222553075 W)**

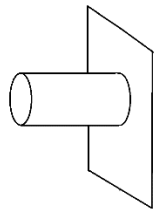
Natural convective finned type heat exchangers will be modelled as a finite difference model. This type of heat exchangers with smaller distances within fins are commonly used in refrigeration and heating systems as forced convection applications. In natural convection applications fin spacing should be larger in order to allow natural convective flows passing through the fins. General view of the device is shown in the figures. Water flows inside pipes, which usually made of copper or aliminium and outside

air is heated up through finned surfaces and pipes of heat exchanger. Fins are made of aliminium and connected to tubes by plastic deformation by appliying pressure from inside of tubes. In order to simulate heat transfer mechanism in the finned heat exchanger, natural convection finite difference model is considered.





We will consider a single fin and a single pipe section. In this case fin will be exposed to natural convection. Pipe section will also exposed to natural convection, but different heat transfer equations will be used for pipe and fin. Furthermore for fins, fin efficiency should be taken into effect.



$$dz = \frac{L}{N}$$

Where L= pipe length

N= number of finite difference division

$$dQ_i = U_i dA_i (T_{water} - T_{air})$$

$$\frac{1}{U_i} = \frac{1}{h_{water\,i}} + \frac{t_{panel}}{k_{panel}} + \frac{1}{\eta_{fin\,i} h_{air\,i}}$$

$$M = \frac{2hl_{fin}}{kt_{fin}}$$

$$\eta_{fin} = \left(\frac{\tanh(M)}{M}\right)$$

$$dQ_i = m_{water} C_{p_{water}} (T_{water\,i})(T_{water\,i} - T_{water\,i+1})$$

In natural convection air flow rate is difficult parameter, Velocity profile can be approximated as:

$$U_{air} = \sqrt{2gdz\beta(T_s - T_{air})}$$

```
import java.util.Arrays;
public class HT_pipe_finned_natural
{  //all pipes connected serial Natural convective heat exchanger
            air_PG_CS air;
            steamIAPWS_IF97 water;
   double L_fin;
   double D;
   double L_pipe;
   double L_total;
   int n_pipe;  // number of pipes
   int n_fin;  //number of fins
   double P;          // air pressure kPA
   double a[];        // property vector
   double b[];        // property vector
   double avis;
   double ak;
   double aCp;
   double aro;
   double aPr;
   double wvis;
   double wk;
   double wCp;
   double wro;
   double wPr;
   double g; // gravitational constant  m/s^2
   double D_fin;  //fin outside radius
   double t_fin; //fin thickness
   double k_pipe;
   double k_fin;
   double m;//kg/s water mass flow rate
   double s_fin; //distance between fins
   double A_pipe; // pipe area
   double A_fin;   // fin area
   double A_in;   //internal area
   double A_cross_pipe;  //pipe cross-sectional areas
   double T_water[];
   double x_pipe[];
   double P_water; //water presure
   double eps; //surface quality m
   double eod;  //eps/D
   double H;  //height of the fin
   double W; //width of the fin
   double dx;  pipe thickness;
   public HT_pipe_finned_natural(double mi,double Pi)
   {air=new air_PG_CS();P=Pi; g=9.806; //m/s^2
   m=mi;
   D_fin = 60.0e-3;
    D = 18e-3;
    H=160e-3; //m fin
```

```
    W=103.93e-3;
    dx=0.8e-3;  //mm pipe thickness
        t_fin=0.79e-3; //fin thickness
    s_fin=6.85e-3;//72e-3; //distance between fins
    A_cross_pipe=Math.PI*D*D/4.0; //pipe inlet crossectional area
    A_pipe=Math.PI*D*s_fin; // pipe area
    A_in=Math.PI*D*(s_fin+t_fin);
    A_fin=W*H;//Math.PI*(D_fin*D_fin)/4.0*2.0;   // fin area
    L_pipe=1.0; //m
    n_pipe=12;
    L_total=L_pipe*n_pipe;
    n_fin=(int)(L_pipe/(t_fin+s_fin));
    k_pipe=360.0; //W/mK
    k_fin=220.0;// W/mK
    water= new steamIAPWS_IF97();
    T_water=new double[ n_fin*n_pipe];
    x_pipe=new double[ n_fin*n_pipe];
    P_water=200.0; //kPa
    eps=0.26e-3;       //surface rougness m cast iron
    eod=eps/D;
   }
   //Rayleigh number for plate
   public  double Ra_L(double Ts,double Tinf)
   {   double T=(Ts+Tinf)/2.0;
       a=air.property("tp",(T+273.15),P);
       avis=a[11];
       ak=a[12];
       aCp=a[7];
       aro=a[15];
       aPr=a[14];
      double beta=1/T;
      double alpha=ak/(aro*aCp);
      double Ra=g*beta*(Ts-Tinf)* D_fin*D_fin*D_fin/alpha/avis;
                return Ra;
   }

   public  double Ra_SH(double Ts,double Tinf)
   {   double T=(Ts+Tinf)/2.0;
       a=air.property("tp",(T+273.15),P);
       avis=a[11];
       ak=a[12];
       aCp=a[7];
       aro=a[15];
       aPr=a[14];
      double beta=1/T;
      double alpha=ak/(aro*aCp);
      double Ra=g*beta*(Ts-Tinf)* (s_fin/H)* (s_fin/H)*(s_fin/H)/alpha/avis;
                return Ra;
   }
     //Rayleigh number for horizontal pipe
   public  double Ra_D(double Ts,double Tinf)
   {   double T=(Ts+Tinf)/2.0;
       a=air.property("tp",(T+273.15),P);
       avis=a[11];
       ak=a[12];
       aCp=a[7];
       aro=a[15];
       aPr=a[14];
      double beta=1/T;
      double alpha=ak/(aro*aCp);
      double Ra=g*beta*(Ts-Tinf)* D* D* D/alpha/avis;
                return Ra;
   }
    //Fin efficiency
                public double eta_fin(double h)
                {
    double rb=D_fin/2.0;  //fin outside radius
    double rt=D/2.0;  //din radius
                double Rb=rb/rt;  //dimensionless pipe outside diameter
    double t=t_fin;  //fin thickness
    double Rt=1.0; //dimensionless fin diameter
    double M1=Math.sqrt(2.0*h/(k_fin*t))*rt;
    if_x eta=M->2.0/M*Rb/(1-Rb*Rb)*(Mathd.I(1.0,M)*Mathd.K(1.0,M*Rb)-
Mathd.K(1.0,M)*Mathd.I(1.0,M*Rb))/(Mathd.I(1.0,M)*Mathd.K(0.0,M*Rb)+Mathd.K(1.0,M)*Mathd.I(0.0,M*Rb));
   double eta1=eta.func(M1);
   return eta1;
    }
```

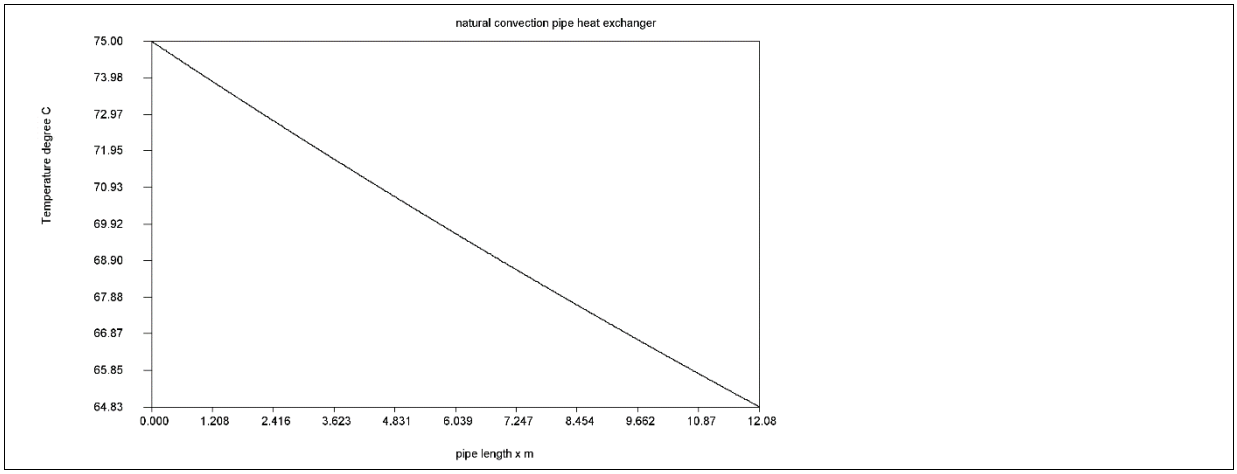
```
  public double Nu_SH(double Ts,double Tinf)
  {   double T=(Tinf+Ts)/2.0;
   a=air.property("tp",(T+273.15),P);
   double Ra=Ra_SH(Ts,Tinf);
   double Pr=a[14];
   double Gr=Ra/Pr;
   double Nu=0.0;
   double Racritic=1e9;
   double Pr_05=Math.sqrt(Pr);
   Nu=1.0/Math.sqrt(576/(Ra*Ra)+2.87/Math.sqrt(Ra));
   return Nu;
   }
  //Nusselt number
   //outside of thepipe Nu number
  public double Nu_L(double Ts,double Tinf,int select)
  {//Nusselt number
   double T=(Tinf+Ts)/2.0;
   a=air.property("tp",(T+273.15),P);
   double Ra=Ra_L(Ts,Tinf);
   double Pr=a[14];
   double Gr=Ra/Pr;
   double Nu=0.0;
   double Racritic=1e9;
   double Pr_05=Math.sqrt(Pr);
   if(Ra>Racritic || select==4 ) {Nu=(0.825+0.387*Math.pow(Ra,(1.0/6.0))/Math.pow((1+Math.pow((0.492/Pr),(9.0/16.0))),(8.0/27.0)));
                Nu=Nu*Nu;
               }
   else
   {if(select==1) Nu=4.0/3.0*Math.pow((Gr/4.0),(1.0/4.0))*0.75*Pr_05/Math.pow((0.609+1.221*Pr_05+1.238*Pr),(1.0/4.0));
    else if(select==2) Nu=4.0/3.0*Math.pow((Gr/4.0),(1.0/4.0))*(0.022830145+0.752624416*Pr_05+2.04e-
4*Pr)/Math.pow((1.222236219+1.010421489*Pr_05+1.389905434*Pr),(1.0/4.0));
    else if(select==3) Nu=(0.68+0.67*Math.pow(Ra,(1.0/4.0))/Math.pow((1+Math.pow((0.492/Pr),(9.0/16.0))),(4.0/9.0)));
   }

   return Nu;

   }
   //inside of the pipe Nu number
  public double Nu_D(double Ts,double Tinf,int select)
  {//Nusselt number of pipe
   double T=(Tinf+Ts)/2.0;
   double Ra=Ra_D(Ts,Tinf);
  a=air.property("tp",(T+273.15),P);
   double aPr=a[14];
   double Gr=Ra/aPr;
   double Nu=0.0;
   double Racritic=1e9;
   double C=0,n=0;
   double Pr_05=Math.sqrt(aPr);
   if(select==1) {
                double x1=0.387*Math.pow(Ra,(1.0/6.0));
                Nu=(0.60+0.387*Math.pow(Ra,(1.0/6.0))/Math.pow((1+Math.pow((0.559/aPr),(9.0/16.0))),(8.0/27.0)));
     Nu=Nu*Nu;
   }
   else if(select==2)
   {  if(Ra>=1e-10 && Ra<1e-2)    {C=0.675;n=0.058;}
     else if(Ra>=1e-2 && Ra<1e2) {C=1.02;n=0.148;}
     else if(Ra>=1e2 && Ra<1e4) {C=0.85;n=0.188;}
     else if(Ra>=1e4 && Ra<1e7)  {C=0.480;n=0.250;}
     else if(Ra>=1e7 && Ra<=1e12){C=0.125;n=0.333;}
               Nu=C*Math.pow(Ra,n);
   }
   return Nu;
   }
    public static double f_Goudar(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
```

```
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
  public static double Nu(double Re,double eod,double Pr)
 { //constant surface temperature internal flow
  double Nu=0;
  double x=Re/1000;
  double f=0;
  if(Re<2300) Nu=3.66;
  else if(Re>=2300 && Re<3100) Nu=2.2407*x*x*x*x-29.499*x*x*x+142.32*x*x-292.51*x+219.88;
  else {f=f_Goudar(Re,eod);
  Nu=(f/8)*(Re-1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8)*(Math.pow(Pr,(2.0/3.0))-1));
  }
 return Nu;
  }
  public double calculate(double T_air,double Twater_in)
  {  double TT= Twater_in;
     //double Ts=(10.0*T_air+1000.0*Twater_in)/1010.0;
     double Ts=(3.0*T_air+150*Twater_in)/153;
     double Q=0.0;
    T_water[0]=Twater_in;
              for(int i=0;i<n_fin*n_pipe-1;i++)
    {  x_pipe[i+1]=L_pipe/(n_fin-1)*i;
                a=air.property("tp",(T_air+273.15),P);
        b=water.propertyC("tp",T_water[i],P_water);
        ak=a[12];
       double h_D=Nu_D(Ts,T_air,1)*ak/D;
       double h_L=Nu_L(Ts,T_air,4)*ak/H;
       wvis=b[11];
       wk=b[12];
       wCp=b[7];
       wro=b[15];
       wPr=b[14];
       double V=wro*A_cross_pipe*m; //i nternal velocity
       double wRe=V*wro*D/wvis;                    // 's number
       double h_w=Nu(wRe,eod,wPr)*wk/D;                  // internal heat transfer coefficients W/m^2K h_w="+h_w);
       double eta=eta_fin(h_D);
       System.out.println("h_D="+h_D+" h_L="+h_L+" h_w="+h_w+"eta="+eta);
       double U1=(h_D*A_pipe+h_L*A_fin*eta_fin(h_L));          //
       double U=1.0/(A_in/U1+1.0/h_w);
       double dQ=U*A_in*(T_water[i]-T_air);
       T_water[i+1]=T_water[i]-dQ/(wCp*1e3*m);
       Q+=dQ;
    }
    return Q ;
  }
public static void main(String arg[])
{  double m=21.751e-3;
    double Pi=120.0;
             HT_pipe_finned_natural heater=new HT_pipe_finned_natural(m,Pi);
             double T_air=20.0; //degree C
             double Twater_in=75.0;
             double Q=heater.calculate(T_air,Twater_in);
             System.out.println("Q="+Q);
               double cc[][]={heater.x_pipe,heater.T_water};
                String s[]={"x m","water temp."};
                Text.printT(cc,s,"natural convection heat exchanger");
                Plot p=new Plot(heater.x_pipe,heater.T_water);
                p.setPlabel("natural convection pipe heat exchanger");
      p.setXlabel("pipe length x m");
      p.setYlabel("Temperature degree C");
                p.plot();
              }
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe" HT_pipe_finned_natural
Q=926.110009550031
```



Fin and tube heat exchanger thermal performance measurements are carried out according to EN 442-2 standard for testing radiators and convectors. According to this standard, Measurements are carried out for three different temperature zones

$$\Delta T = T_m - T_{room} = (30 \mp 2.5)K$$

$$\Delta T = T_m - T_{room} = (50 \mp 2.5)K$$

$$\Delta T = T_m - T_{room} = (60 \mp 2.5)K$$

Where T_m is the arithmetic average temperature between inlet and exit of water

$$T_m = \frac{T_{w_in} + T_{w_out}}{2}$$

And T_{room} is the room temperatures. Room temperature and experiment wall temperatures should be set to a constant temperature of 20 ℃. In order to carry out this test, a laboratuary design with the specification of standards is required. Test results will be fit into a simple curvefitting equation in the form of

$$Q = \dot{m}\left(h_{w_in} - h_{w_out}\right) = K_M \Delta T^n$$

Where Q is the heat transfer, $\dot{m}$ is the mass flow rate of water flowing through radiator, h_w is the water enthalpies at inlet and outlet. K_M and n are the cırve fitting coefficients obtained as a result of experiments. In order to reduce measurements uncertainities, measuremnts of each point should be carried out several times (minimum of three times). A laboratory system according to EN 442-2 is developed and a wide range of radiators are measured by using this facility. Measurement results and curve fitting coefficients are given in below table.

Test Raport No:	2022-227
Test Raport Date:	2.12.2022

Model Name	**Test date**	**S**	**B**	**Measuremen points**		
Fin and tube H.E. prototype	**2.12.2022**			**1**	**2**	**3**
Test Measurement info						
Air Pressure		**P**	**kPa**	**101.373**	**101.316**	**101.413**

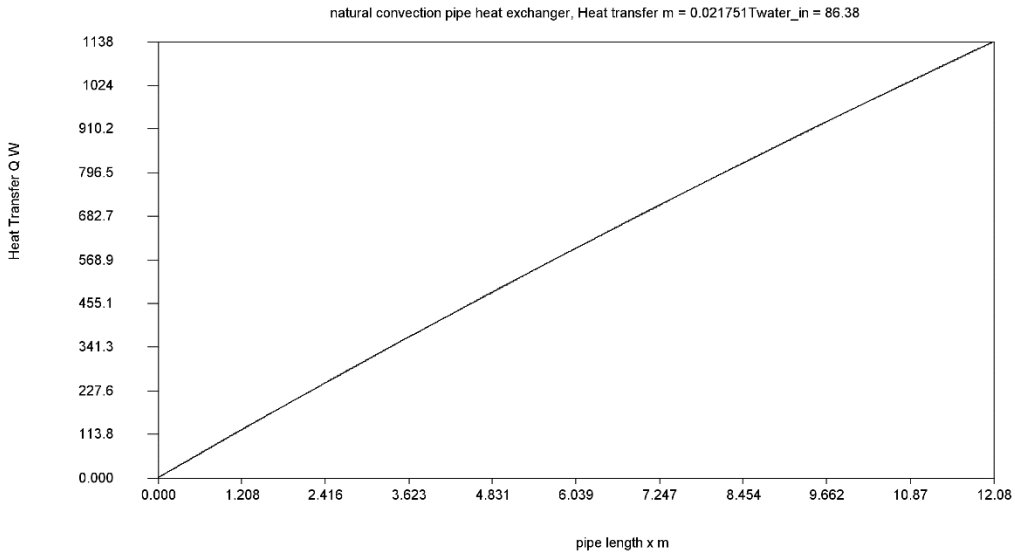
Referance Air Temperature	**t**	**°C**	**19.968**	**20.004**	**20.007**
Water Inlet Temperature	**t1**	**°C**	**86.380**	**74.987**	**52.406**
Water Outlet Temperature	**t2**	**°C**	**73.667**	**65.012**	**47.566**
Temperature Difference	**t1-t2**	**°C**	**12.713**	**9.975**	**4.840**
Inlet Water Enthalpy	**h1**	**j/kg**	**361.823**	**313.974**	**219.407**
Outlet Water Enthalpy	**h2**	**j/kg**	**308.443**	**272.169**	**199.165**
Enthalpy Difference	**Δh**	**j/kg**	**53.380**	**41.806**	**20.242**
Mean Water Enthalpy	**tm**	**°C**	**80.024**	**69.999**	**49.986**
Temperature difference	**Δt**	**°C**	**60.056**	**49.995**	**29.979**
Water mass flow rate	**qm**	**g/s**	**21.751**	**21.751**	**21.751**
Thermal Output Measured	**Φ**	**W**	**1161.1**	**909.3**	**440.3**
Thermal Output for standard air pressure	**Φme**	**W**	**1166.0**	**913.5**	**441.7**

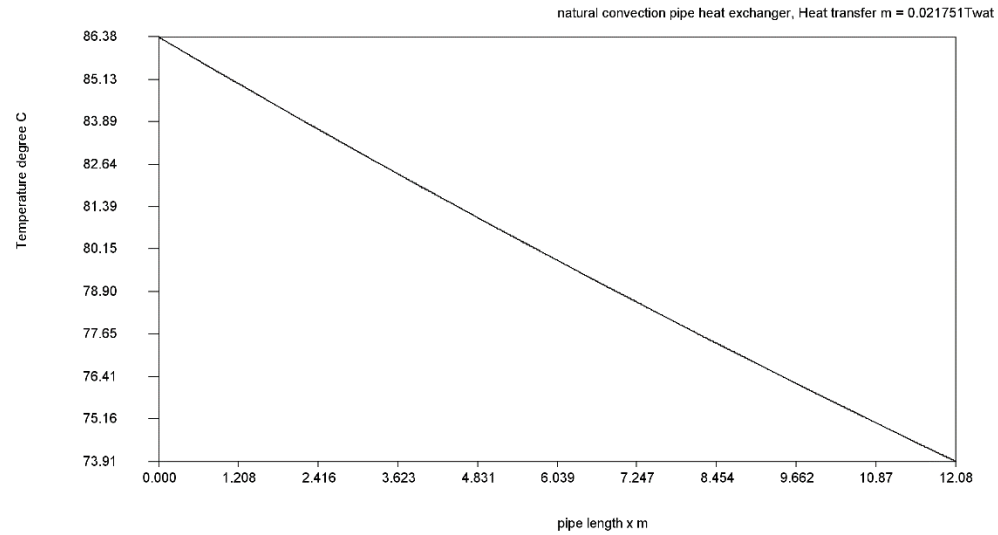
Km	**n**				
3.689111471	**1.407316089**	**47**	**Δ 60**	**Δ 50**	**Δ 30**
			1173.1	**907.6**	**442.3**

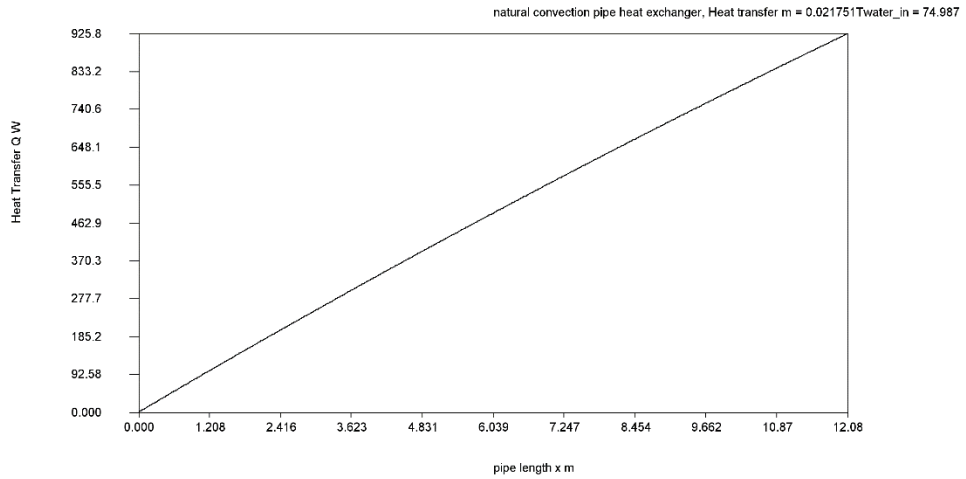
Prototype tube-fin heat exchanger has the following properties:

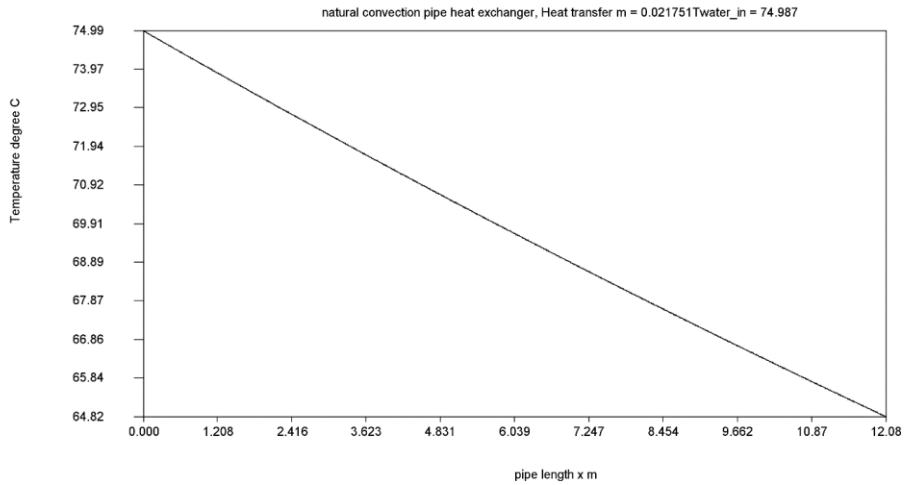
Water mass flow rate	$21.751x10^{-3}$	kg/s
Tube Length	1	m
Tube diameter	$18x10^{-3}$	m
HE Height	$160x10^{-3}$	m
HE Width	$103.93x10^{-3}$	m
Fin thickness	$0.79x10^{-3}$	m
Distance between fins	$6.85x10^{-3}$	m
Pipe thickness	$0.8x10^{-3}$	m
Number of tubes (connected serial)	12	
Pipe material	copper	
Fin material	Aliminium	

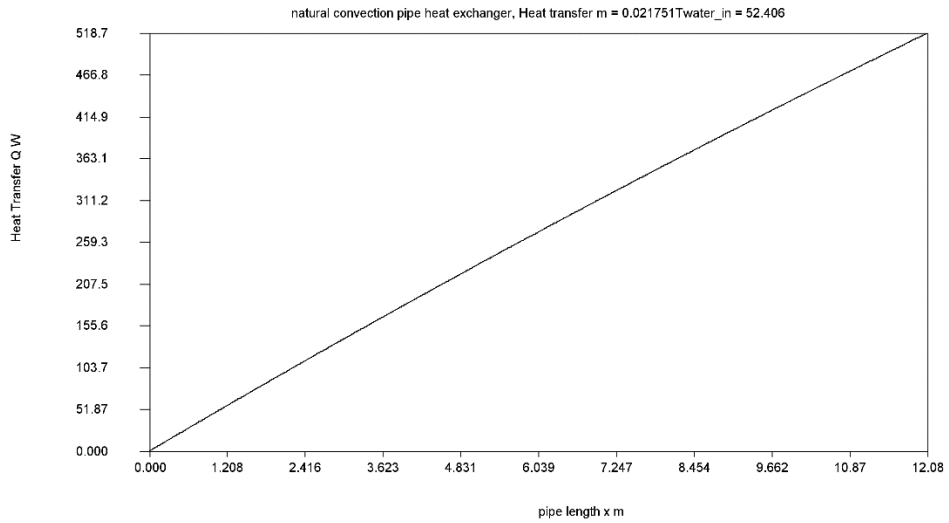
Temperature profile for water inside tubes.

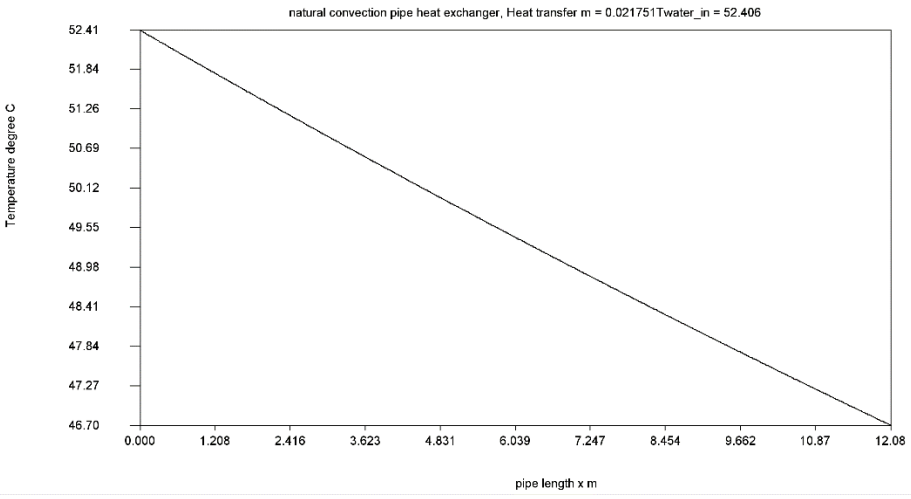

natural convection pipe heat exchanger, Heat transfer m = 0.021751Twater_in = 86.38
Heat Transfer Q W
1138
1024
910.2
796.5
682.7
568.9
455.1
341.3
227.6
113.8
0.000
0.000
1.208
2.416
3.623
4.831
6.039
7.247
8.454
9.662
10.87
12.08
pipe length x m


natural convection pipe heat exchanger, Heat transfer m = 0.021751Twat
Temperature degree C
86.38
85.13
83.89
82.64
81.39
80.15
78.90
77.65
76.41
75.16
73.91
0.000
1.208
2.416
3.623
4.831
6.039
7.247
8.454
9.662
10.87
12.08
pipe length x m


natural convection pipe heat exchanger, Heat transfer m = 0.021751Twater_in = 74.987
Heat Transfer Q W
925.8
833.2
740.6
648.1
555.5
462.9
370.3
277.7
185.2
92.58
0.000
0.000
1.208
2.416
3.623
4.831
6.039
7.247
8.454
9.662
10.87
12.08
pipe length x m







	Air Pressure	**P**	**kPa**	**101.373**	**101.316**	**101.413**
	Referance Air Temperature (measured and simulated)	**t**	**°C**	**19.968**	**20.004**	**20.007**
	Water Inlet Temperature (measured and simulated)	**t1**	**°C**	**86.380**	**74.987**	**52.406**

	Water Outlet Temperature (measured)	t2	°C	73.667	65.012	47.566
	Water Flow (measured and simulated)	qm	g/s	21.751	21.751	21.751
	Thermal Output (measured)	Φ	W	1161.1	909.3	440.3
	Thermal Output (Corrected)	Φme	W	1166.0	913.5	441.7

Km	n				
			Δ 60	Δ 50	Δ 30
3.689111471	1.407316089	47	1173.1	907.6	442.3

	Thermal Output (simulation)	Φme	W	1166.0	913.5	441.7
	Water outlet Temperature (simulation)	t2	°C	73.921	64.824	46.700

As a result, simulated temperature and heat transfer profile is closely representing the actual measured values. The model will be an imporant tool to predict performance and paremeter changes and their effects. It can be used as a first estimation toolsmfor this type of heat excahngers. Model used here has a specific geometry , bu simialar simualtion models can be developed for different geometries as well.

When a two phase heat exchanger finite difference model is designed, basic changing parameter should be enthalpy instead of temperature. An active equation of state should be part of the program to determine phase of fluid that changes the phase. In two phase region basic parameter to be change will be quality, in the one phase region, it still will be temperature.

$U_i = U_i(T_{hi}, T_{ci}, m_h, m_c, D_h, D_c, \varepsilon_h, \varepsilon_c, x)$ where $m_h \; and \; m_c$ are mass flow rates of fluids, D_h and D_c are dimeter or hydrolic diameter of pipes, $\varepsilon_h, \varepsilon_c$ are surface roughness. x is quality(will be $x \leq 0$ for liquid, $0 \leq x \leq 1$ for saturation region and x>1 for vapor. From conservation of energy for surface ΔA_i heat transfer for a condenser will be

$\Delta Q_i = -(m_h)_i[h_{hi+1} - h_{hi}] = \left(m_c C_{pc}\right)_i [T_{ci+1} - T_{ci}]$

$\Delta Q_i = U_i(T_{hi}, T_{ci}, m_h, m_c, D_h, D_c, \varepsilon_h, \varepsilon_c)\Delta A_i[T_{hi} - T_{ci}]$

$h_{hi+1} = h_{hi} - \frac{\Delta Q_i}{m_h}$ and $T_{ci+1} = T_{ci} - \frac{\Delta Q_i}{(m_c C_{pc})_i}$ from h_{hi+1} and Pressure calculate thermodynamic state and if it is one phase fluid (gas or vapor) determine temperature, if it is saturated mixture determine quality. Properties and convective heat transfer equation will be used accordingly

```
import javax.swing.JOptionPane;
```

```
import java.util.Locale;

public class doublepipecondenser
{ refrigerant hot_fluid;
  liquid_CS cold_fluid;  //(boru dışı)
  double d0,di; // inside pipe insede and outside diamecounter
  double D;     // outside pipe diameter
  double L;     // pipe length
  double mi;    // refISO mass flow rate
  double md;    // cooling fluid mass flow rate;
  double k;     // thermal conductivity
  double eps;
  double Tcold[];
  double Thot[];
  double x[];
  double dL;
  double dh;
  double Q;
  double dP;
  double Uortalama;
  double A0,Ai;
  double ts; //saturation temperature
  double Ri,R0;//fouling resistances;
  double dPi,dPd;//pressure drop;
  int N; //finite difference division
  double eod;    // eps/di
  double P;         // refrigerant pressure kPa
  public double xs;         //refrigerant quality x<0 liquid, 0<=x<=1 saturation x>1 gas
  double Cportalama;
  boolean phasechange=false;
  String phase;
  double LMTD;
  double Qlmtd;
  public doublepipecondenser(String fluid1, String fluid2,double Pi,double mii,double mdi,double d0i,double dii,double Di,double
Li,double dLi)
  {  //fluid1 : refISO
     //fluid2 : cooling fluid
     //mi    : mass flow rate refrigerant
     //md    : Mass flow rate cooling fluid
     //di    : inside diameter
     //d0    : outside diameter
     //D     :  Outside pipe inside diameter
     //L     : total pipe length;
     //dL    : finite difference length
   d0=d0i;di=dii;D=Di;L=Li;mi=mii;md=mdi;
   dL=dLi;
   P=Pi;
   N=(int)(L/dLi+0.000001);
   Tcold=new double[N+1];
   Thot=new double[N+1];
   x=new double[N+1];
   eps=45e-6;
   k=54.0;
   Ri=0.000176;
   R0=0.000352;
   hot_fluid=new refrigerant(fluid1);
   cold_fluid=new liquid_CS(fluid2);
   dh=D-d0;
   phase="liquid";
  }
  ///Diğer programdan
  // boru içi
 public double f(double Re)
 {
 //Haaland equation
 //double f1=-1.8*Math.log10(Math.pow((eod/3.7),1.11)+6.9/ Re);
 //f1=1.0/(f1*f1);

 //Swamee-Jain equation
 //double A=Math.log10(eod/3.7+5.74/Math.pow(Re,0.9));
 //double f1=0.25/(A*A);

 //Serghides equation
 //double A1=-2.0*Math.log10(eod/3.7+12.0/Re);
 //double B1=-2.0*Math.log10(eod/3.7+2.51*A1/Re);
 //double C1=-2.0*Math.log10(eod/3.7+2.51*B1/Re);
 //double f1=A1-((B1-A1)*(B1-A1))/(C1-2.0*B1+A1);
```

```
 //f1=1.0/(f1*f1);
 //Goudar equation
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
//boru içi
public double fx(double X,double Re)
{
// colebrook equation to solve
// friction factor for turbulent flow 2000 < Re
double xx=2.0*Math.log10(eod/3.7+2.51/Re*X)+X;
return xx;
}
//boru içi
public double dfx(double X,double Re)
{
//derivative of colebrook equation
double xx;
xx = 1+2.0/(eod/3.7+2.51/Re*X)/Math.log(10.0)*2.51/Re;
return  xx;
}
//boru içi
public  double fcol(double Re)
{
// solution of the colebrook equation
// by using newton method
double fi=f(Re);
double xx=1.0/ Math.pow(fi,0.5);
int nmax=50;
double tolerance=1.0e-10;
for(int i=0;i<nmax;i++)
{
double fx1=fx(xx,Re);
xx-=fx1/dfx(xx,Re);
if(Math.abs(fx1)<tolerance)
{double ff=1.0/(xx*xx);
return ff;}
}
double ff=1.0/(xx*xx);
return ff;
}

//boru içi
public double  velocity(double t)
{
// avarage velocity of the pipe m/x
double x=0.0;
double mboru=mi;
double a[];
if(xs>=0 && xs<=1) a=hot_fluid.property("tx",t,xs);
else               a=hot_fluid.property("tp",t,P);
double ro=a[7];
double A=Math.PI*di*di/4.0;
return mboru/ro/A;
}

public double x(double xs)
{double x=0;
if(xs>1) x=1;
else if(xs<0) x=0;
else x=xs;
return x;
}

public double  velocityl(double t)
{
// avarage velocity of the pipe m/x
```

```
double x=0.0;
double mboru=mi;
double a[];
double rol=hot_fluid.r.rol(t);
double A=Math.PI*di*di/4.0;
return (1-x(xs))*mboru/rol/A;
}
public double  velocityv(double t)
{
// avarage velocity of the pipe m/x
double x=0.0;
double mboru=mi;
double a[];
double rov=hot_fluid.r.rov(t);
double A=Math.PI*di*di/4.0;
return x(xs)*mboru/rov/A;
}
//boru içi  Ref
public double  Re(double t)
{
return Rel(t)+Rev(t);
}
public double  Rel(double t)
{
// Reynold's number boru içi liquid
double rol=hot_fluid.r.rol(t);
double visl=hot_fluid.r.viscosity_l(t);
double vv=velocityl(t);
double res=(di*vv*rol/visl);
return res;
}
public double  Rev(double t)
{
// Reynold's number boru içi vapor
double a[];
double rov=hot_fluid.r.rov(t);
double visv=hot_fluid.r.viscosity_v(t);
double vv=velocityv(t);
double res=(di*vv*rov/visv);
return res;
}

//boru içi
public double  hc(double t)
{
//convective heat transfer coefficient
double Rey;
double kt;
double Nu=0;
double Prt=0;
double ss=0;
if(xs>=1) {  kt=hot_fluid.r.k_v(t);Rey=Rev(t);Prt=hot_fluid.r.Pr_v(t);}
else if(xs<=0) {kt=hot_fluid.r.k_l(t);Rey=Rel(t);Prt=hot_fluid.r.Pr_l(t);}
else
{ kt=(hot_fluid.r.k_l(t)*(1-x(xs))+hot_fluid.r.k_v(t)*x(xs));Rey=Rel(t);Prt=(hot_fluid.r.Pr_v(t)*x(xs)+hot_fluid.r.Pr_l(t)*(1.0-
x(xs)));}
if(xs>=1 || xs<=0)  // single phase heat transfer
{
if(Rey<=2300)  {Nu=3.66;}    //Laminar region
else if(Rey>2300 && Rey<=3100)
{  //Referans : Heat transfer in all pipe flow regimes:laminar,transitional/intermittent, and turbulent
   // J.P. Abraham, E.M. Sparrow, J.C.K. Tong, International Journal of Heat and Mass Transfer 52(2009) 557-563
              double Re1=(Rey/1000.0);
              Nu=2.2407*Re1*Re1*Re1*Re1-29.499*Re1*Re1*Re1+142.32*Re1*Re1-292.51*Re1+219.88;
}
else   //one phase heat transfer
{ double fr  = f(Rey);
  //Prt = hot_fluid.r.Pr(t);
  // Turbulent region Gnielinski equation
  // Referance:
  // V. Gnielinski, New equations for heat and mass transfer in turbulent pipe and
  //channel flow, Int. Chem. Eng. 16 (1976) 359–367.
  // Valid for 2000 < Re <5e5, 0.5 < Pr < 2000
  Nu=fr/8.0*(Rey-1000.0)*Prt/
  (1+12.7* Math.sqrt(fr/8.0)*( Math.pow(Prt,(2.0/3.0))-1.0));
}
```

```
  if(xs>=1)  kt=hot_fluid.r.k_v(t);
  if(xs<=0)  kt=hot_fluid.r.k_l(t);
  ss=Nu*kt/di;
}
else
{//Travis ve arkadaşları formülü
double a[]=hot_fluid.property("tx",t,xs);
double ro=a[7];
double rov=hot_fluid.r.rov(t);
double rol=hot_fluid.r.rol(t);
double visv=hot_fluid.r.viscosity_v(t);
double visl=hot_fluid.r.viscosity_l(t);
double vis=(1.0-xs)*visl+xs*visv;
Rey=Rel(t);
double Xtt=Math.pow(((1-xs)/xs),0.9)*Math.sqrt(rov/ro)*Math.pow((vis/visv),0.1);
double Ftt=0.15*(1.0/Xtt+2.85*Math.pow(Xtt,-0.467));
double F2=0;
double x1=0;
if(Rey<50)
{F2=0.707*Prt*Rey;}
else if(Rey>=50 && Rey<=1125)
{x1=1+Prt*(0.09636*Math.pow(Rey,0.585)-1.0);F2=5.0*Prt+5.0*Math.log(x1);}
else
{x1=0.00313*Math.pow(Rey,0.812);F2=1.0+5*Math.log(1+5.0*Prt)+2.5*Math.log(x1);}
Nu=Prt*Math.pow(Rey,0.9)/F2*Ftt;
kt=hot_fluid.r.k_l(t);
ss=Nu*kt/di;
}
  return ss;
}
  ///=======================
 public  double f(double Re,double eod)
 {
 //Goudar Sonnad equation
 // for Re>2100 and any e/D
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
 //boru dışı ısı transferi
  public double Vd(double t)
  { double ro=cold_fluid.ro(t);
    double Ad=Math.PI*(D*D-d0*d0)/4.0;
    double v=md/ro/Ad;
    return v;
  }

  public double Red(double t)
  {
   double mud=cold_fluid.vis(t);
   double ro=cold_fluid.ro(t);
   double Vd=Vd(t);
   double Re=ro*Vd*dh/mud;
   return Re;
  }
  public  double dPd(double t)
  {
   double Re=Red(t);
   double eod=eps/d0;
   double f=f(Re,eod);
   double ro=cold_fluid.ro(t);
   double v=Vd(t);
   double dP=dL/d0*f*ro*v*v/2.0;
   return dP;
  }

 public double Nud(double t)
 {
```

```
   double eod=eps/dh;
   double Re=Red(t);
              double f=f(Re,eod);
              double Pr=cold_fluid.Pr(t);
              double Nu=(f/8.0)*(Re-1000.0)*Pr;
              double A=1.0+12.7*Math.sqrt(f/8.0)*(Math.pow(Pr,(2.0/3.0))-1.0);
              Nu/=A;
              return Nu;
}

public double hd(double t)
{  double Nu=Nud(t);
   double k=cold_fluid.k(t);
   double h=Nu*k/dh;
              return h;
}

public double U0(double Ti,double Td)
{ double Ai=Math.PI*di*di/4.0;
  double A0=Math.PI*d0*d0/4.0;
  double hi=hc(Ti);
  double hd=hd(Td);
  double U=A0*(1.0/(hi*Ai)+Ri/Ai+Math.log(d0/di)/(2.0*Math.PI*k*dL)+R0/A0+1.0/(hd*A0));
  return 1.0/U;
}

public String toString()
{String s="";
 s+="saturation liquid (internal pipe) = "+hot_fluid.r.refrigerantName+"\n";
 s+="cold liquid (outside pipe) = "+cold_fluid.liquidName+"\n";
 s+="saturation liquid temperatures (internal pipe)  = "+Thot[0]+"    "+Thot[N]+"\n";
 s+="cold liquid temperatres (outside pipe)  = "+Tcold[0]+"    "+Tcold[N]+"\n";
 //s+="sıcak sıvı pressure drop (outside pipe)  = "+dPd+" Pa\n";
 //s+="soğuk sıvı pressure drop (internal pipe)  = "+dPi+" Pa\n";
 s+="Heat transfer = "+Q+"Watt \n";
 s+="surface area = "+Math.PI*d0*L+" m^2 \n";
 return s;
}

public double calculate1(double tic, double tdis1)
{  //tic : refISO inlet temperature degree C
   //tdis1: cooling water inlet temp degree C
              Q=0;
              double xx=0;
   double dtic=0;
   double dhic=0;
   double ticp1=0;
   double hicp1=0;
   double hic=0;
   double xsp1=0;
   double xsp2=0;
   double hv=0,hf=0;
   double Cps,CpsP1;
   double a[],a1[],a2[],a3[];
              double dq=0;
              double roavg=0;
              double Vavg=0;
              double h1,h2;
              double rocold=cold_fluid.ro(tdis1);
   Ai=Math.PI*di*L;
   A0=Math.PI*d0*L;
   a1=hot_fluid.property("px",P,1.0);
   a2=hot_fluid.property("px",P,0.0);
   ts=a1[1];
   if(ts>tic) {phase="liquid";phasechange=false;}
   else if(ts<tic) {phase="vapor";phasechange=false;}
   hv=a1[3];
   hf=a2[3];
   a=hot_fluid.property("tp",tic,P);
   h1=a[3];
   xsp2=(h1-hf)/(hv-hf);
   xs=xsp2;
   a=hot_fluid.property("tp",tic,P);
   a1=hot_fluid.property("tp",(tic-0.1),P);
   Cps=(a[3]-a1[3])/0.1;
   h1=a[3];
   hic=h1;
```

```
            dP=0;
    double dxs=0;
    double dtdis1=0;
    double Cp=0;
    Uortalama=0;
    Vavg=0;
    double uu=0;
    double dA=A0/N;
    double dAi=Ai/N;
    double dq1=0;
    Thot[0]=tic;
            Tcold[0]=tdis1;
            x[0]=xx;
            double Dq1=0;
            Q=0;
            for(int i=1;i<=N;i++)
            {   Cp=cold_fluid.Cp(tdis1);
                Cportalama+=Cp;
                        double Rey=Re(tic);
                double f=fcol(Rey);
                Vavg+=velocity(tic);
                        uu=U0(tic,tdis1);
                        Uortalama+=uu;
                        dq=uu*1e-3*dAi*(tic-tdis1);
                        dq1+=dq;
                        dhic=dq/mi;
                        hicp1=hic+dhic;
                        xsp1=(hicp1-hf)/(hv-hf);
                        if(xsp1<0) {phasechange=false;phase="liquid";}
                        else if (xsp1>1) {phasechange=false;phase="vapor";}
                        else {phasechange=true;phase="saturated";}
                        if(!phasechange)  { ticp1=tic+dhic/Cps;}
        else            {a=hot_fluid.property("px",P,xsp1); ticp1=a[1];}
        hic=hicp1;
                        //cold liquid
                        Dq1=mi*dhic;
                dtdis1=Dq1/(md*Cp);
                Q+=dq1;
                tdis1+=dtdis1;
        Thot[i]=tic;
                Tcold[i]=tdis1;
        xx+=dL;
        x[i]=xx;
        tic=ticp1;
        xs=xsp1;
        //System.out.println("phase = "+phase+"xx="+xx+"ts="+ts+"tic="+tic+"tdis1 = "+tdis1+"hv="+hv+"hf="+hf+" hic="+hic+" pc
"+phasechange+" xsp1="+xsp1+" xs="+xs);
            }
    roavg/=N;
    Vavg/=N;
    Uortalama/=N;
    Cportalama/=N;
    return tic;
}
public void calculate_counter(double tic, double tdis)
{
//counter flow system
//tic  : cold liquid inlet temperature
//tdis : hot liquid inlet temperature

double dT=(ts-tdis)/30;
double tin[]=new double[25];
double tout[]=new double[25];

for(int i=0;i<25;i++)
{tout[i]=tdis+dT*(i+1)+30.0;
 tin[i]=calculate1(tout[i],tdis);
 System.out.println("i="+i+"tout="+tout[i]+"tin="+tin[i]);
}
double e[]=EKK(tin,tout,7);
double tic1=funcEKK(e,tic);
double tic2=calculate1(tic1,tdis);
    double dT1=Thot[0]-Tcold[0];
    double dT2=Thot[N-1]-Tcold[N-1];
LMTD=(dT1-dT2)/Math.log(dT1/dT2);
Qlmtd=Uortalama*A0*LMTD;
}
```

```
//EKK
//en kucuk kareler metodu
  public static double[] pivotlugauss(double a[][],double b[])
  { //kısmi pivotlu gauss eleme yöntemi
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss eleme
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivotlama
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                   if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
                  b[k]=dummy;
                }
                //gauss elemeyi çözme
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
                }
  }
  //geriye doğru yerine koyma
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}



public static double[] EKK(double xi[],double yi[],int n)
{
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  for(j=0;j<n+1;j++)
     {if(i==0 && j==0) A[i][j]=l;
     else  for(k=0;k<l;k++) A[i][j] += Math.pow(xi[k],(i+j));
     }
     for(k=0;k<l;k++) { if(i==0)  B[i]+= yi[k];
                   else      B[i] += Math.pow(xi[k],i)*yi[k];}
  }
//System.out.println(Matrix.toString(A));
//System.out.println(Matrix.toStringT(B));
X=pivotlugauss(A,B);
```

```
//X=B/A;
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
return X;
}

public static double funcEKK(double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=e.length;
double ff;
if(n!=0.0)
  { ff=e[n-1];
  for(int i=n-2;i>=0;i--)
   { ff=ff*x+e[i]; }
  }
else
  ff=0;
return ff;
}

public static double hata(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcEKK(e,x[k]))*(y[k]-funcEKK(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

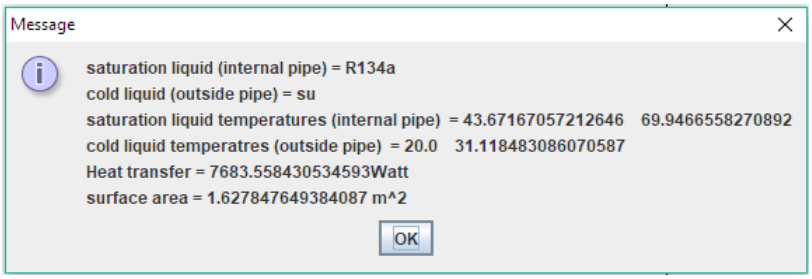
public static double[][]  funcEKK(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
{
//aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
// ara değer sayısı 2 dir
   int n=xi.length;
   int nn=(n-1)*(aradegersayisi+1)+1;
   double z[][]=new double[2][nn];
   double E[]=EKK(xi,yi,polinomkatsayisi);
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
      {z[0][k]=xi[i];z[1][k]=funcEKK(E,z[0][k]);k++;
      for(int j=0;j<aradegersayisi;j++)
      {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
      z[0][k]=z[0][k-1]+dx;z[1][k]=funcEKK(E,z[0][k]);k++;}
      }
      z[0][k]=xi[i];z[1][k]=funcEKK(E,z[0][k]);
   return z;
 }
 public void plot()
{
 Plot pp=new Plot(x,Tcold);
 pp.addData(x,Thot);
 Locale us=new Locale("US");
 String s1="Q = %4.1f kW A = %4.1f m^2 dP = %4.1f kPA U = %4.0f W/m^2K";
 //String s4=String.format(us,s1,Q*1e-3,A0);
 pp.setPlabel("double pipe condenser");
 pp.setXlabel("x, m");
 pp.setYlabel("T temperature , degree C");
 pp.plot();
}

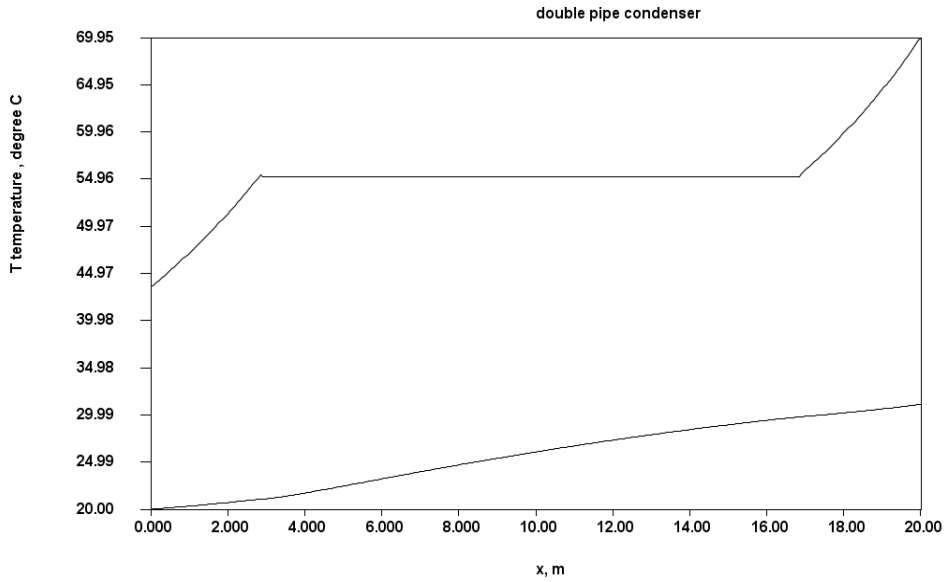
 public static void main(String arg[])
 {  String fluid1="R134a";
    String fluid2="su";
    double Pi=1500.0; //kPa
    //di    : inside diameter
```

```
    double di=25.4e-3;
    double d0=di*1.02;
    double D=1.2*d0;
    double L=20.0;
    double dL=0.05;
    double mi=0.2; //kg/s
    double md=0.8; //kg/s;
     //d0    : outside diameter
     //D     :  Outside pipe inside diameter
     //L     : total pipe length;
     //dL    : finite difference length
               doublepipecondenser iy=new doublepipecondenser(fluid1,fluid2,Pi,mi,md,d0,di,D,L,dL);
    double tdis=20.0;
    double tic=70;
    iy.calculate_counter(tic,tdis);
    iy.plot();
    JOptionPane.showMessageDialog(null,""+iy);

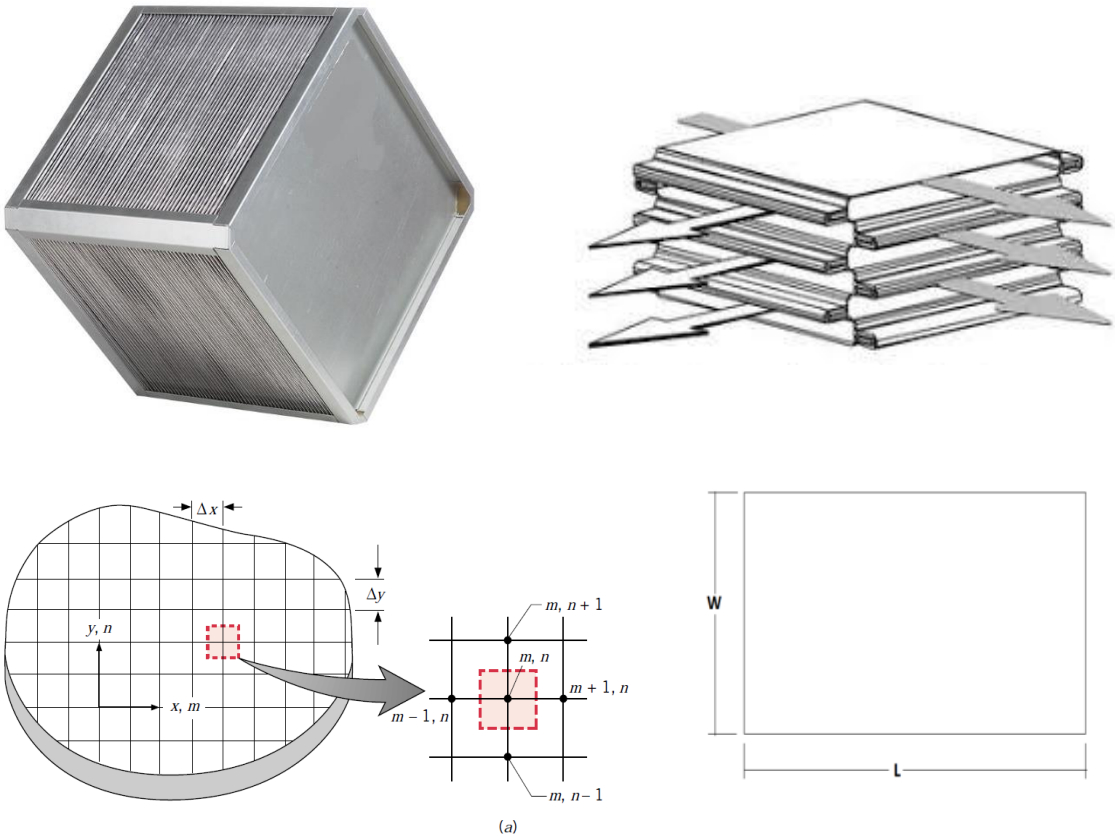
               }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" doublepipecondenser
i=0tout=49.33333333333333tin=83.7625122497982
i=1tout=48.66666666666667tin=81.13717141300293
i=2tout=48.0tin=79.77844621840916
i=3tout=47.33333333333333tin=78.64875546855869
i=4tout=46.66666666666667tin=76.77207535750125
i=5tout=46.0tin=75.00567494331504
i=6tout=45.333333333333336tin=73.6733194237354
i=7tout=44.66666666666667tin=72.71120003438882
i=8tout=44.0tin=70.48355439330206
i=9tout=43.333333333333336tin=69.58201239349478
i=10tout=42.66666666666667tin=68.18803099955473
i=11tout=42.0tin=66.80060461199444
i=12tout=41.333333333333336tin=65.73364561235323
i=13tout=40.66666666666667tin=64.28606408927867
i=14tout=40.0tin=62.98735151866717
i=15tout=39.333333333333336tin=61.833038394321015
i=16tout=38.66666666666667tin=60.660785045863776
i=17tout=38.0tin=59.19679514902403
i=18tout=37.333333333333336tin=58.3363361340472
i=19tout=36.66666666666667tin=56.96242061158853
i=20tout=36.0tin=55.84439441539124
i=21tout=35.333333333333336tin=55.232788492883934
i=22tout=34.66666666666667tin=55.232788492883934
i=23tout=34.0tin=55.232788492883934
i=24tout=33.333333333333336tin=55.232788492883934
```





Now lets look at a two dimensional finite difference heat exchanger problem. In order to recover waste heat from exhaust gases or worm air going out of building we use energy recovery heat exchangers called recuperators. Recuperators are cross flow heat exchangers. By considering a two two dimensional grid, we can investigate temperature profile and energy distribution of this devices. In the representation basic geometry of the finite difference model is shown Plates has an area of W*X and we will be looking two dimensional grid m,n



The basic heat transfer equations can be written as (heat transfer in z direction)

$\Delta Q_{m,n} = U_{m,n}\Delta A_{m,n}\left[T_{hm,n} - T_{c,m,n}\right]$

$x = m * \Delta x \quad y = n * \Delta y$

Boundary conditions for the problem

$T_{hm,n} = T_{y0} = constant$

$T_{cm,n} = T_{x0} = constant$
Note that instead of constant a variable profile can be taken.
Thickness of the plate = Δz
Due to symmetry between the plates $\frac{\Delta z}{2}$ distance will be taken in each side of a single plate and adiabatic boundary will be assumed at this thickness. Therefore mass flow rate for the channels will be taken as $\Delta m_{hm,n} = \frac{1}{2}\frac{m_h}{L}$ $\Delta m_{cm,n} = \frac{1}{2}\frac{m_c}{W}$ if it is assumed that W=L then $\Delta m_{cm,n} = \frac{1}{2}\frac{m_c}{L}$ also $\Delta x = \Delta y$

$\Delta m_{hm,n} = \rho_h(T_{hm,n})\frac{\Delta z}{2}\Delta x V_{hm,n}$ $\Delta m_{cm,n} = \rho_c(T_{cm,n})\frac{\Delta z}{2}\Delta x V_{cm,n}$
$\Delta Q_{m,n} = \Delta m_{hm,n} C_{phm,n}(T_{h(m+1),n} - T_{hm,n})$ $\Delta Q_{m,n} = \Delta m_{cm,n} C_{pcm,n}(T_{hm,(n+1)} - T_{hm,n})$
Since temperature of one corner is known for both cold and hot sides, all temperature and heat transfer profile can be calculated.

PROBLEM: A recuperator has the following properties: $m_h = 0.636\ kg/s$ $m_c = 0.54\ kg/s$ W=0.5 m L=0.5 m channel heigh dZ=2.25 mm, number of plates N=140 and temperature are 80 C and 30 C. Calculate temperature profile and heat transfer

```
import javax.swing.JOptionPane;
import java.util.Locale;

public class recuperator1
{ // recuperator
  Gas hot_sivi; // cold liquid
  Gas cold_sivi;      // hot liquid
  double Thot[][];
  double Tcold[][];
  double x[];      // x dimension coordinates
  double y[];      //y dimension coordinates
  double Q;        // heat transfer
  double Qlmtd;    // heat transfer LMTD
  double LMTD;
  double KL;         // local pressure drop coefficient
  double A;          // surface area
  double mhot;     //  mass flow rate
  double mcold;   //  mass flow rate
  double dmhot;
  double dmcold;
  double Ric,Rdis;  // fouling resistances
  double klevha;     // thermal conductivity of plate
  double dL;        // finite difference step size
  int n_x;     // finite difference x direction number of states
  int n_y;     // finite difference y direction number of states
  double L,W,dz,Z;
  int nplate;
  double kboru;
  double platethickness;
  double eod=1e-4;
  double Thotin,Thotout;
  double Tcoldin,Tcoldout;
  double UU[][];
 public recuperator1(String ihot_sivi,
 String icold_sivi,double mhoti,double mcoldi,double Li, double Wi,double dzi,double dLi,int nplatei)
 {hot_sivi=new Gas(ihot_sivi);
  cold_sivi=new Gas(icold_sivi);
  L=Li;
  W=Wi;
  dL=dLi;
  n_x=(int)(L/dL)+1;
  n_y=(int)(W/dL)+1;
  W=Wi;
  dL=dLi;
  nplate=nplatei;
  dz=dzi;
  Z=dz*nplate;
  mhot=mhoti;
  mcold=mcoldi;
  dmhot=mhot/nplate;
  dmcold=mcold/nplate;
  kboru=210.0; //(w/mK)
```

```
  platethickness=1e-3; //m
  System.out.println("n_x="+n_x+"n_y="+n_y+"L="+L+"W="+W);
  Thot=new double[n_x][n_y];
  Tcold=new double[n_x][n_y];
  UU=new double[n_x][n_y];
  Thotin=0;
  Thotout=0;
  Tcoldin=0;
  Tcoldout=0;
  }
  // eps    : pipe equivalent roughness m
  //         ------------------------------------------
  //         value of equivalent roughness for some pipes:
  //         Pipe            equivalent rougness m
  //         ------------    --------------------
  //         Riveted steel    0.9e-3 - 9e-3
  //         Concrete         0.3e-3 - 3e-3
  //         Wood stave       0.18e-3 - 0.9 e-3
  //         Cast iron        0.26e-3
  //         Galvanised iron  0.15e-3
  //         Commercial steel 0.045e-3
  //         wrought iron     0.045e-3
  //         drawn tubing     0.0015e-3
  //         Plastic,glass    0.0 (smooth)
  //         ------------------------------------------
// boru içi
 public double f(double Re)
 {
 //Haaland equation
 //double f1=-1.8*Math.log10(Math.pow((eod/3.7),1.11)+6.9/ Re);
 //f1=1.0/(f1*f1);

 //Swamee-Jain equation
 //double A=Math.log10(eod/3.7+5.74/Math.pow(Re,0.9));
 //double f1=0.25/(A*A);

 //Serghides equation
 //double A1=-2.0*Math.log10(eod/3.7+12.0/Re);
 //double B1=-2.0*Math.log10(eod/3.7+2.51*A1/Re);
 //double C1=-2.0*Math.log10(eod/3.7+2.51*B1/Re);
 //double f1=A1-((B1-A1)*(B1-A1))/(C1-2.0*B1+A1);
 //f1=1.0/(f1*f1);
 //System.out.println("Sergides equation f1="+f1);
 //Goudar equation
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 //System.out.println("Goudar equation f1="+f1);
 return f1;
 }
//boru içi

//boru içi
public double  velocity_x(double t)
{
// velocity of the pipe m/x
double x=0.0;
double mboru=dmhot/2.0;
double T=t+273.15;
double aa[]=hot_sivi.property(T,1.01325);
double ro=aa[15]/aa[2];
double A=W*dz/2.0;
return (mboru/ro/A);
}
public double  velocity_y(double t)
{
// velocity of the pipe m/x
double x=0.0;
double mboru=dmcold/2.0;
```

```
double T=t+273.15;
double aa[]=cold_sivi.property(T,1.01325);
double ro=aa[15]/aa[2];
double A=L*dz/2.0;
return (mboru/ro/A);
}
//boru içi
public double  Re_x(double t)
{
// reynold's number boru içi
double T=t+273.15;
double aa[]=hot_sivi.property(T,1.01325);
double ro=aa[15]/aa[2];
double vis=aa[13];
double vv=velocity_x(t);
double A=W*dz/2.0;
double P=2.0*W+dz;
double dh=4*A/P;
double res=(dh*vv*ro/vis);
return res;
}
public double  Re_y(double t)
{
// reynold's number boru içi
double T=t+273.15;
double aa[]=cold_sivi.property(T,1.01325);
double ro=aa[15]/aa[2];
double vis=aa[13];
double vv=velocity_x(t);
double A=W*dz/2.0;
double P=2.0*W+dz;
double dh=4*A/P;
double res=(dh*vv*ro/vis);
return res;
}
//boru içi
public double  h_x(double t)
{
//convective heat transfer coefficient
double Rey=Re_x(t);
double Nu=0;
double Prt=1.0;
double T=t+273.15;
double aa[]=hot_sivi.property(T,1.01325);
double A=W*dz/2.0;
double P=2.0*W+dz;
double dh=4*A/P;
if(Rey<2300)
  { Nu=3.66; }
else
  {
  //double fr  = f(m,t);
  double fr  = f(Rey);
  //System.out.println("f="+fr);
  Prt = aa[16];
  // Gnielinski equation
  // Gnielinski, V, Int. Chem Eng., 16, 359,1976
  // Valid for 2000 < Re <5e5, 0.5 < Pr < 2000
  Nu=fr/8.0*(Rey-1000.0)*Prt/
  (1+12.7* Math.sqrt(fr/8.0)*( Math.pow(Prt,(2.0/3.0))-1.0));
  }
  double kt  =  aa[14];
  double ss=Nu*kt/dh;
  //System.out.println("Re="+Rey+"Nu="+Nu+"h="+ss+"Prt="+Prt+"kt="+kt+"di="+di);
  return ss;
}
public double  h_y(double t)
{
//convective heat transfer coefficient
double Rey=Re_y(t);
double Nu=0;
double Prt=1.0;
double T=t+273.15;
double A=L*dz/2.0;
double P=2.0*L+dz;
double dh=4*A/P;
double aa[]=cold_sivi.property(T,1.01325);
```

```
if(Rey<2300)
  { Nu=3.66; }
else
  {
  //double fr  = f(m,t);
  double fr  = f(Rey);
  //System.out.println("f="+fr);
  Prt = aa[16];
  // Gnielinski equation
  // Gnielinski, V, Int. Chem Eng., 16, 359,1976
  // Valid for 2000 < Re <5e5, 0.5 < Pr < 2000
  Nu=fr/8.0*(Rey-1000.0)*Prt/
  (1+12.7* Math.sqrt(fr/8.0)*( Math.pow(Prt,(2.0/3.0))-1.0));
  }
  double kt  =  aa[14];
  double ss=Nu*kt/dh;
  //System.out.println("Re="+Rey+"Nu="+Nu+"h="+ss+"Prt="+Prt+"kt="+kt+"di="+di);
  return ss;
}

public double U0(double t_x,double t_y)
{
double hi=h_x(t_x);    // boru içi ısı transferi
double ho=h_y(t_y); // boru dışı ısı transferi
double Uboru=1.0/(1.0/ho+1.0/hi+platethickness/kboru);
//System.out.println("hi="+hi+"h0="+ho+"Uboru="+Uboru);
return Uboru;
}

public void hesapla(double t_x, double t_y)
{    Thotin=t_x;
     Tcoldin=t_y;
               double Q=0;
               double dQ=0;
    double dA=dL*dL;
    double Cphot=0.0;
    double Cpcold=0.0;
    double U=0;

    int i,j;
               for(j=0;j<n_y;j++)
               {Thot[0][j]=t_y;}
               for(i=0;i<n_x;i++)
               {Tcold[i][0]=t_x;}

               for(j=1;j<n_y;j++)
               {for(i=1;i<n_x;i++)
                {
                              U=U0(Thot[i-1][j-1],Tcold[i-1][j-1]);
                  UU[i][j-1]=U;
                  if(j==1 && i==1) UU[i-1][j-1]=U;
                  dQ=U*dA*(Tcold[i-1][j-1]-Thot[i-1][j-1]);
      double aa[]=hot_sivi.property((Thot[i-1][j-1]+273.15),1.01325);
      Cphot=aa[9]/aa[15]*1e3;
                  Thot[i][j-1]=Thot[i-1][j-1]+dQ/(Cphot*dmhot/2.0);
    }
    for(i=1;i<n_x+1;i++)
    {
               U=U0(Thot[i-1][j-1],Tcold[i-1][j-1]);
               UU[i-1][j]=U;
               dQ=U*dA*(Tcold[i-1][j-1]-Thot[i-1][j-1]);
               Q+=dQ;
               double aa1[]=cold_sivi.property((Tcold[i-1][j-1]+273.15),1.01325);
     Cpcold=aa1[9]/aa1[15]*1e3;
               Tcold[i-1][j]=Tcold[i-1][j-1]-dQ/(Cpcold*dmcold/2.0);
               }
    }

    j=n_y;
    for(i=1;i<n_x;i++)
               {U=U0(Thot[i-1][j-1],Tcold[i-1][j-1]);
               UU[i][j-1]=U;
               dQ=U*dA*(Tcold[i-1][j-1]-Thot[i-1][j-1]);
     double aa[]=hot_sivi.property((Thot[i-1][j-1]+273.15),1.01325);
     Cphot=aa[9]/aa[15]*1e3;
               Thot[i][j-1]=Thot[i-1][j-1]+dQ/(Cphot*dmhot/2.0);
    }
```

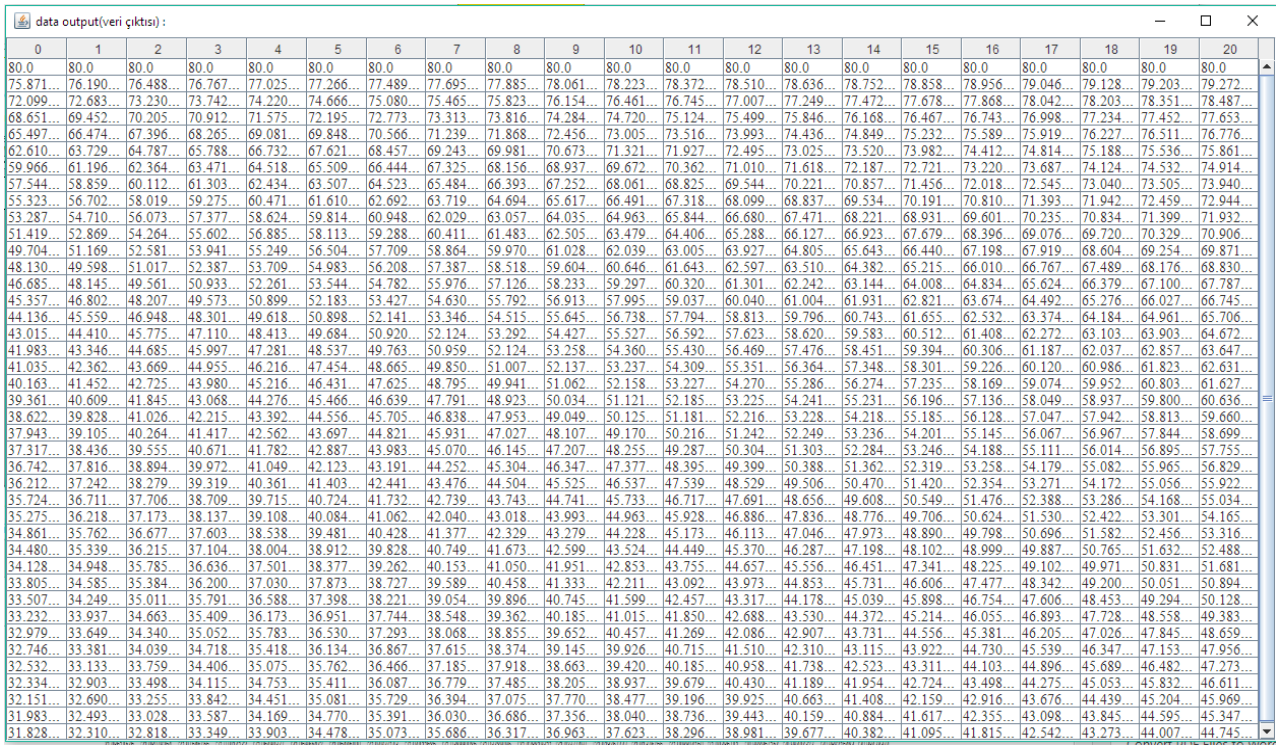
```
    Q*=nplate;
    Thotout=0;
    Tcoldout=0;
    for(j=0;j<n_y;j++)
                {Thotout+=Thot[n_x-1][j];
                 UU[n_x-1][j]=U;
                //System.out.println("Thotout="+Thotout);
                }
                Thotout/=n_y;
                System.out.println("Thotout="+Thotout);
                for(i=0;i<n_x;i++)
                {Tcoldout+=Tcold[i][n_y-1];
                 UU[i][n_y-1]=U;
                }

                Tcoldout/=n_x;
                System.out.println("Tcoldout="+Tcoldout);
                System.out.println("Q="+Q);
}

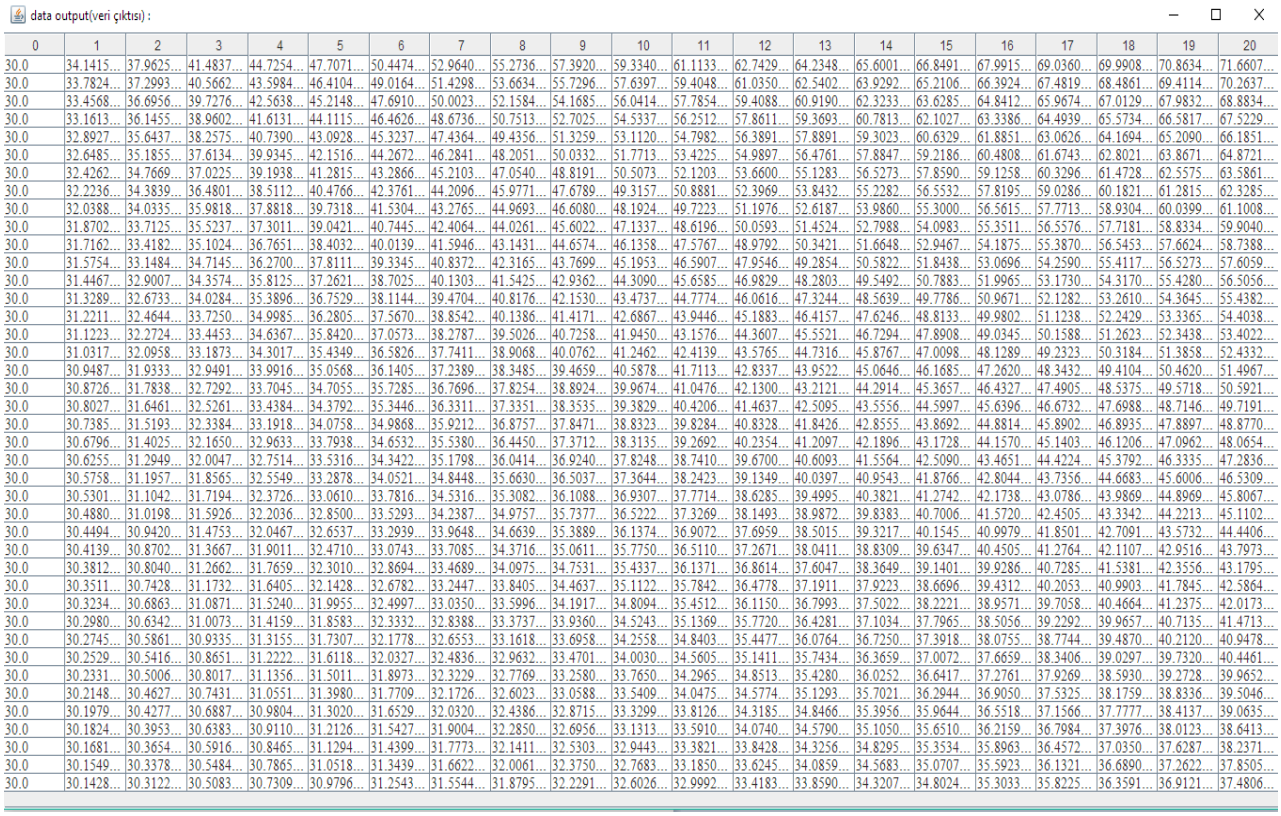
}
```

```
/*
import visad.*;
import visad.java3d.DisplayImplJ3D;
import visad.java2d.DisplayImplJ2D;
import java.rmi.RemoteException;
*/
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class recuperator1test
{
public static void main(String arg[])
{               String ihot_sivi="air";
                String icold_sivi="air";
                double mhoti=0.01;//kg/s
                double mcoldi=0.01;//kg/s
                double Li=0.8; //m
                double Wi=0.4; //m
                double dzi=2.0e-3;//m
                double dLi=0.02; //m
                int nplatei=40;
    recuperator1 r=new recuperator1(ihot_sivi,icold_sivi,mhoti,mcoldi,Li,Wi,dzi,dLi,nplatei);
    r.hesapla(30.0,80.0);
    System.out.println("Thot inlet="+r.Thotin+"Thor exit="+r.Thotout);
    System.out.println("Tcold inlet="+r.Tcoldin+"Tcold exit="+r.Tcoldout);
    Text.print(r.Tcold);
    Text.print(r.Thot);
    Text.print(r.UU);
    ContourPlot h=new ContourPlot(r.Tcold,0.0,Li,0.0,Wi,80,false,"T_hot degree C Recuperator hot side","x, m","y, m");
    ContourPlot c=new ContourPlot(r.Thot,0.0,Li,0.0,Wi,80,false,"T_cold degree C Recuperator cold side","x, m","y, m");
    //ContourPlot u=new ContourPlot(r.UU,0.0,Li,0.0,Wi,40,true,"U ısı transfer katsayısı W/(m2K)","x, m","y, m");
    h.plot();
    c.plot();
    double b1[][]=surfacePlot.dataformat(-1.0,2.0,-1.0,1.0,r.Thot);
    surfacePlot p1=new surfacePlot(b1,"x","y","Thot");
    p1.plot();
    double b2[][]=surfacePlot.dataformat(-1.0,2.0,-1.0,1.0,r.Tcold);
    surfacePlot p2=new surfacePlot(b2,"x","y","Tcold");
    p2.plot();
    //Plot3D.data(r.Tcold,0,r.L,0,r.W,40,"x","y","T");
    //Plot3D.data(r.Thot,0,r.L,0,r.W,40,"x","y","T");
}
}
```

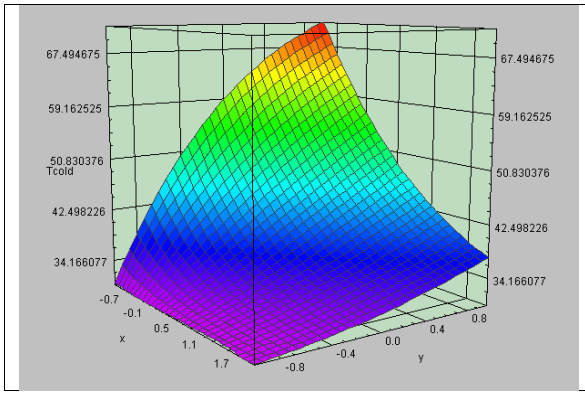
```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" recuperator1test
n_x=41n_y=21L=0.8W=0.4
Thotout=37.86538192287644
Tcoldout=50.925562100756515
Q=-4313.188435010662
Thot inlet=30.0Thor exit=37.86538192287644
Tcold inlet=80.0Tcold exit=50.925562100756515
```

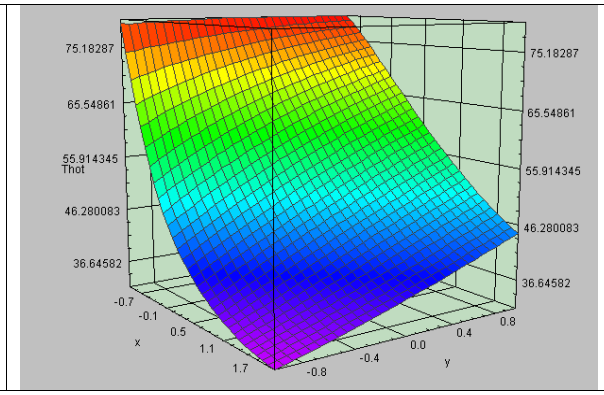


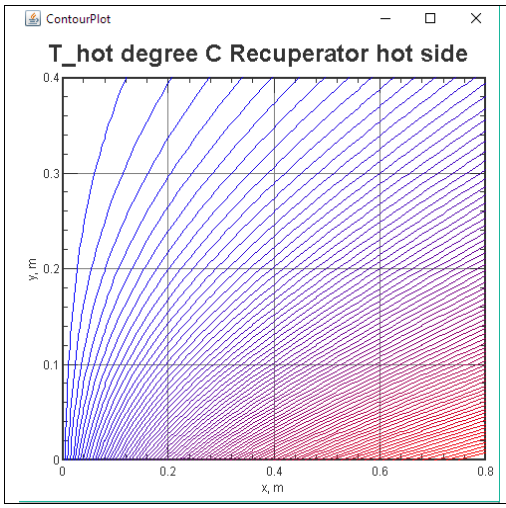
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
80.0	80.0	80.0	80.0	80.0	80.0	80.0	80.0	80.0	80.0	80.0	80.0	80.0	80.0	80.0	80.0	80.0	80.0	80.0	80.0	80.0
75.871...	76.190...	76.488...	76.767...	77.025...	77.266...	77.489...	77.695...	77.885...	78.061...	78.223...	78.372...	78.510...	78.636...	78.752...	78.858...	78.956...	79.046...	79.128...	79.203...	79.272...
72.099...	72.683...	73.230...	73.742...	74.220...	74.666...	75.080...	75.465...	75.823...	76.154...	76.461...	76.745...	77.007...	77.249...	77.472...	77.678...	77.868...	78.042...	78.203...	78.351...	78.487...
68.651...	69.452...	70.205...	70.912...	71.575...	72.195...	72.773...	73.313...	73.816...	74.284...	74.720...	75.124...	75.499...	75.846...	76.168...	76.467...	76.743...	76.998...	77.234...	77.452...	77.653...
65.497...	66.474...	67.396...	68.265...	69.081...	69.848...	70.566...	71.239...	71.868...	72.456...	73.005...	73.516...	73.993...	74.436...	74.849...	75.232...	75.589...	75.919...	76.227...	76.511...	76.776...
62.610...	63.729...	64.787...	65.788...	66.732...	67.621...	68.457...	69.243...	69.981...	70.673...	71.321...	71.927...	72.495...	73.025...	73.520...	73.982...	74.412...	74.814...	75.188...	75.536...	75.861...
59.966...	61.196...	62.364...	63.471...	64.518...	65.509...	66.444...	67.325...	68.156...	68.937...	69.672...	70.362...	71.010...	71.618...	72.187...	72.721...	73.220...	73.687...	74.124...	74.532...	74.914...
57.544...	58.859...	60.112...	61.303...	62.434...	63.507...	64.523...	65.484...	66.393...	67.252...	68.061...	68.825...	69.544...	70.221...	70.857...	71.456...	72.018...	72.545...	73.040...	73.505...	73.940...
55.323...	56.702...	58.019...	59.275...	60.471...	61.610...	62.692...	63.719...	64.694...	65.617...	66.491...	67.318...	68.099...	68.837...	69.534...	70.191...	70.810...	71.393...	71.942...	72.459...	72.944...
53.287...	54.710...	56.073...	57.377...	58.624...	59.814...	60.948...	62.029...	63.057...	64.035...	64.963...	65.844...	66.680...	67.471...	68.221...	68.931...	69.601...	70.235...	70.834...	71.399...	71.932...
51.419...	52.869...	54.264...	55.602...	56.885...	58.113...	59.288...	60.411...	61.483...	62.505...	63.479...	64.406...	65.288...	66.127...	66.923...	67.679...	68.396...	69.076...	69.720...	70.329...	70.906...
49.704...	51.169...	52.581...	53.941...	55.249...	56.504...	57.709...	58.864...	59.970...	61.028...	62.039...	63.005...	63.927...	64.805...	65.643...	66.440...	67.198...	67.919...	68.604...	69.254...	69.871...
48.130...	49.598...	51.017...	52.387...	53.709...	54.983...	56.208...	57.387...	58.518...	59.604...	60.646...	61.643...	62.597...	63.510...	64.382...	65.215...	66.010...	66.767...	67.489...	68.176...	68.830...
46.685...	48.145...	49.561...	50.933...	52.261...	53.544...	54.782...	55.976...	57.126...	58.233...	59.297...	60.320...	61.301...	62.242...	63.144...	64.008...	64.834...	65.624...	66.379...	67.100...	67.787...
45.357...	46.802...	48.207...	49.573...	50.899...	52.183...	53.427...	54.630...	55.792...	56.913...	57.995...	59.037...	60.040...	61.004...	61.931...	62.821...	63.674...	64.492...	65.276...	66.027...	66.745...
44.136...	45.559...	46.948...	48.301...	49.618...	50.898...	52.141...	53.346...	54.515...	55.645...	56.738...	57.794...	58.813...	59.796...	60.743...	61.655...	62.532...	63.374...	64.184...	64.961...	65.706...
43.015...	44.410...	45.775...	47.110...	48.413...	49.684...	50.920...	52.124...	53.292...	54.427...	55.527...	56.592...	57.623...	58.620...	59.583...	60.512...	61.408...	62.272...	63.103...	63.903...	64.672...
41.983...	43.346...	44.685...	45.997...	47.281...	48.537...	49.763...	50.959...	52.124...	53.258...	54.360...	55.430...	56.469...	57.476...	58.451...	59.394...	60.306...	61.187...	62.037...	62.857...	63.647...
41.035...	42.362...	43.669...	44.955...	46.216...	47.454...	48.665...	49.850...	51.007...	52.137...	53.237...	54.309...	55.351...	56.364...	57.348...	58.301...	59.226...	60.120...	60.986...	61.823...	62.631...
40.163...	41.452...	42.725...	43.980...	45.216...	46.431...	47.625...	48.795...	49.941...	51.062...	52.158...	53.227...	54.270...	55.286...	56.274...	57.235...	58.169...	59.074...	59.952...	60.803...	61.627...
39.361...	40.609...	41.845...	43.068...	44.276...	45.466...	46.639...	47.791...	48.923...	50.034...	51.121...	52.185...	53.225...	54.241...	55.231...	56.196...	57.136...	58.049...	58.937...	59.800...	60.636...
38.622...	39.828...	41.026...	42.215...	43.392...	44.556...	45.705...	46.838...	47.953...	49.049...	50.125...	51.181...	52.216...	53.228...	54.218...	55.185...	56.128...	57.047...	57.942...	58.813...	59.660...
37.943...	39.105...	40.264...	41.417...	42.562...	43.697...	44.821...	45.931...	47.027...	48.107...	49.170...	50.216...	51.242...	52.249...	53.236...	54.201...	55.145...	56.067...	56.967...	57.844...	58.699...
37.317...	38.436...	39.555...	40.671...	41.782...	42.887...	43.983...	45.070...	46.145...	47.207...	48.255...	49.287...	50.304...	51.303...	52.284...	53.246...	54.188...	55.111...	56.014...	56.895...	57.755...
36.742...	37.816...	38.894...	39.972...	41.049...	42.123...	43.191...	44.252...	45.304...	46.347...	47.377...	48.395...	49.399...	50.388...	51.362...	52.319...	53.258...	54.179...	55.082...	55.965...	56.829...
36.212...	37.242...	38.279...	39.319...	40.361...	41.403...	42.441...	43.476...	44.504...	45.525...	46.537...	47.539...	48.529...	49.506...	50.470...	51.420...	52.354...	53.271...	54.172...	55.056...	55.922...
35.724...	36.711...	37.706...	38.709...	39.715...	40.724...	41.732...	42.739...	43.743...	44.741...	45.733...	46.717...	47.691...	48.656...	49.608...	50.549...	51.476...	52.388...	53.286...	54.168...	55.034...
35.275...	36.218...	37.173...	38.137...	39.108...	40.084...	41.062...	42.040...	43.018...	43.993...	44.963...	45.928...	46.886...	47.836...	48.776...	49.706...	50.624...	51.530...	52.422...	53.301...	54.165...
34.861...	35.762...	36.677...	37.603...	38.538...	39.481...	40.428...	41.377...	42.329...	43.279...	44.228...	45.173...	46.113...	47.046...	47.973...	48.890...	49.798...	50.696...	51.582...	52.456...	53.316...
34.480...	35.339...	36.215...	37.104...	38.004...	38.912...	39.828...	40.749...	41.673...	42.599...	43.524...	44.449...	45.370...	46.287...	47.198...	48.102...	48.999...	49.887...	50.765...	51.632...	52.488...
34.128...	34.948...	35.785...	36.636...	37.501...	38.377...	39.262...	40.153...	41.050...	41.951...	42.853...	43.755...	44.657...	45.556...	46.451...	47.341...	48.225...	49.102...	49.971...	50.831...	51.681...
33.805...	34.585...	35.384...	36.200...	37.030...	37.873...	38.727...	39.589...	40.458...	41.333...	42.211...	43.092...	43.973...	44.853...	45.731...	46.606...	47.477...	48.342...	49.200...	50.051...	50.894...
33.507...	34.249...	35.011...	35.791...	36.588...	37.398...	38.221...	39.054...	39.896...	40.745...	41.599...	42.457...	43.317...	44.178...	45.039...	45.898...	46.754...	47.606...	48.453...	49.294...	50.128...
33.232...	33.937...	34.663...	35.409...	36.173...	36.951...	37.744...	38.548...	39.362...	40.185...	41.015...	41.850...	42.688...	43.530...	44.372...	45.214...	46.055...	46.893...	47.728...	48.558...	49.383...
32.979...	33.649...	34.340...	35.052...	35.783...	36.530...	37.293...	38.068...	38.855...	39.652...	40.457...	41.269...	42.086...	42.907...	43.731...	44.556...	45.381...	46.205...	47.026...	47.845...	48.659...
32.746...	33.381...	34.039...	34.718...	35.418...	36.134...	36.867...	37.615...	38.374...	39.145...	39.926...	40.715...	41.510...	42.310...	43.115...	43.922...	44.730...	45.539...	46.347...	47.153...	47.956...
32.532...	33.133...	33.759...	34.406...	35.075...	35.762...	36.466...	37.185...	37.918...	38.663...	39.420...	40.185...	40.958...	41.738...	42.523...	43.311...	44.103...	44.896...	45.689...	46.482...	47.273...
32.334...	32.903...	33.498...	34.115...	34.753...	35.411...	36.087...	36.779...	37.485...	38.205...	38.937...	39.679...	40.430...	41.189...	41.954...	42.724...	43.498...	44.275...	45.053...	45.832...	46.611...
32.151...	32.690...	33.255...	33.842...	34.451...	35.081...	35.729...	36.394...	37.075...	37.770...	38.477...	39.196...	39.925...	40.663...	41.408...	42.159...	42.916...	43.676...	44.439...	45.204...	45.969...
31.983...	32.493...	33.028...	33.587...	34.169...	34.770...	35.391...	36.030...	36.686...	37.356...	38.040...	38.736...	39.443...	40.159...	40.884...	41.617...	42.355...	43.098...	43.845...	44.595...	45.347...
31.828...	32.310...	32.818...	33.349...	33.903...	34.478...	35.073...	35.686...	36.317...	36.963...	37.623...	38.296...	38.981...	39.677...	40.382...	41.095...	41.815...	42.542...	43.273...	44.007...	44.745...

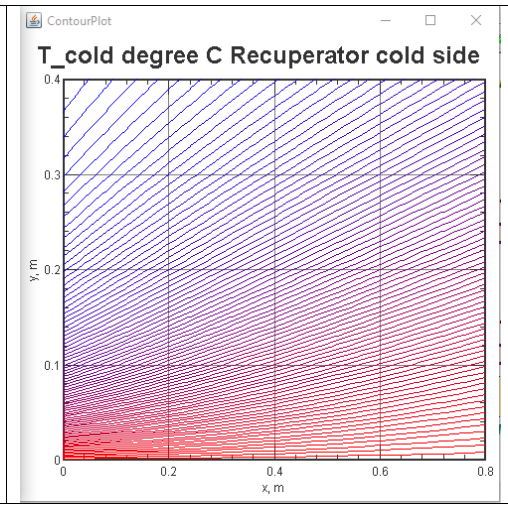


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
30.0	34.1415...	37.9625...	41.4837...	44.7254...	47.7071...	50.4474...	52.9640...	55.2736...	57.3920...	59.3340...	61.1133...	62.7429...	64.2348...	65.6001...	66.8491...	67.9915...	69.0360...	69.9908...	70.8634...	71.6607...
30.0	33.7824...	37.2993...	40.5662...	43.5984...	46.4104...	49.0164...	51.4298...	53.6634...	55.7296...	57.6397...	59.4048...	61.0350...	62.5402...	63.9292...	65.2106...	66.3924...	67.4819...	68.4861...	69.4114...	70.2637...
30.0	33.4568...	36.6956...	39.7276...	42.5638...	45.2148...	47.6910...	50.0023...	52.1584...	54.1685...	56.0414...	57.7854...	59.4088...	60.9190...	62.3233...	63.6285...	64.8412...	65.9674...	67.0129...	67.9832...	68.8834...
30.0	33.1613...	36.1455...	38.9602...	41.6131...	44.1115...	46.4626...	48.6736...	50.7513...	52.7025...	54.5337...	56.2512...	57.8611...	59.3693...	60.7813...	62.1027...	63.3386...	64.4939...	65.5734...	66.5817...	67.5229...
30.0	32.8927...	35.6437...	38.2575...	40.7390...	43.0928...	45.3237...	47.4364...	49.4356...	51.3259...	53.1120...	54.7982...	56.3891...	57.8891...	59.3023...	60.6329...	61.8851...	63.0626...	64.1694...	65.2090...	66.1851...
30.0	32.6485...	35.1855...	37.6134...	39.9345...	42.1516...	44.2672...	46.2841...	48.2051...	50.0332...	51.7713...	53.4225...	54.9897...	56.4761...	57.8847...	59.2186...	60.4808...	61.6743...	62.8021...	63.8671...	64.8721...
30.0	32.4262...	34.7669...	37.0225...	39.1938...	41.2815...	43.2866...	45.2103...	47.0540...	48.8191...	50.5073...	52.1203...	53.6600...	55.1283...	56.5273...	57.8590...	59.1258...	60.3296...	61.4728...	62.5575...	63.5861...
30.0	32.2236...	34.3839...	36.4801...	38.5112...	40.4766...	42.3761...	44.2096...	45.9771...	47.6789...	49.3157...	50.8881...	52.3969...	53.8432...	55.2282...	56.5532...	57.8195...	59.0286...	60.1821...	61.2815...	62.3285...
30.0	32.0388...	34.0335...	35.9818...	37.8818...	39.7318...	41.5304...	43.2765...	44.9693...	46.6080...	48.1924...	49.7223...	51.1976...	52.6187...	53.9860...	55.3000...	56.5615...	57.7713...	58.9304...	60.0399...	61.1008...
30.0	31.8702...	33.7125...	35.5237...	37.3011...	39.0421...	40.7445...	42.4064...	44.0261...	45.6022...	47.1337...	48.6196...	50.0593...	51.4524...	52.7988...	54.0983...	55.3511...	56.5576...	57.7181...	58.8334...	59.9040...
30.0	31.7162...	33.4182...	35.1024...	36.7651...	38.4032...	40.0139...	41.5946...	43.1431...	44.6574...	46.1358...	47.5767...	48.9792...	50.3421...	51.6648...	52.9467...	54.1875...	55.3870...	56.5453...	57.6624...	58.7388...
30.0	31.5754...	33.1484...	34.7145...	36.2700...	37.8111...	39.3345...	40.8372...	42.3165...	43.7699...	45.1953...	46.5907...	47.9546...	49.2854...	50.5822...	51.8438...	53.0696...	54.2590...	55.4117...	56.5273...	57.6059...
30.0	31.4467...	32.9007...	34.3574...	35.8125...	37.2621...	38.7025...	40.1303...	41.5425...	42.9362...	44.3090...	45.6585...	46.9829...	48.2803...	49.5492...	50.7883...	51.9965...	53.1730...	54.3170...	55.4280...	56.5056...
30.0	31.3289...	32.6733...	34.0284...	35.3896...	36.7529...	38.1144...	39.4704...	40.8176...	42.1530...	43.4737...	44.7774...	46.0616...	47.3244...	48.5639...	49.7786...	50.9671...	52.1282...	53.2610...	54.3645...	55.4382...
30.0	31.2211...	32.4644...	33.7250...	34.9985...	36.2805...	37.5670...	38.8542...	40.1386...	41.4171...	42.6867...	43.9446...	45.1883...	46.4157...	47.6246...	48.8133...	49.9802...	51.1238...	52.2429...	53.3365...	54.4038...
30.0	31.1223...	32.2724...	33.4453...	34.6367...	35.8420...	37.0573...	38.2787...	39.5026...	40.7258...	41.9450...	43.1576...	44.3607...	45.5521...	46.7294...	47.8908...	49.0345...	50.1588...	51.2623...	52.3438...	53.4022...
30.0	31.0317...	32.0958...	33.1873...	34.3017...	35.4349...	36.5826...	37.7411...	38.9068...	40.0762...	41.2462...	42.4139...	43.5765...	44.7316...	45.8767...	47.0098...	48.1289...	49.2323...	50.3184...	51.3858...	52.4332...
30.0	30.9487...	31.9333...	32.9491...	33.9916...	35.0568...	36.1405...	37.2389...	38.3485...	39.4659...	40.5878...	41.7113...	42.8337...	43.9522...	45.0646...	46.1685...	47.2620...	48.3432...	49.4104...	50.4620...	51.4967...
30.0	30.8726...	31.7838...	32.7292...	33.7045...	34.7055...	35.7285...	36.7696...	37.8254...	38.8924...	39.9674...	41.0476...	42.1300...	43.2121...	44.2914...	45.3657...	46.4327...	47.4905...	48.5375...	49.5718...	50.5921...
30.0	30.8027...	31.6461...	32.5261...	33.4384...	34.3792...	35.3446...	36.3311...	37.3351...	38.3535...	39.3829...	40.4206...	41.4637...	42.5095...	43.5556...	44.5997...	45.6396...	46.6732...	47.6988...	48.7146...	49.7191...
30.0	30.7385...	31.5193...	32.3384...	33.1918...	34.0758...	34.9868...	35.9212...	36.8757...	37.8471...	38.8323...	39.8284...	40.8328...	41.8426...	42.8555...	43.8692...	44.8814...	45.8902...	46.8935...	47.8897...	48.8770...
30.0	30.6796...	31.4025...	32.1650...	32.9633...	33.7938...	34.6532...	35.5380...	36.4450...	37.3712...	38.3135...	39.2692...	40.2354...	41.2097...	42.1896...	43.1728...	44.1570...	45.1403...	46.1206...	47.0962...	48.0654...
30.0	30.6255...	31.2949...	32.0047...	32.7514...	33.5316...	34.3422...	35.1798...	36.0414...	36.9240...	37.8248...	38.7410...	39.6700...	40.6093...	41.5564...	42.5090...	43.4651...	44.4224...	45.3792...	46.3335...	47.2836...
30.0	30.5758...	31.1957...	31.8565...	32.5549...	33.2878...	34.0521...	34.8448...	35.6630...	36.5037...	37.3644...	38.2423...	39.1349...	40.0397...	40.9543...	41.8766...	42.8044...	43.7356...	44.6683...	45.6006...	46.5309...
30.0	30.5301...	31.1042...	31.7194...	32.3726...	33.0610...	33.7816...	34.5316...	35.3082...	36.1088...	36.9307...	37.7714...	38.6285...	39.4995...	40.3821...	41.2742...	42.1738...	43.0786...	43.9869...	44.8969...	45.8067...
30.0	30.4880...	31.0198...	31.5926...	32.2036...	32.8500...	33.5293...	34.2387...	34.9757...	35.7377...	36.5222...	37.3269...	38.1493...	38.9872...	39.8383...	40.7006...	41.5720...	42.4505...	43.3342...	44.2213...	45.1102...
30.0	30.4494...	30.9420...	31.4753...	32.0467...	32.6537...	33.2939...	33.9648...	34.6639...	35.3889...	36.1374...	36.9072...	37.6959...	38.5015...	39.3217...	40.1545...	40.9979...	41.8501...	42.7091...	43.5732...	44.4406...
30.0	30.4139...	30.8702...	31.3667...	31.9011...	32.4710...	33.0743...	33.7085...	34.3716...	35.0611...	35.7750...	36.5110...	37.2671...	38.0411...	38.8309...	39.6347...	40.4505...	41.2764...	42.1107...	42.9516...	43.7973...
30.0	30.3812...	30.8040...	31.2662...	31.7659...	32.3010...	32.8694...	33.4689...	34.0975...	34.7531...	35.4337...	36.1371...	36.8614...	37.6047...	38.3649...	39.1401...	39.9286...	40.7285...	41.5381...	42.3556...	43.1795...
30.0	30.3511...	30.7428...	31.1732...	31.6405...	32.1428...	32.6782...	33.2447...	33.8405...	34.4637...	35.1122...	35.7842...	36.4778...	37.1911...	37.9223...	38.6696...	39.4312...	40.2053...	40.9903...	41.7845...	42.5864...
30.0	30.3234...	30.6863...	31.0871...	31.5240...	31.9955...	32.4997...	33.0350...	33.5996...	34.1917...	34.8094...	35.4512...	36.1150...	36.7993...	37.5022...	38.2221...	38.9571...	39.7058...	40.4664...	41.2375...	42.0173...
30.0	30.2980...	30.6342...	31.0073...	31.4159...	31.8583...	32.3332...	32.8388...	33.3737...	33.9360...	34.5243...	35.1369...	35.7720...	36.4281...	37.1034...	37.7965...	38.5056...	39.2292...	39.9657...	40.7135...	41.4713...
30.0	30.2745...	30.5861...	30.9335...	31.3155...	31.7307...	32.1778...	32.6553...	33.1618...	33.6958...	34.2558...	34.8403...	35.4477...	36.0764...	36.7250...	37.3918...	38.0755...	38.7744...	39.4870...	40.2120...	40.9478...
30.0	30.2529...	30.5416...	30.8651...	31.2222...	31.6118...	32.0327...	32.4836...	32.9632...	33.4701...	34.0030...	34.5605...	35.1411...	35.7434...	36.3659...	37.0072...	37.6659...	38.3406...	39.0297...	39.7320...	40.4461...
30.0	30.2331...	30.5006...	30.8017...	31.1356...	31.5011...	31.8973...	32.3229...	32.7769...	33.2580...	33.7650...	34.2965...	34.8513...	35.4280...	36.0252...	36.6417...	37.2761...	37.9269...	38.5930...	39.2728...	39.9652...
30.0	30.2148...	30.4627...	30.7431...	31.0551...	31.3980...	31.7709...	32.1726...	32.6023...	33.0588...	33.5409...	34.0475...	34.5774...	35.1293...	35.7021...	36.2944...	36.9050...	37.5325...	38.1759...	38.8336...	39.5046...
30.0	30.1979...	30.4277...	30.6887...	30.9804...	31.3020...	31.6529...	32.0320...	32.4386...	32.8715...	33.3299...	33.8126...	34.3185...	34.8466...	35.3956...	35.9644...	36.5518...	37.1566...	37.7777...	38.4137...	39.0635...
30.0	30.1824...	30.3953...	30.6383...	30.9110...	31.2126...	31.5427...	31.9004...	32.2850...	32.6956...	33.1313...	33.5910...	34.0740...	34.5790...	35.1050...	35.6510...	36.2159...	36.7984...	37.3976...	38.0123...	38.6413...
30.0	30.1681...	30.3654...	30.5916...	30.8465...	31.1294...	31.4399...	31.7773...	32.1411...	32.5303...	32.9443...	33.3821...	33.8428...	34.3256...	34.8295...	35.3534...	35.8963...	36.4572...	37.0350...	37.6287...	38.2371...
30.0	30.1549...	30.3378...	30.5484...	30.7865...	31.0518...	31.3439...	31.6622...	32.0061...	32.3750...	32.7683...	33.1850...	33.6245...	34.0859...	34.5683...	35.0707...	35.5923...	36.1321...	36.6890...	37.2622...	37.8505...
30.0	30.1428...	30.3122...	30.5083...	30.7309...	30.9796...	31.2543...	31.5544...	31.8795...	32.2291...	32.6026...	32.9992...	33.4183...	33.8590...	34.3207...	34.8024...	35.3033...	35.8225...	36.3591...	36.9121...	37.4806...



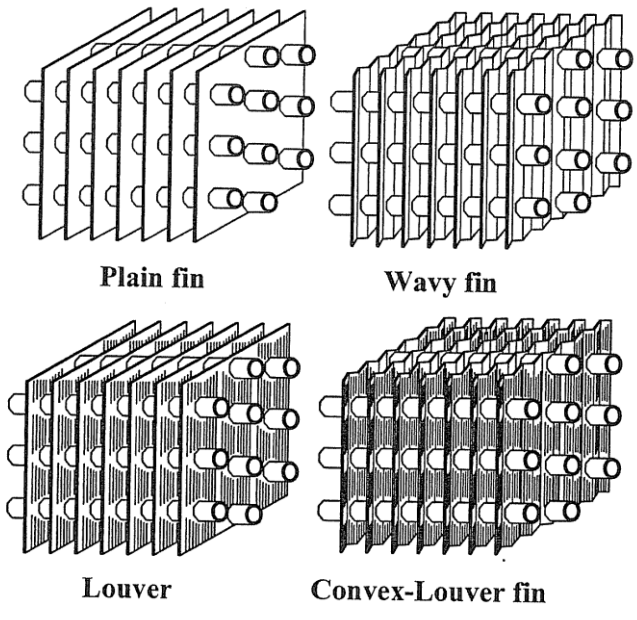






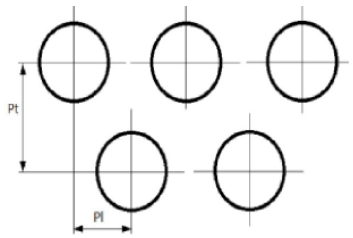
16.6 FINNED SURFACES HEAT TRANSFER FOR HEAT EXCHANGER APPLICATIONS

Now let us looked at a finned surface heat exchanger. In this geometry tubes connected to the fins throufg a bended region of the plate called çöller. In most of the heat exchangers fin and pipe connection based only touching contact by expanding pipe diameter by applying high internal pressure. Braze connections are also used but not as common. For finned heat exchangers a set of equation [70],[71]

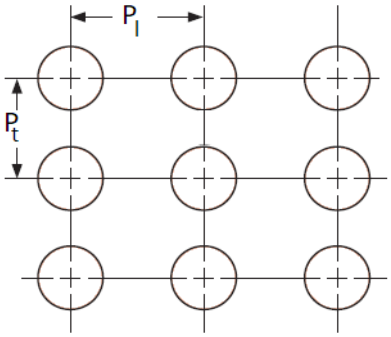


Can be given as:
$\frac{1}{UA} = \frac{1}{\eta_0 h_0 A_0} + \frac{\delta_w}{k_w A_w} + \frac{1}{A_i h_i} + \frac{R_i}{A_i} + \frac{R_w}{A_w}$ where h_0 average fin site heat transfer coefficient, η_0 fin efficiency A_0 fin site total area (including pipe), A_i inside area, U is overall heat transfer coefficient and R_i and R_w internal and external fouling factors. δ_w is fin thickness and δ_f fin thickness

$\eta_0 = \frac{\tanh(mr\phi)}{mr\phi} \quad m = \sqrt{\frac{2h_0}{k_f \delta_f}}$
$\phi = \left[\frac{R_{eq}}{r} - 1\right]\left[\frac{X_L}{X_M} - 0.35ln\left(\frac{R_{eq}}{r}\right)\right]$



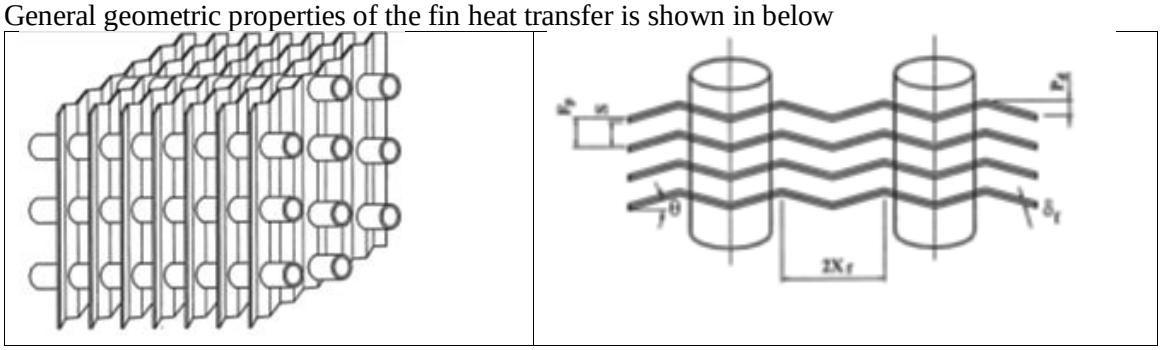
Staggered configuration



Aligned configuration

For staggered pipes:
$\frac{R_{eq}}{r} = 1.27\frac{X_M}{r}\left[\frac{X_L}{X_M} - 0.3\right]^{1/2}$ where r is the pipe radius
$X_L = \sqrt{\left(\frac{P_t}{2}\right)^2 + P_l^2} \quad X_M = \left(\frac{P_t}{2}\right)$ where P_t is vertical distance between the pipes, P_l is horizontal distance between the pipes
For aligned pipes:
$\frac{R_{eq}}{r} = 1.28\frac{X_M}{r}\left[\frac{X_L}{X_M} - 0.2\right]^{1/2}$

For this composition
$X_L = \left(\frac{P_T}{2}\right)$
For plain fins:$St = \frac{h}{GC_p}$ stanton number $Re_D = \frac{\rho V D_c}{\mu}$ where D_c is fin collar outside diameter $D_H = 2R_{eq}$
$j = StPr^{2/3} = \frac{h}{GC_p}Pr^{2/3}$ Colburn module
General geometric properties of the fin heat transfer is shown in below



Where P_d waffle height
F_p fin pitch
S fin spacing

X_f projected fin pattern length
δ_f fin thickness
θ corrugation angle
D_c Fin collar outside diameter

Plain fins

Single raw plain fin (N=1)

$$j = StPr^{2/3} = \frac{h}{GC_p}Pr^{2/3} = 0.173Re_D^{-0.346}\left(\frac{P_t}{P_l}\right)^{P1}\left(\frac{D_c}{F_p}\right)^{1.161}\left(\frac{D_h}{F_p}\right)^{1.035}\left(\frac{F_p}{P_t}\right)^{P2}$$

$$P1 = -0.22\ln(Re_D) + 1.88$$

$$P2 = 0.106\ln(Re_D)$$

Two or more raws ($N \geq 2$)

$$j = StPr^{2/3} = \frac{h}{GC_p}Pr^{2/3} = 0.1078Re_D^{P3}N^{P4}\left(\frac{F_p}{D_h}\right)^{P5}\left(\frac{P_t}{F_p}\right)^{1.026}$$

$$P3 = 0.16\ln\left(N\left(\frac{F_p}{D_c}\right)^{0.42}\right) - 0.349$$

$$P3 = \frac{-0.094\left(\frac{P_l}{D_h}\right)^{1.38}}{\ln(Re_D)} - 1.405$$

$$P4 = 1.263ln\left(\frac{Re_D}{N}\right) - 5.97$$

$D_h = \frac{4A_cL}{A_0}$ hydraulic diameter

Fanning friction factor:

$$f = 0.0146Re_D^{P6}\left(\frac{P_t}{P_l}\right)^{1.959}\left(\frac{F_p}{D_c}\right)^{P7}N^{0.021}$$

$$P4 = -0.0535 + \frac{0.01166}{ln\left(\frac{P_t}{P_l}\right)} + 0.123\left(\frac{F_p}{D_c}\right)$$

$$P7 = 2.319 - \frac{19.59}{\ln(Re_D)}$$

wavy fins:

$$j = StPr^{2/3} = \frac{h}{GC_p}Pr^{2/3} = -0.17017 - Re_D^{W1}\left(\frac{P_l}{\delta_f}\right)^{-0.456}N^{-0.27}\left(\frac{F_p}{D_c}\right)^{-1.343}\left(\frac{P_d}{X_f}\right)^{0.317}$$

$$W1 = -0.1707 - 1.374\left(\frac{P_l}{\delta_f}\right)^{-0.493}\left(\frac{F_p}{D_c}\right)^{-0.886}N^{-0.143}\left(\frac{P_d}{X_f}\right)^{-0.0296}$$

$$f = 0.05273Re_D^{f1}\left(\frac{P_d}{X_f}\right)^{f2}\left(\frac{F_p}{P_t}\right)^{f3}\left(ln\left(\frac{A_0}{A_t}\right)\right)^{-2.726}\left(\frac{D_h}{D_c}\right)^{0.1325}N^{0.02305}$$

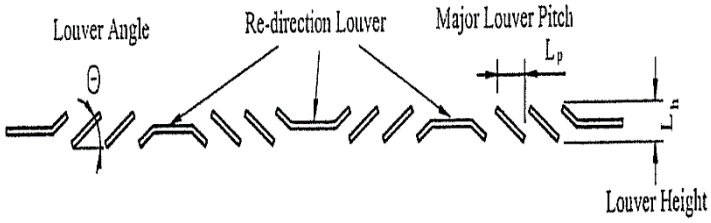
$$f1 = 0.1714 - 0.07372\left(\frac{F_p}{P_l}\right)^{0.25}ln\left(\frac{A_0}{A_t}\right)\left(\frac{P_d}{X_f}\right)^{-0.2}$$

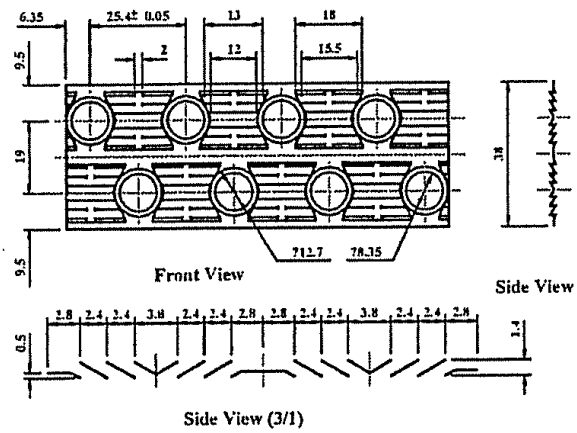
$$f2 = 0.426\left(\frac{F_p}{P_t}\right)^{0.3}ln\left(\frac{A_0}{A_t}\right)$$

$$f3 = \frac{-10.2192}{\ln(Re_D)}$$

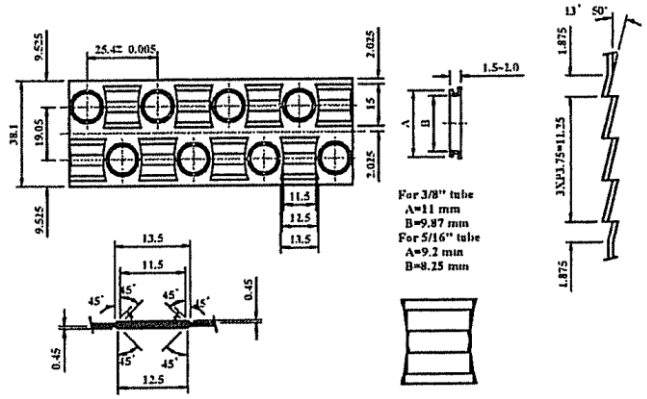
Fanning friction factor:

Correlations for lower fins:

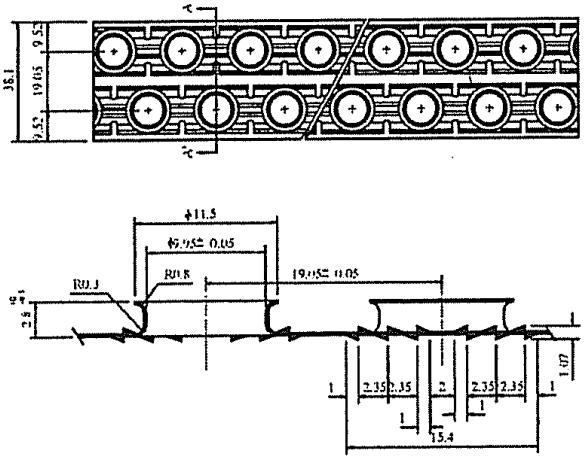




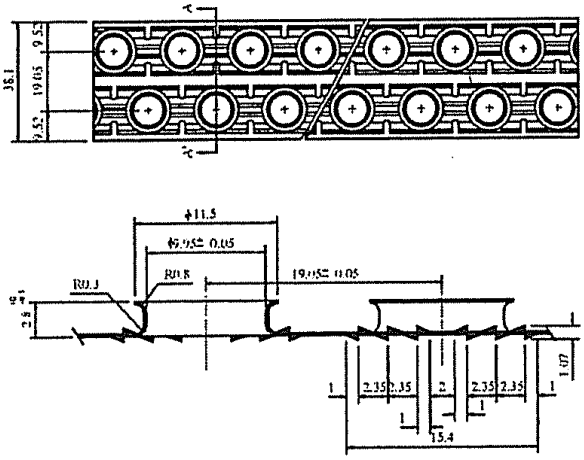
type I Louver



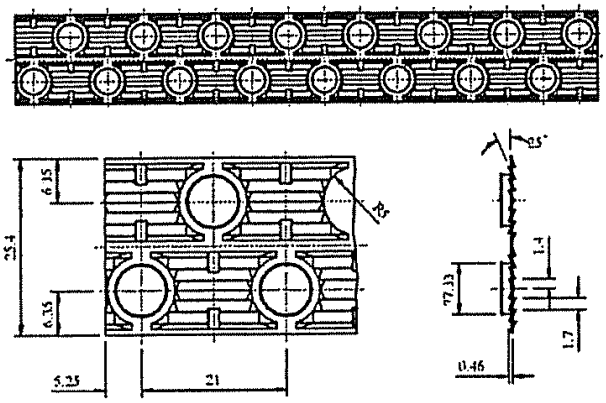
type II Louver



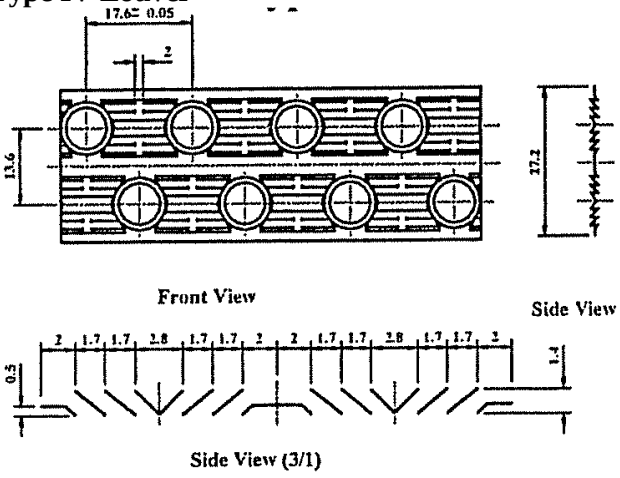
type III Louver



Type IV Louver



type V Louver



type VI Louver

Type I,III,IV and VI $Re_D < 1000$

$$j = StPr^{2/3} = \frac{h}{GC_p}Pr^{2/3} = 0.501Re_D^{0.094}\left(\frac{P_l}{P_t}\right)^{1.44}\left(\frac{F_p}{D_c}\right)^{0.322}N^{-0.809}\left(\frac{L_p}{F_p}\right)^{-0.177}\left(\frac{L_h}{L_p}\right)^{-0.254}$$

Equation valid for $N \geq 4$ $\frac{F_p}{D_c} \geq 0.14$ or $N = 2,3$ $\frac{F_p}{D_c} < 0.14$

Type II, and lover fins with re-direction louver and N=1 and lover fins with re-direction louver $Re_D > 1000$

$$j = StPr^{2/3} = \frac{h}{GC_p}Pr^{2/3} = 0.962Re_D^{L1}\left(\frac{P_l}{P_t}\right)^{-1.51}\left(\frac{F_p}{D_c}\right)^{-0.107}N^{-1.06}\left(\frac{L_h}{L_p}\right)^{-0.433}$$

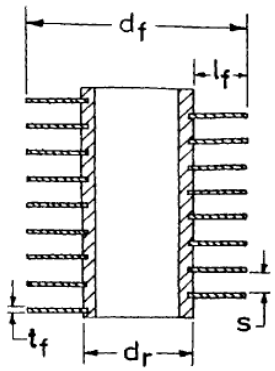
$$L1 = \left[-0.501\left(\frac{P_l}{P_t}\right)^{-0.424} N^{-0.253}\left(\frac{L_h}{L_p}\right)^{-0.127}\right]$$

Correlation of fanning friction factor

$$f = 0.72 Re_D^{Z1}\left(\frac{F_p}{D_c}\right)^{Z2} N^{0.322}\left(\frac{L_h}{L_p}\right)^{0.352}\left(\frac{L_p}{F_p}\right)^{1.69}$$

$$Z1 = -0.392\left(\frac{P_l}{P_t}\right)^{-0.726} N^{0.0782}\left(\frac{L_h}{L_p}\right)^{0.352}\left(\frac{L_p}{F_p}\right)^{0.63}$$

$$Z2 = -3.3 + 3.72\left(\frac{P_l}{P_t}\right)^{-1} - 1.29\left(\frac{P_l}{P_t}\right)^{-2}$$



Now let us look at another finned structure: tube with a fin. In this type of fnned surfaces fin can be integral part of the tube , or welded or brazed to the tube surface. A general wiew of this type of finned tube is given above. Where s=distance between fins, t_f fin thickness, d_r pipe external diameter, l_f fin height and d_f total diameter. N_f number of fins in 1 m tube length. Heat transfer equation for few different geometries are given below:

Low fin height and frequent fins (Briggs & Young Correlation[95],[97]) :

$$\text{Nu} = 0.1507 Re_d^{0.667} Pr^{1/3}\left(\frac{s}{l_f}\right)^{0.164}\left(\frac{s}{t_f}\right)^{0.075} \qquad 1000 \le Re \le 20000 \text{ for Staggered configuration}$$

High-fin tube banks

$$\text{Nu} = 0.1378 Re_d^{0718} Pr^{1/3}\left(\frac{s}{l_f}\right)^{0.296}$$

For all fin heights:

$$\text{Nu} = 0.134 Re_d^{0.681} Pr^{1/3}\left(\frac{s}{l_f}\right)^{0.2}\left(\frac{s}{t_f}\right)^{0.1134}$$

$$j = StPr^{2/3} = \frac{h}{GC_p} Pr^{2/3} = 0.134 Re_d^{-0.3191}\left(\frac{s}{d_f}\right)^{0.2}\left(\frac{s}{t_f}\right)^{0.1134}$$

Small fin height and frequent fins (Rabas, Eckel and Sabatino equation) :

$$j = StPr^{2/3} = \frac{h}{GC_p} Pr^{2/3} = 0.292 Re_d^{-m}\left(\frac{s}{d_f}\right)^{0.257}\left(\frac{t_f}{s}\right)^{0.666}\left(\frac{d_f}{d_r}\right)^{0.473}\left(\frac{d_f}{t_f}\right)^{0.7717}\alpha_h\alpha_n$$

$$m = 0.415 - 0.0346 ln\left(\frac{d_f}{s}\right)$$

$$\alpha_h = \left(\frac{T_b + 273.15}{T_w + 273.15}\right)^{0.25}$$

$$\alpha_n = \left(\frac{\text{Pr}\,(T_b)}{\text{Pr}\,(T_w)}\right)^{0.25}$$

Fanning friction factor

$$f = 3.805 Re_d^{-0.234} Pr^{1/3}\left(\frac{s}{d_f}\right)^{0.2251}\left(\frac{l_f}{s}\right)^{0.759}\left(\frac{d_r}{P_t}\right)^{0.709}\left(\frac{P_l}{P_t}\right)^{0.379}$$

l_f < 6.35 mm, 1000<Re<25000 4.76<d<33.75 mm, 246<N_f<1181fin/m, 15.08<P_t<111 mm, P_l <= P_t 10.32<P_l<96.11 mm , d_f/s<40 ve N_r>6 for Staggered configuration

A sample program: circular fin heat exchanger

```
 import javax.swing.JOptionPane;
import java.util.Locale;

public class circular_fin_HE_FD
{ // circular fin heat exchanger
  liquid_CS soguk_sivi; // cold fluid (inside tubes)
  Gmix sicak_sivi;      // hot fluid (outside tubes)
  steam st1;            // water - steam Eqaution of state
  String g[];   // gas names in the mixture
  double gn[];  // mol numbers in the gas mixtures
  double s;     //distance between fins
  int nkanat;
  double delta; //fin thickness
  double hf;    //fin height
  int n_sira;   //number of rows (vertical)
  int n_boru;   //number of lines (horizontal)
  double eps;   //surface roughness
  double di;    //pipe internal diameter
  double dout;  //pipe external diameter
  double D;     //fin diameter
  double fi;
  double st;    // borular arası dikey mesafe
  double sl;    // borular arası yatay mesafe
  double dP;    // basınç düşümü
  double Uortalama;
  double Cportalama;
  double L; //fin height
  double W; //channel w
  double H; //chnnel height
  double eod;      // eps/di
  double Tsoguk[];
  double Tsicak[];
  double x[];      // heat exchanger length coordinate
  double Q;        // heat transfer
  double Qlmtd;    // heat transfer LMTD mehod
  double LMTD;
  double KL;       // local pressure drop coefficient
  double Aboru;    // pipe surface area
  double msoguk;   // cold fluid mass flow rate
  double Vsicak;   //hot gas velocity m/s
  double msicak;    // hot gas mass flow rate
  double Ric,Rdis;  // fouling factors
  double kboru;     // pipe and fin thermal conductivity
  double dL;        // finite difference length
  int n_finite;     // finite difference step
  double rosicak;   // hot gas temp.
 public circular_fin_HE_FD(String isoguk_sivi,
 String isicak_sivi,String gi[],double gni[],
 double si,double deltai,double Di,int n_sirai,int n_borui, double epsi,
 double dii, double douti, double Li, double Wi, double Hi,double Vsicaki,
 double msoguki,double Kboru, double Rici, double Rdisi,double KLi,
 double dLi)
 {soguk_sivi=new liquid_CS(isoguk_sivi);
  sicak_sivi=new Gmix(isicak_sivi,gi,gni);
  st1=new steam();
  double nsteam=0;
  for(int i=0;i<gi.length;i++)
  {if(gi[i].equals("h2o")) nsteam=gni[i];}
  s=si;
  delta=deltai;
  D=Di;
  n_sira=n_sirai;
  n_boru=n_borui;
  eps=epsi;
  di=dii;
  dout=douti;
  fi=(D/dout-1)*(1+0.35*Math.log(D/dout));
  hf=(D-dout)/2.0;
  L=Li;
  W=Wi;
```

```
  H=Hi;
  nkanat=(int)(L/(s+delta));
  Ric=Rici;
  Rdis=Rdisi;
  dL=dLi;
  KL=KLi;
  n_finite=(int)(L/dL+0.0001)+1;
  Tsoguk=new double[n_finite*n_sira];
  // pipes connected in series
  Tsicak=new double[n_finite*n_sira];  //each row has a constant temperature
  x=new double[n_finite*n_sira];
  Q=0;
  Vsicak=Vsicaki;
  kboru=209; //Aliminium W/mK
  msoguk=msoguki;
  }
  // eps    : pipe equivalent roughness m
  //         ------------------------------------------
  //         value of equivalent roughness for some pipes:
  //         Pipe            equivalent rougness m
  //         ------------    --------------------
  //         Riveted steel    0.9e-3 - 9e-3
  //         Concrete         0.3e-3 - 3e-3
  //         Wood stave       0.18e-3 - 0.9 e-3
  //         Cast iron        0.26e-3
  //         Galvanised iron  0.15e-3
  //         Commercial steel 0.045e-3
  //         wrought iron     0.045e-3
  //         drawn tubing     0.0015e-3
  //         Plastic,glass    0.0 (smooth)
  //         ------------------------------------------
// inside tube
 public double f(double Re)
 {
 //Haaland equation
 //double f1=-1.8*Math.log10(Math.pow((eod/3.7),1.11)+6.9/ Re);
 //f1=1.0/(f1*f1);
 //Swamee-Jain equation
 //double A=Math.log10(eod/3.7+5.74/Math.pow(Re,0.9));
 //double f1=0.25/(A*A);
 //Serghides equation
 //double A1=-2.0*Math.log10(eod/3.7+12.0/Re);
 //double B1=-2.0*Math.log10(eod/3.7+2.51*A1/Re);
 //double C1=-2.0*Math.log10(eod/3.7+2.51*B1/Re);
 //double f1=A1-((B1-A1)*(B1-A1))/(C1-2.0*B1+A1);
 //f1=1.0/(f1*f1);
 //System.out.println("Sergides equation f1="+f1);
 //Goudar equation
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 //System.out.println("Goudar equation f1="+f1);
 return f1;
 }
//boru içi
public double fx(double X,double Re)
{
// colebrook equation to solve
// friction factor for turbulent flow 2000 < Re
double xx=2.0*Math.log10(eod/3.7+2.51/Re*X)+X;
return xx;
}
//boru içi
public double dfx(double X,double Re)
{
//derivative of colebrook equation
double xx;
xx = 1+2.0/(eod/3.7+2.51/Re*X)/Math.log(10.0)*2.51/Re;
return  xx;
```

```
}
//boru içi
public  double fcol(double Re)
{
// solution of the colebrook equation
// by using newton method
double fi=f(Re);
double xx=1.0/ Math.pow(fi,0.5);
int nmax=50;
double tolerance=1.0e-10;
for(int i=0;i<nmax;i++)
{
double fx1=fx(xx,Re);
xx-=fx1/dfx(xx,Re);
if(Math.abs(fx1)<tolerance)
{double ff=1.0/(xx*xx);
return ff;}
}
double ff=1.0/(xx*xx);
//System.out.println("fcol="+ff);
return ff;
}
//inside tube
public double  velocity(double t)
{
// velocity of the pipe m/x
double x=0.0;
double mboru=msoguk/n_boru;
double ro=soguk_sivi.rd.rol(t+273.15);
double A=Math.PI*di*di/4.0;
return mboru/ro/A;
}

//inside tube
public double  Re(double t)
{
// reynold's number boru içi
double ro=soguk_sivi.rd.rol(t+273.15);
double vis=soguk_sivi.rd.viscosity_l(t+273.15);
double vv=velocity(t);
double res=(di*vv*ro/vis);
return res;
}
//inside tube
public double  hc(double t)
{
//convective heat transfer coefficient
double Rey=Re(t);
double Nu=0;
double Prt=1.0;
if(Rey<=2300)
{  //Laminar region
                Nu=3.66;
}
else if(Rey>2300 && Rey<=3100)
{  //Referans : Heat transfer in all pipe flow regimes:laminar,transitional/intermittent, and turbulent
   // J.P. Abraham, E.M. Sparrow, J.C.K. Tong, International Journal of Heat and Mass Transfer 52(2009) 557-563
                double Re1=(Rey/1000.0);
                Nu=2.2407*Re1*Re1*Re1*Re1-29.499*Re1*Re1*Re1+142.32*Re1*Re1-292.51*Re1+219.88;}
else
  {
  double fr  = f(Rey);
  Prt = soguk_sivi.rd.Pr(t+273.15);
  // Turbulent region Gnielinski equation
  // Referance:
  // V. Gnielinski, New equations for heat and mass transfer in turbulent pipe and
  //channel flow, Int. Chem. Eng. 16 (1976) 359–367.
  // Valid for 2000 < Re <5e5, 0.5 < Pr < 2000
  Nu=fr/8.0*(Rey-1000.0)*Prt/
  (1+12.7* Math.sqrt(fr/8.0)*( Math.pow(Prt,(2.0/3.0))-1.0));
  }
  double kt  =  soguk_sivi.rd.k_l(t+273.15);
  double ss=Nu*kt/di;
  //System.out.println("Re="+Rey+"Nu="+Nu+"h="+ss+"Prt="+Prt+"kt="+kt+"di="+di);
  return ss;
}
```

```
// outside tube
public double Reout(double t)
{
//======= Reynold sayısı boru dışı ==========
//x : % karışım miktarı
//t sıvı sıcaklığı
//boru geçiş alanı
double As=L*W-n_boru*((L-nkanat*delta)*dout+D*delta*nkanat);  //m^2 giriş alanı
double A=L*W;
double V=Vsicak*A/As; //düzeltilmiş hız
double ro=sicak_sivi.M/sicak_sivi.v((t+273.15),1.01325);
double vis=sicak_sivi.vis(t+273.15);
double res=V*ro*dout/vis;
msicak=ro*V*A;
//System.out.println("As="+As+"L="+L+"W="+W+"n_boru="+n_boru+"nkanat"+nkanat+"delta="+delta+"dout="+dout+"A="+A+"
V="+V+"Vsicak="+Vsicak+"Re="+res+"ro="+ro+"dout="+dout);
return res;
}
public double hout(double t)
{
// t  sıvı sıcaklığı
//====== Boru dışı taşınım ısı transferi katsayısı W/M^2 K ===========
//                  kanatlı boru dizini
double Re=Reout(t);
double vis=sicak_sivi.vis(t);
double k=sicak_sivi.k(t);
double Pr=sicak_sivi.Prandtl(t);
double A_At0_ratio=1.0+2.0*hf*(hf+dout+delta)/(s*dout);
double Nu=0.134*Math.pow(Re,0.681)*Math.pow((s/hf),0.2)*Math.pow((s/delta),0.1134)*Math.pow(Pr,(1.0/3.0));
double ho=k/dout*Nu;
//System.out.println("Re="+Re+"vis="+vis+"k="+k+"Pr="+Pr+"A_At0_ratio="+A_At0_ratio+"Nu="+Nu+"ho="+ho);
return ho;
}

public double X(double t)
{double hm=hout(t);
 return fi*dout/2.0*Math.sqrt(2.0*hm/(kboru*delta));
}
public double eta(double t)
{ //kanat verimi
  double x=X(t);
  return Math.tanh(x)/x;
}

public double houtv(double t)
{ double eta=eta(t);
  double Af=2.0*Math.PI/4.0*(D*D-dout*dout)*nkanat;
  double A=Math.PI*dout*(L-nkanat*delta)+Af;
  return hout(t)*(1-(1-eta)*Af/A);
}

public double U0(double tic,double tdis)
{
double hi=hc(tic);    // boru içi ısı transferi
double ho=houtv(tdis); // boru dışı ısı transferi
double Ai=Math.PI*di*L;
double Af=2.0*Math.PI/4.0*(D*D-dout*dout)*nkanat;
double A=Math.PI*dout*(L-nkanat*delta)+Af;
Aboru=A*n_boru*n_sira;
double Uboru=1.0/(1.0/ho+A/Ai*(1.0/hi+(dout-di)/(2.0*kboru)));
return Uboru;
}

public void hesapla(double tic, double tdis)
{   Q=0;
            double dq=0;
            double roavg=0;
            double Vavg=0;
            double ms=msoguk/n_boru; //bir borudaki kütlesel debi
            double rosicak=sicak_sivi.M/sicak_sivi.v((tic+273.15),1.01325);
            msicak=rosicak*Vsicak*L*W;  //sabit
            double msic=msicak/n_boru;
            double Af=2.0*Math.PI/4.0*(D*D-dout*dout)*nkanat;
   double A=Math.PI*dout*(L-nkanat*delta)+Af;
   Aboru=A*n_boru*n_sira;
            dP=0;
   double xx=0;
```

```
    double dtic=0;
    double dtdis=0;
    double Cp=0;
    Uortalama=0;
    Vavg=0;
    double uu=0;
    double dA=A/n_finite;
    double dq1=0;
                for(int k=0;k<n_sira;k++)
                {
                 Cp=sicak_sivi.Cp(tdis+273.15)/sicak_sivi.M;
                 Cportalama+=Cp;
                for(int i=0;i<n_finite;i++)
                {   double Rey=Re(tic);
                    double f=fcol(Rey);
                    Vavg+=velocity(tic);
                                uu=U0(tic,tdis);
                                Uortalama+=uu;
                                dq=uu*dA*(tic-tdis);
                                dq1+=dq;
                                dtic=dq/(ms*soguk_sivi.rd.Cpl(tic+273.15)*1.0e3);
                                dtdis=dq/(msic*Cp*1.0e3);
                    Tsoguk[k*n_finite+i]=tic;
                    Tsicak[k*n_finite+i]=tdis;
                    x[k*n_finite+i]=xx;
                    tic-=dtic;
                    tdis+=dtdis;
        xx+=dL;
                }
                Q+=dq1;
                //dtdis=dq1/(msic*Cp*1.0e3);
                tdis-=dtdis;
                }
    Q*=n_boru;
    roavg/=n_finite;
    Vavg/=n_finite;
    Uortalama/=(n_finite*n_boru);
    Cportalama/=n_boru;
    double dT1=Tsoguk[0]-Tsicak[0];
    double dT2=Tsoguk[n_finite*n_boru-1]-Tsicak[n_finite*n_boru-1];
    LMTD=(dT1-dT2)/Math.log(dT1/dT2);
    Qlmtd=Uortalama*Aboru*LMTD;
    dP+=KL*roavg*Vavg*Vavg/2.0;
}

public double hesapla_ters1(double tic, double tdisa)
{
    Q=0;
                double dq=0;
                double roavg=0;
                double Vavg=0;
                double ms=msoguk/n_boru; //bir borudaki kütlesel debi
                double rosicak=sicak_sivi.M/sicak_sivi.v((tic+273.15),1.01325);
                msicak=rosicak*Vsicak*L*W;  //sabit
                double msic=msicak/n_boru;
                double Af=2.0*Math.PI/4.0*(D*D-dout*dout)*nkanat;
    double A=Math.PI*dout*(L-nkanat*delta)+Af;
    Aboru=A*n_boru*n_sira;
    //System.out.println("A="+A+"Aboru="+Aboru+"Af="+Af+"D="+D+"dout="+dout+"L="+L+"nkanat="+nkanat+"delta="+delta);
                dP=0;
    double xx=0;
    double dtic=0;
    double dtdis=0;
    double Cp=0;
    Uortalama=0;
    Vavg=0;
    double uu=0;
    double dA=A/n_finite;
    double dq1=0;
                for(int k=0;k<n_sira;k++)
                {
                 Cp=sicak_sivi.Cp(tdisa+273.15)/sicak_sivi.M;
                 Cportalama+=Cp;
                for(int i=0;i<n_finite;i++)
                {   double Rey=Re(tic);
                    double f=fcol(Rey);
                    Vavg+=velocity(tic);
```

```
                        uu=U0(tic,tdisa);
                        Uortalama+=uu;
                        dq=uu*dA*(tic-tdisa);
                        dq1+=dq;
                        dtic=dq/(ms*soguk_sivi.rd.Cpl(tic+273.15)*1.0e3);
                        dtdis=dq/(msic*Cp*1.0e3);
                Tsoguk[k*n_finite+i]=tic;
                Tsicak[k*n_finite+i]=tdisa;
                x[k*n_finite+i]=xx;
                tic+=dtic;
                tdisa+=dtdis;
        xx+=dL;
                }
                Q+=dq1;
                }
    Q*=n_boru;
    roavg/=n_finite;
    Vavg/=n_finite;
    Uortalama/=(n_finite*n_boru);
    Cportalama/=n_boru;
    double dT1=Tsicak[0]-Tsoguk[0];
    double dT2=Tsicak[n_finite*n_boru-1]-Tsoguk[n_finite*n_boru-1];
    LMTD=(dT1-dT2)/Math.log(dT1/dT2);
    Qlmtd=Uortalama*Aboru*LMTD;
    dP+=KL*roavg*Vavg*Vavg/2.0;
    return  Tsoguk[n_boru*n_finite-1];
}
public void plot()
{
 Plot pp=new Plot(x,Tsoguk);
 pp.addData(x,Tsicak);
 Locale us=new Locale("US");
 String s1="Q = %4.1f kW A = %4.1f m^2 dP = %4.1f kPA U = %4.0f W/m^2K";
 String s4=String.format(us,s1,Q*1e-3,Aboru,dP*1e-3,Uortalama);
 pp.setPlabel(s4);
 pp.setXlabel("x, m");
 pp.setYlabel("T sıcaklık , derece C");
 pp.plot();
}

public String toString()
{String s="Heat transfer round fin \n";
s+="internal fluid = "+soguk_sivi.rd.liquidName+"\n";
s+="external fluid = "+sicak_sivi.gasName+"\n";
s+=sicak_sivi.toString(sicak_sivi.gasName)+"\n";
s+="m_internal = "+msoguk+"\n kg/s";
s+="m_external = "+msicak+"\n kg/s";
s+="inside pipe surface roughness : "+eps+"\n";
s+="internal diameter : "+di+"\n";
s+="external diameter : "+dout+"\n";
s+="Pressure drop : "+dP+"\n";
s+="average heat transfer coefficient : "+Uortalama+"\n";
s+="average heat capacity = "+Cportalama+"kJ/kg K\n";
s+="Heat transfer : "+Q+"\n";
s+="heat transfer area : "+Aboru+"\n";
s+="cold fluid inlet-exit temperatures : "+"\n";
s+=Tsoguk[0]+"   "+Tsoguk[n_boru*n_finite-1]+"\n";
s+="hot fluid inlet-exit temperatures : "+"\n";
s+=Tsicak[0]+"   "+Tsicak[n_boru*n_finite-1]+"\n";
s+="LMTD ="+LMTD+"derece K\n";
s+="Q_LMTD ="+Qlmtd+"W\n";
return s;
}



public void hesapla_ters(double tic, double tdis)
{
//ters akışlı sistem
//tic  : soğuk sıvı giriş sıcaklığı
//tdis : sıcak sıvı giriş sıcaklığı
double dT=(tic-tdis)/30;
double Tgir[]=new double[30];
double Tcik[]=new double[30];
for(int i=0;i<30;i++)
{ Tgir[i]=tic-dT*i;
```

```
  Tcik[i]=hesapla_ters1(Tgir[i],tdis);
  //System.out.println("i="+i+" "+Tcik[i]+" "+Tgir[i]);
}
double e[]=EKK(Tcik,Tgir,7);
double tgir1=funcEKK(e,tic);
double tdis2=hesapla_ters1(tgir1,tdis);
    double dT1=Tsoguk[0]-Tsicak[0];
    double dT2=Tsoguk[n_boru*n_finite-1]-Tsicak[n_boru*n_finite-1];
    LMTD=(dT1-dT2)/Math.log(dT1/dT2);
    Qlmtd=Uortalama*Aboru*LMTD;
}


//en kucuk kareler metodu
  public static double[] pivotlugauss(double a[][],double b[])
  { //kısmi pivotlu gauss eleme yöntemi
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss eleme
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivotlama
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                   if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
                  b[k]=dummy;
                }
                //gauss elemeyi çözme
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
                }
  }
  //geriye doğru yerine koyma
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}

public static double[] EKK(double xi[],double yi[],int n)
{
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  for(j=0;j<n+1;j++)
     {if(i==0 && j==0) A[i][j]=l;
```

```
      else  for(k=0;k<l;k++) A[i][j] += Math.pow(xi[k],(i+j));
      }
      for(k=0;k<l;k++) { if(i==0)  B[i]+= yi[k];
                  else      B[i] += Math.pow(xi[k],i)*yi[k];}
   }
//System.out.println(Matrix.toString(A));
//System.out.println(Matrix.toStringT(B));
X=pivotlugauss(A,B);
//X=B/A;
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
return X;
}

public static double funcEKK(double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=e.length;
double ff;
if(n!=0.0)
  { ff=e[n-1];
  for(int i=n-2;i>=0;i--)
   { ff=ff*x+e[i]; }
  }
else
  ff=0;
return ff;
}

public static double hata(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcEKK(e,x[k]))*(y[k]-funcEKK(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

public static double[][]  funcEKK(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
{
//aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
// ara değer sayısı 2 dir
   int n=xi.length;
   int nn=(n-1)*(aradegersayisi+1)+1;
   double z[][]=new double[2][nn];
   double E[]=EKK(xi,yi,polinomkatsayisi);
   System.out.println("katsayılar :\n"+Matrix.toStringT(E));
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
      {z[0][k]=xi[i];z[1][k]=funcEKK(E,z[0][k]);k++;
      for(int j=0;j<aradegersayisi;j++)
      {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
      z[0][k]=z[0][k-1]+dx;z[1][k]=funcEKK(E,z[0][k]);k++;}
      }
      z[0][k]=xi[i];z[1][k]=funcEKK(E,z[0][k]);
   return z;
 }

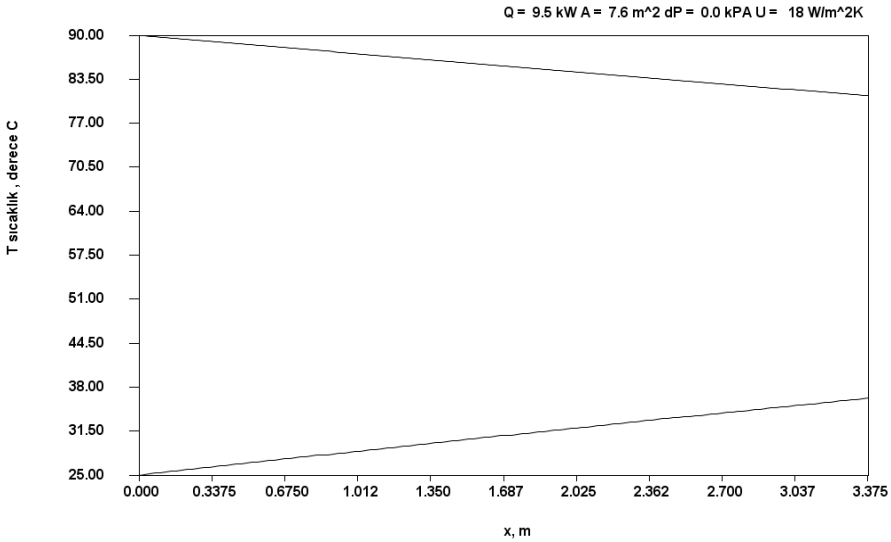
}
```

```
 import java.util.*;
import java.lang.*;
import java.io.*;
import javax.swing.*;
```

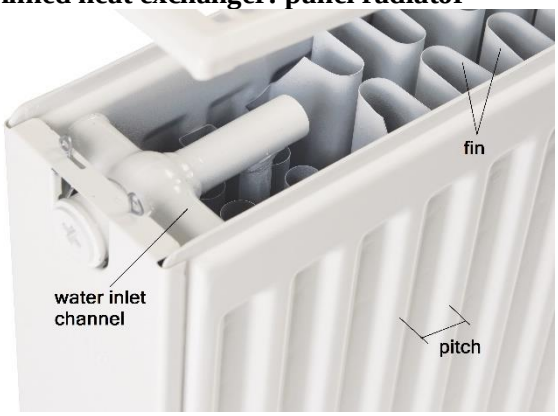
```
class circular_fin_HE_FDtest{
    public static void main (String args[]) throws IOException
    {
    String isoguk_sivi="water"; //water
    String isicak_sivi="air"; //air
    String gi[]={"air"};
    double gni[]={1.0};
    double si=1.2e-3;//distance between fins
    double deltai=0.1e-3; // fin thickness
    double Di=32e-3;       // fin diameter
    int n_sirai=4;         // fin rows
    int n_borui=2;         // number of pipes in each row
    double epsi=1.0e-4;    // pipe inlet surface roughness m
    double dii=9.0e-3;     // pipe inlet diameter
    double douti=10.0e-3;  // pipe outlet diameter
    double Li=0.275*3;     // pipe length in a row
    double Wi=0.08;        // heat exchanger width
    double Hi=0.61;        // heat exchanger height
    double Vsicaki=5;    // gas velocity m/s
    double msoguki=0.1;  // water mass flow rate kg/s
    double Kboru=50;      // boru
    double Rici=0.0;  // fouling resistance inside pipe
    double Rdisi=0.0;     // fouling resistanceoutside pipe
    double KLi=5.0;        // local pressure drop coefficient
    double dLi=25e-3;      // finite difference step length
    circular_fin_HE_FD id1=new circular_fin_HE_FD(isoguk_sivi,
    isicak_sivi,gi,gni,si,deltai,Di,n_sirai,n_borui,epsi,
     dii, douti, Li, Wi, Hi,Vsicaki,msoguki,Kboru, Rici, Rdisi,KLi,dLi);
    id1.hesapla(90,25);
    id1.plot();
    System.out.println(id1.toString());
    }
}
```

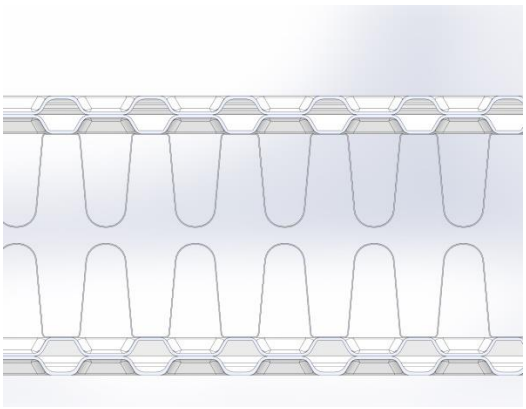
```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" circular_fin_HE_FDtest
Heat transfer round fin
internal fluid = su
external fluid = air

m_internal = 0.1
 kg/sm_external = 0.5317654376668579
 kg/sinside pipe surface roughness : 1.0E-4
internal diameter : 0.009
external diameter : 0.01
Pressure drop : 0.0
average heat transfer coefficient : 18.400592953827353
average heat capacity = 2.009772585324642kJ/kg K
Heat transfer : 9540.765388934233
heat transfer area : 7.552991925019351
cold fluid inlet-exit temperatures :
90.0   85.46983956870368
hot fluid inlet-exit temperatures :
25.0   30.81973790314732
LMTD =59.675538523677126derece K
Q_LMTD =8293.678296268454W
```



A finned heat exchanger: panel radiator





Panel radiators are finned natural convective heat exchangers. General view of the device and fin profile is shown in the figures. Water flows inside chanels and outside air is heated up through finned and bare surfaces of heat exchanger. Basic simulation is similar to the previous ones except natural convection is considered.

```
public class HT_radiator_elba
{  public double Ra,Gr,Re,U,m,Ts,Tw;
   public double Pr,mu,nu,ro,Cp,k,beta,n,alpha,g,kmetal; //m/s^2
   //input variables
   public double l_fin;    // length of fin
   public double l_base1;  // length outside region
   public double L;        // length of radiator
   public double t_fin;    // fin thickness
   public double t_panel;  // panel thickness
   public double P;        // pressure
   public double dx;       // finite element height of one step
   public int N;           // number of finite element steps
   public double m_water;  // water mass flow rate
   public double km;       // Thermal conductivity of metal
   public double AC1;      // cross sectional area AC1
   public double AC2;      // cross sectional area AC2
   public double ACW;      // water channel area
   public double l_water;  // length of the
   public double A1;       // fin+base facial area 1
   public double A2;       // fin+base facial area 2
   public double Twi;      //water inlet temperature
   public double Tai;      //air inlet temperature
   public double Dh;       //hydrolic diameter of water channel
   public double Uair;     //natural convection air velocity
   public double Rea1,Rea2,Rew; //air channels and water reynolds numbers
     double eps;
   double eod;
   steamIAPWS_IF97 st;
   public HT_radiator_elba(double l_fini,double l_base1i,double l_base2i,double Li,double t_fini,double t_paneli,double Pi,int Ni,
```

```
   double m_wateri,double AC1i,double AC2i,double ACWi,double l_wateri,double epsi)
   {st=new steamIAPWS_IF97();
              l_fin=l_fini;

    eps=epsi;
    l_base1=l_base1i;
    L=Li;
    t_fin=t_fini;
    t_panel=t_paneli;
   P=Pi;
   N=Ni;
   m_water=m_wateri;

   AC1=AC1i;
   AC2=AC2i;
   ACW=ACWi;
   l_water=l_wateri;
   Dh=4*ACW/l_water;
   eod=eps/Dh;
   }
   // air viscosity P=101.325 kPa
  public static  double mu(double T)
  {double a[]={-7.00796942E-01,
8.85836561E-02,
-1.20909367E-04,
1.73074497E-07,
-1.45011707E-10,
-4.33936641E-14,
3.14868466E-16,
-4.41548734E-19,
3.25615211E-22,
-1.28963912E-25,
2.16661936E-29};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-6;
return y;
}
//air thermal conductivity P=101.325 kPa
public static double k(double T)
{double a[]={-5.96297846E-01,
1.00247973E-01,
7.32722776E-05,
-1.02111800E-06,
4.21691040E-09,
-1.04115589E-11,
1.67088379E-14,
-1.75371889E-17,
1.16134919E-20,
-4.40290627E-24,
7.28284943E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
y*=1e-3;
return y;
}
//air Prandtl number P=101.325 kPa
public static double Pr(double T)
{double a[]={1.47352515E+00,
-1.73708682E-02,
1.80473009E-04,
-1.07436904E-06,
3.99304707E-09,
-9.67956666E-12,
1.55567247E-14,
-1.64359568E-17,
1.09674418E-20,
-4.18869082E-24,
6.97523663E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
return y;
}
// air specific heat P=101.325 kPa
public static  double Cp(double T)
{double a[]={1.89668613E+00,
-2.23937197E-02,
```

```
2.40885259E-04,
-1.45254290E-06,
5.43552393E-09,
-1.32310554E-11,
2.13239465E-14,
-2.25744420E-17,
1.50864395E-20,
-5.76865768E-24,
9.61540784E-28};
double T2=T*T,T3=T2*T,T4=T2*T2,T5=T4*T,T6=T4*T2,T7=T6*T,T8=T4*T4,T9=T8*T,T10=T9*T;
double y=a[0]+a[1]*T+a[2]*T2+a[3]*T3+a[4]*T4+a[5]*T5+a[6]*T6+a[7]*T7+a[8]*T8+a[9]*T9+a[10]*T10;
return y*1e3;
}
  //air density ro(T,P)
  public  static double ro(double T,double P)
  {   double M=2.8965000E+01;
      double R=8314.5/M;
                return P*1e3/R/T;
  }

  // air Nusselt number
  public  double h(double Ts,double Tinf,double P,double L,double D)
  {//Nusselt number
   this.Ts=Ts;
   double T=(Tinf+Ts)/2;
   double dx=L/N;
   Ra=Ra(Ts,Tinf,P,dx);
   Pr=Pr(T+273.15);
   k=k(T+273.15);
   Gr=Ra/Pr;
   Re=Math.sqrt(Gr);
   ro=ro(T+273.15,P);
   Uair=Re*mu/(ro*D);
   double Nu=0.0;
   double Racritic=1e9;
   double Pr_05=Math.sqrt(Pr);
   if(Ra>Racritic )
{Nu=(0.825+0.387*Math.pow(Ra,(1.0/6.0))/Math.pow((1+Math.pow((0.492/Pr),(9.0/16.0))),(8.0/27.0)));Nu=Nu*Nu;}
   else
   { Nu=(0.68+0.67*Math.pow(Ra,(1.0/4.0))/Math.pow((1+Math.pow((0.492/Pr),(9.0/16.0))),(4.0/9.0)));}
   double h=Nu*k/dx;
   return h;
   }

   //metal thermal conductivity
   public  double kmetal(double t)
  {double T=t;
   double a=71.62587939698594-0.03447361809045638*T-7.927135678388277E-6*T*T;
  return a;
  }
  //fin efficiency
   public  double eta_fin(double Ts,double h,double l_fin,double l_base,double t_fin)
   {   double M=Math.sqrt(2*h/kmetal(Ts)/t_fin)*l_fin;
       double eta=(l_fin*Math.tanh(M)/M+l_base)/(l_fin+l_base);
       return eta;
   }
   //heat transfer
   public  double Q_fin(double Ts,double Tinf,double dx,double h, double k,double l_fin,double l_base,double t_fin)
   {  // Ts surface temperature
      // Tinf air temperature
      // dx finite difference length
      // h convective heat transfer coefficient
      // k thermal conductivity of metal
      // l_fin length of fin
      // A area
                double eta=eta_fin(Ts,h,l_fin,l_base,t_fin);
                double A=(l_fin+l_base)*dx;
       double Q=eta*A*h*(Ts-Tinf);
       return Q;
   }

   public  double Ra(double Ts,double Tinf,double P,double L)
   {  // Ts surface temperature
      // Tinf air temperature
      // T average temperature
      // P air pressure
      // L radiator length (full length)
```

```
                double T=(Ts+Tinf)/2.0;
                mu=mu(T+273.15);
     ro=ro(T+273.15,P);
     Cp=Cp(T+273.15);
     k=k(T+273.15);
     beta=1/(T+273.15);
     nu=mu/ro;
     alpha=k/(ro*Cp);
     g=9.806; //m/s^2
     Ra=g*beta*(Ts-Tinf)*L*L*L/alpha/nu;
                return Ra;
   }

  //pipe inlet region
  public double f(double Re)
 {
 //Goudar equation
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }

public double  velocity(double t)
{
// velocity of the pipe m/x
double x=0.0;
double a[]=st.propertyC("pt",P,t);
//double mpipe=mass_flow_rate_inlet/n;
double ro=a[15]; //kg/m^3
return m_water/(ro*ACW);
}

public double  Re(double t)
{
// reynold's number pipe inlet
double a[]=st.propertyC("pt",P,t);
//double mpipe=mass_flow_rate_inlet/n;
double ro=a[15]; //kg/m^3
double vis=a[11];
double vv=velocity(t);
double res=(Dh*vv*ro/vis);
return res;
}

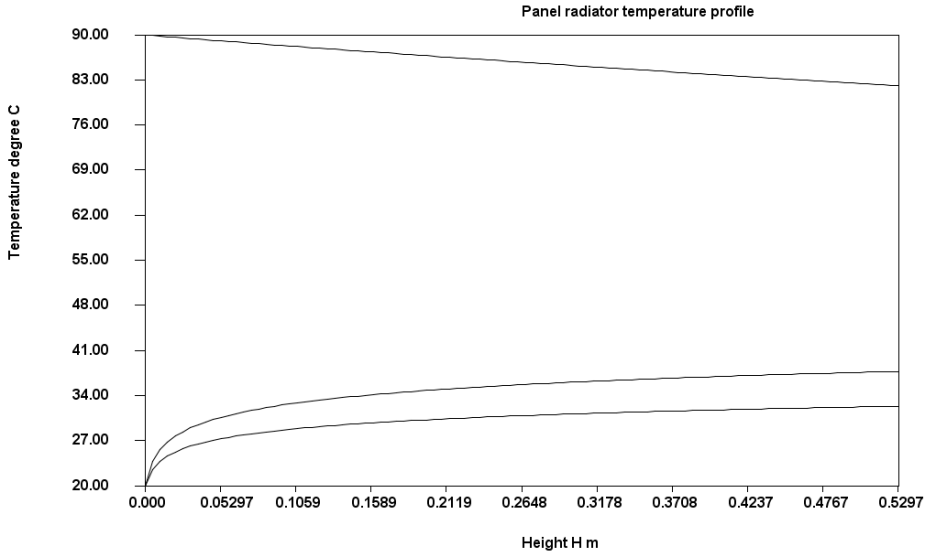
public double  hc(double t)
{
// convective heat transfer coefficient
double Rey=Re(t);
double Nu=0;
if(Rey<2300)
  { Nu=3.66; }
else
  {
  double fr  = f(Rey);
  double a[]=st.propertyC("pt",P,t);
  double Prt = a[14];
  // Gnielinski equation
  // Gnielinski, V, Int. Chem Eng., 16, 359,1976
  // Valid for 2000 < Re <5e5, 0.5 < Pr < 2000
  Nu=(fr/2.0)*(Rey)*Prt/(1+8.7* Math.sqrt(fr/2.0)*(Prt-1.0));
  }
  double a[]=st.propertyC("pt",P,t);
  double kt  =  a[12];
  double ss=Nu*kt/Dh;
  return ss;
}

  public static void main(String arg[])
```

```
{
 double L=535e-3; // m
 int N=100;
 double dx=L/N;
 double W=1;
 double l_fin=41.6e-2;
 double l_base1=19.76e-3;
 double l_base2=3.53e-3;
 double t_fin=0.35e-3;
 double t_panel=1.3e-3*2;
 double m_water=0.0014;   // kg/s
 double m_air1,m_air2;
 double  V1=0,V2=0;
 double AC1=275.4958e-6; //m^2
 double AC2=1372.8645e-6/4.0; //m^2
 double ACW=60e-6; //m^2
 double l_water=22e-3; //m
 double eps=1e-6;
 double P=101.325; //kPa
 HT_radiator_elba hte=new HT_radiator_elba(l_fin,l_base1,l_base2,L,t_fin,t_panel,P,N,m_water,AC1,AC2,ACW,l_water,eps);
 double tw=90;
 double ts1=20;
 double ts2=20;
 double ts3=20;
 double hw,dha1,dha2,Nu1,k1,h1,U1,Nu2,h2,k2,U2,Nu3,h3,k3,U3,eta1,eta2,dA1,dA2,dA3,dQ1=0,dQ2=0,dQ3=0,dQ=0;
 double Q1=0;
 double Q2=0;
 double Q3=0;
 double Q=0;
 double Cpw=0;
 dA1=(l_fin+l_base1)*dx;
 dA2=(l_fin+l_base2)*dx;
 dA3=(l_base1+l_base2)*dx;
 double T1[]=new double[hte.N];
 double T2[]=new double[hte.N];
 double xx[]=new double[hte.N];
 double TW[]=new double[hte.N];
 for(int i=0;i<hte.N;i++)
 {   TW[i]=tw;
          T1[i]=ts1;
          T2[i]=ts2;
          xx[i]=i*dx;
        double a[]=hte.st.propertyC("pt",P,tw);
        Cpw=a[7]*1e3;
 hw=hte.hc(tw);
 dha1=4.0*AC1/(l_fin+l_base1);
 dha2=4.0*AC2/(l_fin+l_base2);
 h1=hte.h(tw,ts1,P,L,dha1);
 V1+=hte.Uair;
 U1=1.0/(1.0/hw+1/h1);
 h2=hte.h(tw,ts2,P,L,dha2);
 V2+=hte.Uair;
 U2=1.0/(1.0/hw+1/h2);
 h3=hte.h(tw,ts3,P,L,dha2);
 U3=1.0/(1.0/hw+1/h3);
 eta1=hte.eta_fin(ts1,h1,l_fin,l_base1,t_fin);
 eta2=hte.eta_fin(ts2,h2,l_fin,l_base2,t_fin);
 dQ1=dA1*eta1*U1*(tw-ts1);
 Q1+=dQ1;
 dQ2=dA2*eta2*U2*(tw-ts2);
 Q2+=dQ2;
 dQ3=dA3*U3*(tw-ts3);
 Q3+=dQ3;
 dQ=dQ1+dQ2+dQ3;
 Q+=dQ;
 tw=tw-dQ/(Cpw*m_water);
 m_air1=V1*ro(ts1+273.15,P)*AC1;
 double Cp= Cp(ts1+273.15);
 ts1=ts1+dQ1/(m_air1*Cp);
 if(ts1>tw) ts1=tw-0.01;
 m_air2=V2*ro(ts2+273.15,P)*AC2;
 Cp= Cp(ts1+273.15);
 ts2=ts2+dQ2/(m_air2*Cp);
 if(ts2>tw) ts2=tw-0.01;
 }
 Plot p=new Plot(xx,TW);
 p.addData(xx,T1);
```

```
    p.addData(xx,T2);
    p.setPlabel("Panel radiator temperature profile");
    p.setXlabel("Height H m");
    p.setYlabel("Temperature degree C");
    p.plot();
    System.out.println("dQ1="+dQ1+"dQ2="+dQ2+"dQ3="+dQ3+"dQ="+dQ);
    System.out.println("Q1="+Q1+"Q2="+Q2+"Q="+Q);
    System.out.println("total Q="+(29*(Q1+Q2)+30*Q3));
    }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" HT_radiator_elba
dQ1=0.1678773262658881dQ2=0.12554780100475665dQ3=0.12261414738580062dQ=0.41603927465644536
Q1=19.76832797361413Q2=14.233562872437522Q=47.15275114510011
```



16.7 CHEVRON PLATE HEAT EXCHANGER HEAT TRANSFER

Şevron Levha tipi condenser ısı transferi denklemleri içinde ısı transferi ve basınç düşümü denklemlerine göz atalım. In chevron type plate heat exchangers chevron angle β is an important property for overall heat transfer and pressure drop.

Friction factor can be calculated s :

$$\frac{1}{\sqrt{f}} = \frac{cos\beta}{\sqrt{0.045tan\beta+0.09sin\beta+f_0/cos\beta}} + \frac{1-cos\beta}{\sqrt{3.8f_1}}$$

f_0, in this equation

$f_0 = \frac{20}{Re}$ for $Re \leq 2000$

$f_0 = (1.56\ln(Re) - 3.0)^{-2}$ for Re>2000

f_1, in this eqaution

$f_1 = \frac{149}{Re} + 9.965$ for $Re \leq 2000$

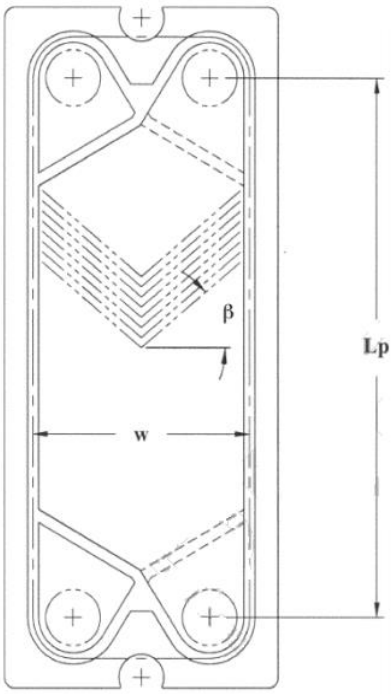
$f_1 = \frac{9.75}{Re^{0.289}}$ for Re>2000

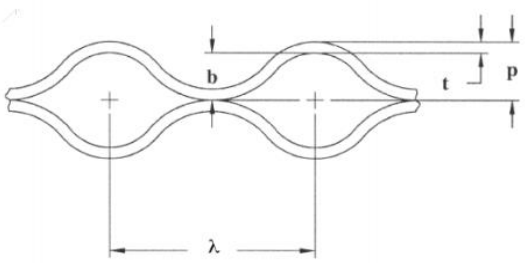
Hydrolic diameter: $d_h = \frac{2b}{\Phi}$ where Φ is an area increment factor

$\Phi(X) = \frac{1}{6}\left(1 + \sqrt{1+X^2} + 4\sqrt{1+X^2/2}\right)$ $X = \frac{2\pi b}{\lambda}$

where b is the distance from center of chevron to the top of chenron (as shown in the figure), λ is the length of each chevron wave. Velocity can be defined as

U=$\frac{m}{\rho bw}$ where fluid mass flow rate, b is the chevrin height and w is the plate width





With this definition we can use standart Re number definition for equations

Re=$\frac{\rho U d_h}{\mu}$

Nu number can be calculated as:

$Nu = 0.205 Pr^{1/3} \left(\frac{\mu_m}{\mu_w}\right)^{1/6} [f Re^2 \sin(2\beta)]^{0.374}$ (for liquids)

$Nu = 0.205 Pr^{1/3} [f Re^2 \sin(2\beta)]^{0.374}$ (for gases)

Error range of the equation for chevron angle of 10-80 degree is in the range of +/- 20%

```
import javax.swing.JOptionPane;
import java.util.Locale;

public class chevron_plate_heat_exchanger
{ liquid_CS soguk_sivi; //cold liquid
  liquid_CS sicak_sivi; //hot liquid
  String akis;
  double b;      // channel chevron height
  double fi;     // chevron angle
  double lambda; // chevron wave length
  double Lp;     // length of plate
  double Bp;     // width of plate
  double dP;     // pressure drop
  double t;      // metal thickness
  double Uortalama;//average heat transfer coefficient
  double Tsoguk[];
  double Tsicak[];
  double x[];      // heat transfer length coordinate
  double Q;        // heat transfer
  double Qlmtd;    // heat transfer LMTD method
  double LMTD;
  double di;       //hydraulic diameter
  double dL;       // finite difference length
  int nboru;       // finite difference step number
  int n;           // number of plates
  double msoguk;   // cold fluid flow rate
  double msicak;   // hot fluid flow rate
```

```
  double Ric,Rdis; // fouling factors
  double klevha;   // plate thermal conductivity
  double Aboru; //  total plate area
  double dP_soguk_sivi;
  double dP_sicak_sivi;
  double Uavg;

  public chevron_plate_heat_exchanger(String isoguk_sivi,
  String isicak_sivi,
  double bi, double fii,double lambdai, double Lpi,double Bpi,
  double imsoguk,
  double imsicak,
  double dLi,
  int ni,
  double klevhai,
  double ti)
 {soguk_sivi=new liquid_CS(isoguk_sivi);
  sicak_sivi=new liquid_CS(isicak_sivi);
  akis="tanımsız";
  b=bi;
  fi=fii;
  lambda=lambdai;
  Lp=Lpi;
  Bp=Bpi;
  Uortalama=0;
  dL=dLi;
  n=ni;
  nboru=(int)(Lp/dL+0.0001)+1;
  Tsoguk=new double[nboru];
  Tsicak=new double[nboru];
  x=new double[nboru]; //sıcaklık koordinatı
  Q=0;
  t=ti;
  Ric=0.00001;
  Rdis=0.00001;
  msoguk=imsoguk;
  msicak=imsicak;
  klevha=klevhai;
  double X=2.0*Math.PI*b/lambda;
  double F=1.0/6.0*(1+Math.sqrt(1+X*X)+4.0*Math.sqrt(1.0+X*X/2.0));
  di=2*b/F;
  }

public double  hiz_soguk_sivi(double t)
{
double mlevha=msoguk/n;
double ro=soguk_sivi.rd.rol(t+273.15);
return (mlevha/ro/b/Bp);
}

public double  hiz_sicak_sivi(double t)
{
double mlevha=msicak/n;
double ro=sicak_sivi.rd.rol(t+273.15);
return (mlevha/ro/b/Bp);
}

public double  Re_soguk_sivi(double t)
{
double ro=soguk_sivi.rd.rol(t+273.15);
double vis=soguk_sivi.rd.viscosity_l(t+273.15);
double vv=hiz_soguk_sivi(t);

double res=(di*vv*ro/vis);
return res;
}
public double  Re_sicak_sivi(double t)
{
double ro=sicak_sivi.rd.rol(t+273.15);
double vis=sicak_sivi.rd.viscosity_l(t+273.15);
double vv=hiz_sicak_sivi(t);
double res=(di*vv*ro/vis);
return res;
}

public double f_soguk_sivi(double t)
{
```

```
double Re=Re_soguk_sivi(t);
double f0,f1;
double xx=1.0;
if(Re<2000) f0=20.0/Re;
else        {xx=1.56*Math.log(Re)-3.0;f0=1.0/(xx*xx);}
if(Re<2000) {f1=149.0/Re+0.9625;}
else        {f1=9.75/Math.pow(Re,0.289);}
xx=Math.cos(fi)/Math.sqrt(0.045*Math.tan(fi)+0.09*Math.sin(fi)+f0/Math.cos(fi))+
   (1-Math.cos(fi))/Math.sqrt(3.8*f1);
double f=1.0/(xx*xx);
return f;
}

public double f_sicak_sivi(double t)
{
double Re=Re_sicak_sivi(t);
double f0,f1;
double xx=1.0;
if(Re<2000) f0=20.0/Re;
else        {xx=1.56*Math.log(Re)-3.0;f0=1.0/(xx*xx);}
if(Re<2000) {f1=149.0/Re+0.9625;}
else        {f1=9.75/Math.pow(Re,0.289);}
xx=Math.cos(fi)/Math.sqrt(0.045*Math.tan(fi)+0.09*Math.sin(fi)+f0/Math.cos(fi))+
   (1-Math.cos(fi))/Math.sqrt(3.8*f1);
double f=1.0/(xx*xx);
return f;
}

public double  h_soguk_sivi(double t,double tw)
{
double Re=Re_soguk_sivi(t);
double Pr = soguk_sivi.rd.Pr(t+273.15);
double visw=sicak_sivi.rd.viscosity_l(tw+273.15);
double vis=sicak_sivi.rd.viscosity_l(t+273.15);
double kt  =  sicak_sivi.rd.k_l(t+273.15);
double f=f_soguk_sivi(t);
double Nu=0.205*Math.pow(Pr,(1.0/3.0))*Math.pow((visw/vis),(1.0/6.0))*Math.pow((f*Re*Re*Math.sin(2.0*fi)),0.374);
double ss=Nu*kt/di;
return ss;
}
public double  h_sicak_sivi(double t,double tw)
{
double Re=Re_sicak_sivi(t);
double Pr = sicak_sivi.rd.Pr(t+273.15);
double visw=sicak_sivi.rd.viscosity_l(tw+273.15);
double vis=sicak_sivi.rd.viscosity_l(t+273.15);
double kt  =  sicak_sivi.rd.k_l(t+273.15);
double f=f_sicak_sivi(t);
double Nu=0.205*Math.pow(Pr,(1.0/3.0))*Math.pow((visw/vis),(1.0/6.0))*Math.pow((f*Re*Re*Math.sin(2.0*fi)),0.374);
double ss=Nu*kt/di;
return ss;
}

public double  dP_soguk_sivi(double t)
{
double ro=soguk_sivi.rd.rol(t+273.15);
double V=hiz_soguk_sivi(t);
double f=f_sicak_sivi(t);
double dP=f*dL/di*ro*V*V/2.0;
return dP;
}

public double  dP_sicak_sivi(double t)
{
double ro=sicak_sivi.rd.rol(t+273.15);
double V=hiz_sicak_sivi(t);
double f=f_sicak_sivi(t);
double dP=f*dL/di*ro*V*V/2.0;
return dP;
}

public double U0(double tc,double th)
{
double tw=(tc+th)/2.0;
double hi=h_soguk_sivi(tc,tw);    // boru içi ısı transferi
double ho=h_sicak_sivi(th,tw); // boru dışı ısı transferi
```

```
double Uboru=1.0/(1.0/hi+Ric+t/klevha+Rdis+1/ho);
//System.out.println("tw="+tw+"hi="+hi+"h0="+ho+"U0="+Uboru);
return Uboru;
}

public void calculate_parallel(double tc, double th)
{
		Q=0;
		double dq=0;
		double X=2.0*Math.PI*b/lambda;
   double F=1.0/6.0*(1+Math.sqrt(1+X*X)+4.0*Math.sqrt(1.0+X*X/2.0));
		double dA=Bp*dL*F;
		double roavg=0;
		double Vavg=0;
		akis="paralel";
		Aboru=nboru*dA;
		dP_soguk_sivi=0.0;
		dP_sicak_sivi=0.0;
   double xx=0;
   Uavg=0;
   double uu=0;
   double mlevha_soguk=msoguk/n;
   double mlevha_sicak=msoguk/n;
		for(int k=0;k<nboru;k++)
		{   uu=U0(tc,th);
			Uavg+=uu;
			dq=uu*dA*(th-tc);
		   dP_soguk_sivi+=dP_soguk_sivi(tc);
		   dP_sicak_sivi+=dP_sicak_sivi(th);

		   Q+=dq;
		   Tsoguk[k]=tc;
		   Tsicak[k]=th;
		   x[k]=xx;
		   double dtc=dq/(mlevha_soguk*soguk_sivi.rd.Cpl(tc+273.15)*1.0e3);
      double dth=dq/(mlevha_sicak*sicak_sivi.rd.Cpl(th+273.15)*1.0e3);
      tc+=dtc;
      th-=dth;
      xx+=dL;
      System.out.println("k="+k+"u="+uu+"dq="+dq+"dtc="+dtc);
   }
   roavg/=nboru;
   Vavg/=nboru;
   Uavg/=nboru;
   double dT1=Tsicak[0]-Tsoguk[0];
   double dT2=Tsicak[nboru-1]-Tsoguk[nboru-1];
   Aboru=Bp*Lp*F*n;
   LMTD=(dT1-dT2)/Math.log(dT1/dT2);
   Qlmtd=Uavg*Aboru*LMTD;
}

public double  calculate_counter1(double tc, double th1)
{
		Q=0;
		double dq=0;
		double X=2.0*Math.PI*b/lambda;
   double F=1.0/6.0*(1+Math.sqrt(1+X*X)+4.0*Math.sqrt(1.0+X*X/2.0));
		double dA=Bp*dL*F;
		double roavg=0;
		double Vavg=0;
		akis="ters";
		double Aboru=nboru*dA;
		dP_soguk_sivi=0.0;
		dP_sicak_sivi=0.0;
   double xx=0;
   Uavg=0;
   double uu=0;
   double mlevha_soguk=msoguk/n;
   double mlevha_sicak=msoguk/n;
   double dtc,dth;
		for(int k=0;k<nboru;k++)
		{   uu=U0(tc,th1);
			Uavg+=uu;
			dq=uu*dA*(th1-tc);
		   dP_soguk_sivi+=dP_soguk_sivi(tc);
		   dP_sicak_sivi+=dP_sicak_sivi(th1);
		   //System.out.println("dP="+dP+"f="+f+"dout="+dout+"ro="+ro);
```

```
                Q+=dq;
                Tsoguk[k]=tc;
                Tsicak[k]=th1;
                x[k]=xx;
                dtc=dq/(mlevha_soguk*soguk_sivi.rd.Cpl(tc+273.15)*1.0e3);
        dth=dq/(mlevha_sicak*sicak_sivi.rd.Cpl(th1+273.15)*1.0e3);
        tc+=dtc;
        th1+=dth;
        xx+=dL;
    }
    roavg/=nboru;
    Vavg/=nboru;
    Uavg/=nboru;
    double dT1=Tsicak[0]-Tsoguk[0];
    double dT2=Tsicak[nboru-1]-Tsoguk[nboru-1];
    Aboru=Bp*Lp*F*n;
    LMTD=(dT1-dT2)/Math.log(dT1/dT2);
    Qlmtd=Uavg*Aboru*LMTD;
    return Tsicak[nboru-1];
}

public void calculate_counter(double tc, double th)
{
//counter flow sistem
//tic  : cold liquid inlet temp.
//tdis : hot liquid inlet temp.
double dT=(th-tc)/30.0;
double Tgir[]=new double[25];
double Tcik[]=new double[25];
akis="ters";
for(int i=0;i<25;i++)
{Tcik[i]=th-dT*i;
 Tgir[i]=calculate_counter1(tc, Tcik[i]);
}
double e[]=EKK(Tgir,Tcik,7);
double tdis1=funcEKK(e,th);
double tdis2=calculate_counter1(tc,tdis1);
    double dT1=Tsicak[0]-Tsoguk[0];
    double dT2=Tsicak[nboru-1]-Tsoguk[nboru-1];
    LMTD=(dT1-dT2)/Math.log(dT1/dT2);
    Qlmtd=Uortalama*Aboru*LMTD;
}

public void plot()
{
 Plot pp=new Plot(x,Tsoguk);
 pp.addData(x,Tsicak);
 Locale us=new Locale("US");
 String s1="Q = %4.1f kW A = %4.1f m^2 U = %4.0f W/m^2K";
 String s4=String.format(us,s1,Q*1e-3,Aboru,dP_soguk_sivi*1e-3,Uavg);
 pp.setPlabel(s4);
 pp.setXlabel("x, m");
 pp.setYlabel("T temperature , degree C");
 pp.plot();
}

//en kucuk kareler metodu
  public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivotlama
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                   if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
```

```
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                 }
                 dummy=b[p];
                 b[p]=b[k];
                 b[k]=dummy;
               }
               //solving gauss elimination
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
                }
  }
  //back subsitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}
public static double[] EKK(double xi[],double yi[],int n)
{
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  for(j=0;j<n+1;j++)
     {if(i==0 && j==0) A[i][j]=l;
     else  for(k=0;k<l;k++) A[i][j] += Math.pow(xi[k],(i+j));
     }
     for(k=0;k<l;k++) { if(i==0)  B[i]+= yi[k];
                  else      B[i] += Math.pow(xi[k],i)*yi[k];}
  }
X=pivotlugauss(A,B);
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
return X;
}

public static double funcEKK(double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=e.length;
double ff;
if(n!=0.0)
  { ff=e[n-1];
  for(int i=n-2;i>=0;i--)
   { ff=ff*x+e[i]; }
  }
else
  ff=0;
return ff;
}

public static double hata(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
```

```
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcEKK(e,x[k]))*(y[k]-funcEKK(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

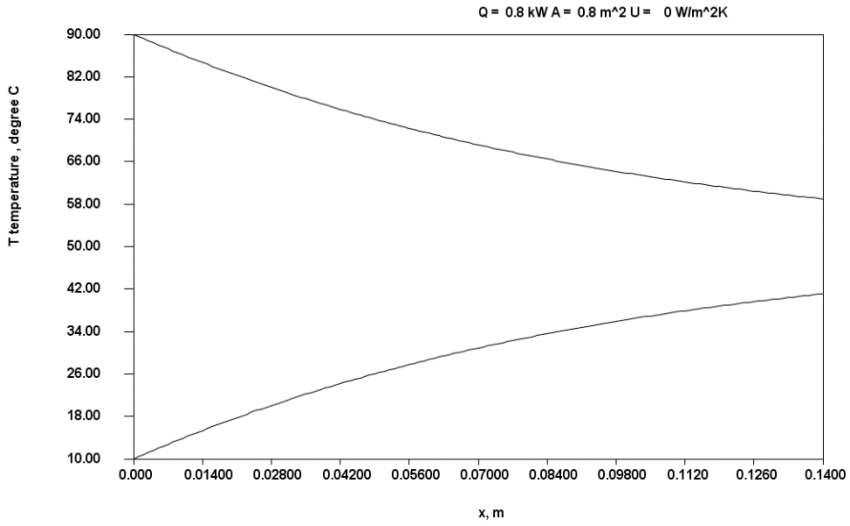
public static double[][]  funcEKK(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
{
//aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
// ara değer sayısı 2 dir
   int n=xi.length;
   int nn=(n-1)*(aradegersayisi+1)+1;
   double z[][]=new double[2][nn];
   double E[]=EKK(xi,yi,polinomkatsayisi);
   System.out.println("katsayılar :\n"+Matrix.toStringT(E));
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
      {z[0][k]=xi[i];z[1][k]=funcEKK(E,z[0][k]);k++;
      for(int j=0;j<aradegersayisi;j++)
      {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
      z[0][k]=z[0][k-1]+dx;z[1][k]=funcEKK(E,z[0][k]);k++;}
      }
      z[0][k]=xi[i];z[1][k]=funcEKK(E,z[0][k]);
   return z;
 }
public String toString()
{String s="Chevron plate heat exchanger \n";
s+="cold fluid = "+soguk_sivi.rd.liquidName+"\n";
s+="hot fluid = "+sicak_sivi.rd.liquidName+"\n";
s+="m cold = "+msoguk+"kg/s \n ";
s+="m hot = "+msicak+"kg/s \n ";
s+="b channel chevron height : "+b+" m\n";
s+="lambda chevron wave length: "+lambda+" m\n";
s+="Lp plate length: "+Lp+" m\n";
s+="Bp plate width: "+Bp+" m\n";
s+="t plate thickness: "+t+" m\n";
s+="pressure drop cold fluid: "+dP_soguk_sivi+" kPa\n";
s+="pressure drop hot fluid: "+dP_sicak_sivi+" kPa\n";
s+="heat transfer : "+Q+" W\n";
s+="LMTD logaritmik mean temperature difference : "+LMTD+" K\n";
s+="cold fluid inlet-exit temp. : "+"\n";
s+=Tsoguk[0]+"   "+Tsoguk[nboru-1]+"\n";
s+="hot fluid inlet-exit temp. : "+"\n";
s+=Tsicak[0]+"   "+Tsicak[nboru-1]+"\n";
return s;
}
}
```

```
import java.util.*;
import java.lang.*;
import java.io.*;

class chevron_plate_heat_exchanger_test{
    public static void main (String args[]) throws IOException
    {
    String isoguk_sivi="water";
    String isicak_sivi="water";
    double bi=5e-3;//chevron height (depth)
    double fii=Math.PI/4.0;//chevron angle
    double lambdai=8e-3;//shevron wave length
    double Lpi=140e-3;//length of plate heat exchanger
    double Bpi=70e-3;//width of plate heat exchanger
    double imsoguk=0.69/3.6; //kg/s cold fluid
    double imsicak=0.69/3.6; //kg/s hot fluid
    double dLi=1e-3; //finite difference length
    int ni=30;  // number of plates
    double klevhai=25.0;//plate thermal conductivity
    double ti=3.0e-3;//plate thickness
    chevron_plate_heat_exchanger lid=new chevron_plate_heat_exchanger(isoguk_sivi,isicak_sivi,
    bi,fii,lambdai,Lpi,Bpi,imsoguk,imsicak,dLi,
    ni,klevhai,ti);
```

```
    double th=90.0;
    double tc=10.0;
    lid.calculate_parallel(tc,th);
    //lid.calculate_counter(tc,th);
    lid.plot();
    System.out.println(lid.toString());
    }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" chevron_plate_heat_exchanger_test
Chevron plate heat exchanger
cold fluid = water
hot fluid = water
m cold = 0.19166666666666665kg/s
 m hot = 0.19166666666666665kg/s
 b channel chevron height : 0.005 m
lambda chevron wave length: 0.008 m
Lp plate length: 0.14 m
Bp plate width: 0.07 m
t plate thickness: 0.003 m
pressure drop cold fluid: 5.3187934970577855 kPa
pressure drop hot fluid: 3.272772495332654 kPa
heat transfer : 833.764856105945 W
LMTD logaritmik mean temperature difference : 41.46083695554668 K
cold fluid inlet-exit temp. :
10.0   41.09641705190255
hot fluid inlet-exit temp. :
90.0   58.978381611543945
```

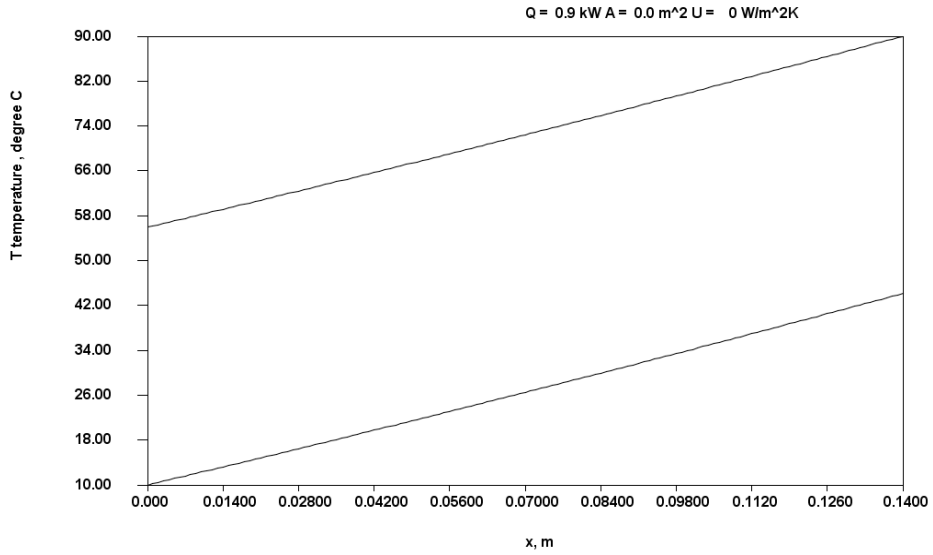



```
import java.util.*;
import java.lang.*;
import java.io.*;

class chevron_plate_heat_exchanger_test{
    public static void main (String args[]) throws IOException
    {
    String isoguk_sivi="water";
    String isicak_sivi="water";
    double bi=5e-3;//chevron height (depth)
    double fii=Math.PI/4.0;//chevron angle
    double lambdai=8e-3;//shevron wave length
    double Lpi=140e-3;//length of plate heat exchanger
    double Bpi=70e-3;//width of plate heat exchanger
    double imsoguk=0.69/3.6; //kg/s cold fluid
    double imsicak=0.69/3.6; //kg/s hot fluid
    double dLi=1e-3; //finite difference length
    int ni=30;  // number of plates
    double klevhai=25.0;//plate thermal conductivity
    double ti=3.0e-3;//plate thickness
    chevron_plate_heat_exchanger lid=new chevron_plate_heat_exchanger(isoguk_sivi,isicak_sivi,
    bi,fii,lambdai,Lpi,Bpi,imsoguk,imsicak,dLi,
    ni,klevhai,ti);
```

```
    double th=90.0;
    double tc=10.0;
    //lid.calculate_parallel(tc,th);
    lid.calculate_counter(tc,th);
    lid.plot();
    System.out.println(lid.toString());
    }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" chevron_plate_heat_exchanger_test
Chevron plate heat exchanger
cold fluid = water
hot fluid = water
m cold = 0.19166666666666665kg/s
 m hot = 0.19166666666666665kg/s
 b channel chevron height : 0.005 m
lambda chevron wave length: 0.008 m
Lp plate length: 0.14 m
Bp plate width: 0.07 m
t plate thickness: 0.003 m
pressure drop cold fluid: 5.563468421506597 kPa
pressure drop hot fluid: 3.2381046919934056 kPa
heat transfer : 919.3408504385529 W
LMTD logaritmik mean temperature difference : 45.89967762091959 K
cold fluid inlet-exit temp. :
10.0   44.13851908482931
hot fluid inlet-exit temp. :
55.938037509195716   89.99985817799062
```



Muley and Manglik[80] suggested the following equation:

$F_1 = [0.2668 - 0.006967\beta + 7.244x10^{-5}\beta^2]$

$F_2 = \left[20.78 - 50.94\Phi + 41.16\Phi^2 - 10.51\Phi^3\right]$

$F_3 = \left[0.728 + 0.0543sin[(\pi\beta/45) + 3.7]\right]$ where β should be taken in degrees

$Nu = F_1F_2Re^{F_3}Pr^{1/3}\left(\frac{\mu}{\mu_w}\right)^{0.14}$

$F_4 = [2.917 - 0.1277\beta + 2.016x10^{-3}\beta^2]$

$F_5 = \left[5.474 - 19.02\Phi + 18.93\Phi^2 - 5.341\Phi^3\right]$

$F_6 = \left[0.2 + 0.0577sin[(\pi\beta/45) + 2.1]\right]$

$f = F_4F_5Re^{-F_6}$

In this equations angle β will be taken as degree. Equation is valid for the range of 30 to 60 degrees Re > 1000

Dong at al [73] suggested following correlations for **R410A ve R22 condensers**

$$Nu = A_1 Re^{A_3} Pr^{1/3}$$

$$A_1 = 11.22\left(\frac{\lambda}{2d_h}\right)^{-2.83}\left(\frac{\pi}{2}-\beta\right)^{-4.5}$$

$$A_2 = 0.35\left(\frac{\lambda}{2d_h}\right)^{0.23}\left(\frac{\pi}{2}-\beta\right)^{1.48}$$

$$f = A_3 Re^{A_4}$$

$$A_3 = 3521.1\left(\frac{\lambda}{2d_h}\right)^{4.17}\left(\frac{\pi}{2}-\beta\right)^{-7.75}$$

$$A_4 = -1.024\left(\frac{\lambda}{2d_h}\right)^{0.0925}\left(\frac{\pi}{2}-\beta\right)^{-1.3}$$

$$G_{eq} = m\left[1 - x + x\left(\frac{\rho_f}{\rho_g}\right)^{0.5}\right]$$

$Re = \frac{G_{eq} d_h}{\mu_f}$ equations are valid for $300 \leq Re \leq 4000$

Muthuraman[74] is given the following correlations for liquid

$$h_w = 0.295 Re_d^{0.64} Pr^{0.32}\left(\frac{\pi}{2}-\beta\right)^{0.09}$$

Akturk et al [72] is given the following correlation for liquids

$$Nu = 0.32592 Re^{0.1625} Pr^{1/3}\left(\frac{\mu}{\mu_w}\right)^{0.14} \quad 450 \leq Re \leq 5250 \;\; \beta = 30\, degree$$

Friction factor:

$$f = 4291 Re^{-1.278} + 0.3343$$

They also listed results of another similar equation form other researchers

Kumar[77] equation:

$$Nu = 0.348 Re^{0.663} Pr^{1/3}\left(\frac{\mu}{\mu_w}\right)^{0.17} \; 410 \leq Re \;\; \beta = 30\, degree$$

Focke[75] equation:

$$Nu = 1.112 Re^{0.6} Pr^{0.5} \quad 600 \leq Re \leq 16000 \;\; \beta = 30\, degree$$

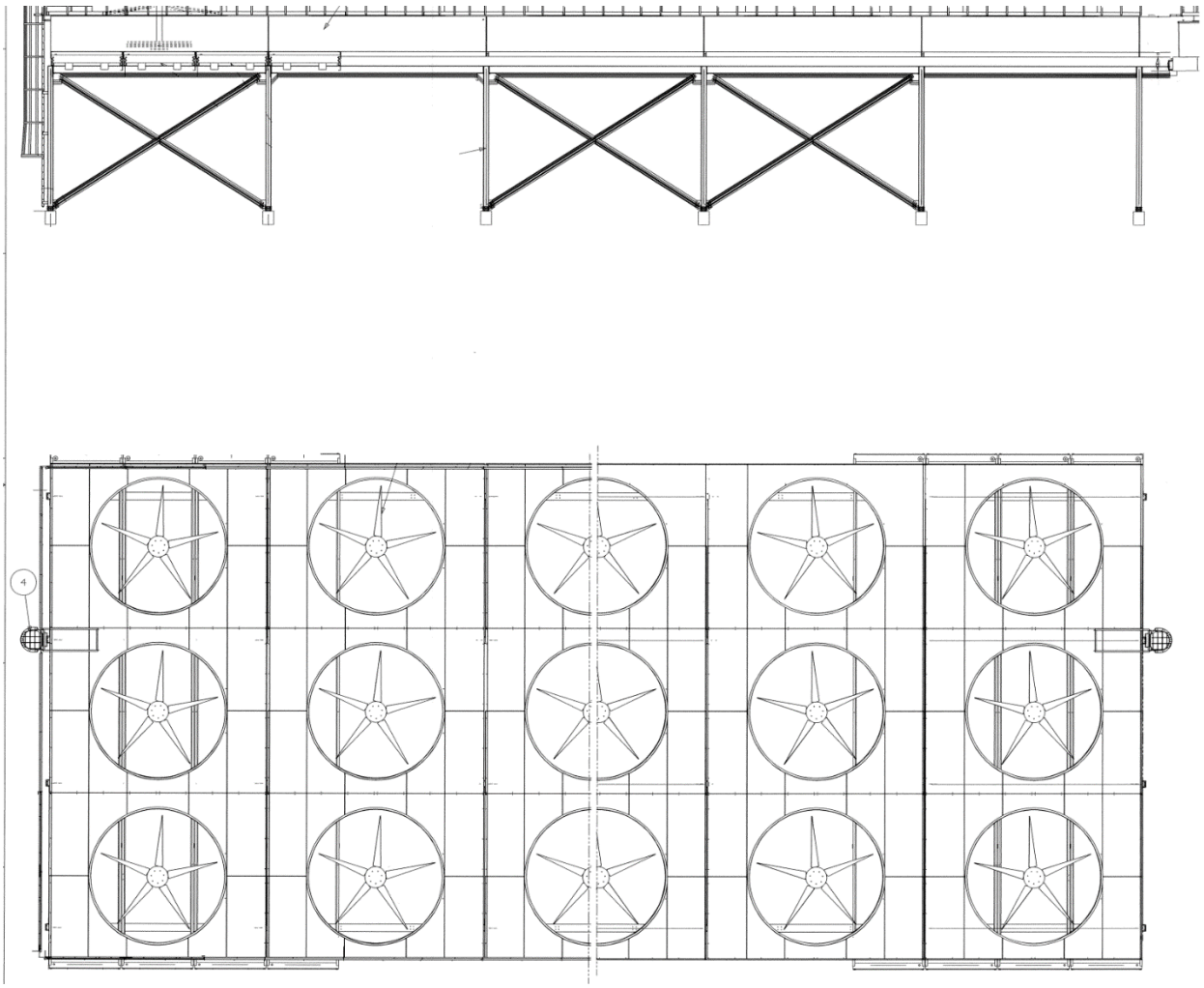
$$Nu = 0.57 Re^{0.7} Pr^{0.5} \quad 150 \leq Re \leq 600 \;\; \beta = 30\, degree$$

Okada[76] equation:

$$Nu = 0.1528 Re^{0.66} Pr^{0.4} \quad 400 \leq Re \leq 15000 \;\; \beta = 30\, degree$$

GEOTHERMAL POWER PLANT AIR COOLED CONDENSER MODELLING

Air cooled, finned pipe condensers are widely used in geothermal power plants. Air passsed through finned pipes forced by fans. A secondary fluid wih relatively low saturation temperature condenses inside of the pipes.



```
public class HT_pipe_round_finned
{  // round finned pipes
   public double s; //distance between fins
    public double tf; //fin thickness
    public int n_fin; //number of fins in length L
    public double df; //fin diameter
    public double dr; //pipe outside dimension
    public double di; //pipe inlet dimension
    public double lf;  //fin length
    public double L;  //pipe length;
    double k;  //Thermal conductivity of pipe
    double T; //temperature degree K
    double P; //pressure kPa
    double avis;
    double ak;
    double aCp;
    double aro;
    double Pr;
    double Re;
    double Pt; //vertical distance between pipe
    double Pl; //horizontal distance between two row of pipesin
    double A_fin;  //fin area per pipe
    double A_pipe_base; //pipe area per pipe
    double A; // corrected pipe area=a_pipe_base+eta_fin*A_fin per pipe
```

```
    //  -    0      0      0
    //  Pt
    //             0      0
    //         |Pl |
    double a[];// properties of gas
    String material;
    // working fluid: air
    air_PG_CS air;
    //constructor method
    public HT_pipe_round_finned(String materiali,double dri,double dii,double dfi,double si,double tfi,double Li,dou
Pli)
    {s=si;dr=dri;di=dii;df=dfi;tf=tfi;lf=(df-dr)/2.0;air=new air_PG_CS();L=Li;
    material=materiali;
     if(material.equals("copper") ) k=360.0;
     else if(material.equals("steel"))  k=50.0;
     //else k=50.0;
     T=29.0+273.15; //degree K
     P=101.325; //kPa
     n_fin=(int)(L/(s+tf));

     System.out.println("n_fin="+n_fin);
    }
    //setting of temperature and pressure of gas
    public void setTP(double Ti,double Pi)
    { T=Ti;
      P=Pi;
      a=air.property("tp",T,P);
      avis=a[11];
      ak=a[12];
      aCp=a[7];
      aro=a[15];
      Pr=a[14];
    }
    //Re numbers
    public double Re(double Ui,double Ti,double Pi)
    {setTP(Ti,Pi);
      return aro*Ui*dr/avis;
    }
    //outside heat transfer correlations
    //  Low fin height and frequent fins (Briggs & Young Correlation[95])
   //  95. Briggs, D.E. and Young, E.H. , Convection heat transfer and pressure drop of air flowing across
    //  triangular pitch banks of finned tubes, Chem. Eng. Prog. Symp. Ser., 59, 1-10 (1963)
    public double h1(double Ui,double Ti,double Pi)
    { double T1=Ti+273.15;
              Re=Re(Ui,T1,Pi);
              double Nu=0.1505*Math.pow(Re,0.667)*Math.pow(Pr,(1.0/3.0))*Math.pow((s/lf),0.164)*Math.pow((s/t
      return Nu*ak/dr;
    }
    // High fin tube banks
     public double h2(double Ui,double Ti,double Pi)
    {  double T1=Ti+273.15;
              Re=Re(Ui,T1,Pi);
               double Nu=0.1378*Math.pow(Re,0.718)*Math.pow(Pr,(1.0/3.0))*Math.pow((s/lf),0.296);
       return Nu*ak/dr;
    }
    //For all fin heights
```

```
    public double h3(double Ui,double Ti,double Pi)
    {  double T1=Ti+273.15;
       Re=Re(Ui,T1,Pi);
              double Nu=0.134*Math.pow(Re,0.681)*Math.pow(Pr,(1.0/3.0))*Math.pow((s/lf),0.2)*Math.pow((s/tf),0
      return Nu*ak/dr;
    }
    // Fanning friction factor
    public double f(double Ui,double Ti,double Pi)
    {  double T1=Ti+273.15;
              Re=Re(Ui,T1,Pi);
             double f=3.805*Math.pow(Re,-
0.234)*Math.pow(Pr,(1.0/3.0))*Math.pow((s/df),0.2251)*Math.pow((lf/s),0.759)*Math.pow((dr/Pt),0.709)*Math.po
      return f;
           }
           //fin efficiency
           public double eta_fin(double h)
           {
   double rb=dr/2.0;  //pipe outside radius
   double rt=df/2.0;  //din radius
          double Rb=rb/rt;  //dimensionless pipe outside diameter
   double t=tf;  //fin thickness
   double Rt=1.0; //dimensionless fin diameter
   double M1=Math.sqrt(2.0*h/(k*t))*rt;
   if_x eta=M->2.0/M*Rb/(1-Rb*Rb)*(Mathd.I(1.0,M)*Mathd.K(1.0,M*Rb)-
Mathd.K(1.0,M)*Mathd.I(1.0,M*Rb))/(Mathd.I(1.0,M)*Mathd.K(0.0,M*Rb)+Mathd.K(1.0,M)*Mathd.I(0.0,M*Rb)
  double eta1=eta.func(M1);
  A_fin=eta1*n_fin*Math.PI*(df*df-dr*dr)/4.0;
  A_pipe_base=n_fin*s*Math.PI*dr;
  A=A_fin+A_pipe_base;  //area per pipe
  //System.out.println("A_fin = "+A_fin+"A_pipe_base="+A_pipe_base+"A="+A);
  return eta1;
  }
   public static void main(String arg[])
   { double dri=25.4e-3; //m pipe outlet diameter
    double dii=24.3e-3; //m pipe inlet diameter
    double dfi=57e-3; //  fin diameter
    double si=1.651e-3;  //m distance between fins
    double tfi=0.88e-3; // m fin thickness
    double Li=18.292; // m pipe length
    String materiali="copper"; // m/s
    double U=5.0;
    double T=29.0; //degree K
    double P=101.325; //kPa
    double Pt=50.8356912e-3; //
    double Pl=58.7e-3;  //m
   HT_pipe_round_finned pipe=new HT_pipe_round_finned(materiali,dri,dii,dfi,si,tfi,Li,Pt,Pl);
    double h1=pipe.h1(U,T,P);
    double h2=pipe.h2(U,T,P);
    double h3=pipe.h3(U,T,P);
    double eta1=pipe.eta_fin(h1);
    double eta2=pipe.eta_fin(h2);
    double eta3=pipe.eta_fin(h3);
    System.out.println("h1="+h1+"W/m K");
    System.out.println("h2="+h2+"W/m K");
    System.out.println("h3="+h3+"W/m K");
    System.out.println("eta1="+eta1);
```

```
    System.out.println("eta2="+eta2);
    System.out.println("eta3="+eta3);
   }

}
```

```
public class HT_inside_pipe_condensing
{
public refrigerant re;
public String flow_pattern;
public double xs;        //soğutucu akışkan fazı x<0 sıvı, 0<=x<=1 karışım x>1 gaz
public double mi;    // internal fluid mass flow rate kg/s
public double di;  //internal pipe diameter
public double eps;  //surface quality
public double eod;
public double L;  //pipe length
public double A;
public HT_inside_pipe_condensing(String namei,double mii,double dic,double Li,double epsi)
{re=new refrigerant(namei);mi=mii;di=dic;eps=epsi;eod=eps/di;L=Li;A=Math.PI*di*L;}

//  Dobson and Chato equation
//  108. Dobson, M.K. and Chato, J.C. (1998). Condensation in Smooth Horizontal Tubes, J. Heat
//  tranfer   Vol. 120, No. 1, pp. 193-213.
public double h_condensing_chato(double x, double tsat,double tw)
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // L vertical length of vertical plate
  // Dobson & Chato
  double D=di;
    double tf=(tsat+tw);
    double muv=re.r.viscosity_v(tf);
    double mul=re.r.viscosity_l(tf);
    double rov=re.r.rov(tf);
    double rol=re.r.rol(tf);
    double A=Math.PI*D*D/4.0;
    double G=mi/A;
    double Rev=mi*D/(A*muv);
     double Rel=mi*(1.0-x)*D/(A*mul);
    double Prl=re.r.Pr_l(tf);
    double hfg=(re.r.h_v(tsat)-re.r.h_l(tsat))*1e3; //J/kg
    double dT=tsat-tw;
    double kl=re.r.k_l(tf);
    double Cpl=re.r.Cpl(tf)*1e3;  //J/kgK
    double hfgp=hfg+0.375*Cpl*dT;
    double Nuanuular=0.0;
    double Nustratified_wavy=0.0;
    double hh=0;
    double g=9.81; //m/s^2
    double Xtt=Math.pow(((1.0-x)/x),0.9)*Math.sqrt(rov/rol)*Math.pow((mul/muv),0.1);
    Nuanuular=0.023*Math.pow(Rel,0.8)*Math.pow(Prl,0.4)*(1.0+2.22/Math.pow(Xtt,0.89));
    if(G>500) hh=Nuanuular*(kl/D);
    else
    {double eps1=1.0/(1.0+(1.0-x)/x*Math.pow((rov/rol),2.0/3.0));
     double teta_strat=1.0-Math.acos(2.0*eps1-1.0);
     double Gal=g*rol*(rol-rov)*D*D*D/(mul*mul);
     double Jal=Cpl*(tsat-tw)/hfg;
     double Frl=mi*mi/(rol*rol*g*D);
```

```
    double Frso=0.0;
    double C1=0.0,C2=0.0;
    if(Frl<=0.7)
    {C1=4.172+5.48*Frl-1.564*Frl*Frl;
     C2=1.773-0.169*Frl;
    }
    else
    {C1=7.242;C2=1.655;}
    double Nustrat=0.0195*Math.pow(Rel,0.8)*Math.pow(Prl,0.4)*(1.376+C1/Math.pow(Xtt,C2));
    Nustratified_wavy=0.23*Math.pow(Rev,0.12)/(1.0+1.11*Math.pow(Xtt,0.58)*Math.pow((Gal*Prl/Jal),0.25))+(1
teta_strat/Math.PI)*Nustrat;
    if(Rel<=1250)
    {Frso=0.025*Math.pow(Rel,1.59)*Math.pow((1.0+1.09*Math.pow(Xtt,0.039)/Xtt),1.5)/Math.sqrt(Gal);}
    else
    {Frso=1.26*Math.pow(Rel,1.04)*Math.pow((1.0+1.09*Math.pow(Xtt,0.039)/Xtt),1.5)/Math.sqrt(Gal);}
    if(G<=500 && Frso<20) {hh=Nustratified_wavy*(kl/D);}
    else if(G<=500 && (Frso<20 && Frso>=7)) {hh=(kl/D)*(Nuanuular+(Frso-7.0)/13.0*(Nustratified_wavy-Nuan
    }
   return hh;
}
//Rohsenov equation
public double h_condensing_Rohsenov(double x, double tsat,double tw)
{ // tsat : saturation temperature degree C
  // tw surface temperature degree C
  // m mass flow rate
  // x vapor quality
  // Reference: Traviss, D.P., Rohseno
  // Rohsenow equation
//   w, W.M. and Baron A.B.,
  // Forced convection condensation inside tubes:a heat trasnfer equation for
  // condenser design, ASHRAE Transactions, 79, part1, 157-165 (1973)
     double D=di;
   double tf=(tsat+tw)/2.0;
   double muv=re.r.viscosity_v(tsat);
   double mul=re.r.viscosity_l(tsat);
   double rov=re.r.rov(tsat);
   double rol=re.r.rol(tf);
   double A=Math.PI*D*D/4.0;
   double Rev=mi*x*D/(A*muv);
   double Rel=mi*(1.0-x)*D/(A*mul);
   double Prl=re.r.Pr_l(tf);
   double hfg=(re.r.h_v(tsat)-re.r.h_l(tsat))*1e3; //J/kg
   double dT=tsat-tw;
   double kl=re.r.k_l(tf);
   double Cpl=re.r.Cpl(tf)*1e3;  //J/kgK
   double hfgp=hfg+0.375*Cpl*dT;
   double hh=0.0;
   double g=9.81; //m/s^2
   double F=0;
   double hL=0;
   double Xtt=Math.pow(((1.0-x)/x),0.9)*Math.sqrt(rov/rol)*Math.pow((mul/muv),0.1);
   double FXtt=0.15*(1.0/Xtt+2.85/Math.pow(Xtt,0.476));
   if(Rel>=1125.0) F=5.0*Prl+5.0*Math.log(1.0+5.0*Prl)+2.5*Math.log(0.00313*Math.pow(Rel,0.812));
   else if(Rel>50 && Rel<1125) F=5.0*Prl+5.0*Math.log(1.0+Prl*(0.09636*Math.pow(Rel,0.585)-1.0));
   else F=0.707*Prl*Math.sqrt(Rel);
   if(FXtt>=0.1 && FXtt<=1.0)
```

```
    {hL=(kl/D)*Rel*Prl/F*FXtt;}
   else if(FXtt>1 && FXtt<=15.0)
    {hL=(kl/D)*Rel*Prl/F*Math.pow(FXtt,1.15);}
   return hL;
}
//Shah (2009) correlation
public double h_condensing_Shah(double x, double tsat,double tw)
{ // tsat : saturation temperature degree C
 // tw surface temperature degree C
 // m mass flow rate
 // x vapor quality
 // Shah, M.M. (1979). A General Correlation for Heat Transfer during Film
 // Condensation inside of Pipes, Int. J. Heat Mass Transfer, Vol. 22, pp. 547-556.
   double D=di;
   double tf=(tsat+tw)/2.0;
   double Ps=re.Psb(tsat);
   double Pc=re.r.Pc;
   Double Pr=Ps/Pc;
   double muf=re.r.viscosity_l(tf);
   double mug=re.r.viscosity_l(tsat);
   double rol=re.r.rol(tf);
   double rog=re.r.rov(tsat);
   double g=9.806; //m/s^2
   double A=Math.PI*D*D/4.0;
   double G=mi/A;
   double Relt=mi*D/(A*muf);
   double Rels=mi*D*(1.0-x)/(A*muf);
   double Prl=re.r.Pr_l(tf);
   double kl=re.r.k_l(tf);
   double Z=Math.pow((1.0/x-1),0.8)*Math.pow(Pr,0.4);
   double hlt=0.023*(kl/D)*Math.pow(Relt,0.8)*Math.pow(Prl,0.4);
   double hI=hlt*Math.pow((muf/(14.0*mug)),(0.0058+0.557*Pr))*(1.0+3.8/Math.pow(Z,1.8));
   double hNu=1.32*Math.pow(Rels,(-1.0/3.0))*Math.pow((rol*(rol-rog)*g*kl*kl*kl/muf/muf),(1.0/3.0));

   double Jg=x*G/Math.sqrt(g*D*rog*(rol-rog));
   double hTP=0;
   if(Jg>=1.0/(2.4*Z+0.73)) hTP=hI;
   else if(Jg<= (0.89-0.93*Math.exp(-0.087*Math.pow(Z,-1.17)))) hTP=hNu;
   else hTP=hI+hNu;
   return hTP;
}
// El Hajal, Thome & Cavallini correlation
public double h_condensing_ElHajal(double x, double tsat,double tw)
{ // tsat : saturation temperature degree C
 // tw surface temperature degree C
 // m mass flow rate of refrigerant kg(s
 // x vapor quality of refrigerant kg vapor/kg total
 // L vertical length of vertical plate
 // El Hajal, J., Thome, J.R. and Cavallini, A.(2003).Condensation in Horizontal Tubes, Part 1: Two-
 // Phase Flow Pattern Map, Int. J. Heat Mass Transfer, Vol. 46, pp. 3349-3363.
 // Condensation in Horizontal Tubes, Part 2: Two Phase Flow Pattern Map, Int. J. Heat Mass Transfer,
 // Vol. 46, pp. 3365-3387.
   double D=di;
    double tf=(tsat+tw)/2.0;
   double muv=re.r.viscosity_v(tsat);
   double mul=re.r.viscosity_l(tf);
```

```
    double rov=re.r.rov(tsat);
    double rol=re.r.rol(tf);
    double A=Math.PI*D*D/4.0;
    double G=mi/A;
    double Rev=mi*D/(A*muv);
    double Rel=mi*(1.0-x)*D/(A*mul);
    double Prl=re.r.Pr_l(tf);
    double hfg=(re.r.h_v(tsat)-re.r.h_l(tsat))*1e3; //J/kg
    double dT=tsat-tw;
    double g=9.806; //m/s^2
    double kl=re.r.k_l(tf);
    double Cpl=re.r.Cpl(tf)*1e3;  //J/kgK
    double hfgp=hfg+0.375*Cpl*dT;
    double sigma=re.r.sigma(tsat);
    //Homegeneous void fraction
    double epsh=1.0/(1+(1.0-x)/x*(rov/rol));
    //Non-homegeneous void fraction
    double epsra=x/rov/((1.0+0.12*(1.0-x))*(x/rov+(1.0-x)/rol)+
    1.18*(1.0-x)*Math.pow((g*sigma*(rol-rov)),0.25)/G/Math.sqrt(rol));
    double eps=(epsh-epsra)/Math.log(epsh/epsra);
    double Al=A*(1.0-eps);
    double Av=A*eps;
    double Ald=Al/(D*D);
    double Avd=Av/(D*D);
    double ul=G*(1.0-x)/(rol*(1.0-eps));
    double uv=G*x/(rov*eps);
    double teta=0;
    double teta_strat=2.0*Math.PI-2.0*(Math.PI*(1.0-eps)+Math.pow(1.5*Math.PI,(1.0/3.0))*(1.0-2.0*(1.0-eps)+Ma
eps),(1.0/3.0))-Math.pow(eps,1.0/3.0))-
    1.0/200.0*(1.0-eps)*eps*(1.0-2.0*(1.0-eps))*(1.0+4.0*((1.0-eps)*(1.0-eps)+eps*eps)));
    if_x f2=tet->Ald-1.0/8.0*((2.0*Math.PI-tet)-Math.sin(2.0*Math.PI-tet));
    System.out.println("teta_strat="+teta_strat);
    teta_strat=bisection(f2,0.0,1.5*Math.PI);
    System.out.println("teta_strat="+teta_strat);
    double hld=0.5*(1.0-Math.cos((2.0*Math.PI-teta)/2.0));
    double Pid=Math.sin((2.0*Math.PI-teta)/2.0);
    double Gwavy=(16.0*Avd*Avd*Avd*g*D*rol*rov/(x*x*Math.PI*Math.PI*Math.sqrt(1.0-(2.0*hld-1.0)*(2.0*hl
    (Math.PI*Math.PI/(25.0*hld*hld)*Math.pow((g*D*D*rol/sigma),-1.023)+1.0)+50.0-75.0*Math.exp((-x*x-0.97)/
    double Gstrat=Math.pow((51211.69*Ald*rov*(rol-rov)*mul*g)/(x*x*(1.0-x)*Math.PI*Math.PI*Math.PI),(1.0/3.
    double xIA=1.0/((0.2914*Math.pow((rov/rol),(-1.0/1.75))*Math.pow((muv/mul),(-1.0/7.0)))+1.0);
    double Gmist=Math.sqrt(7680.0*Avd*Avd*g*D*rol*Math.pow((1.138+2.0*Math.log(Math.PI/(1.5*Ald))),2.0)/
    (x*x*Math.PI*Math.PI*(g*D*D*rol/sigma)));
    if(G>Gwavy & G<Gmist && x>xIA)
    {flow_pattern="annular";}
    else if(G>Gwavy & G<Gmist && x<xIA)
    {flow_pattern="intermittent";}
    else if(G<Gwavy & G>Gstrat)
    {flow_pattern="Stratified_wavy";}
    else if(G<Gstrat)
    {flow_pattern="Fully_Stratified";}
    else
    {flow_pattern="Mist";}
    System.out.println("flow pattern = "+flow_pattern);
    double htp=0;
    double hc=0,hf=0;
    double delta=0;
```

```
    if(flow_pattern.equals("annular") || flow_pattern.equals("intermittent") || flow_pattern.equals("Mist"))
    {   teta=0.0;
        final double teta1=teta;
        if_x f3=delt->Al-(2.0*Math.PI-teta1)/8.0*(D*D-(D-2.0*delt)*(D-2.0*delt));
        delta=bisection(f3,0.0,0.6*D);
        double fi=1.0+Math.sqrt(uv/ul)*Math.pow((rol-rov)*g*delta*delta/sigma,0.25);
        hc=0.003*Math.pow(Rel,0.74)*Math.sqrt(Prl)*kl/delta*fi;
              htp=hc;
           }
    else if(flow_pattern.equals("Stratified_wavy"))
    {teta=teta_strat*Math.sqrt((Gwavy-G)/(Gwavy-Gstrat));
     final double teta1=teta;
     if_x f3=delt->Al-(2.0*Math.PI-teta1)/8.0*(D*D-(D-2.0*delt)*(D-2.0*delt));
     delta=bisection(f3,0.0,0.6*D);
     double fi=1.0+Math.sqrt(uv/ul)*Math.pow((rol-rov)*g*delta*delta/sigma,0.25);
     hc=0.003*Math.pow(Rel,0.74)*Math.sqrt(Prl)*kl/delta*fi;
     hf=0.728*Math.pow((rol*(rol-rov)*g*hfg*kl*kl*kl/(mul*D*dT)),0.25);
     htp=(hf/2.0*teta+(2*.0*Math.PI-teta)/2.0*hc)/(Math.PI);
    }
    else if(flow_pattern.equals("Fully_Stratified"))
    {teta=teta_strat;
     final double teta1=teta;
     if_x f3=delt->Al-(2.0*Math.PI-teta1)/8.0*(D*D-(D-2.0*delt)*(D-2.0*delt));
     delta=bisection(f3,0.0,0.6*D);
     double fi=1.0+Math.sqrt(uv/ul)*Math.pow((rol-rov)*g*delta*delta/sigma,0.25);
     hc=0.003*Math.pow(Rel,0.74)*Math.sqrt(Prl)*kl/delta*fi;
     hf=0.728*Math.pow((rol*(rol-rov)*g*hfg*kl*kl*kl/(mul*D*dT)),0.25);
     htp=(hf/2.0*teta+(2*.0*Math.PI-teta)/2.0*hc)/(Math.PI);
    }
    return htp;
}
public static double bisection(if_x f,double a,double b)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   double eps=1.0e-8;
   int nmax=100;
   int i=0;
   while(Math.abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;
    i++;
   }
   if(i>=nmax) r=bisection(f,a,b1);
   return r;
  }
 public double  Rel(double T)
{
// Reynold's number boru içi liquid
double rol=re.r.rol(T);
double visl=re.r.viscosity_l(T);
double vv=velocityl(T);
double res=(di*vv*rol/visl);
return res;
}
//velocities
```

```
public double  velocityl(double T)
{
// avarage velocity of the pipe m/x
double mboru=mi;
double a[];
double rol=re.r.rol(T);
double A=Math.PI*di*di/4.0;
double Vl= (1.0-xs)*mboru/rol/A;
return Vl;
}
public double f(double Re)
 {
 //Goudar equation
 double a=2/Math.log(10);
 double b=eod/3.7;
 double d=Math.log(10.0)/5.02*Re;
 double s=b*d+Math.log(d);
 double q=Math.pow(s,(s/(s+1)));
 double g=b*d+Math.log(d/q);
 double z=Math.log(q/g);
 double dLA=(g/(g+1))*z;
 double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
 double f1=a*(Math.log(d/q)+dCFA);
 f1=1.0/(f1*f1);
 return f1;
 }
public double  velocityv(double T)
{
// avarage velocity of the pipe m/x
double x=0.0;
double a[];
double rov=re.r.rov(T);
double A=Math.PI*di*di/4.0;
x=xs;
double y=x*mi/rov/A;
return y;
}
public double  Rev(double T)
{
// Reynold's number boru içi vapor
double a[];
double rov=re.r.rov(T);
double visv=re.r.viscosity_v(T);
double vv=velocityv(T);
double res=(di*vv*rov/visv);
return res;
}
//boru içi
// single phase internal pipe convective heat trasnfer
public double  hc(double T)
{
// convective heat transfer coefficient
double Rey=0.0;
double kt;
double Nu=0;
double Prt=0;
```

```
double ss=0;
double fr=0;
if(xs>=1) { kt=re.r.k_v(T);Rey=Rev(T);Prt=re.r.Pr_v(T);}
else if(xs<=0) {kt=re.r.k_l(T);Rey=Rel(T);Prt=re.r.Pr_l(T);}
else
{ kt=(re.r.k_l(T)*(1-xs)+re.r.k_v(T)*xs);Rey=Rel(T);Prt=(re.r.Pr_v(T)*xs+re.r.Pr_l(T)*(1.0-xs));}



if(Rey<=2300)  {Nu=3.66;
//System.out.println("Re="+Rey+"Laminar flow");
}    //Laminar region
else if(Rey>2300 && Rey<=3100)
{  //Referans : Heat transfer in all pipe flow regimes:laminar,transitional/intermittent, and turbulent
   // J.P. Abraham, E.M. Sparrow, J.C.K. Tong, International Journal of Heat and Mass Transfer 52(2009) 557-563
          double Re1=(Rey/1000.0);
          Nu=2.2407*Re1*Re1*Re1*Re1-29.499*Re1*Re1*Re1+142.32*Re1*Re1-292.51*Re1+219.88;
          //System.out.println("Re="+Rey+"transitional/intermittent, flow");
}
else   //one phase heat transfer
{ fr  = f(Rey);
//System.out.println("Re="+Rey+"turbulent flow");
  //Prt = sicak_sivi.cs.Pr(t);
  // Turbulent region Gnielinski equation
  // Referance:
  // V. Gnielinski, New equations for heat and mass transfer in turbulent pipe and
  //channel flow, Int. Chem. Eng. 16 (1976) 359–367.
  // Valid for 2000 < Re <5e5, 0.5 < Pr < 2000
  Nu=fr/8.0*(Rey-1000.0)*Prt/
  (1+12.7* Math.sqrt(fr/8.0)*( Math.pow(Prt,(2.0/3.0))-1.0));
}
  if(xs>=1)  kt=re.r.k_v(T);
  if(xs<=0)  kt=re.r.k_l(T);
  else kt=re.r.k_l(T)*(1-xs)+re.r.k_v(T)*xs;
  ss=Nu*kt/di;
  return ss;
}
          public static void main(String arg[])
          {double mi=0.01;// kg/s;
            double dii=24.3e-3; //m pipe inlet diameter
     double Li=18.292; // m pipe length
     double xi=0.999;
     double epsi=1e-4;
     String name="R134a";
     double tsat=70.0;
     double tw=20.0;
     HT_inside_pipe_condensing htpc=new HT_inside_pipe_condensing(name,mi,dii,Li,epsi);
     double h1=htpc.h_condensing_chato(xi, tsat,tw);
     System.out.println("chato h1="+h1);
      double h2=htpc.h_condensing_Rohsenov(xi, tsat,tw);
      System.out.println(" Rohsenov h2="+h2);
      double h3=htpc.h_condensing_Shah(xi, tsat,tw);
      System.out.println(" Shah h3="+h3);
      double h4=htpc.h_condensing_ElHajal(xi, tsat,tw);
       System.out.println("El hajal h4="+h4);
       htpc.xs=1.1;
```

```
        double h5=htpc.hc(tsat);
          System.out.println("di="+htpc.di+"single phase liquid h5="+h5);
            }
}
```

```
// refrigerant property values
// r[0] P Basınç kPa
// r[1] t Sıcaklık derece C
// r[2] v Özgül hacim m^3/kg
// r[3] h KJ/kg
// r[4] u KJ/kg
// r[5] s KJ/kgK
// r[6] x kg buhar/kg toplam
// r[7] ro yoğunluk kg/m^3
import java.util.Arrays;
public class HT_geothermal_condenser
{ HT_pipe_round_finned p_out;
  HT_inside_pipe_condensing p_in;
  int n;  //number of division in finite difference
  public double T[];   // temperature internal pipe degree C
  public double P[];   // pressure inernal pipe kPa
  public double s[];   // entrpy internal pipe kJ/kgK
  public double h[];   // enthalpy internal pipe kJ/kg
  public double x[]; //quality internal pipe kg vapor/kg mix
  public double Cp[]; //specific heat
  double hp,xp,tp; //properties at t+dt
  double Ta1; //boundary layer temperature for air side
  double dT; //temperature increment for test
  // namei : binary cycle working fluid name
  // epsi : pipe internal surface
  double ar[]; //property loading array for refrigerant
   double ar1[]; //property loading array for refrigerant
   double ar2[]; //property loading array for refrigerant
  double aa[]; //property loading array for refrigerant
  double ha; //convective heat transfer coefficient for air site
  double hr; //convective heat transsfer coefficient for secondary fluid site
  double hv,hl;  //saturated vapor and liquid entalpies;
  double xx[];  // x coordinate
  public HT_geothermal_condenser(int ni)
  {  //air site of the condenser (outside of the pipe)
     n=ni;
     T=new double[n];
     h=new double[n];
     s=new double[n];
     P=new double[n];
     x=new double[n];
    Cp=new double[n];
     xx=new double[n];
           double dri=25.4e-3; //m pipe outlet diameter
     double dii=24.3e-3; //m pipe inlet diameter
     double dfi=57e-3; //  fin diameter
     double si=1.651e-3;  //m distance between fins
     double tfi=0.88e-3; // m fin thickness
     double Li=18.292; // m pipe length
     String materiali="copper"; // m/s
     double U=5.0;
     double T_air=29.0+273.15; //degree K
```

```
 double P_air=101.325; //kPa
 double Pt=50.8356912e-3; //
 double Pl=58.7e-3;  //m
 dT=0.1;
p_out=new HT_pipe_round_finned(materiali,dri,dii,dfi,si,tfi,Li,Pt,Pl);
//power plant working fluid (condensation) site of the condenser (inner pipe)
double mi=0.015;// kg/s;
 double xi=0.1;
 double epsi=1e-4;
 String name="R134a";
 p_in=new HT_inside_pipe_condensing(name,mi,dii,Li,epsi);
}
public void calculate(double T_air,double P_air,double Twater_in,double Pwater_in,double U)
{ T[0]=Twater_in;
  P[0]=Pwater_in;
  ar=p_in.re.property("tp",T[0],P[0]);
  ar1=p_in.re.property("tp",(T[0]+dT),P[0]);
  h[0]=ar[3];
  s[0]=ar[5];
  x[0]=ar[6];
  xx[0]=0.0;
  hp=ar1[3];
  xp=ar1[3];
  Cp[0]=(hp-h[0])/dT;
  double tw=(T[0]*1000.0+T_air*10.0)/1100.0;
  System.out.println("tw="+tw);
  if(x[0]>=0 &&x[0]<=1)
     {//hr=p_in.h_condensing_chato(x[0], T[0],tw);
       hr=p_in.h_condensing_chato(x[0], T[0],tw);
     }
  else
     {hr=p_in.hc((T[0]+tw)/2.0);}
  p_out.setTP(T_air,P_air);
  ha=p_out.h3(U,T_air,P_air); //air convective heat transfer coefficient
  p_out.eta_fin(ha);
  double dQ=0.0;
  // =================for ============
  for(int i=1;i<n;i++)
  {xx[i]=p_in.L/n*i;
          double UU=1.0/(1.0/(hr)+p_in.A*Math.log(p_out.dr/p_out.di)/(2.0*Math.PI*p_out.k*p_out.L)+p_in.A/(h
    dQ=UU*p_in.A/n*(T[i-1]-T_air);
    h[i]=h[i-1]-dQ*1e-3/p_in.mi;
    ar1=p_in.re.property("px",P[0],0.0);
    ar2=p_in.re.property("px",P[0],1.0);
    hl=ar1[3];
    hv=ar2[3];
    x[i]=(h[i]-hl)/(hv-hl);
    if(x[i]>=0 && x[i]<=1) //saturation zone
    {T[i]=ar[1];
     //hr=h_condensing_ElHajal(x[0], T[0],tw);
    hr=p_in.h_condensing_chato(x[i], T[i],tw);
     System.out.println("saturation i="+i+"x="+x[i]+"T="+T[i]);
    }
    else if(x[i]>1)  //vapor zone
    {          ar=p_in.re.property("ph",P[0],h[i]);
              T[i]=ar[1];         s[i]=ar[5];
```

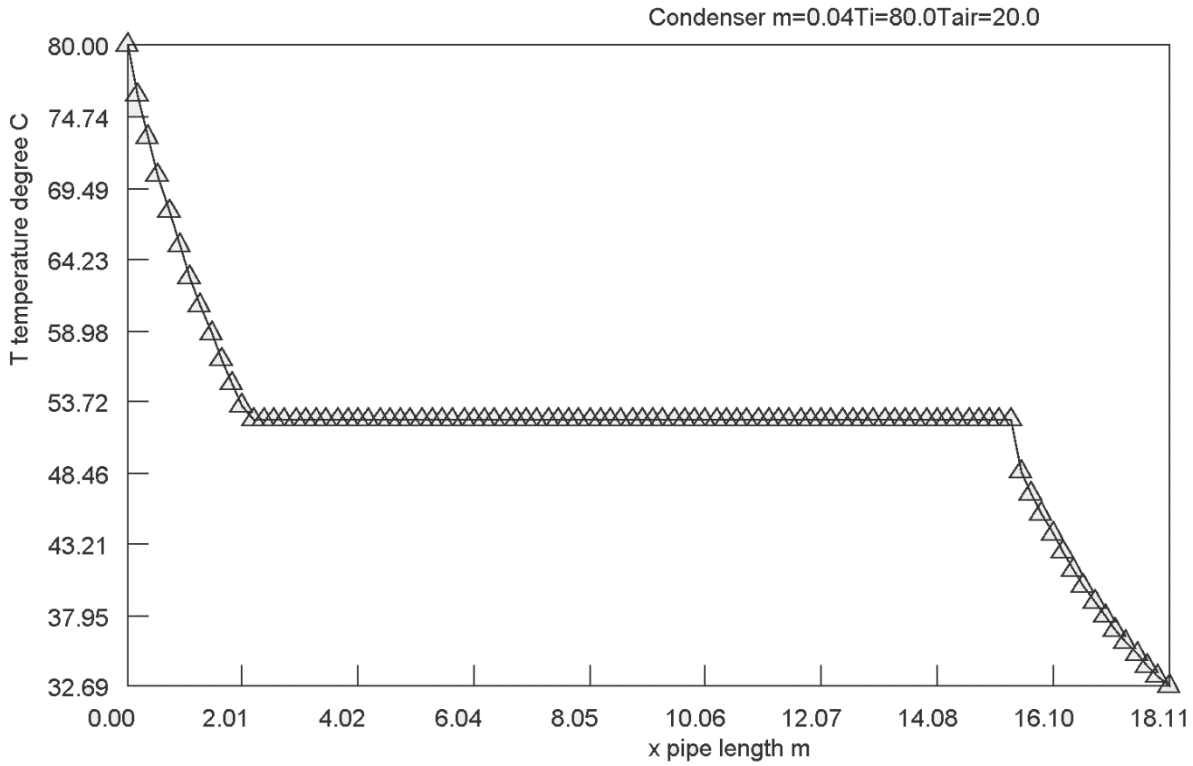
```
        x[i]=ar[6];hr=p_in.hc((T[i]+tw)/2.0);
         System.out.println("vapor zone i="+i+"x="+x[i]+"T="+T[i]);
        }
        else if(x[i]<0) //liquid zone
        {         ar=p_in.re.property("ph",P[0],h[i]);
                 T[i]=ar[1];         s[i]=ar[5];
        x[i]=ar[6];hr=p_in.hc((T[i]+tw)/2.0);
         System.out.println("liquid zone i="+i+"x="+x[i]+"T="+T[i]);
        }

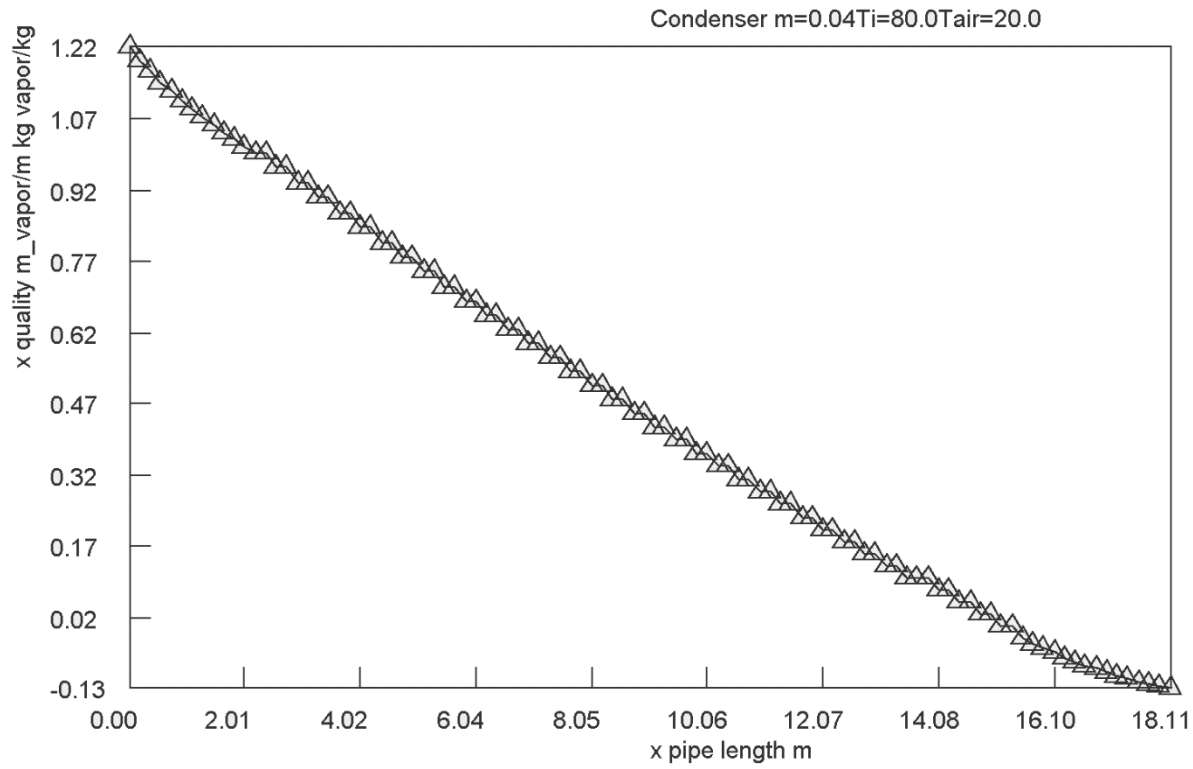

        //==properties==

      p_out.setTP(tw,P_air);
      ha=p_out.h3(U,T_air,P_air); //air convective heat transfer coefficient
      // weighted average
      tw=(p_out.A*ha*T_air+p_in.A*hr*T[0])/(p_out.A*ha+p_in.A*hr);
      //tw=(10.0*T_air+1000.0*T[0])/(T_air+T[0]);
      System.out.println("i="+i+"tw="+tw+"Ti="+T[i]+"tw="+tw);

      if(x[i]>=0 &&x[i]<=1)
        {hr=p_in.h_condensing_chato(x[i], T[0],tw);}
      else
        {hr=p_in.hc((T[i]+tw)/2.0);}
      p_out.setTP(tw,P_air);
      ha=p_out.h3(U,T_air,P_air); //air convective heat transfer coefficient
   }
           double bb[][]={xx,T};
      Text.printT(bb);
      graph2D p=new graph2D();
      p.addData(xx,T);
      p.addPoint(xx,T);
      //Plot p=new Plot(xx,T);
      p.plot();
    }

public static void main(String arg[])
{HT_geothermal_condenser htgc=new HT_geothermal_condenser(100);
double T_air=20.0; //degree C
double T_ref_in=80.0;
double P_ref_in=1400.0;
double P_air_in=100.0;
double U=5.0; //air velocity m/s
 htgc.calculate(T_air,P_air_in,T_ref_in,P_ref_in,U);
}
}
```





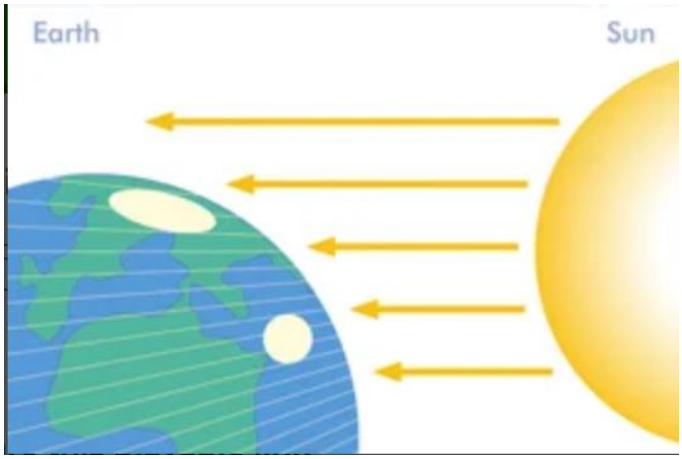
x m pipe length	T degree C	x quality
0	80	1.216183334
0.18292	76.3621003	1.18904021
0.36584	73.2921337	1.165968341
0.54876	70.4434689	1.144397529

0.73168	67.7869695	1.124111427
0.9146	65.3083272	1.105005313
1.09752	62.9940794	1.086981359
1.28044	60.8324205	1.069954722
1.46336	58.8130548	1.053852525
1.64628	56.92673	1.038610332
1.8292	55.1644104	1.024165504
2.01212	53.517217	1.010456407
2.19504	52.4227524	0.997428374
2.37796	52.4227524	0.997428374
2.56088	52.4227524	0.965402255
2.7438	52.4227524	0.965402255
2.92672	52.4227524	0.933617489
3.10964	52.4227524	0.933617489
3.29256	52.4227524	0.902020116
3.47548	52.4227524	0.902020116
3.6584	52.4227524	0.87059749
3.84132	52.4227524	0.87059749
4.02424	52.4227524	0.839346074
4.20716	52.4227524	0.839346074
4.39008	52.4227524	0.808266583
4.573	52.4227524	0.808266583
4.75592	52.4227524	0.777362282
4.93884	52.4227524	0.777362282
5.12176	52.4227524	0.746638205
5.30468	52.4227524	0.746638205
5.4876	52.4227524	0.716100719
5.67052	52.4227524	0.716100719
5.85344	52.4227524	0.685757257
6.03636	52.4227524	0.685757257
6.21928	52.4227524	0.655616119
6.4022	52.4227524	0.655616119
6.58512	52.4227524	0.62568631
6.76804	52.4227524	0.62568631
6.95096	52.4227524	0.595977377
7.13388	52.4227524	0.595977377
7.3168	52.4227524	0.566499248
7.49972	52.4227524	0.566499248
7.68264	52.4227524	0.537262042
7.86556	52.4227524	0.537262042
8.04848	52.4227524	0.508275849
8.2314	52.4227524	0.508275849
8.41432	52.4227524	0.479550476
8.59724	52.4227524	0.479550476
8.78016	52.4227524	0.451095145
8.96308	52.4227524	0.451095145
9.146	52.4227524	0.422918148

9.32892	52.4227524	0.422918148
9.51184	52.4227524	0.395026444
9.69476	52.4227524	0.395026444
9.87768	52.4227524	0.367425223
10.0606	52.4227524	0.367425223
10.24352	52.4227524	0.340117415
10.42644	52.4227524	0.340117415
10.60936	52.4227524	0.313103178
10.79228	52.4227524	0.313103178
10.9752	52.4227524	0.286379359
11.15812	52.4227524	0.286379359
11.34104	52.4227524	0.259938962
11.52396	52.4227524	0.259938962
11.70688	52.4227524	0.233770627
11.8898	52.4227524	0.233770627
12.07272	52.4227524	0.20785815
12.25564	52.4227524	0.20785815
12.43856	52.4227524	0.182180041
12.62148	52.4227524	0.182180041
12.8044	52.4227524	0.156709143
12.98732	52.4227524	0.156709143
13.17024	52.4227524	0.131412267
13.35316	52.4227524	0.131412267
13.53608	52.4227524	0.106249802
13.719	52.4227524	0.106249802
13.90192	52.4227524	0.106249802
14.08484	52.4227524	0.081175127
14.26776	52.4227524	0.081175127
14.45068	52.4227524	0.056133468
14.6336	52.4227524	0.056133468
14.81652	52.4227524	0.031059104
14.99944	52.4227524	0.031059104
15.18236	52.4227524	0.005866698
15.36528	52.4227524	0.005866698
15.5482	48.6307628	-0.01960128
15.73112	46.9922738	-0.0306058
15.91404	45.4498128	-0.04096536
16.09696	43.9994963	-0.05070606
16.27988	42.6353316	-0.05986814
16.4628	41.3517821	-0.06848879
16.64572	40.143804	-0.07660188
16.82864	39.006751	-0.08423862
17.01156	37.9362326	-0.0914285
17.19448	36.9280758	-0.09819954
17.3774	35.9783486	-0.10457815
17.56032	35.0833939	-0.1105889
17.74324	34.2398586	-0.1162543

17.92616	33.4446522	-0.12159511
18.10908	32.6949086	-0.12663058

PART III RADIATION HEAT TRANSFER



17.0 RADIATION BASIC DEFINITIONS

Thermal rediative energy may be viewed as consisting of electromagnetic waves (as predictited by electromagnetic theory) or as consisting of massless photons (as predicted by quantum mechanics). All electromagnetic waves, or photons, are known to propogate through any medium at high velocities. This velocity is known as speed of light. Spped of light depends o the medium through which it travels.

$c = \frac{c_0}{n}$ where c is speed of light, c_0=2.998x10^8 m/s is speed of light in vacuum an n is refractive index.

Some of the basic definition for electromagnetic radiation is as follows

Frequency ν (measured in cycles/s = Hz);

Wavelength, λ (measured in μm=10^{-6}m

Wavenumber, η(measured in 1/cm)

Angular frequency, ω (measured in radian/s)

$\nu = \frac{\omega}{2\pi} = \frac{c}{\lambda} = c\eta$

Each photon carries with it an amount of energy , ε, determined from quantum mechanics as $\varepsilon = h\nu$ where h=6.626069934x10^{-34} J/K is Plank's constant. Electromagnetic spectrum is shown in the following figure.

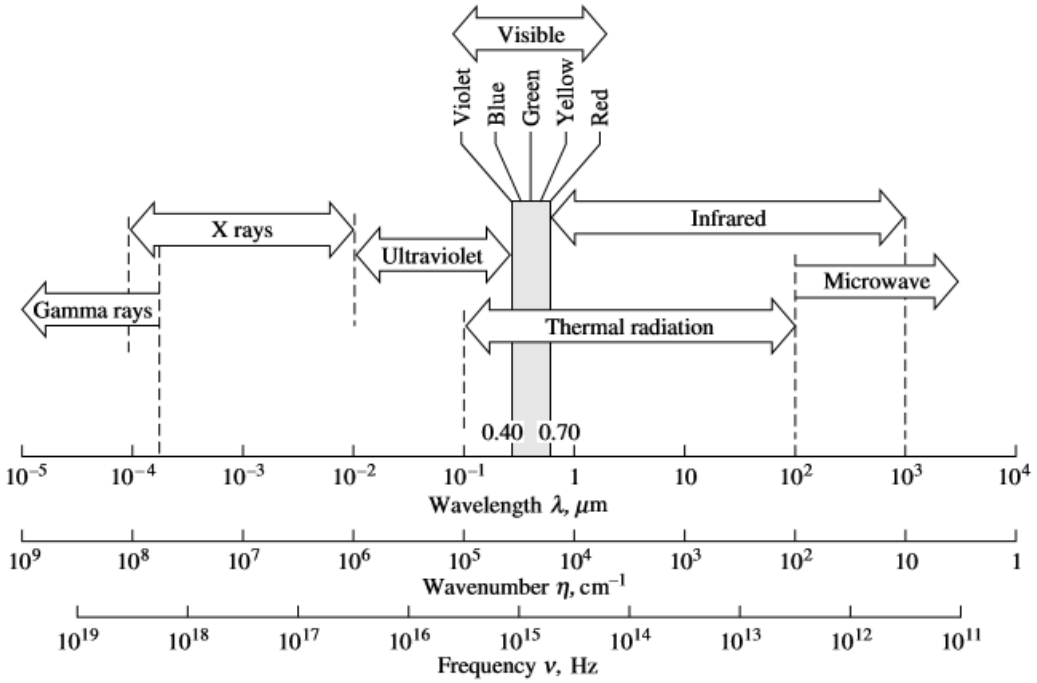
Detailed information about refraction index of different mediums are given at internet side https://refractiveindex.info/?shelf=main&book=Ag&page=Johnson

It should be note that refraction index is also function of wavelength (or other related properties). Some example common material refractive indexes are given here

substance	n	n(λ) formula
Standard air: dry air at 15 °C, 101.325 kPa and with 450 ppm CO_2 content. (λ=1.69 μm)	1.00027316	$n - 1 = \frac{0.05792105}{238.0185-\lambda^{-2}} + \frac{0.00167917}{57.362-\lambda^{-2}}$
Methane CH_4 (λ=1.69 μm)	1.00043647	
Ethane (λ=1.69 μm)	1.0007483	
Glass BK7 (λ=1.69 μm)	1.5140	$n^2 - 1 = \frac{1.03961212\lambda^2}{\lambda^2-0.00600069867} + \frac{0.231792344\lambda^2}{\lambda^2-0.0200179144} + \frac{1.01046945\lambda^2}{\lambda^2-103.560653}$

Glass Fused slica (quartz) (λ=1.69 μm)	1.4561	$n^2 = 1.7606279 - 0.0079050\lambda^2 - 0.0066381\lambda^4 + \frac{0.00646123}{\lambda^2} + \frac{0.000111406}{\lambda^4}$
Carbondioxide (CO_2) (λ=0.58929 μm)	1.001	
Water (H_2O) (λ=0.58929 μm)	1.33	$n^2 - 1 = \frac{0.6961663\lambda^2}{\lambda^2 - 0.004679148} + \frac{0.4079426\lambda^2}{\lambda^2 - 0.013512063} + \frac{0.8974794\lambda^2}{\lambda^2 - 97.93400254}$

Each photon carries with it an amount of energy ε, determined from quantum mechanics as $\varepsilon=h\nu$ where $h=6.626\times10^{-34}$ Js and known as Plank's constant. Electromagnetic waves has a range from $\lambda=0$ μm to $\lambda=\infty$ μm. Smaller wavelengths(bigger frequencies) carries more energy. Ranges of wavelengths (electromagnetic spectrum) and specific names given to the electromagnetic radiation in that particular wavelengths are given in the figure below.



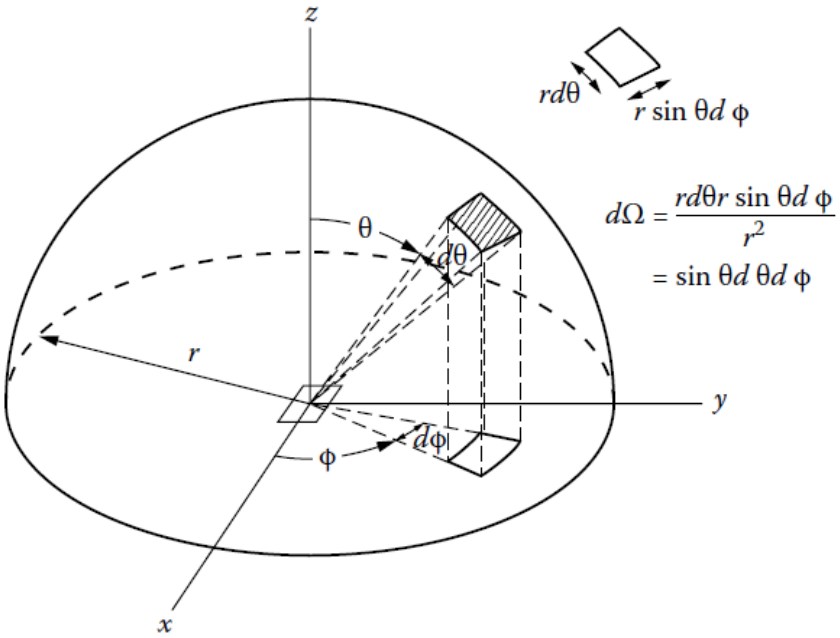
Electromagnetic spectrum.

17.1 SOLID ANGLE AND SPECTRAL RADIATIVE INTENSITY

Radiative transfer happens between surfaces in three dimensional space (volume), therefore angular relations of the directions defined with solid angles. The unit of solid angle is steradians. Definition of solid angle for a semispherical surface can be given as

$d\Omega=\frac{rd\theta r\sin(\theta)d\phi}{r^2} = sin(\theta)d\theta d\phi$ The integral of dW over all angles of surrounding hemisphere yields 2π steradians

$$\Omega=\int_{\theta=0}^{\pi/2}\int_{\phi=0}^{2\pi} sin(\theta)d\theta d\phi = 2\pi \ \ steradian$$



The spectral radiative intensity I_λ is defined as the spectral radiative energy rate at time t, $dQ_\lambda(S,\Omega,t)$, per unit projected area dA along path S, per unit solid angle $d\Omega$ around Ω, and between a small wavelength interval $d\lambda$ at at around λ

$I_\lambda(S,\Omega,t)=\lim\left[\frac{dQ_\lambda(S,\Omega,t)}{dA d\lambda d\Omega}\right]$ In this definition lim means all properties ar efor small intervals. The unit of spectral radiative intensity is W/(m^2 μm sr). It should be note that the measurable quantitiy is energy which requires of integration of spectral radiative intensity over time t, solid angle Ω and path S yielding the units f Joule. The spectral intensity is a function of seven independent parameters, including three space coordinates (*x*, *y*, *z*) along *S*, two angular coordinates (θ, ϕ) for direction Ω, one wavelength (λ) (or, frequency, ν), and one time (*t*). It is always specified at a wavelength λ per unit wavelength interval around that λ. The temperature dependence of intensity is implicit in these expressions, as temperature is specified with the spatial coordinates within the medium. Specifically, the ***spectral intensity*** refers to radiation in an interval *d*λ around a *single* wavelength, while the ***total intensity*** refers to combined radiation including *all* wavelengths. Radiative energy may loses some of its strength as it propogates through a path dS due to scattering and absorbtion. Some of the gases on the path such as carbondioxide
, water, methane are absorbed and emit significant amount of radiation.

Every material at finite temperature (a temperature above 0 K) emits radiative energy. In principle emission is directional in natura and ia a function of local material properties, placement geometry and temperature:
$E_\lambda = f(T,\theta,\phi)$ where T(x,y,z) An ideal body would emit maximum energy uniformly in all direction(independent of direction) **.** Such a body is called **blackbody**. An ideal blackbody also absorbs all the radiative energy coming into it.

In calculation of radiative energy loss by a surface, the angular profile of emmisison needs to be determined over a hemispherical envelop placed over a black surface. The hemispherical spectral emmissive power $E_{\lambda,b}$, is the energy emitted by a black surface per unit time per unit area and per unit wavelength interval around λ

$I_{\lambda,b}\cos(\theta)\, d\Omega d\lambda = I_{\lambda,b}\cos(\theta)\sin(\theta) d\theta d\phi d\lambda$

To obtain the blackbody spectral emission passing through the entire hemisphere in terms of the blackbody intensity above equation is integrated over all solid angles of hemisphere to give

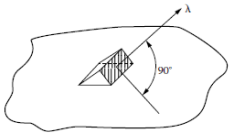
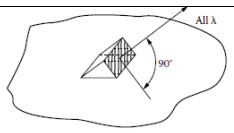
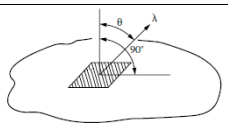
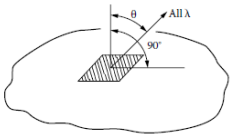
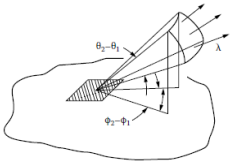
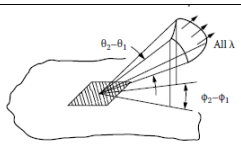
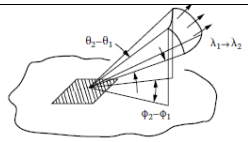
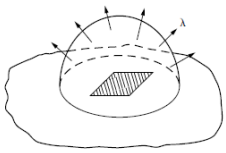
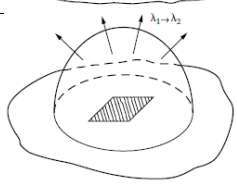
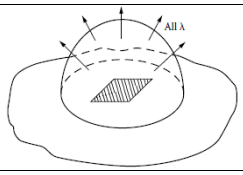
$E_{\lambda,b}d\lambda = I_{\lambda,b}\, d\lambda \int_{\phi=0}^{2\pi}\int_{\theta=0}^{\pi/2}\sin(\theta)\cos(\theta) d\theta d\phi = \pi I_{\lambda,b} d\lambda$

Which is a remarkably simple relation: The blackbody hemispherical emissive power is π times the blackbody intensity. If emission is desired only part of the hemispherical space from θ_1 to θ_2 and ϕ_1 to ϕ_2

$E_{\lambda,b}(\theta_1 \text{ to } \theta_2, \phi_1 \text{ to } \phi_2)d\lambda =$

$I_{\lambda,b}\, d\lambda \int_{\phi=\phi_1}^{\phi_2} \int_{\theta=\theta_1}^{\theta_2} \sin(\theta)\cos(\theta) d\theta d\phi = (\phi_2 - \phi_1)\left[\frac{sin^2(\theta_2) - sin^2(\theta_1)}{2}\right] I_{\lambda,b} d\lambda$

As it is seen so far several variables are existed to define intensities and emissive powers of blackbody radiation. In order to reduce confusion a list of blackbody radiation related quantities are given as a list[98]

Symbol	Name	Definition	Geometry	Formula
$I_{\lambda,b}(T)$	Spectral intensity	Emission in any direction per unit of projected area normal to that direction, and per unit time, wavelength interval about λ, and solid angle		$\frac{C_1/\pi}{n^2\lambda^5[exp(C_2/(n\lambda T) - 1]}$
$I_b(T)$	Total intensity	Emission, including all wavelengths in any direction per unit of projected area normal to that direction, and per unit time and solid angle		$\sigma T^4/\pi$
$E_{\lambda,b}(\theta,T)$	Directional spectral emissive power	Emission per unit solid angle in direction θ per unit surface area, wavelength interval, and time		$I_{\lambda,b}(T)\cos(\theta)$
$E_b(\theta,T)$	Directional total emissive power	Emission, including all wavelengths, in direction θ per unit surface area, solid angle, and time		$\sigma T^4/\pi\cos(\theta)$
$E_{\lambda,b}(\theta_1 \to \theta_2, \phi_1 \to \phi_2, T)$	Finite solid angle spectral emissive power	Emission in solid angle $\theta_1 \le \theta \le \theta_2$, $\phi_1 \le \phi \le \phi_2$ per unit surface area, wavelength interval, and time		$I_{\lambda,b}(T)(\phi_2 - \phi_1)\left[\frac{sin^2(\theta_2) - sin^2(\theta_1)}{2}\right]$
$E_b(\theta_1 \to \theta_2, \phi_1 \to \phi_2, T)$	Finite solid angle total emissive power	Emission, including all wave lengths, in solid angle $\theta_1 \le \theta \le \theta_2$, $\phi_1 \le \phi \le \phi_2$ per unit surface area and time.		$\frac{\sigma T^4}{\pi}\left[\frac{sin^2(\theta_2) - sin^2(\theta_1)}{2}\right](\phi_2 - \phi_1)$
$E_{\lambda,b}(\lambda_1 \to \lambda_2, \theta_1 \to \theta_2, \phi_1 \to \phi_2, T)$	Finite solid-angle band emissive power	Emission in solid angle $\theta 1 \le \theta \le \theta 2, \phi 1 \le \phi \le \phi 2$ and wavelength band $\lambda 1 \to \lambda 2$ per unit surface area and time		$\frac{\sigma T^4}{\pi}\left[\frac{sin^2(\theta_2) - sin^2(\theta_1)}{2}\right](\phi_2 - \phi_2)(F_{0\to\lambda_2} - F_{0\to\lambda_1})$
$E_{\lambda,b}(T)$	Hemispherical spectral emissive power	Emission into hemispherical solid angle per unit surface area, wavelength interval, and time		$\pi I_{\lambda,b}(T)$
$E_{\lambda,b}(\lambda_1 \to \lambda_2, T)$	Hemispherical band emissive power	Emission in wavelength band $\lambda 1 \to \lambda 2$ into hemispherical solid angle per unit surface area and time		$(F_{0\to\lambda_2} - F_{0\to\lambda_1})\sigma T^4$
$E_b(T)$	Hemispherical total emissive power	Emission, including all wavelengths, into hemispherical solid angle per unit surface area and time		σT^4

18 BLACK BODY RADIATION FUNCTION

Blackbody spectral diffuse emissive power (Plank's law)

$$\pi I_{\lambda,b}(\lambda,T)=E_{\lambda,b}(\lambda,T)=\frac{2\pi hc_0^2}{n^2\lambda^5[exp(hc_0/(n\lambda kT)-1]}$$

Coefficients used in the equations : k=1.380649x10^{-23} J/K is Boltsmann's constant, h=6.62607015x10^{-34} Js is Planck' constant, c0= 2.99792458x10^{8} m/s is the speed of light. N is the refraction index. Note that above values of Planck' constant, Boltsmann's constant and speed of light is now SI standards(Plank' constant, Boltsmann's constant are effective from 20 May 2019, speed of light is already in effect) . Latest NIST- National Institutes of Standarts and Technology of USA measure for Plank's constant is h= 6.626069934 x10^{-34} Js with 34 parts per billion uncertinity. Combining constants together:

$$\pi I_{\lambda,b}(\lambda,T)=E_{\lambda,b}(\lambda,T)=\frac{C_1}{n^2\lambda^5[exp(C_2/(n\lambda T)-1]}$$

$$\frac{E_{\lambda,\beta}(\lambda,T)}{n^3T^5}=\frac{C_1}{(n\lambda T)^5[exp(C_2/(n\lambda T)-1]}$$

h	Plank's constant	6.6260701500E-34	J s		
k	Boltzman's constant	1.3806490000E-23	J/K		
c_o	Speed of light in vacuum	2.9979245800E+08	m/s		
C_1	$2\pi hc_0^2$	3.7417718522E-16	Wm^2	3.7417718522E+08	W μm^4/m^2
C_2	hc_0	1.4387768775E-02	W/mK	1.4387768775E+04	W/μmK

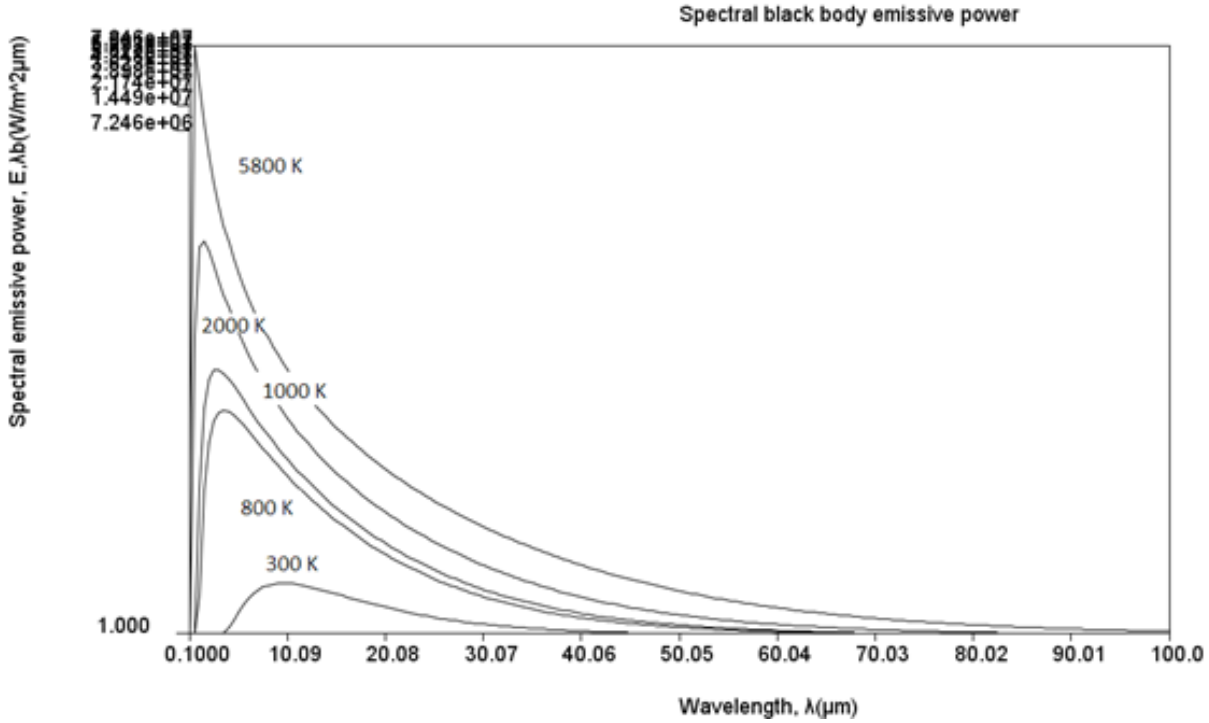
```
 import java.io.*;

class f4 extends f_x
{ public double T;
  double C1,C2;
  //lambda mikrometers
public f4(double Ti)
{T=Ti;
  C1=3.7417718522e8;    //  Wmikrometre^4/m^2
  C2=1.4387768775e4;          //  mikrometre K
  }
  public void setT(double Ti)
  {T=Ti;}
public double func(double lambda)
{
double ff;
double l5=lambda*lambda*lambda*lambda*lambda;
ff= C1/l5/(Math.exp(C2/lambda/T)-1);
return ff; //maksimum testi
}
}

class PlotT03
{public static void main (String args[]) throws IOException
  {
  double T[]={50,100,300,800,1000,2000,5800}; //K
  f4 f1=new f4(T[0]);
  Plot pp=new Plot(f1,0.1,100.0);
  for(int i=0;i<T.length;i++)
  {f1.setT(T[i]);
    pp.addFunction(f1,0.1,100.0,200);
   }
   pp.setYlogScaleOn();
    String sp="Spectral black body emissive power";
   String sx="Wavelength, "+'\u03BB'+"("+'\u03BC'+"m)";
```

```
   String sy="Spectral emissive power, E,"+"\u03BB"+"b(W/m^2"+"\u03BC"+"m)";
   pp.setPlabel(sp);
   pp.setXlabel(sx);
    pp.setYlabel(sy);
   pp.plot();
}}
```

$$\frac{E_{\lambda,\beta}(\lambda,T)}{n^3T^5} = \frac{C_1}{(n\lambda T)^5[exp(C_2/(n\lambda T)-1]}$$



Derivative of this function will give the maximum curve

$$\frac{d}{d(n\lambda T)}\left(\frac{E_{\lambda,\beta}(\lambda,T)}{n^3T^5}\right) = \frac{d}{d(n\lambda T)}\left(\frac{C_1}{(n\lambda T)^5[exp(C_2/(n\lambda T)-1]}\right) = -\frac{1}{u^2}\left[5x^4\left(\exp\left(\frac{C_2}{x}\right)-1\right)+x^5\left(-\frac{C_2}{x^2}\right)\exp\left(\frac{C_2}{x}\right)\right] = 0$$

$$x = n\lambda T \quad u = x^5\left[\exp\left(\frac{C_2}{x}\right)-1\right]$$

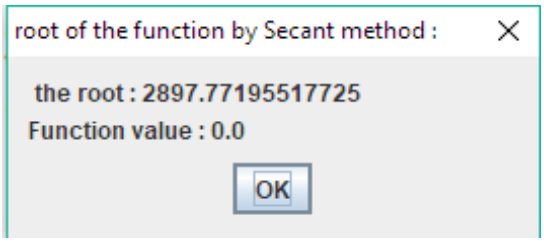
```
 ///Numerical Analysis
// Secant method with one root estimation
// Linear difference formula
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

//============= function definition ================

class fb extends f_x
{  public double C2= 1.4387768775E+04;
   public double func (double x)
   { return 5*x*x*x*x*x*(Math.exp(C2/x)-1.0)+x*x*x*x*x*(-C2/(x*x)*Math.exp(C2/x));}
}
public class NA8
{
public static double secant(f_x f,double x)
{
int nmax=100;
double tolerance=1.0e-10;
double fx,dfx;
double dx=0.01;
for(int i=0;i<nmax;i++)
{
```

```
  fx=f.func(x);
  dfx=(f.func(x+dx)-f.func(x-dx))/(2.0*dx);
  x-=fx/dfx;
  if(Math.abs(fx)<tolerance) { return x;}
 }
JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration is exceeded \n"+
   " result may not be valid","MAXİMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}

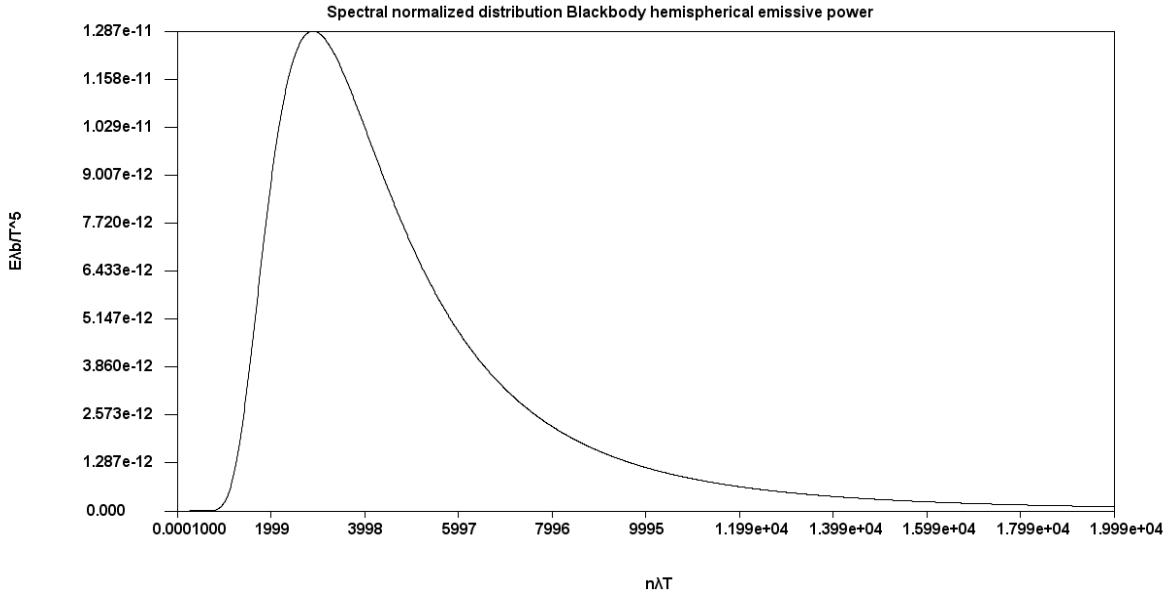
    public static void main (String args[])
    {
              fb f=new fb();
    double x0;
    x0=Double.parseDouble(JOptionPane.showInputDialog(" First estimation for the root : "));
    double r= secant(f,x0);
    JOptionPane.showMessageDialog(null," the root : "+r+"\nFunction value : "+f.func(r),
    "root of the function by Secant method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



So $(n\lambda T)_{max} = 2897.77195\ \mu mK$ This is called Wien's displacement law

We can also see this maximum by looking normalized function plotted directly

```
public class Black_body_radiation
{ public static double E(double x)
  { double C1=3.7417718522E+08;
             double C2=1.4387768775E+04;
    double y=C2/x;
  double E=C1/(x*x*x*x*x*(Math.exp(C2/x)-1.0));
  return E;
  }
  public static void main(String arg[])
  {
  double xx[]=new double[2000];
  double ff[]=new double[2000];
  int i=0;
  for(double x=0.0001;x<=20000;x+=10)
  {xx[i]=x;ff[i]=E(x);i++;}
  String s="Spectral normalized distribution Blackbody hemispherical emissive power";
  String s1[]={"n"+'\u03BB'+"T","E"+'\u03BB'+"b/T^5"};
  double a[][]={xx,ff};
  Text.printT(a,s1,s);
  Plot p=new Plot(xx,ff);
  p.setPlabel(s);
  p.setXlabel(s1[0]);
  p.setYlabel(s1[1]);
  p.plot();
  }
}
```



Now we would like to integrate Blackbody spectral diffuse power equation for all the wavelengths to find the total blackbody power for all spectrum.

$\frac{E_b(\mathrm{T})}{n^3T^5}=\int_{\lambda=0}^{\infty}\frac{C_1 d\lambda}{(n\lambda T)^5[exp(C_2/(n\lambda \mathrm{T})-1]}$ Converting variables gives:

$\xi=\frac{C_2}{\mathrm{n}\lambda\mathrm{T}}\quad \lambda=\frac{C_2}{\mathrm{n}\xi\mathrm{T}}\quad d\lambda=-\frac{C_2}{\mathrm{n}\xi^2\mathrm{T}}d\xi$

$\frac{E_b(\mathrm{T})}{n^3T^5}=\int_{\xi=\infty}^{0}\frac{C_1\xi^5}{C_2^5}\left(-\frac{C_2}{\xi^2\mathrm{T}}d\xi\right)$

$E_{\mathrm{b}}(T)=\left[\left(\frac{C_1}{C_2^4}\right)n^2T^4\int_{\xi=0}^{\infty}\frac{\xi^3}{[exp(\xi)-1]}d\xi\right]$

$\int_{\xi=0}^{\infty}\frac{\xi^3}{[\exp(\xi)-1]}d\xi=\int_{\xi=0}^{\infty}\frac{e^{-\xi}\xi^3}{e^{-\xi}[e^{\xi}-1]}d\xi=\int_{\xi=0}^{\infty}\frac{e^{-\xi}\xi^3}{[1-e^{-\xi}]}d\xi=\int_{\xi=0}^{\infty}\xi^3\sum_{n=1}^{\infty}e^{-n\xi}d\xi$

In order to solve the integral a new variable change will be applied. Each summation in the integral can be evaluated seperately.

$u=n\xi\quad du=nd\xi$

$\int_{\xi=0}^{\infty}\xi^3\sum_{n=1}^{\infty}e^{-n\xi}d\xi=\frac{1}{n^4}\int_{u=0}^{\infty}u^3e^{-u}du$

$\int_{u=0}^{\infty}u^3e^{-u}du=-u^3e^{-u}|_{u=0}^{\infty}+3\int_{u=0}^{\infty}u^2e^{-u}du$

$\int_{u=0}^{\infty}u^2e^{-u}du=-u^2e^{-u}|_{u=0}^{\infty}+2\int_{u=0}^{\infty}ue^{-u}du$

$\int_{u=0}^{\infty}ue^{-u}du=-ue^{-u}|_{u=0}^{\infty}+2\int_{u=0}^{\infty}e^{-u}du$

$\int_{u=0}^{\infty}e^{-u}du=-e^{-u}|_{u=0}^{\infty}=-(0-1)=1$

$\int_{\xi=0}^{\infty}\xi^3\sum_{n=1}^{\infty}e^{-n\xi}d\xi=\sum_{n=1}^{\infty}\frac{6}{n^4}=6\frac{\pi^4}{90}=\frac{\pi^4}{15}$

Therefore

$E_{\mathrm{b}}(T)=\left[\left(\frac{C_1}{C_2^4}\right)\frac{\pi^4}{15}\right]n^2T^4=\sigma n^2T^4$

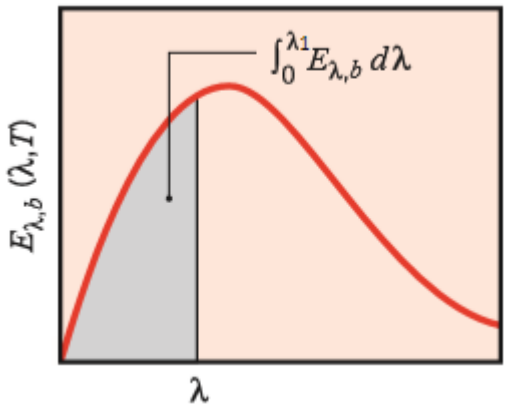
$\sigma=\left[\left(\frac{C_1}{C_2^4}\right)\frac{\pi^4}{15}\right]=5.6703744192\text{x}10^{-8}\ \ \mathrm{W/m^2K^4}$

This equation is called Stephan-Boltzmann law and constant σ is called Stephan-Boltzmann constant.

So $I_{\mathrm{b}}(T)=\left[\left(\frac{C_1}{C_2^4}\right)\frac{\pi^4}{15}\right]\frac{n^2T^4}{\pi}=\frac{\sigma n^2T^4}{\pi}$

If an emissive power of a spectral band is required only,

$$E_{(0\to\lambda 1),\mathrm{b}}(T)=\int_{\lambda=0}^{\lambda 1}E_{\lambda,\mathrm{b}}(T)d\lambda$$



Black body emmissive power of a spectral band is usually given as a ratio to total black body emmissive power. This ratio is called blackbody radiation function

$$F_{(0\to\lambda 1)}(T)=\frac{E_{(0\to\lambda 1),\mathrm{b}}(T)}{E_{\mathrm{b}}(T)}=\frac{\int_{\lambda=0}^{\lambda 1}E_{\lambda,\mathrm{b}}(T)d\lambda}{\int_{\lambda=0}^{\infty}E_{\lambda,\mathrm{b}}(T)d\lambda}=\frac{\int_{\lambda=0}^{\lambda 1}E_{\lambda,\mathrm{b}}(T)d\lambda}{\sigma n^2T^4}$$

$$\xi=\frac{C_2}{\mathrm{n}\lambda\mathrm{T}}\quad \lambda=\frac{C_2}{\xi\mathrm{nT}}\quad d\lambda=-\frac{C_2}{n\xi^2\mathrm{T}}d\xi$$

$$F_{(0\to\lambda 1)}(T)=\frac{E_{(0\to\lambda 1),\mathrm{b}}(T)}{E_{\mathrm{b}}(T)}=\frac{\left[\left(\frac{C_1}{C_2^4}\right)\int_{\xi=\xi 1}^{\infty}\frac{\xi^3}{[e^{\xi}-1]}d\xi\right]n^2T^4}{\left[\left(\frac{C_1}{C_2^4}\right)\frac{\pi^4}{15}\right]n^2T^4}=\frac{15}{\pi^4}\int_{\xi=\xi 1}^{\infty}\frac{\xi^3}{[e^{\xi}-1]}d\xi$$

$$F_{(0\to\lambda 1)}(T)=1-\frac{15}{\pi^4}\int_{\xi=0}^{\xi 1}\frac{\xi^3}{[e^{\xi}-1]}d\xi=1-\frac{15}{\pi^4}\int_{\xi=0}^{\xi 1}\xi^3\sum_{n=1}^{\infty}e^{-n\xi}\,d\xi$$

$$u=n\xi\quad du=nd\xi$$

$$\int_{\xi=0}^{\xi 1}\xi^3\sum_{n=1}^{\infty}e^{-n\xi}\,d\xi=\frac{1}{n^4}\int_{u=0}^{\mathrm{u}1}u^3e^{-u}du$$

$$\int_{u=0}^{\mathrm{u}1}u^3e^{-u}du=-u^3e^{-u}|_{u=0}^{u1}+3\int_{u=0}^{\mathrm{u}1}u^2e^{-u}du$$

$$\int_{u=0}^{\mathrm{u}1}u^2e^{-u}du=-u^2e^{-u}|_{u=0}^{u1}+2\int_{u=0}^{\mathrm{u}1}ue^{-u}du$$

$$\int_{u=0}^{\mathrm{u}1}ue^{-u}du=-ue^{-u}|_{u=0}^{u1}+2\int_{u=0}^{\mathrm{u}1}e^{-u}du$$

$$\int_{u=0}^{\mathrm{u}1}e^{-u}du=-e^{-u}|_{u=0}^{u1}=-(e^{-u1}-1)=-e^{-u1}+1$$

$$\int_{u=0}^{\mathrm{u}1}u^3e^{-u}du=-e^{-u1}(u1^3+3u1^2+6u1+6)+1$$

$$F_{(0\to\lambda 1)}(T)=1-\frac{15}{\pi^4}\sum_{n=1}^{\infty}\frac{1}{n^4}\left[-e^{-n\xi}\left((n\xi)^3+3(n\xi)^2+6(n\xi)+6\right)\right]-1$$

$$F_{(0\to\lambda 1)}(T)=\frac{15}{\pi^4}\sum_{n=1}^{\infty}\frac{e^{-n\xi}}{n}\left[\left(\xi^3+\frac{3\xi^2}{n}+\frac{6\xi}{n^2}+\frac{6}{n^3}\right)\right]\quad \text{where } \xi=\frac{C_2}{\lambda\mathrm{T}}$$

A java code for the function

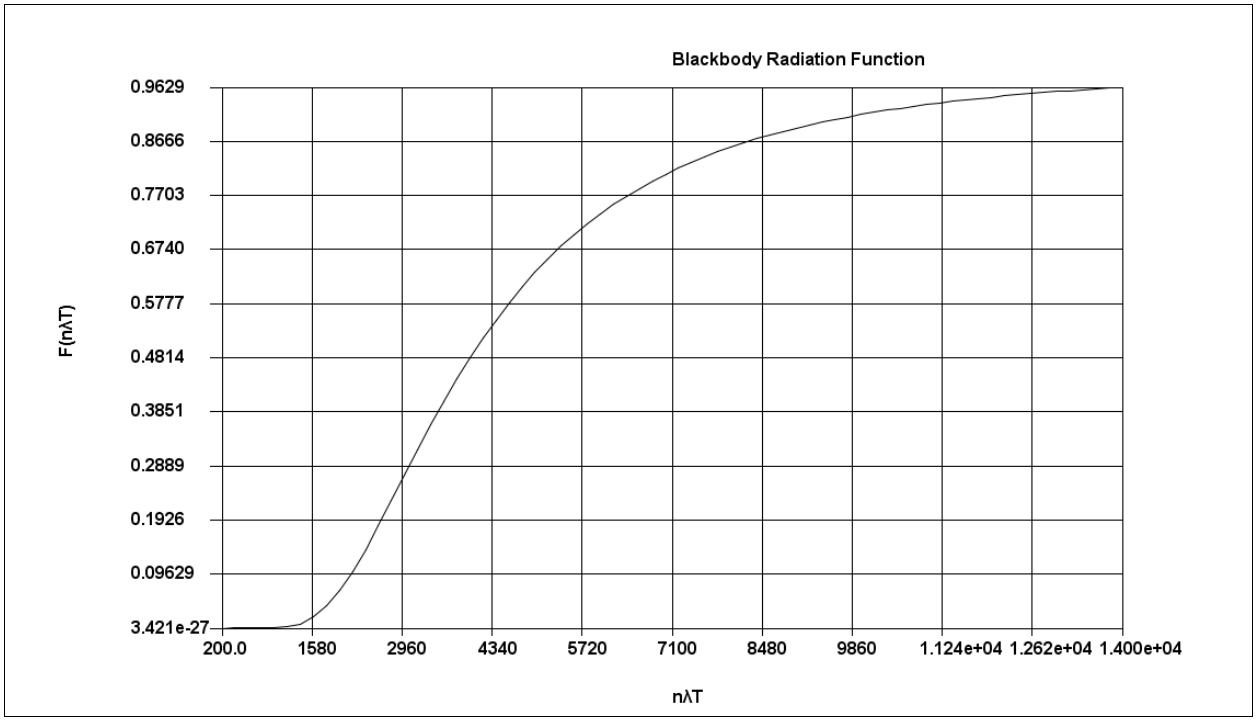
```
public class F_radiation
{ public static double F(double x)
 {double C2=1.438769e4;
 double y=C2/x;
 double total=0;
 for(int n=1;n<80;n++)
 {total+=Math.exp(-n*y)/n*(y*y*y+3.0*y*y/n+6.0*y/(n*n)+6.0/(n*n*n));}
 double pi=Math.PI;
```

```
  total=15.0/(pi*pi*pi*pi)*total;
  return total;
  }
  public static void main(String arg[])
  {
  double xx[]=new double[70];
  double ff[]=new double[70];
  int i=0;
  for(double x=200;x<=14000;x+=200)
  {xx[i]=x;ff[i]=F(x);i++;}
  String s="Blackbody Radiation Function";
  String s1[]={"n"+\u03BB'+"T","F(n"+\u03BB'+"T)"};
  double a[][]={xx,ff};
  Text.printT(a,s1,s);
  Plot p=new Plot(xx,ff);
  p.setPlabel(s);
  p.setXlabel(s1[0]);
  p.setYlabel(s1[1]);
  p.plot();
  }
}
```

Blackbody radiation function values listed

nλT	F(0->nλT)	nλT	F(0->nλT)	nλT	F(0->nλT)	nλT	F(0->nλT)
200	3.4208698924E-27	17800	9.8023063374E-01	35400	9.9705067896E-01	53000	9.9907388328E-01
400	1.8652896018E-12	18000	9.8081267065E-01	35600	9.9709751893E-01	53200	9.9908392844E-01
600	9.2944420531E-08	18200	9.8137220988E-01	35800	9.9714337356E-01	53400	9.9909382893E-01
800	1.6436330973E-05	18400	9.8191032141E-01	36000	9.9718826859E-01	53600	9.9910358735E-01
1000	3.2079017070E-04	18600	9.8242801514E-01	36200	9.9723222895E-01	53800	9.9911320624E-01
1200	2.1343161517E-03	18800	9.8292624476E-01	36400	9.9727527882E-01	54000	9.9912268807E-01
1400	7.7907129692E-03	19000	9.8340591133E-01	36600	9.9731744162E-01	54200	9.9913203527E-01
1600	1.9719854672E-02	19200	9.8386786664E-01	36800	9.9735874004E-01	54400	9.9914125024E-01
1800	3.9343321141E-02	19400	9.8431291627E-01	37000	9.9739919609E-01	54600	9.9915033529E-01
2000	6.6731637674E-02	19600	9.8474182249E-01	37200	9.9743883112E-01	54800	9.9915929271E-01
2200	1.0089198163E-01	19800	9.8515530693E-01	37400	9.9747766582E-01	55000	9.9916812473E-01
2400	1.4026010230E-01	20000	9.8555405307E-01	37600	9.9751572027E-01	55200	9.9917683354E-01
2600	1.8312374980E-01	20200	9.8593870856E-01	37800	9.9755301394E-01	55400	9.9918542127E-01
2800	2.2789309407E-01	20400	9.8630988737E-01	38000	9.9758956572E-01	55600	9.9919389001E-01
3000	2.7323297650E-01	20600	9.8666817181E-01	38200	9.9762539395E-01	55800	9.9920224184E-01
3200	3.1810106319E-01	20800	9.8701411442E-01	38400	9.9766051643E-01	56000	9.9921047874E-01
3400	3.6173287444E-01	21000	9.8734823971E-01	38600	9.9769495042E-01	56200	9.9921860270E-01
3600	4.0360249961E-01	21200	9.8767104578E-01	38800	9.9772871269E-01	56400	9.9922661563E-01
3800	4.4337629438E-01	21400	9.8798300587E-01	39000	9.9776181953E-01	56600	9.9923451944E-01
4000	4.8086862061E-01	21600	9.8828456980E-01	39200	9.9779428675E-01	56800	9.9924231596E-01
4200	5.1600357981E-01	21800	9.8857616528E-01	39400	9.9782612969E-01	57000	9.9925000702E-01
4400	5.4878384176E-01	22000	9.8885819916E-01	39600	9.9785736329E-01	57200	9.9925759439E-01
4600	5.7926624918E-01	22200	9.8913105862E-01	39800	9.9788800202E-01	57400	9.9926507980E-01
4800	6.0754328332E-01	22400	9.8939511226E-01	40000	9.9791805996E-01	57600	9.9927246497E-01
5000	6.3372931881E-01	22600	9.8965071111E-01	40200	9.9794755081E-01	57800	9.9927975157E-01
5200	6.5795065026E-01	22800	9.8989818964E-01	40400	9.9797648785E-01	58000	9.9928694124E-01
5400	6.8033841345E-01	23000	9.9013786659E-01	40600	9.9800488401E-01	58200	9.9929403557E-01
5600	7.0102368458E-01	23200	9.9037004589E-01	40800	9.9803275186E-01	58400	9.9930103615E-01
5800	7.2013419210E-01	23400	9.9059501742E-01	41000	9.9806010362E-01	58600	9.9930794452E-01
6000	7.3779220558E-01	23600	9.9081305775E-01	41200	9.9808695115E-01	58800	9.9931476219E-01
6200	7.5411327148E-01	23800	9.9102443089E-01	41400	9.9811330603E-01	59000	9.9932149064E-01
6400	7.6920554939E-01	24000	9.9122938888E-01	41600	9.9813917948E-01	59200	9.9932813134E-01
6600	7.8316956612E-01	24200	9.9142817248E-01	41800	9.9816458244E-01	59400	9.9933468569E-01
6800	7.9609825444E-01	24400	9.9162101172E-01	42000	9.9818952554E-01	59600	9.9934115510E-01
7000	8.0807717942E-01	24600	9.9180812644E-01	42200	9.9821401913E-01	59800	9.9934754095E-01
7200	8.1918488310E-01	24800	9.9198972685E-01	42400	9.9823807328E-01	60000	9.9935384457E-01
7400	8.2949329782E-01	25000	9.9216601398E-01	42600	9.9826169781E-01	60200	9.9936006729E-01
7600	8.3906819390E-01	25200	9.9233718017E-01	42800	9.9828490223E-01	60400	9.9936621040E-01
7800	8.4796963762E-01	25400	9.9250340948E-01	43000	9.9830769585E-01	60600	9.9937227516E-01
8000	8.5625244362E-01	25600	9.9266487811E-01	43200	9.9833008770E-01	60800	9.9937826283E-01
8200	8.6396661120E-01	25800	9.9282175481E-01	43400	9.9835208659E-01	61000	9.9938417461E-01

8400	8.7115773819E-01	26000	9.9297420120E-01	43600	9.9837370110E-01	61200	9.9939001172E-01
8600	8.7786740859E-01	26200	9.9312237217E-01	43800	9.9839493958E-01	61400	9.9939577532E-01
8800	8.8413355265E-01	26400	9.9326641613E-01	44000	9.9841581016E-01	61600	9.9940146656E-01
9000	8.8999077867E-01	26600	9.9340647542E-01	44200	9.9843632076E-01	61800	9.9940708659E-01
9200	8.9547067750E-01	26800	9.9354268650E-01	44400	9.9845647911E-01	62000	9.9941263650E-01
9400	9.0060210052E-01	27000	9.9367518032E-01	44600	9.9847629273E-01	62200	9.9941811739E-01
9600	9.0541141294E-01	27200	9.9380408250E-01	44800	9.9849576895E-01	62400	9.9942353033E-01
9800	9.0992272378E-01	27400	9.9392951362E-01	45000	9.9851491492E-01	62600	9.9942887637E-01
10000	9.1415809451E-01	27600	9.9405158945E-01	45200	9.9853373761E-01	62800	9.9943415653E-01
10200	9.1813772793E-01	27800	9.9417042118E-01	45400	9.9855224379E-01	63000	9.9943937183E-01
10400	9.2188013913E-01	28000	9.9428611560E-01	45600	9.9857044009E-01	63200	9.9944452327E-01
10600	9.2540231002E-01	28200	9.9439877533E-01	45800	9.9858833297E-01	63400	9.9944961180E-01
10800	9.2871982903E-01	28400	9.9450849899E-01	46000	9.9860592871E-01	63600	9.9945463840E-01
11000	9.3184701733E-01	28600	9.9461538141E-01	46200	9.9862323346E-01	63800	9.9945960400E-01
11200	9.3479704294E-01	28800	9.9471951375E-01	46400	9.9864025320E-01	64000	9.9946450952E-01
11400	9.3758202377E-01	29000	9.9482098371E-01	46600	9.9865699376E-01	64200	9.9946935587E-01
11600	9.4021312086E-01	29200	9.9491987564E-01	46800	9.9867346086E-01	64400	9.9947414393E-01
11800	9.4270062263E-01	29400	9.9501627074E-01	47000	9.9868966003E-01	64600	9.9947887459E-01
12000	9.4505402119E-01	29600	9.9511024715E-01	47200	9.9870559672E-01	64800	9.9948354869E-01
12200	9.4728208131E-01	29800	9.9520188009E-01	47400	9.9872127620E-01	65000	9.9948816709E-01
12400	9.4939290295E-01	30000	9.9529124202E-01	47600	9.9873670364E-01	65200	9.9949273060E-01
12600	9.5139397793E-01	30200	9.9537840273E-01	47800	9.9875188408E-01	65400	9.9949724004E-01
12800	9.5329224128E-01	30400	9.9546342943E-01	48000	9.9876682244E-01	65600	9.9950169621E-01
13000	9.5509411788E-01	30600	9.9554638691E-01	48200	9.9878152353E-01	65800	9.9950609990E-01
13200	9.5680556478E-01	30800	9.9562733763E-01	48400	9.9879599202E-01	66000	9.9951045186E-01
13400	9.5843210967E-01	31000	9.9570634179E-01	48600	9.9881023251E-01	66200	9.9951475286E-01
13600	9.5997888590E-01	31200	9.9578345742E-01	48800	9.9882424947E-01	66400	9.9951900365E-01
13800	9.6145066427E-01	31400	9.9585874054E-01	49000	9.9883804725E-01	66600	9.9952320495E-01
14000	9.6285188206E-01	31600	9.9593224514E-01	49200	9.9885163012E-01	66800	9.9952735748E-01
14200	9.6418666950E-01	31800	9.9600402335E-01	49400	9.9886500225E-01	67000	9.9953146194E-01
14400	9.6545887382E-01	32000	9.9607412547E-01	49600	9.9887816770E-01	67200	9.9953551904E-01
14600	9.6667208131E-01	32200	9.9614260005E-01	49800	9.9889113045E-01	67400	9.9953952945E-01
14800	9.6782963745E-01	32400	9.9620949398E-01	50000	9.9890389439E-01	67600	9.9954349383E-01
15000	9.6893466525E-01	32600	9.9627485251E-01	50200	9.9891646331E-01	67800	9.9954741285E-01
15200	9.6999008214E-01	32800	9.9633871939E-01	50400	9.9892884091E-01	68000	9.9955128716E-01
15400	9.7099861536E-01	33000	9.9640113685E-01	50600	9.9894103082E-01	68200	9.9955511739E-01
15600	9.7196281608E-01	33200	9.9646214572E-01	50800	9.9895303659E-01	68400	9.9955890417E-01
15800	9.7288507238E-01	33400	9.9652178543E-01	51000	9.9896486167E-01	68600	9.9956264810E-01
16000	9.7376762113E-01	33600	9.9658009413E-01	51200	9.9897650944E-01	68800	9.9956634980E-01
16200	9.7461255892E-01	33800	9.9663710868E-01	51400	9.9898798323E-01	69000	9.9957000985E-01
16400	9.7542185211E-01	34000	9.9669286474E-01	51600	9.9899928625E-01	69200	9.9957362884E-01
16600	9.7619734610E-01	34200	9.9674739678E-01	51800	9.9901042168E-01	69400	9.9957720735E-01
16800	9.7694077377E-01	34400	9.9680073815E-01	52000	9.9902139261E-01	69600	9.9958074594E-01
17000	9.7765376339E-01	34600	9.9685292114E-01	52200	9.9903220206E-01	69800	9.9958424517E-01
17200	9.7833784581E-01	34800	9.9690397697E-01	52400	9.9904285299E-01	70000	9.9958770557E-01
17400	9.7899446114E-01	35000	9.9695393586E-01	52600	9.9905334829E-01		
17600	9.7962496490E-01	35200	9.9700282708E-01	52800	9.9906369080E-01		



So the blackbody band radiation for a given temperature and wavelength band

$$E_{(\lambda 1\to\lambda 2),\mathrm{b}}(T) = \left[F_{(0\to\lambda 2)}(T) - F_{(0\to\lambda 1)}(T)\right]E_{\mathrm{b}}(T) = \left[F_{(0\to\lambda 2)}(T) - F_{(0\to\lambda 1)}(T)\right]\sigma T^4$$

An excel program calculating this property

Blackbody Radiation Function

T	1.5000000000E+03	K	
λ1	2.0000000000E+00	λ2	4.0000000000E+00
λ1T	3.0000000000E+03	λ2T	6.0000000000E+03
σ	5.6703744192E-08	W/m^2K^4	
ξ1	4.7958966667E+00	ξ2	2.3979483333E+00
Eb	2.8706922510E+05	W/m^2	
F(0-λ1T)	2.7323297650E-01	F(0-λ2T)	7.3779220558E-01
dF	4.6455922908E-01		
E	1.3336065791E+05	W/m^2	

n			
1	2.7242585552E-01		7.1988811110E-01
2	8.0313721143E-04		1.7026615970E-02
3	3.9603714755E-06		8.2359974015E-04
4	2.3238728400E-08		5.0196075715E-05
5	1.4871107285E-10		3.4148965482E-06
6	1.0022356324E-12		2.4752321722E-07
7	6.9909368369E-15		1.8691757440E-08
8	4.9973174841E-17		1.4524205250E-09
9	3.6382617989E-19		1.1525588039E-10
10	2.6867330663E-21		9.2944420529E-12
11	2.0067095438E-23		7.5910716249E-13
12	1.5127637318E-25		6.2639727024E-14
13	1.1492340212E-27		5.2129638574E-15
14	8.7877493416E-30		4.3693355231E-16
15	6.7572738615E-32		3.6845226702E-17
16	5.2211332410E-34		3.1233234276E-18

17	4.0512911780E-36		2.6596722913E-19
18	3.1552982343E-38		2.2739136243E-20
19	2.4656127175E-40		1.9509901703E-21
20	1.9323896195E-42		1.6792081664E-22
21	1.5185207619E-44		1.4493763322E-23
22	1.1961684119E-46		1.2541934649E-24
23	9.4430869380E-49		1.0878050825E-25
24	7.4696924982E-51		9.4547733238E-27
25	5.9195128242E-53		8.2335371692E-28
26	4.6989398519E-55		7.1827126325E-29
27	3.7358381180E-57		6.2761998985E-30
28	2.9744064467E-59		5.4923433385E-31
29	2.3713296432E-61		4.8130846016E-32
30	1.8928787453E-63		4.2232961635E-33
31	1.5127140020E-65		3.7102526405E-34
32	1.2102119866E-67		3.2632082756E-35
33	9.6918604397E-70		2.8730569669E-36
34	7.7690355840E-72		2.5320569863E-37
35	6.2332834938E-74		2.2336067819E-38
36	5.0053460982E-76		1.9720613964E-39
37	4.0225211118E-78		1.7425813923E-40
38	3.2351224666E-80		1.5410079484E-41
39	2.6037181461E-82		1.3637591476E-42
40	2.0969693252E-84		1.2077435122E-43

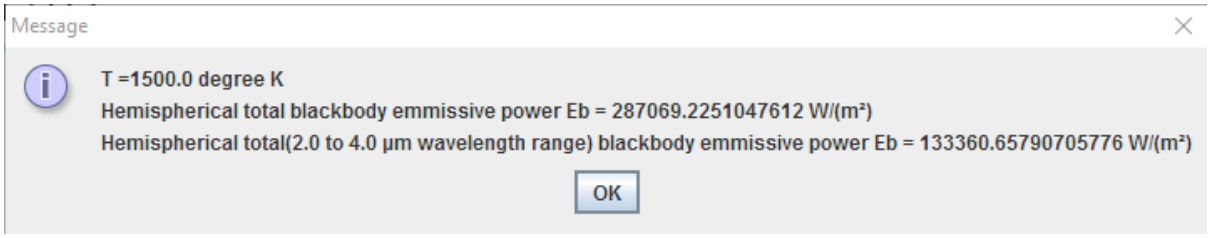
Java program to calculate bandwise blackbody emissive power

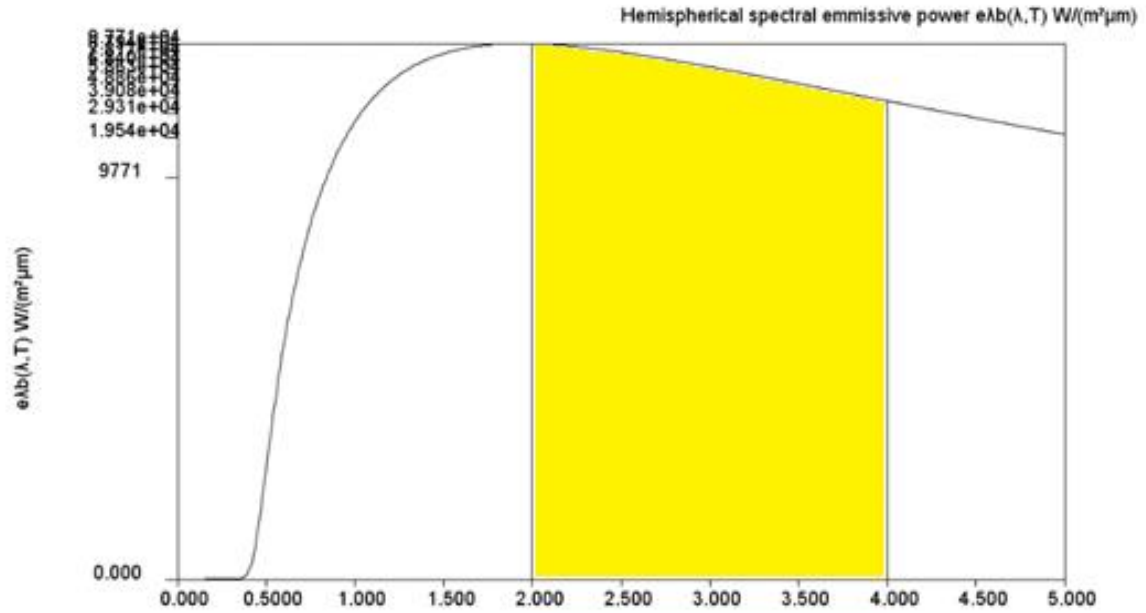
```
 import java.io.*;
import javax.swing.*;

class radiation_ch1_P2
{
            public static double Eb(double lambda,double T)
{   //lambda in micrometers
            double C1=3.741774892e8; //    W/(mikrometer)^4/(m^2sr)
   double C2=14387.69;  //mikrometer K
            double l5=lambda*lambda*lambda*lambda*lambda;
            double e=C1/(Math.exp(C2/(lambda*T))-1.0)/l5;
 return e;
}
  public static double F(double x)
  {
  double C2=14387.69;
  double y=C2/x;
  double total=0;
  for(int n=1;n<80;n++)
  {total+=Math.exp(-n*y)/n*(y*y*y+3.0*y*y/n+6.0*y/(n*n)+6.0/(n*n*n));}
  double pi=Math.PI;
  total=15.0/(pi*pi*pi*pi)*total;
  return total;
  }

  public static void main(String args[])
  {  //Gauss-Legendre integral of any desired point (general)
  double T=Double.parseDouble(JOptionPane.showInputDialog("T degree K="));
  double l1=Double.parseDouble(JOptionPane.showInputDialog("lambda1 mikrometers"));
  double l2=Double.parseDouble(JOptionPane.showInputDialog("lambda2 mikrometers"));
  double PI=Math.PI;
  double C1=3.741774892e8; //    W/(mikrometer)^4/(m^2sr)
  double C2=14387.69;  //mikrometer K
  double sigma=C1/(C2*C2*C2*C2)*PI*PI*PI*PI/15.0;
  double x1=C2/(l1*T);
  double x2=C2/(l2*T);
  double dF=F(T*l2)-F(T*l1);
  double Eb=sigma*T*T*T*T;
   char lambda='\u03BB';
 char sq='\u00B2';
 char mu='\u03BC';
 String ss1="";
  ss1+="T ="+T+" degree K\n";
  ss1+="Hemispherical total blackbody emmissive power Eb = "+Eb+" W/(m"+sq+")\n";
```

```
  ss1+="Hemispherical total("+l1+" to "+l2+" "+mu+"m wavelength range) blackbody emmissive power Eb = "+Eb*dF+"
W/(m"+sq+")\n";
  JOptionPane.showMessageDialog(null,ss1);
  double l[]=Mathd.setX(0.0,0.01,l2+1.0);
  double E[]=new double[l.length];
  double emax=-1.0e40;
  double emin=1.0e40;
  for(int i=0;i<l.length;i++) {E[i]=Eb(l[i],T);if(E[i]<emin) emin=E[i];if(E[i]>emax) emax=E[i];}
  Plot pp=new Plot(l,E);
  double y1[]={emin,Eb(l1,T)};
  double y2[]={emin,Eb(l2,T)};
  double xx1[]={l1,l1};
  double xx2[]={l2,l2};
  pp.addData(xx1,y1);
  pp.addData(xx2,y2);
  pp.setYlogScaleOn();
  pp.setXlabel("wavelength "+mu+"meters");
 String s1="e"+lambda+"b("+lambda+",T) W/(m"+sq+mu+"m)";
 String s="Hemispherical spectral emmissive power e"+lambda+"b("+lambda+",T) W/(m"+sq+mu+"m)";
 pp.setYlabel(s1);
 pp.setPlabel(s);
 pp.plot();
}}
```





PROBLEM: A light bulb filament is 3600 K. Assuming that filament radiates as blackbody, what fraction of it emmission is in the visible rigion (visible region is within λ=0.4 to 0.7 μm)

Black body Radiation function

C2	1.43876900E+04
T	3.60000000E+03
λ1	4.00000000E-01
λ2	7.00000000E-01

λ1T	1440	ξ1
λ2T	2520	ξ2

n

1	6.24055738E-02	1.07447621E+00
2	1.21914357E-06	1.34217121E-03

3	3.53354128E-11	2.70387408E-06	
4	1.18272559E-15	6.42174979E-09	
5	4.26663125E-20	1.65720949E-11	
6	1.61157201E-24	4.49581244E-14	
7	6.27942884E-29	1.26104852E-16	
8	2.50228909E-33	3.62255408E-19	
9	1.01419395E-37	1.05942511E-21	
10	4.16550459E-42	3.14175074E-24	
11	1.72920418E-46	9.42128891E-27	
12	7.24151189E-51	2.85106134E-29	
13	3.05486836E-55	8.69362139E-32	
14	1.29674305E-59	2.66799670E-34	
15	5.53392602E-64	8.23306379E-37	
16	2.37260611E-68	2.55276172E-39	
17	1.02136797E-72	7.94826766E-42	
18	4.41263398E-77	2.48390308E-44	
19	1.91249931E-81	7.78787370E-47	
20	8.31280890E-86	2.44892491E-49	
21	3.62254945E-90	7.72106224E-52	
22	1.58232316E-94	2.44013886E-54	
23	6.92626314E-99	7.72847328E-57	
24	3.03770889E-103	2.45263333E-59	
25	1.33464871E-107	7.79757580E-62	
26	5.87354139E-112	2.48319925E-64	
27	2.58875508E-116	7.92013738E-67	
28	1.14259254E-120	2.52972580E-69	
29	5.04961265E-125	8.09076327E-72	
30	2.23435520E-129	2.59084332E-74	
31	9.89780518E-134	8.30601459E-77	
32	4.38921189E-138	2.66570353E-79	
33	1.94834795E-142	8.56385652E-82	
34	8.65670809E-147	2.75384005E-84	
35	3.84965266E-151	8.86330311E-87	
36	1.71336979E-155	2.85507745E-89	
37	7.63170391E-160	9.20418375E-92	
38	3.40184440E-164	2.96947105E-94	
39	1.51744519E-168	9.58698735E-97	
40	6.77332049E-173	3.09726659E-99	
total	6.24067930E-02	1.07582109E+00	
F(0->λT)	**9.61000544E-03**	**1.65665403E-01**	**0.1560554**

PROBLEM: The sun radiaties as a blackbody at an effective temperature of T=5762 K. What fraction of the total energy is in

a) Ultraviolet (UV) range ($\lambda_1 = 0.01\ \mu m - \ \lambda_2 = 0.4\ \mu m$)
b) Visible light range ($\lambda_2 = 0.4\ \mu m - \ \lambda_3 = 0.7\ \mu m$)
c) Infrared region ($\lambda_3 = 0.7\ \mu m - \ \lambda_4 = 1000\ \mu m$)

Blackbody Radiation Function

T	5.7620000000E+03	K	
λ_1	1.0000000000E-02	λ_2	4.0000000000E-01
λ_1T	5.7620000000E+01	λ_2T	2.3048000000E+03
σ	5.6705032119E-08	W/m^2K^4	
ξ1	2.4969958348E+02	ξ2	6.2424895869E+00
Eb	6.2504979425E+07	W/m^2	
F(0-λ_1T)	8.7462553396E-103	F(0-λ_2T)	1.2098038007E-01
dF	**1.2098038007E-01**		
E	7.5618761670E+06	W/m^2	

n			
1	8.7462553396E-103		1.2088944367E-01
2	1.5668385700E-211		9.0827998461E-05
3	0.0000000000E+00		1.0825095675E-07
4	0.0000000000E+00		1.5147024468E-10

5	0.0000000000E+00		2.2991576623E-13
6	0.0000000000E+00		3.6657750704E-16
7	0.0000000000E+00		6.0402801032E-19
8	0.0000000000E+00		1.0190245244E-21
9	0.0000000000E+00		1.7498583627E-24
10	0.0000000000E+00		3.0465647297E-27
11	0.0000000000E+00		5.3630753370E-30
12	0.0000000000E+00		9.5267391166E-33
13	0.0000000000E+00		1.7050965792E-35
14	0.0000000000E+00		3.0713238596E-38
15	0.0000000000E+00		5.5626164713E-41
16	0.0000000000E+00		1.0122627693E-43
17	0.0000000000E+00		1.8497412561E-46
18	0.0000000000E+00		3.3925048494E-49
19	0.0000000000E+00		6.2423078278E-52
20	0.0000000000E+00		1.1519572446E-54
21	0.0000000000E+00		2.1314127186E-57
22	0.0000000000E+00		3.9530320557E-60
23	0.0000000000E+00		7.3473554469E-63
24	0.0000000000E+00		1.3683217699E-65
25	0.0000000000E+00		2.5528811975E-68
26	0.0000000000E+00		4.7708461223E-71
27	0.0000000000E+00		8.9294998884E-74
28	0.0000000000E+00		1.6736954417E-76
29	0.0000000000E+00		3.1412263401E-79
30	0.0000000000E+00		5.9027720466E-82
31	0.0000000000E+00		1.1104811983E-84
32	0.0000000000E+00		2.0913787710E-87
33	0.0000000000E+00		3.9426783562E-90
34	0.0000000000E+00		7.4397903211E-93
35	0.0000000000E+00		1.4051302906E-95
36	0.0000000000E+00		2.6560544664E-98
37	0.0000000000E+00		5.0246012596E-101
38	0.0000000000E+00		9.5124388813E-104
39	0.0000000000E+00		1.8021489282E-106
40	0.0000000000E+00		3.4165059920E-109

Blackbody Radiation Function

T	5.7620000000E+03	K	
λ_2	4.0000000000E-01	λ_3	7.0000000000E-01
λ1T	2.3048000000E+03	λ2T	4.0334000000E+03
σ	5.6705032119E-08	W/m^2K^4	
ξ1	6.2424895869E+00	ξ2	3.5671369068E+00
Eb	6.2504979425E+07	W/m^2	
F(0-λ_2T)	1.2098038007E-01	F(0-λ_3T)	4.8690088115E-01
dF	**3.6592050108E-01**		
E	2.2871853391E+07	W/m^2	

Blackbody Radiation Function

T	5.7620000000E+03	K	
λ3	7.0000000000E-01	λ4	1.0000000000E+03
λ1T	4.0334000000E+03	λ2T	5.7620000000E+06
σ	5.6705032119E-08	W/m^2K^4	
ξ1	3.5671369068E+00	ξ2	2.4969958348E-03
Eb	6.2504979425E+07	W/m^2	
F(0-λ3T)	4.8690088115E-01	F(0-λ4T)	9.9999536521E-01
dF	**5.1309448407E-01**		
E	3.2070960170E+07	W/m^2	

```
public class F_radiation1B
{ public static double F(double x)
```

```
  {double C2=1.438769e4;
  double y=C2/x;
  double total=0;
  for(int n=1;n<80;n++)
  {total+=Math.exp(-n*y)/n*(y*y*y+3.0*y*y/n+6.0*y/(n*n)+6.0/(n*n*n));}
  double pi=Math.PI;
  total=15.0/(pi*pi*pi*pi)*total;
  return total;
  }

  public static void main(String arg[])
  {
  double lambda1=0.01;
  double lambda2=0.4;
  double lambda3=0.7;
  double lambda4=1000;
  double n=1.0; //refractive index for space(vacuum)
  double T=5762.0; //degree K
  double x1=n*lambda1*T;
  double x2=n*lambda2*T;
  double x3=n*lambda3*T;
  double x4=lambda4*T;
  double F1=F(x1);
  double F2=F(x2);
  double F3=F(x3);
  double F4=F(x4);
  double dF1=(F2-F1);
  double dF2=(F3-F2);
  double dF3=(F4-F3);
  System.out.println("Ultraviolet percentage dF1="+dF1);
  System.out.println("Visible light percentage dF2="+dF2);
  System.out.println("Infrared percentage dF3="+dF3);
  }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" F_radiation1B
Ultraviolet percentage dF1=0.12098038006833149
Visible light percentage dF2=0.36592050107794827
Infrared percentage dF3=0.5130985058495146

> Terminated with exit code 0.
```

PROBLEM: Iron at 1200 C. If it is assumed that electromagnetic radiation emmision is assumed to be blackbody radiative emmision. What fraction of the total energy is in

a) Ultraviolet (UV) range ($\lambda_1 = 0.01\ \mu m - \ \lambda_2 = 0.4\ \mu m$)
b) Visible light range ($\lambda_2 = 0.4\ \mu m - \ \lambda_3 = 0.7\ \mu m$)
c) Infrared region ($\lambda_3 = 0.7\ \mu m - \ \lambda_4 = 1000\ \mu m$)

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" F_radiation1B
Ultraviolet percentage dF1=4.407445556522024E-10
Visible light percentage dF2=3.372543669769459E-5
Infrared percentage dF3=0.999965617221291
> Terminated with exit code 0.
```

If blackbody radiation function is given and nλT is required numerical root finding methods can be used to achieve this.

PROBLEM : In air we have a blackbody source of T=500 K. Which wavelength (μm) would be transfer 10% of the total blackbody radiation ($F_{0\rightarrow n\lambda T} = 0.1\ \ \lambda = ?$)

Function interface **if_x** to define lambda variable functions

```
import static java.lang.Math.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
  // single function single independent variable
  // example f=x*x
```

```
  // includes full set of derivatives
  // Reference :"Generation of Finite Difference Formulas on Arbitrary Spaced Grids",
  // Bength Fornberg, Mathematics of Computation, Volume 51, Number 184, October 1988
  // pages 699-706
  //interface version
@FunctionalInterface
interface if_x
{  public double func(double x);
//first order derivative
default double dfunc(double x)
{double h=1.0e-3;
 int n=1;
 int M=10;
 return dfunc(x,n,M,h);
}
//second order derivative
default double dfunc2(double x)
{double h=1.0e-3;
 int n=2;
 int M=10;
 return dfunc(x,n,M,h);
}
default double dfunc(double x,int N,int Mi,double hi)
{// order of the maximum derivative
 // N order of derivative
 // M degree of difference formula
double c[][][];
double alpha[];
double h;
int M;
double a[]=new double[0];
h=0.01;
double x0=0;
M=20;
double alphai[]={0,1,-1,2,-2,3,-3,4,-4,5,-5,6,-6,7,-7,8,-8,9,-9,10,-10,11,-11,12,-12,13,-13,14,-14,15,
-15,16,-16,17,-17,18,-18,19,-19,20,-20,21,-21,22,-22,23,-23,24,-24,25,-25,26,-26,27,-27,28,-28,29,-29,30,-30,
31,-31,32,-32,33,-33,34,-34,35,-35,36,-36,37,-37,38,-38,39,-39,40,-40,41,-41,42,-42,43,-43,44,-44,45,-45,46,
-46,47,-47,48,-48,49,-49,50,-50,51,-51,52,-52,53,-53,54,-54,55,-55,56,-56,57,-57,58,-58,59,-59,60,-60,
-61,61,-62,62,-63,63,-64,64,-65,65,-66,66,-67,67,-68,68,-70,70,-71,71,-72,72,-73,73,-74,74,-75,75,
-76,76,-77,77,-78,78,-79,79,-80,80,-81,81,-82,82,-83,83,-84,84,-85,85,-86,86,-87,87,
-88,88,-89,89,-90,90,-91,91,-92,92,-93,93,-94,94,-95,95,-96,96,-97,97,-98,98,-99,99,-100,100};
alpha=alphai;
int N1=alpha.length-1;
// M   degree of highest derivative
// N+1 number of coefficients
double delta[][][]=new double[N1+1][N1+1][M+1];
double c1,c2,c3;
delta[0][0][0]=1.0;
c1=1.0;
for(int n=1;n<=N1;n++)
{  c2=1;
   for(int nu=0;nu<=(n-1);nu++)
   {c3=alpha[n]-alpha[nu];
    c2=c2*c3;
    if(n<=M) delta[n-1][nu][n]=0.0;
    for(int m=0;m<=Math.min(n,M);m++)
    {
       if(m==0)
       { delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m])/c3;}
       else
       {delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m]-m*delta[n-1][nu][m-1])/c3;}
    }//next m
   }//next nu
   for(int m=0;m<=Math.min(n,M);m++)
   {   if(m==0)
                {delta[n][n][m]=c1/c2*(-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
                else
                {delta[n][n][m]=c1/c2*(m*delta[n-1][n-1][m-1]-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
   }//next m
   c1=c2;
}//next n
c=delta;
  if(Mi<N) M=N;
  else M=Mi;
  h=hi;
  double deriv=0;
  double h1=1/h;
```

```
  double h2=1;
  for(int j=0;j<N;j++)
  {h2*=h1;}
  for(int i=0;i<c[0].length;i++)
  {   deriv+=c[M][i][N]*func(x+alpha[i]*h);}
  return deriv*h2;}
default double dfunc(double x,int N)
{int M=30;double h=0.05*N;return dfunc(x,N,M,h);}
}
```

```
public class F_radiation1
{ public static double F(double x)
  {double C2=1.438769e4;
  double y=C2/x;
  double total=0;
  for(int n=1;n<80;n++)
  {total+=Math.exp(-n*y)/n*(y*y*y+3.0*y*y/n+6.0*y/(n*n)+6.0/(n*n*n));}
  double pi=Math.PI;
  total=15.0/(pi*pi*pi*pi)*total;
  return total;
  }
  public static double bisection(if_x f,double a,double b)
  {double b1=2.0*b;
   double r=(a+b)/2.0;
   double eps=1.0e-8;
   int nmax=100;
   int i=0;
   while(Math.abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;
    i++;
   }
   if(i>=100) r=bisection(f,a,b1);
   return r;
  }
  public static void main(String arg[])
  {
  double F1=0.1;
  double n=1.00027316; //refractive index for air
  double T=500.0; //degree K
  if_x f1=x->F(x)-F1;
  double a=0.0;
  double b=250.0;
  double x=bisection(f1,a,b);
  System.out.println("x="+x);
  double lambda=x/(n*T);
  System.out.println("lambda="+lambda+" micrometers");
  }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" F_radiation1
x=2195.176601409912
lambda=4.389154261441768 micrometers

> Terminated with exit code 0.
```

19 EMMISIVITY

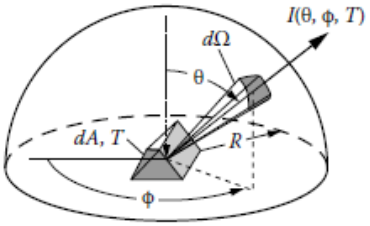
In real surfacess emissive Power values are less than the black surface emissive power values. The ratio of actual emissive power to black body emissive power is called emissivity. Emissivity specifies radiation from a real surface as compared with emission from a blackbody at the same temperature. It can be spectral, and in general depends on surface temperature and direction. In detailed radiative exchange calculations, data for spectral directional emissivity at the correct surface temperature is needed. Such detailed data for particular materials is scarce. Most available data are for a limited spectral range, and are mostly for the normal direction. Because of this, averaged emissivity values are often used. Averages may be over direction, wavelength, or both. Averages for both direction and wavelength is called total emissivity.

Unlike the intensity from a blackbody, the intensity emitted from a real body *does* depend on direction. The energy leaving a real surface dA of temperature T per unit time in the wavelength interval $d\lambda$ and within the solid angle $d\Omega = \sin\theta d\theta d\varphi$ is then given by

Directional spectral emissivity $\varepsilon_\lambda(\theta,\phi,T)$

$$\varepsilon_\lambda(\theta,\phi,T) = \frac{I_\lambda(\theta,\phi,T)}{I_{\lambda b}(T)} = \frac{E_\lambda(\theta,\phi,T)}{E_{\lambda,b}(T)}$$

Directional total emissivity $\varepsilon(\theta,\phi,T)$:



$$\varepsilon(\theta,\phi,T) = \frac{I(\theta,\phi,T)}{I_b(T)} = \frac{\pi\int_{\lambda=0}^{\infty} I_\lambda(\theta,\phi,T)d\lambda}{E_b(T)} = \frac{\pi\int_{\lambda=0}^{\infty} I_\lambda(\theta,\phi,T)d\lambda}{\sigma n^2 T^4}$$

Directional total emissivity in terms of directional spectral emissivity:

$$\varepsilon(\theta,\phi,T) = \frac{\pi\int_{\lambda=0}^{\infty} \varepsilon_\lambda(\theta,\phi,T) I_{\lambda b}(T)d\lambda}{\sigma n^2 T^4} = \frac{\int_{\lambda=0}^{\infty} \varepsilon_\lambda(\theta,\phi,T) E_{\lambda b}(T)d\lambda}{\sigma n^2 T^4}$$

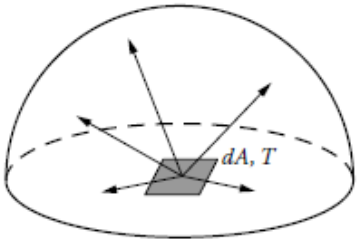
Hemispherical spectral emissivity $\varepsilon_\lambda(T)$

hemispherical spectral emissive power found by integrating the spectral energy per unit solid angle over all solid angles.

$E_\lambda(\mathrm{T}) = \int_{\phi=0}^{2\pi}\int_{\theta=0}^{\pi/2} I_\lambda(\theta,\phi,T)\sin(\theta)\cos(\theta)d\theta d\phi =$

$\int_\Omega I_\lambda(\theta,\phi,T)\cos(\theta)d\Omega = I_{\lambda b}(T)\int_\Omega \varepsilon_\lambda(\theta,\phi,T)\cos(\theta)d\Omega$

Hemispherical spectral emissivity (in terms of directional spectral emissivity)



$$\varepsilon_\lambda(T) = \frac{E_\lambda(T)}{E_{\lambda b}(T)} = \frac{1}{\pi}\int_\Omega \varepsilon_\lambda(\theta,\phi,T)\cos(\theta)d\Omega$$

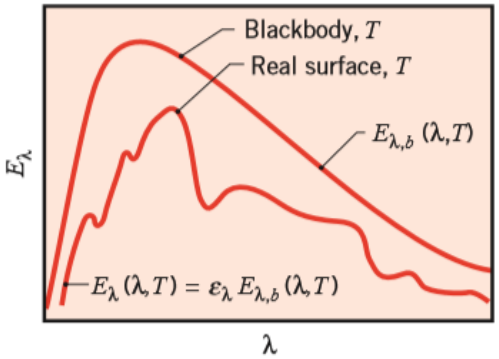
Hemispherical total emissivity $\varepsilon(T)$

$$\varepsilon(T) = \frac{E(T)}{E_b(T)} = \frac{\int_{\lambda=0}^{\infty}\int_\Omega[\varepsilon_\lambda(\theta,\phi,T)I_{\lambda b}(T)d\lambda]\cos(\theta)d\Omega}{\sigma n^2 T^4}$$

Hemispherical total emissivity (in terms of directional total emissivity)

$$\varepsilon(T) = \frac{1}{\pi}\int_\Omega \varepsilon_\lambda(\theta,\phi,T)\cos(\theta)d\Omega = \frac{1}{\pi}\int_{\phi=0}^{2\pi}\int_{\theta=0}^{\pi/2}\varepsilon_\lambda(\theta,\phi,T)\sin(\theta)\cos(\theta)d\theta d\phi$$

Hemispherical total emissivity (in terms of hemispherical spectral emissivity)



$$\varepsilon(T) = \frac{\pi \int_{\lambda=0}^{\infty} \varepsilon_\lambda(T) I_{\lambda b}(T) d\lambda}{\sigma n^2 T^4} = \frac{\int_{\lambda=0}^{\infty} \varepsilon_\lambda(T) E_{\lambda b}(T) d\lambda}{\sigma n^2 T^4}$$

If geometry dependence is uniform in everywhere, this type of surface is called grey surface.

Total emmisivities of some substances are given in the table below:

Alloys		
20Ni-25Cr-55Fe, oxidized	200	0.90
	500	0.97
60Ni-12Cr-28Fe, oxidized	270	0.89
	560	0.82
80Ni-20Cr, oxidized	100	0.87
	600	0.87
	1300	0.89
Aluminum		
Polished	100	0.095
Highly Polished	50-500	0.04-0.06
Unoxidized	25	0.022
	100	0.028
	500	0.060
Oxidized	200	0.11
	600	0.19
Commercial Sheet	100	0.090
Anodized Sheet, Chromic Acid Proc	100	0.55
Heavily Oxidized	93-504	0.2-0.31
Aluminum Oxide	500-827	0.42-0.26
Asbestos		
Board	20	0.96
Cement	0-200	0.96
Cloth	93	0.90
Paper	0-100	0.95
Asphalt	Ambient	0.90-0.98
Oil, on polished metal		
.001” Thick	Ambient	0.27
.002” Thick	Ambient	0.46
.005” Thick	Ambient	0.72
Bismuth, Unoxidized	25	0.048
	100	0.061
Brass		
Polished	200	0.03
Unoxidized	25	0.035
	100	0.035
Oxidized	200	0.61
	600	0.59
Rolled Sheet	20	0.06
Brick		
Building	1000	0.450
Red, rough, no gross irregularities	20	0.930
Grog, Brick, glazed	1100	0.750
Silica Brick	1000	0.80
	1100	0.85
Fire Brick	1000	0.750
Plate, heated long time, covered		
with thick oxide layer	25	0.78
Plate, heated at 600°C	200-600	0.570
Cuprous Oxide	800-1100	0.66-0.54
Polished	50-100	0.02-.05
Oxidized	50	0.6-0.7
	200	0.60

.....500.....0.88
Unoxidized.....100.....0.02
..... Liquid.....0.15
Dow Metal .....232-400.....0.24-0.20
Enamel, White, fused on Iron.....19.....0.900
Glass
Smooth ..... 0-200.....0.95
.....250-1000.....0.87-0.72
.....1100-1500.....0.70-0.67
Fused Quartz .....320.....0.75
Covex D Glass .....320.....0.76
Nonex Glass .....320.....0.82
Pyrex ..... 0-300.....0.90
Gold
Pure, highly polished .....100.....0.02
Carefully Polished .....200-600.....0.02-0.03
Unoxidized.....100.....0.02
.....500.....0.03
Enamel .....100.....0.37
Graphite ..... 0-3600.....0.70-0.80
Gypsum 0.02” thick on smooth
or blackened plate.....20.....0.93
Human Skin .....36-7-37.2.....0.985
Inconel
Type X 0.550-0.780
Type B .....450-1620.....0.350-0.550
Iron
Cast
Oxidized.....200-600.....0.64-0.78
Strongly Oxidized.....40.....0.95
.....250.....0.95
Unoxidized.....100.....0.21
Polished.....200.....0.210
Newly Turned.....22.....0.440
Turned and Heated .....882-990.....0.600-0.700
Liquid Unoxidized..... --.....0.29
Rusted .....25.....0.65
Wrought, Dull .....100.....0.50
Wrought Iron, dull oxidized .....21-360.....0.940
Wrought, highly polished .....38-250.....0.280
Oxidized.....100.....0.74
.....500.....0.84
Unoxidized.....1200.....0.89
Plate, pickled, then rusted red .....20.....0.610
Plate, completely rusted .....19.....0.690
Smooth oxidized electrolytic iron .....127-527.....0.780-0.820
Iron Oxide .....500-1200.....0.85-0.89
Rough-ingot iron .....927-1116.....0.870-0.950
Cast Plate, oxidized, smooth .....23.....0.8
Cast Plate, oxidized, rough.....23.....0.82
Molten Pure Iron .....1516-1771.....0.420-0.450
Molten Armco Iron.....1521-1689.....0.400-0.410
Lead
Pure (99.96%) Unoxidized.....127-227.....0.057-0.075
Oxidized.....200.....0.63
Oxidized, Gray .....24.....0.280
Magnesium
Magnesium Oxide .....227-826.....0.550-0.200
Magnesium Oxide .....900-1704.....0.200
Magnesite
Refractory Brick .....1000.....0.380
Marble, Light Grey Polished..... 0-100.....0.903
Mercury, Unoxidized..... 0.....0.09
.....25.....0.10
.....100.....0.12
Molybdenum
Polished.....538.....0.05
.....1482.....0.17
Oxidized.....538.....0.82
Unoxidized.....1000.....0.13
.....1500.....0.19
.....2000.....0.24
Filament.....827-2593.....0.096-0.202
Monel Metal, Oxidized .....200.....0.43
.....600.....0.43
Nichrome Wire
Clean .....50.....0.65

.....500-1000.....0.71-0.79
Oxidized.....50-500.....0.95-0.98
Nickel
Polished..... low.....0.12
.....1204.....0.32
Oxidized.....200.....0.37
.....871.....0.85
.....1200.....0.85
Unoxidized.....25.....0.045
.....100.....0.06
.....500.....0.12
.....1000.....0.19
Electroplated, Polished.....23.....0.045
Electroplated, not Polished .....20.....0.110
Wire.....187-1007.....0.096-0.186
Plate, oxidized by heating at 600°C .....200-600.....0370-0.480
Nickel Oxide.....650-1254.....0.590-0.860
Chromnickel.....52-1034.....0.640-0.760
Nickel-Silver Polished.....100.....0.135
Oak, Planed .....21.....0.900
Oil Layers on Aluminum Foil
(Linseed Oil)
Aluminum Foil .....100.....0.087
+1, 2 coats oil.....100.....0.561-0.574
Paint, Lacquers, Varnishes
Alum. Paint ..... 0-100.....0.55
Bronze Paint ..... 0-100.....0.80
Black Glass Paint ..... 0-100.....0.90
White Lacquer..... 0-100.....0.95
Green Paint..... 0-100.....0.95
Gray Paint..... 0-100.....0.95
Lamp Black..... 0-100.....0.95
Gold Enamel ..... 0-100.....0.37
Snow White Enamel varnish on
rough iron plate .....23.....0.906
Black Shiny Lacquer, sprayed
on iron.....24.....0.875
Black Shiny shellac on
tinned iron sheet.....21.....0.821
Black Matte shellac .....77-146.....0.910
Black on White Lacquer.....38-93.....0.800-0.950
Flat Black Lacquer .....38-93.....0.960-0.980
Oil Paints, 16 diff. (all colors) .....100.....0.920-0.960
rough or smooth surface.....100.....0.520
Other A1 paints, varying age and
Al content.....100.....0.270-0.670
A1 Lacquer, Varnish binder on
rough plate.....21.....0.390
A1 Paint after heating to 326°C .....150-316.....0.350
Radiator Paint: .....100.....0.790, 0.770, 0.840
White, Cream, Bleach
Radiator Paint, bronze.....100.....0.510
Lacquer coatings, 0.001-0.015”
thick on Alum. alloys.....38-150.....0.870-0.970
3M Nextel101-C10 ..... 0-300.....98
Mikron High Temp Test Paint
(Spirex SP102).....Ambient-650.....0.999
Clear Silicone Vehicle Coating
0.001-0.150” thick:
On mild steels .....260.....0.660
On stainless steels 316, 301, 347.....260.....0.680, 0.750, 0.750
On Dow Metal .....260.....0.740
On Al Alloys, 24ST, 75ST.....260.....0.770, 0.820
Aluminum Paint with silicone
vehicle paint on Inconel.....260.....0.290
Dull Black Varnish .....40-100.....0.80-0.95
Glossy Black Varnish sprayed on
iron .....20.....0.87
.....40.....0.96-0.98
Paper, Any Color..... 0-100.....0.94
Thinipasted on Tinned or
Blackened Plate .....19.....0.920-0.940
Plaster..... 0-200.....0.91
Plastics, Opaque any color .....25.....0.950
Platinum
Cleaned Polished .....200-600.....0.05-0.10
Filament.....27-1227.....0.036-0.192

Unoxidized.....25.....0.037
.....100.....0.047
.....500.....0.096
.....1000.....0.152
.....1500.....0.191
Wire.....50-200.....0.06-0.07
.....500-1000.....0.10-0.16
.....1400.....0.18
Propellant:
Liquid rocket engine .....600-4500.....0.900
Quartz
Rough, fused.....21.....0.930
Glass, 1.98mm Thick.....282-838.....0.900-0.410
Glass, 6.88mm Thick.....300-838.....0.930-0.470
Opaque.....300-838.....0.920-0.680
Roofing Paper .....21.....0.910
Silica (98 Si O2, Fe-free) effect of
grain size, microns
10 microns .....1010-1566.....0.420-0.330
70-600 microns .....1010-1566.....0.620-0.460
Silver
Polished.....100.....0.052
Cleaned Polished .....200-600.....0.02-0.03
Unoxidized.....100.....0.02
.....500.....0.035
Stainless Steel 18-8
Buffed .....20.....0.160
Polished.....93.....0.16
.....371.....0.19
Oxidized.....93-371.....0.83
Stainless Steel 303.....316.....0.74
Oxidized.....1093.....0.87
Stainless Steel 304 (8Cri 18Ni)
light silvery, rough brown,
after heating.....216-490.....0.440-0.360
After 42 hours of heating at 527°C.....216-527.....0.620-0.730
Stainless Steel 310 (25Cr, 20Ni)
Brown, splotched, oxidized from
furnace service.....216-527.....0.900-0.970
Stainless Steel
Allegheny metal No. 4, polished .....100.....0.130
Allegheny metal No. 66, polished .....100.....0.110
Steel
Alloyed (8%Ni, 18%Cr).....500.....0.35
Aluminized .....50-500.....0.79
Table of Emissivity of Various Surfaces For Infrared Thermometry Page 9
Dull Nickel Plated .....20.....0.11
Flat, Rough Surface .....50.....0.95-0.98
Cast, Polished.....750-1050.....0.52-0.56
Calorized, Oxidized .....200.....0.52
.....600.....0.57
Sheet Steel, Ground.....938-1100.....0.550-0.610
Sheet Steel, Rolled .....21.....0.660
Sheet Steel, Strong, Rough Oxide
Layer.....24.....0.800
Sheet with Shiny layer of oxide.....20.....0.82
Oxidized.....25.....0.80
.....200.....0.79
.....600.....0.79
Unoxidized.....100.....0.08
Molten Steel.....1500-1650.....0.420-0.530
.....1520-1650.....0.430-0.40
Molten Mild Steel .....1600-1800.....0.280
Molten Steel, various with
0.25-1.2% (slightly oxidized
surfaces.) .....1560-1710.....0.270-0.390
Molten Steel, unoxidized ..... Liquid.....0.280
Steel Plate, Rough .....40.....0.94
.....400.....0.97
.....600.....0.57
Tantalum
Unoxidized.....1500.....0.21
.....2000.....0.26
Filament.....1327-3000.....0.190-0.310
Thorium Oxide .....277-500.....0.580-0.360
Tin
Unoxidized.....25.....0.05

Commercial tin-plated sheet iron .....100.....0.070-0.080
Tungsten
Filament, aged .....27-3316.....0.320-0.350
Filament.....3316.....0.390
Unoxidized.....25.....0.024
.....100.....0.032
.....500.....0.071
.....1000.....0.15
.....1500.....0.23
.....2000.....0.28
Turbojet Engine Operating.....350-600.....0.900
Water .....Ambient.....0.96
Wood
Spruce, sanded.....93.....0.82
Oak, planed ..... 0-200.....0.89

Some of the spectral emissivity values

Iron

T=1078	K	T=1122	K	T=1321	K	T=1418	K
λ	ε	λ	ε	λ	ε	λ	λ
1.2	0.31	0.99	0.324	1.15	0.26	0.65	0.4705714
1.35	0.285	1.18	0.264	1.6	0.24	1	0.7427143
1.5	0.27	1.41	0.264	2	0.22	1.1	0.8648571
2	0.235	1.57	0.244	2.9	0.29	1.2	1.0684286
2.5	0.22	1.77	0.24	4	0.18	1.3	1.2514286
3	0.205	1.99	0.226	5	0.155	1.4	1.4408571
3.5	0.19	2.16	0.218	6.2	0.15	1.5	1.6968571
4	0.18	2.55	0.205	7	0.14	1.75	1.9207143
5	0.155	2.76	0.194	8	0.135	2	2.0634286
6	0.14	2.96	0.188	9	0.13	2.25	2.2054286
7	0.13	3.45	0.175	9.6	0.125	2.5	2.2314286
8	0.125	3.97	0.169	11	0.12	2.75	2.4862857
9.6	0.115	4.48	0.166	12	0.12	3	2.4587143
12	0.115	4.92	0.164	13	0.115	3.25	2.1991429
14	0.11	5.42	0.147				
		5.97	0.134				
		6.38	0.135				
		6.87	0.134				
		7.41	0.135				
		7.91	0.126				
		8.45	0.123				
		8.89	0.122				
		9.51	0.117				
		9.84	0.117				
		10.20	0.115				
		10.90	0.115				
		11.30	0.114				
		11.80	0.112				
		12.40	0.105				
		13.40	0.107				

99.6 Al, 0.11 Fe, 0.11 Si, 001 Cu, 0.11 Mg cylindrical tube heated for 15 hours polished with Carnu and Selvyt cloth; surface roughness 3 μm

Aluminium

T=599	K	T=697	K	T=805	
λ	ε	λ	ε	λ	ε
1.8	0.11	1.7	0.1	1.7	0.13
2	0.1	2	0.09	2	0.11
2.5	0.086	2.5	0.075	2.5	0.094
3	0.076	3	0.067	3	0.083
3.5	0.068	3.5	0.061	3.5	0.075
4	0.062	4	0.057	4	0.068
4.5	0.057	4.5	0.053	4.5	0.065
5	0.053	5	0.05	5	0.061

5.5	0.05	5.5	0.048	5.5	0.058
6	0.048	6	0.046	6	0.056
6.5	0.046	6.5	0.044	6.5	0.054
7	0.044	7	0.043	7	0.052
7.5	0.042	7.5	0.042	7.5	0.051
8	0.041	8	0.041	8	0.05
8.5	0.039	8.5	0.04	8.5	0.049
9	0.038	9	0.039	9	0.048
9.5	0.037	9.5	0.038	9.5	0.047
10	0.04	10	0.039	10	0.049
10.5	0.054	10.6	0.041	10.5	0.057
10.7	0.056	11	0.04	10.9	0.059
11	0.054	11.5	0.038	11.3	0.055
11.5	0.044	12	0.037	11.5	0.051
12	0.039	12.5	0.036	12	0.049
12.5	0.036	13	0.035	12.5	0.047
		13.5	0.035	13	0.045
		14	0.034	13.5	0.043
				14	0.042

99.6 Al, 0.11 Fe, 0.11 Si, 001 Cu, 0.11 Mg cylindtrical tube heated for 25 hours roughened andknurled with grade 180 slicon carbide paper; surface roughness 115 μm

Aliminium							
T=462		T=599			T=715		T=803
λ	ε	λ	ε	λ	ε	λ	ε
1.8	0.25	2	0.284	1.25	0.33	3.5	0.284
2	0.26	2.5	0.28	1.5	0.34	4	0.282
2.5	0.264	3	0.277	2	0.33	4.5	0.277
3	0.261	3.5	0.273	2.5	0.319	5	0.268
3.5	0.258	4	0.271	3	0.31	5.5	0.274
4	0.254	4.5	0.27	3.5	0.301	6	0.272
4.5	0.251	5	0.27	4	0.292	6.2	0.272
5	0.247	5.5	0.266	4.5	0.28	6.7	0.271
5.5	0.244	6	0.266	5.1	0.274	7.4	0.272
6	0.241	6.5	0.265	5.6	0.28	7.6	0.272
6.5	0.238	7	0.264	5.9	0.281	8	0.263
7	0.235	7.5	0.263	6.5	0.276	8.5	0.268
7.5	0.233	8	0.262	7	0.273	9	0.27
8	0.233	8.5	0.266	7.5	0.271	9.4	0.27
8.5	0.238	9	0.268	8	0.27	9.6	0.272
9	0.239	9.4	0.267	8.5	0.272	10	0.282
9.5	0.237	10	0.27	8.8	0.272	10.5	0.345
10	0.246	10.6	0.288	9.4	0.272	11	0.384
10.5	0.256	11	0.28	10	0.279	12	0.343
11	0.248	12	0.269	10.5	0.312	12.6	0.337
12	0.234	12.6	0.266	11	0.328	13	0.339
12.5	0.233	13.1	0.273	12	0.308	13.4	0.339
13	0.233	13.5	0.271	12.5	0.294	14	0.333
13.5	0.233	14	0.269	13	0.31	14.5	0.328
14	0.233			13.3	0.312		
14.5	0.231			13.6	0.31		
				14	0.304		
				14.5	0.301		

99.9 copper steel-shot blasted preheted roughness 348 microinches

Copper							
T=652	K	T=757	K	T=867	K	T=913	K
λ	ε	λ	ε	λ	ε	λ	ε

		1.5	0.262	1.5	0.28	1.5	0.227
2	0.228	2	0.214	2	0.229	2	0.231
4	0.145	4	0.146	4	0.154	4	0.16
6	0.102	6	0.111	6	0.119	6	0.124
8	0.088	8	0.095	8	0.106	8	0.111
10	0.08	10	0.097	10	0.108	10	0.109

99.9 Cu precleaned, electropolished, cold water rinsed, alcohol rinsed, surface roughness 5.2 mikroinches

Copper							
T=755	K	T=865	K	T=915	K	T=1035	K
λ	ε	λ	ε	λ	ε	λ	ε
		1.5	0.024	1.5	0.026	1.5	0.031
2	0.019	2	0.022	2	0.026	2	0.029
4	0.018	4	0.022	4	0.024	4	0.025
6	0.018	6	0.02	6	0.023	6	0.023
8	0.013	8	0.016	8	0.023	8	0.023
10	0.012	10	0.017	10	0.021	10	0.021

99.9 Cu Glass-shot blasted surface roughness 76 microinches

Copper							
T=636	K	T=763	K	T=873	K	T=907	K
λ	ε	λ	ε	λ	ε	λ	ε
		1.5	0.07	1.5	0.104	1.5	0.104
2	0.063	2	0.063	2	0.076	2	0.081
4	0.039	4	0.059	4	0.053	4	0.053
6	0.031	6	0.06	6	0.047	6	0.042
8	0.026	8	0.06	8	0.04	8	0.037
		10	0.063	10	0.047	10	0.051

For a grey surface

$$\varepsilon_\lambda(\lambda,T)=\frac{E_\lambda(\lambda,T)}{E_{\lambda,\mathrm{b}}(\lambda,T)}$$

$$\varepsilon\,(T)=\frac{\int_{\lambda=0}^{\infty}E_\lambda(\lambda,T)d\lambda}{E_\mathrm{b}(T)}=\frac{\int_{\lambda=0}^{\infty}\varepsilon_\lambda(\lambda,T)E_{\lambda,\mathrm{b}}(\lambda,T)d\lambda}{E_\mathrm{b}(T)}$$

$$\varepsilon\,(T)=\frac{\int_{\lambda=0}^{\infty}E_\lambda(\lambda,T)d\lambda}{E_\beta(T)}=\frac{\left[\left(\frac{C_1}{C_2^4}\right)n^2T^4\int_{\xi=0}\frac{\varepsilon(\xi)\xi^3}{[e^\xi-1]}d\xi\right]}{\left[\left(\frac{C_1}{C_2^4}\right)\frac{\pi^4}{15}\right]n^2T^4}=\frac{15}{\pi^4}\int_{\xi=0}^{\infty}\frac{\varepsilon(\xi)\xi^3}{[e^\xi-1]}d\xi$$

An analytical solution is not available integral can be solved by using numerical integration methods such as Gauss-Legendre method or numerical finite difference method.

In finite difference methods, spectral emissivity values are assumed to be constant for a finite difference band of spectrum. Therefore integration can be converted to a summation process as a function of black body radiation function.

$$\varepsilon\,(T)=\sum_{i=0}^{\infty}\left(F_{(0\to\lambda\mathrm{i})}(T)-F_{(0\to\lambda\mathrm{i}+1)}(T)\right)\varepsilon_{\lambda\mathrm{i}}(\lambda\mathrm{i},T)$$

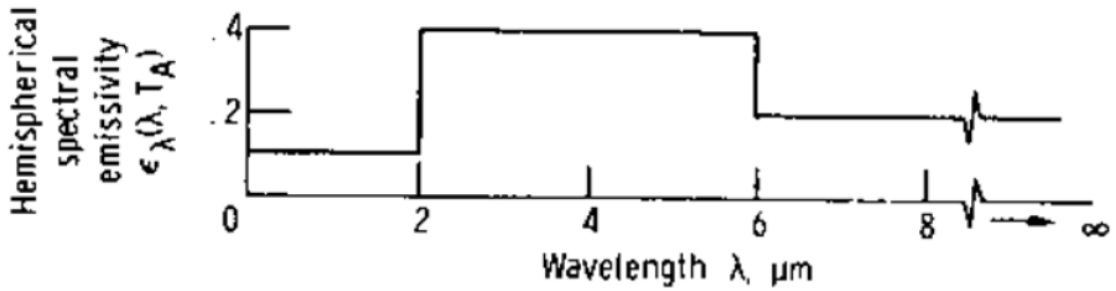
It should also be note that spectrumwise available numerical datas are limited . Therefore normalizing the data will also be applied into the equations.

$$\varepsilon\,(T) = \frac{15}{\pi^4\left(F_{(0\to\lambda 2)}(T) - F_{(0\to\lambda 1)}(T)\right)} \int_{\xi=\xi 1}^{\xi 2} \frac{\varepsilon(\xi)\xi^3}{[e^{\xi}-1]}\, d\xi \quad \text{where } \xi = \frac{C_2}{n\lambda T}$$

$$\varepsilon\,(T) = \frac{\sum_{i=0}^{\infty}\left(F_{(0\to\lambda i)}(T) - F_{(0\to\lambda i+1)}(T)\right)\varepsilon_{\lambda i}(\lambda i, T)}{\left(F_{(0\to\lambda 2)}(T) - F_{(0\to\lambda 1)}(T)\right)}$$

PROBLEM

Surface temperature of a substance is T=1000 K, and spectral emmisivity can be approximated as the figure below. Calculate the total emissivity coefficient.



c.txt data for spectral emissivity

```
0 0.1
1 0.1
2 0.1
2.001 0.4
3 0.4
4 0.4
5 0.4
6 0.4
6.001 0.2
7 0.2
8 0.2
9 0.2
10 0.2
50 0.2
100 0.2
```

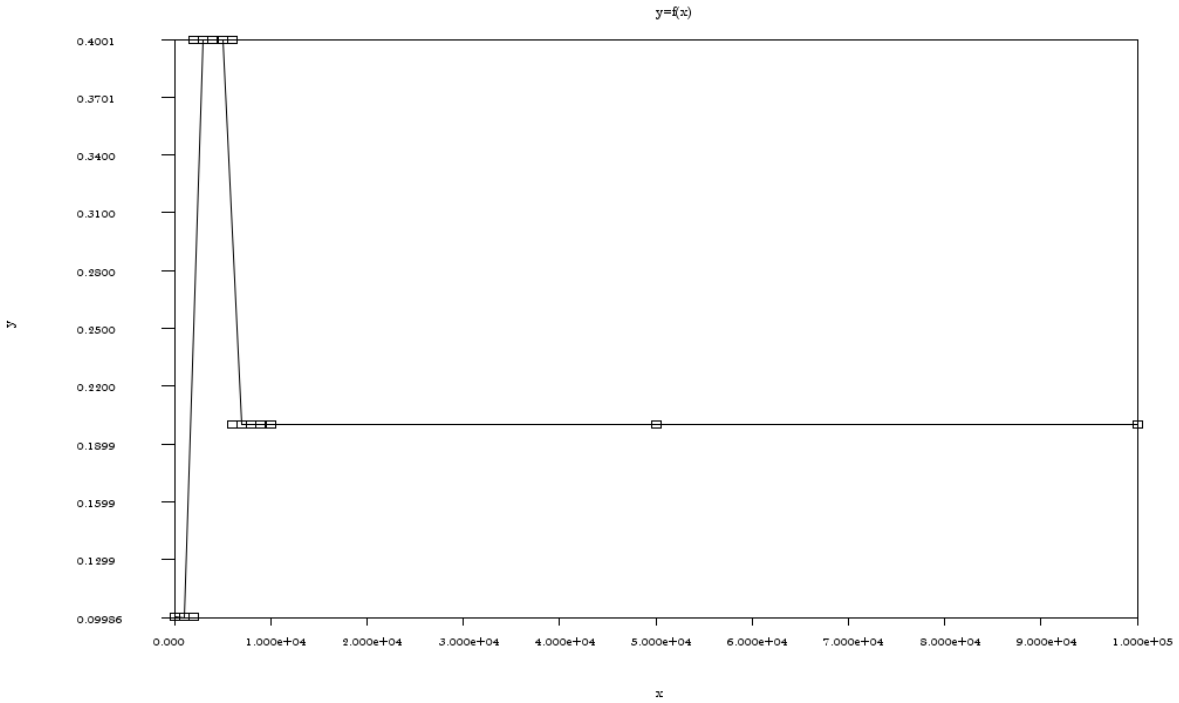
```
//=====================================================
// Radiation heat transfer
// total emmissivity (wavelength based emmisisvity is given as
// data- partial continuous pulynomial curve fittin ids applied)
// Dr. Turhan Coban
// =====================================================
import java.io.*;
import javax.swing.*;
class fh extends f_xr
{   double func(double x,int n)
            { double xx=1.0;
     if(n==0) xx=1.0;
     else if(n==1) xx=x;
     else if(n==2) xx=x*x;
     else if(n==3) xx=x*x*x;
     else if(n==4) xx=x*x*x*x;
     else if(n==5) xx=x*x*x*x*x;
     else if(n==6) xx=x*x*x*x*x*x;
     else if(n==7) xx=x*x*x*x*x*x*x;
     else if(n==8) xx=x*x*x*x*x*x*x*x;
     else if(n==9) xx=x*x*x*x*x*x*x*x*x;
     else if(n==10) xx=x*x*x*x*x*x*x*x*x*x;
     else
     {          for(int i=1;i<=n;i++) xx*=x;}
     return xx;
            }
}
```

```
class f1 extends f_x
{ public double C1=3.741774892e8; //    W/(mikrometer)^4/(m^2sr)
   public double C2=14387.69;  //mikrometer K
  public double sigma=C1/(C2*C2*C2*C2)*PI*PI*PI*PI/15.0;
  double a[][]; //coefficients of partial continuous curve fitting
  double T;
  double xmin,xmax;
  double ymin,ymax;
  static char lambda='\u03BB';
  static char  epsilon='\u03B5';
  static char  ksi='\u03BE';
  static String s1=""+lambda+"*T";
  static String s2=""+epsilon+"_"+lambda+"*T";
  fh f2;
  public f1(double x[][],double Ti)
  {   T=Ti;
      f2=new fh();
      xmin=1.0e200;
      xmax=0;
                for(int i=0;i<x[0].length;i++)
        {x[0][i]=T*x[0][i];
         if(x[0][i]<xmin) {xmin=x[0][i];ymin=x[1][i];}
         if(x[0][i]>xmax) {xmax=x[0][i];ymax=x[1][i];}
        }
        String label[]={s1,s2};
        Text.printT(x,label,"data read");
        a=partial_continuous.EKKgeneral(f2,x[0],x[1],10,6);
        Text.print(a,"a");
        partial_continuous.funcEKKPlot(f2,x[0],x[1],10,6);
  }
  public f1(String name,double Ti)
  {double x[][]=Text.readDoubleT(name);T=Ti;
   T=Ti;
      f2=new fh();
      xmin=1.0e200;
      xmax=0;
                for(int i=0;i<x[0].length;i++)
        {x[0][i]=T*x[0][i];
         if(x[0][i]<xmin) {xmin=x[0][i];ymin=x[1][i];}
         if(x[0][i]>xmax) {xmax=x[0][i];ymax=x[1][i];}
        }
        String label[]={s1,s2};
        Text.printT(x,label,"data read");
        a=partial_continuous.EKKgeneral(f2,x[0],x[1],5,3);
        Text.print(a,"a");
         partial_continuous.funcEKKPlot(f2,x[0],x[1],5,3);
  }
  public f1(double Ti)
  { T=Ti;
    double x[][]=Text.readDoubleT("copper1.txt");T=Ti;
    T=Ti;
      f2=new fh();
      xmin=1.0e200;
      xmax=0;
                for(int i=0;i<x[0].length;i++)
        {x[0][i]=T*x[0][i];
         if(x[0][i]<xmin) xmin=x[0][i];
         if(x[0][i]>xmax) xmax=x[0][i];
        }
        String label[]={s1,s2};
        Text.printT(x,label,"data read");
        a=partial_continuous.EKKgeneral(f2,x[0],x[1],5,3);
        Text.print(a,"a");
         partial_continuous.funcEKKPlot(f2,x[0],x[1],5,3);
  }
  public double func(double x)
  {//x ksi
    return eps(x)*15.0/(PI*PI*PI*PI)*x*x*x/(exp(x)-1.0);}

  public double eps(double x)
  {//x ksi
   //y lambda*T
   //z lambda
   double C2=14387.69;  //mikrometer K
   double eps=0;
   double y=C2/x;
```

```
   double z=y/T;
   if(y<xmin) eps=ymin;
   else if(y>xmax) eps=ymax;
   else eps=partial_continuous.funcEKKgeneral(f2,a,y);
   return eps;
  }
}
class radiation_ch1_P3
{
  public static double F(double x)
  {
  if(x<=0) return 0;
   double C2=14387.69;  //mikrometer K
  double y=C2/x;
  double total=0;
  for(int n=1;n<80;n++)
  {total+=Math.exp(-n*y)/n*(y*y*y+3.0*y*y/n+6.0*y/(n*n)+6.0/(n*n*n));}
  double pi=Math.PI;
  total=15.0/(pi*pi*pi*pi)*total;
  return total;
  }
public static double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{
            //calculates legendre gauss-coefficients as coefficients of the integral
            //for n terms
            double EPS=3.0e-15;
            int m,j,i;
            double z1,z,xm,xl,pp,p3,p2,p1;
    //double x[]=new double[n];
    //double w[]=new double[n];
    double a[][]=new double[2][n];//a[0][i]=x[i]   a[1][i]=w[i]
            m=(n+1)/2;
            xm=0.5*(x2+x1);
            xl=0.5*(x2-x1);
            for (i=1;i<=m;i++)  {
                        z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                        do {
                                    p1=1.0;
                                    p2=0.0;
                                    for (j=1;j<=n;j++) {
                                                p3=p2;
                                                p2=p1;
                                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                                    }
                                    pp=n*(z*p1-p2)/(z*z-1.0);
                                    z1=z;
                                    z=z1-p1/pp;
                        } while (Math.abs(z-z1) > EPS);
                        a[0][i-1]=xm-xl*z;
                        a[0][n-i]=xm+xl*z;
                        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                        a[1][n-i]=a[1][i-1];
                                    }
return a;
}
public static double integral(f_x f_xnt,double x1,double x2,int n)
{
//n : number of integral coefficients
// this routine first generates gauss legendre coefficients
// for [x1,x2] band
// then calculates gauss legendre integral
double a[][]=new double[2][n];
a=gauss_legendre_coefficients(x1,x2,n);
double z=0;
for(int i=0;i<n;i++)
{ z+=a[1][i]*f_xnt.func(a[0][i]);}
return z;
}
public static double total_emmissivity(String name,double T,double lambda1,double lambda2)
{ // integration method
  f1 b1=new f1(name,T);
  double C2=14387.69;  //mikrometer K
  double x1=C2/(lambda1*T);
  double x2=C2/(lambda2*T);
  double dF=F(lambda2*T)-F(lambda1*T);
  double z=integral(b1,x2,x1,200)/dF;
  return z;
```

```
 }
public static double total_emmissivity_FD(String name,double T,double lambda1,double lambda2)
{return total_emmissivity_FD(name,T,lambda1,lambda2,300);}
public static double total_emmissivity_FD(String name,double T,double lambda1,double lambda2,int n)
{ //finite difference addition (euler integration) method
  f1 b1=new f1(name,T);
  double dlambdaT=(lambda2-lambda1)*T/(double)(n-1);
  double z=0;
  double lambdaT=lambda1*T;
  double lambdaT1;
  double y;
  double eps;
  double C2=14387.69;  //mikrometer K
  double x1=C2/(lambda1*T);
  double x2=C2/(lambda2*T);
  for(int i=0;i<n;i++)
  {   y=C2/(lambdaT+dlambdaT/2.0);
              lambdaT1=lambdaT+dlambdaT;
              z+=b1.eps(y)*(F(lambdaT1)-F(lambdaT));
              lambdaT=lambdaT1;
  }
  double dF=F(lambda2*T)-F(lambda1*T);
  z/=dF;
  return z;
}
  public static void main(String args[])
  {  double eps=total_emmissivity("c.txt",1000.0,0.01,100.0);
     System.out.println("eps total  = "+eps);
     double eps1=total_emmissivity_FD("c.txt",1000.0,0.01,100.0);
     System.out.println("eps1 total  = "+eps1);
  }
}
```




```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" radiation_ch1_P3
eps total  = 0.32564263508764707
eps1 total  = 0.3251220240080172
```

Spectral emissivity values can be given as a function as well

```
//=====================================================
// Radiation heat transfer
// total emmissivity (wavelength based emmisisvity is given as a function)
// Dr. Turhan Coban
// =====================================================
import java.io.*;
import javax.swing.*;
```

```
class f3 extends f_x
{   public double C1=3.741774892e8; //    W/(mikrometer)^4/(m^2sr)
    public double C2=14387.69;  //mikrometer K
  public double sigma=C1/(C2*C2*C2*C2)*PI*PI*PI*PI/15.0;
  double T;
  static char lambda='\u03BB';
  static char  epsilon='\u03B5';
  static char  ksi='\u03BE';
  static String s1=""+lambda+"*T";
  static String s2=""+epsilon+"_"+lambda+"*T";
              public f3(double Ti)
  {T=Ti;}
   public double func(double x)
  {//x ksi
    return eps(x)*15.0/(PI*PI*PI*PI)*x*x*x/(exp(x)-1.0);}
  public double eps(double x)
    {//x ksi
     //y lambda*T
     //z lambda
     double eps=0;
       double y=C2/x;
       double z=y/T;
       if(z<2) eps=0.1;
       else if(z>=2 && z<6) eps=0.4;
       else eps=0.2;
       return eps;
}
}
class radiation_ch1_P4
{
  public static double F(double x)
  {
  if(x<=0) return 0;
  double C2=14387.69;  //mikrometer K
  double y=C2/x;
  double total=0;
  for(int n=1;n<80;n++)
  {total+=Math.exp(-n*y)/n*(y*y*y+3.0*y*y/n+6.0*y/(n*n)+6.0/(n*n*n));}
  double pi=Math.PI;
  total=15.0/(pi*pi*pi*pi)*total;
  return total;
  }
public static double total_emmissivity(double T,double lambda1,double lambda2)
{ // integration method
  f3 b1=new f3(T);
  double C2=14387.69;  //mikrometer K
  double x1=C2/(lambda1*T);
  double x2=C2/(lambda2*T);
  double dF=F(lambda2*T)-F(lambda1*T);
  double z=radiation_ch1_P3.integral(b1,x2,x1,200)/dF;
  return z;
}
public static double total_emmissivity_FD(double T,double lambda1,double lambda2)
{return total_emmissivity_FD(T,lambda1,lambda2,300);}
public static double total_emmissivity_FD(double T,double lambda1,double lambda2,int n)
{ //finite difference addition (euler integration) method
  f3 b1=new f3(T);
  double dlambdaT=(lambda2-lambda1)*T/(double)(n-1);
  double z=0;
  double lambdaT=lambda1*T;
  double lambdaT1;
  double y;
  double eps;
  double C2=14387.69;  //mikrometer K
  double x1=C2/(lambda1*T);
  double x2=C2/(lambda2*T);
  for(int i=0;i<n;i++)
  {   y=C2/(lambdaT+dlambdaT/2.0);
              lambdaT1=lambdaT+dlambdaT;
              z+=b1.eps(y)*(F(lambdaT1)-F(lambdaT));
              lambdaT=lambdaT1;
  }
  double dF=F(lambda2*T)-F(lambda1*T);
  z/=dF;
  return z;
}
  public static void main(String args[])
```

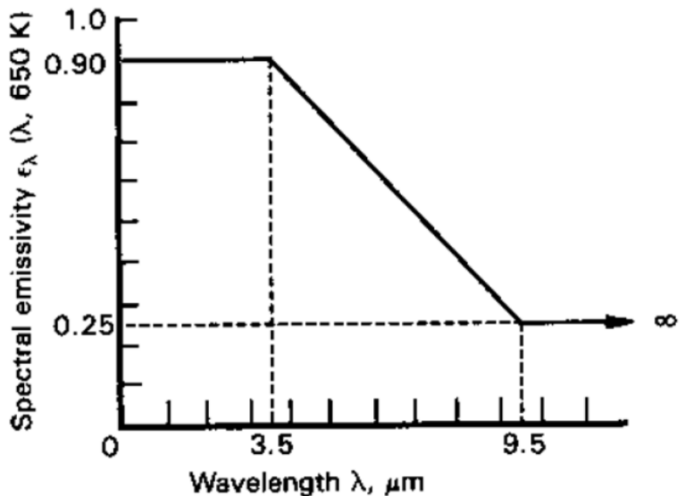
```
  {  double eps=total_emmissivity(1000,0.01,100.0);
     System.out.println("eps total  = "+eps);
     double eps1=total_emmissivity_FD(1000.0,0.01,100.0);
     System.out.println("eps1 total  = "+eps1);
  }
}
```

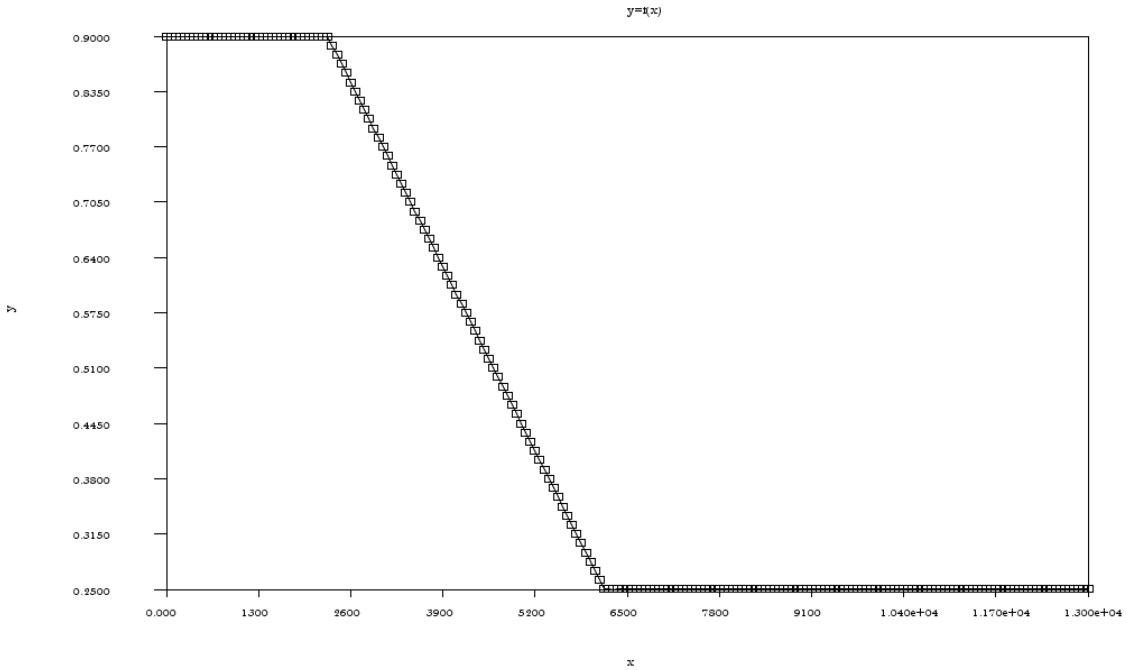
```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" radiation_ch1_P4
eps total  = 0.3263737765962589
eps1 total  = 0.3272811064431412

> Terminated with exit code 0.
```

PROBLEM

Surface temperature of a substance is T=650 K, and spectral emmisivity can be approximated as the figure below. Calculate the total emissivity coefficient.






```
  public static void main(String args[])
  {  double eps=total_emmissivity("d.txt",650.0,0.01,20.0);
     System.out.println("eps total  = "+eps);
```

```
    double eps1=total_emmissivity_FD("d.txt",650.0,0.01,20.0);
    System.out.println("eps1 total  = "+eps1);
  }
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" radiation_ch1_P3
eps total  = 0.5824152853506223
eps1 total  = 0.5819970639553191
```

Spectral emissivities are given as function:

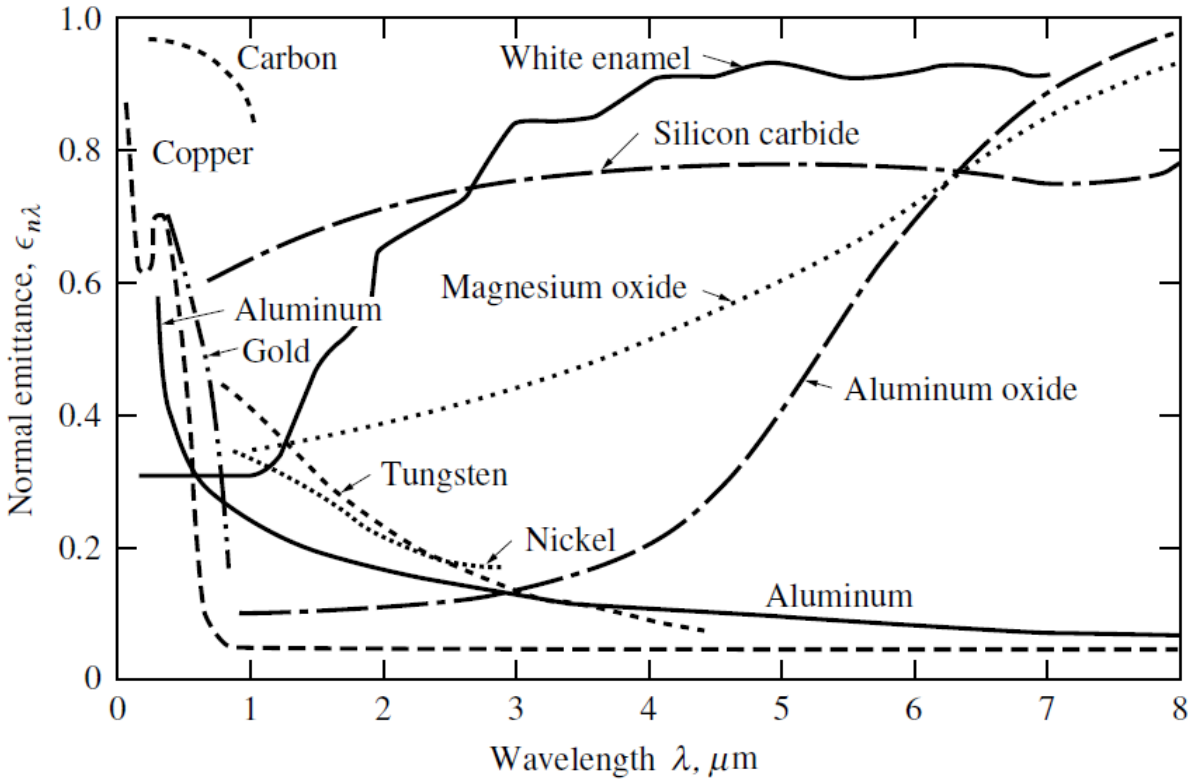
```
   public double func(double x)
  {//x ksi
    return eps(x)*15.0/(PI*PI*PI*PI)*x*x*x/(exp(x)-1.0);}
  public double eps(double x)
    {//x ksi
     //y lambda*T
     //z lambda
     double eps=0;
       double y=C2/x;
       double z=y/T;
       if(z<3.5) eps=0.9;
       else if(z>=3.5 && z<9.5) eps=1.27917-0.10833*z;
       else eps=0.25;
       return eps;
}
}
public static void main(String args[])
  {  double eps=total_emmissivity(650,0.01,20.0);
     System.out.println("eps total  = "+eps);
     double eps1=total_emmissivity_FD(650.0,0.01,20.0);
     System.out.println("eps1 total  = "+eps1);
  }
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" radiation_ch1_P4
eps total  = 0.5824154554285608
eps1 total  = 0.5819971034613677

> Terminated with exit code 0.
```

Actual datas are not change that smooth all the time. Actual datas can be read to the computer environmet than tha same programs can be used to evaluate total emissivity. If the data is in graphic format, digitizing of the data might be required.

PROBLEM. Let us consider total emissivity of aluminium oxide data is given the the figure below.



Program called digitizer is given below.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.filechooser.*;

 public class digitizer extends JApplet implements ActionListener
 {
 private JButton B[];
 private JLabel l[];
 JLabel altbar;
 private JTextField t[];
 private JTextArea ta;
 private String S[]={"input page","graphic digitizing page"};
 private Container c;
 private JPanel J[];
 scanP fp;
 double xmin,xmax,ymin,ymax;
 String resim,dosya;
 JTabbedPane jtp;

 public void init()
 {
              xmin=0.0;
              xmax=8.0;
              ymin=0.0;
              ymax=1.0;
              resim="emmisivity.JPG";
              dosya="a.txt";
 c=getContentPane();
 c.setLayout(new BorderLayout(5,5));
 B=new JButton[2];
 l=new JLabel[8];
 t=new JTextField[6];
 altbar=new JLabel();
 ta=new JTextArea(toString());
 jtp=new JTabbedPane();
 J=new JPanel[3];
 J[0]=new JPanel();
```

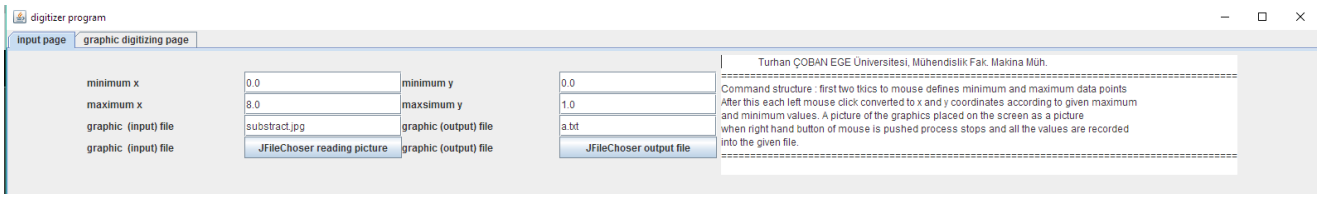
```
J[0].setLayout(new GridLayout(4,4));
J[1]=new JPanel();
J[2]=new JPanel();
B[0]=new JButton("JFileChoser reading picture");
B[1]=new JButton("JFileChoser output file");
l[0]=new JLabel("minimum x");
l[1]=new JLabel("minimum y");
l[2]=new JLabel("maximum x");
l[3]=new JLabel("maxsimum y");
l[4]=new JLabel("graphic  (input) file");
l[5]=new JLabel("graphic (output) file");
l[6]=new JLabel("graphic  (input) file");
l[7]=new JLabel("graphic (output) file");
t[0]=new JTextField(""+xmin);
t[1]=new JTextField(""+ymin);
t[2]=new JTextField(""+xmax);
t[3]=new JTextField(""+ymax);
t[4]=new JTextField(resim);
t[5]=new JTextField(dosya);
for(int i=0;i<5;i++) t[i].addActionListener(this);
for(int i=0;i<2;i++) B[i].addActionListener(this);
J[0].add(l[0]);
J[0].add(t[0]);
J[0].add(l[1]);
J[0].add(t[1]);
J[0].add(l[2]);
J[0].add(t[2]);
J[0].add(l[3]);
J[0].add(t[3]);
J[0].add(l[4]);
J[0].add(t[4]);
J[0].add(l[5]);
J[0].add(t[5]);
J[0].add(l[6]);
J[0].add(B[0]);
J[0].add(l[7]);
J[0].add(B[1]);
J[1].add(J[0],BorderLayout.NORTH);
J[1].add(ta,BorderLayout.CENTER);
jtp.addTab(S[0],J[1]);
fp=new scanP(resim,dosya,xmin,xmax,ymin,ymax);
//fp.setSize(1000,600);
//J[2].setLayout(new FlowLayout());
J[2].add(fp,BorderLayout.CENTER);
//J[2].add(fp.jp,BorderLayout.SOUTH);
jtp.addTab(S[1],fp);
c.add(jtp);
}

public String toString()
{String s;
s="            Turhan ÇOBAN EGE Üniversitesi, Mühendislik Fak. Makina Müh.\n";
s+="=============================================================================================\n";
s+="Command structure : first two tkics to mouse defines minimum and maximum data points\n";
s+="After this each left mouse click converted to x and y coordinates according to given maximum \n";
s+="and minimum values. A picture of the graphics placed on the screen as a picture\n";
s+="when right hand button of mouse is pushed process stops and all the values are recorded \n";
s+="into the given file.\n";
s+="=============================================================================================\n";
return s;
}

public void actionPerformed(ActionEvent e)
{
if(e.getSource()==t[0])
{            Double V0=new Double(t[0].getText());
   xmin=V0.doubleValue();
}
else if(e.getSource()==t[1])
{            Double V0=new Double(t[1].getText());
   ymin=V0.doubleValue();
}
else if(e.getSource()==t[2])
{            Double V0=new Double(t[2].getText());
   xmax=V0.doubleValue();
}
else if(e.getSource()==t[3])
```

```
{               Double V0=new Double(t[3].getText());
   ymax=V0.doubleValue();
}
else if(e.getSource()==t[4])
{               resim=t[4].getText();
  fp.setResimismi(resim);}
else if(e.getSource()==t[5])
{               dosya=t[4].getText();
   fp.setDosyaismi(dosya);
}
else if(e.getSource()==B[0])
{               File file=new File("emmisivity.jpg");
                JFileChooser fc=new JFileChooser();
   if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {file = fc.getSelectedFile();resim=file.getName();}
   t[4].setText(resim);
   fp.setResimismi(resim);
}
else if(e.getSource()==B[1])
{               File file=new File("a.txt");
                JFileChooser fc=new JFileChooser();
   if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {file = fc.getSelectedFile();dosya=file.getName();}
   t[5].setText(dosya);
   fp.setDosyaismi(dosya);
}
System.out.println("xmin = "+xmin+"xmax="+xmax+"ymin="+ymin+"ymax="+ymax);
fp.setMinMax(xmin,xmax,ymin,ymax);
repaint();
}


  public static void main(String s[])
  {
     JFrame f = new JFrame("digitizer program");
     JApplet applet = new digitizer();
     applet.init();
     f.add(applet);
     f.pack();
     f.setSize(1600,1000);
     f.setVisible(true);
  }
}
```

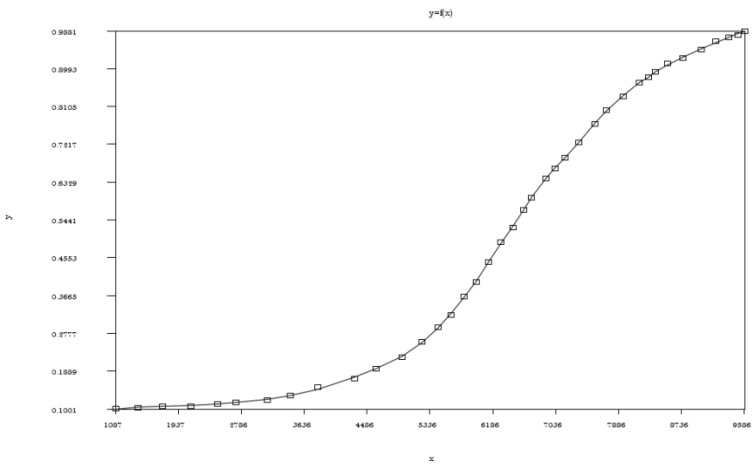


digitiser output data : a.txt file

0	1
0.9055472263868065	0.10039630118890357
1.1574212893553224	0.10303830911492734
1.4332833583208395	0.10700132100396301
1.7511244377811095	0.10700132100396301
2.0569715142428784	0.11096433289299867
2.2668665667166414	0.11624834874504623
2.6146926536731634	0.1215323645970938
2.8725637181409294	0.130779392338177
3.184407796101949	0.15059445178335534
3.598200899550225	0.17040951122853368
3.83808095952024	0.1955085865257596
4.131934032983508	0.22192866578599735
4.353823088455772	0.2575957727873184
4.539730134932533	0.2919418758256275
4.683658170914542	0.32100396301188905
4.833583208395802	0.3632760898282695
4.9655172413793105	0.3976221928665786
5.109445277361319	0.44517833553500663
5.247376311844078	0.4914134742404227
5.379310344827586	0.5257595772787318
5.505247376311844	0.5680317040951123
5.589205397301349	0.5970937912813739
5.751124437781109	0.6406869220607662
5.859070464767616	0.6644649933949802
5.967016491754123	0.6908850726552179

Data is recorded into file a.txt. data range is between 0.9055472263868065 and 7.988005997001499 mikrometer, and temperature is 1200 K So:

```
public static void main(String args[])
{  double eps=total_emmissivity("a.txt",1200,0.91,7.98);
   System.out.println("eps total  = "+eps);
   double eps1=total_emmissivity_FD("a.txt",1200,0.91,7.98);
   System.out.println("eps1 total  = "+eps1);
}
```




```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" radiation_ch1_P3
eps total  = 0.26101710218117774
eps1 total  = 0.2617421719867452
```

It should also be note that both approximation methods are given close values.

PROBLEM: Directional total emissivity $\varepsilon(\theta,T)$ values are given in the table below as a function of θ. ε is independent of ϕ angle and Temperature is T=1000 K. Find hemispherical total emissivity $\varepsilon(T)$
Hemispherical total emissivity (in terms of directional total emissivity)

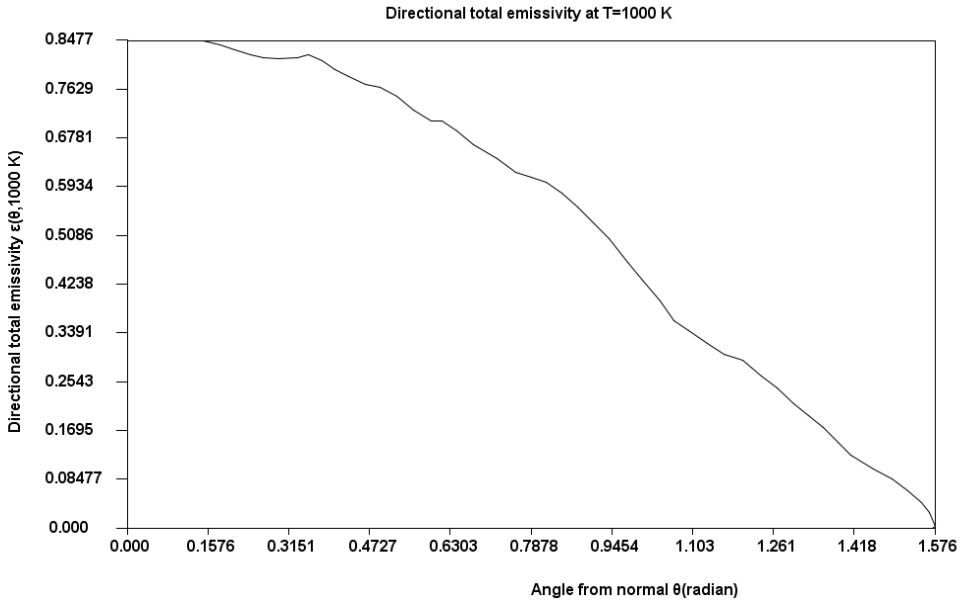
$$\varepsilon(T) = \frac{1}{\pi}\int_{\Omega} \varepsilon_\lambda(\theta,\phi,T)\cos(\theta)d\Omega = \frac{1}{\pi}\int_{\phi=0}^{2\pi}\int_{\theta=0}^{\pi/2} \varepsilon_\lambda(\theta,\phi,T)\sin(\theta)\cos(\theta)d\theta d\phi$$

$$\varepsilon(T) = 2\int_{\theta=0}^{\pi/2} \varepsilon_\lambda(\theta,\phi,T)\sin(\theta)\cos(\theta)d\theta = \int_{\theta=0}^{\pi/2} \varepsilon_\lambda(\theta,\phi,T)\sin(2\theta)d\theta$$

θ (radian)	ε(θ,T)	ε(θ,T)sin(2θ)
0	0.8476727	0
0.0399355	0.8462623	0.067519976
0.07745067	0.8476727	0.130781172
0.1125455	0.8476727	0.189196382
0.1488505	0.8476727	0.248642014
0.181525	0.8406205	0.298527132
0.210569	0.8307475	0.339608995
0.239613	0.8222849	0.379149405
0.2650265	0.8180536	0.413590774
0.29649083	0.8166431	0.456369705
0.33279583	0.8180536	0.505168232
0.35457883	0.8222849	0.53546761
0.37999233	0.8124118	0.559678913
0.40540583	0.7968970	0.57762809
0.4320295	0.7842031	0.596374347
0.464704	0.7715091	0.618183973
0.493748	0.7658674	0.63923081
0.5264225	0.7503526	0.651932835
0.559097	0.7263751	0.653238007
0.59419183	0.7080394	0.656895695
0.61476466	0.7080394	0.667207833
0.6425985	0.6911142	0.663119262
0.67648316	0.6657263	0.649994402
0.72125933	0.6417489	0.636476138
0.7587745	0.6177715	0.616895939
0.7878185	0.6093088	0.609301747
0.8168625	0.6008462	0.599656974
0.84832683	0.5825105	0.577903161
0.87979116	0.5585331	0.548609568
0.90883516	0.5303244	0.514245533
0.9402995	0.5021156	0.478211812
0.9766045	0.4612129	0.427898191
1.0056485	0.4301833	0.389117321
1.04074333	0.3949224	0.344533206
1.067367	0.3596615	0.3039704
1.09520083	0.3427362	0.279024222
1.13150583	0.3201692	0.246477171
1.16418033	0.3018335	0.219285662
1.20169549	0.2919605	0.196477395
1.23800049	0.2637517	0.162873206
1.26704449	0.2425952	0.138478088
1.30092916	0.2157969	0.110899775
1.32876299	0.1946403	0.090582199
1.35780699	0.1748942	0.072268422
1.38685099	0.1495063	0.053769664
1.41226449	0.1255289	0.039137146
1.45583049	0.1029619	0.023466153
1.49213549	0.0846262	0.013258689
1.52359983	0.0648801	0.006115138
1.54901333	0.0437235	0.001904258
1.56474549	0.0267983	0.000324296
1.57079633	0	0

Data can be plotted to see the variation

```
import java.io.*;
 class PlotT1
 {  public static void main(String args[]) throws IOException
   {Plot pp=new Plot("b.txt");
    pp.setPlabel("Directional total emissivity at T=1000 K");
    pp.setXlabel("Angle from normal "+'\u03B8'+"(radian)");
    pp.setYlabel("Directional total emissivity "+'\u03B5'+"("+'\u03B8'+",1000 K)");
    pp.plot();}}
```



Direct data integration with cubic spline interpolation will be used to solve the problem.
Cubic spline interpolation equation:

$$s_k(x) = a_k(x - x_k) + b_k(x_{k+1} - x) + \frac{[(x-x_k)^3 c_{k+1} + (x_{k+1}-x)^3 c_k]}{6h_k} \quad 1 \le k \le n$$

Defining $h_k = x_{k+1} - x_k$ $1 \le k \le n$
derivative equations:

$s'_k(x) = a_k - b_k + [(x-x_k)^2 c_{k+1} - (x_{k+1}-x)^2 c_k]/(2h_k)$ $1 \le k \le n$

$s''_k(x) = [(x-x_k) c_{k+1} + (x_{k+1}-x) c_k]/h_k$ $1 \le k \le n$

a_k ve b_k coefficients can be expressed as a function of c_k

$b_k = [6y_k - h_k c_k]/(6h_k)$, $1 \le k \le n$

$a_k = [6y_{k+1} - h_k^2 c_{k+1}]/(6h_k)$, $1 \le k \le n$

In this case only c_k terms left in the system of equations to be solved.

$$h_{k-1}c_{k-1} + 2(h_{k-1}-h_k)c_k + h_k c_{k+1} = 6\left[\frac{y_{k+1}-y_k}{h_k} - \frac{y_k - y_{k-1}}{h_{k-1}}\right] \quad , \quad 1 \le k \le n$$

This system of equation has only n-2 terms to be solved. By making definition

$$w_k = \frac{y_{k+1}-y_k}{h_k}, \quad 1 \le k \le n$$

System of equation becomes

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ h_1 & 2(h_1+h_2) & h_2 & \dots & 0 & 0 & 0 \\ 0 & h_2 & 2(h_2+h_3) & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 2(h_{n-3}+h_{n-2}) & h_{n-2} & 0 \\ 0 & 0 & 0 & \dots & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \dots \\ c_{n-2} \\ c_{n-1} \\ c_n \end{bmatrix} = \begin{bmatrix} A \\ 6(w_2 - w_1) \\ 6(w_3 - w_2) \\ \dots \\ 6(w_{n-2} - w_{n-3}) \\ 6(w_{n-1} - w_{n-2}) \\ B \end{bmatrix}$$

Where A and B are the second derivative end conditions. A and B should be defined by user.
Integration of the equation could be described with the equation

$$\int_a^b S_k(x)dx = \sum_{x=a}^{b} a_k \frac{(x-x_k)^2}{2} - b_k \frac{(x_{k+1}-x)^2}{2} + c_k \frac{(x-x_k)^4}{24h_k} - d_k \frac{(x_{k+1}-x)^4}{24h_k}$$

For a, x_0, x_1, x_2, …. x_n,.

```
//=====================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Direct data integration by using cubic spline
// curve fitting
// Dr. Turhan Coban
// =====================================================
import java.io.*;
import javax.swing.*;

class NA79
{

  public static double [] thomas(double a[][],double r[])
  {
  //
  int n=a.length;
  double f[]=new double[n];
  double e[]=new double[n];
  double g[]=new double[n];
  double x[]=new double[n];
  for(int i=0;i<n;i++)     {f[i]=a[i][i];}
  for(int i=0;i<(n-1);i++) {g[i]=a[i][i+1];}
  for(int i=0;i<(n-1);i++) {e[i+1]=a[i+1][i];}
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
     }
  for(int k=1;k<n;k++)
     {r[k]=r[k]-e[k]*r[k-1];
     }
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
     {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
  return x;
  }

  public static double [] thomas(double f[],double e[],double g[],double r[])
  {
  int n=f.length;
  double x[]=new double[n];
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
     }
  for(int k=1;k<n;k++)
     {r[k]=r[k]-e[k]*r[k-1];
     }
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
     {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
  return x;
  }


  public static double [][] cubic_spline(double xi[],double yi[],double c0,double cn)
  {
  // c0 second derivative of the first point
```

```
// cn second derivative of the last point
int n=xi.length;
double h[]=new double[n];
double w[]=new double[n];
double f[]=new double[n];
double e[]=new double[n];
double g[]=new double[n];
double d[]=new double[n];
double x[]=new double[n];
double S[][]=new double[4][n];
int k;
for(k=0;k<(n-1);k++)
   {h[k]=xi[k+1]-xi[k];
    w[k]=(yi[k+1]-yi[k])/h[k];
            }
d[0]=c0;
d[n-1]=cn;
for(k=1;k<(n-1);k++)
   {d[k]=6.0*(w[k]-w[k-1]);}
f[0]=1.0;
f[n-1]=1.0;
g[0]=0.0;
g[n-1]=0.0;
e[0]=0.0;
e[n-1]=0.0;
for(k=1;k<(n-1);k++)
{f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];}
S[2]=thomas(f,e,g,d);
S[3]=xi;
for(k=0;k<(n-1);k++)
  {S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
   S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
  }
return S;
}

public static double funcSpline(double S[][],double x)
{
int n=S[0].length;
double xx1=0;
double xx2=0;
double y=0;
double hk=0;
for(int k=0;k<(n-1);k++)
   {if(S[3][k]<=x && x<=S[3][k+1])
    {hk=(S[3][k+1]-S[3][k]);
     xx1=(x-S[3][k]);
     xx2=(S[3][k+1]-x);
     y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
     break;
     }
   }
     if(y==0 && S[3][n-2]<=x )
     {
              int k=n-2;
              hk=(S[3][k+1]-S[3][k]);
     xx1=(x-S[3][k]);
     xx2=(S[3][k+1]-x);
     y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
```

```
      }
  return y;
  }

// Derivative formulations
// =================
  public static double dfSpline(double xi[],double yi[],double c0,double cn,double x)
  { //kübik şerit türev formülü
            double S[][]=cubic_spline(xi,yi,c0,cn);
    return dfSpline(S,x);
  }

  public static double dfSpline(double S[][],double x)
  {
    //derivative of the cubic spline formula
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y=0;
  double hk=0;
  for(int k=0;k<(n-1);k++)
     {if(S[3][k]<=x && x<=S[3][k+1])
      {hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]-S[1][k]+(xx1*xx1*S[2][k+1]-xx2*xx2*S[2][k])/(2.0*hk);
       break;
       }
     }
       if(y==0 && S[3][n-2]<=x )
       {
              int k=n-2;
              hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
       }
  return y;
  }

 public static double  dfSpline(double xi[],double yi[],double x)
 {
   //natural cubic spline formula
            double S[][]=cubic_spline(xi,yi,0,0);
   return dfSpline(S,x);
 }
// Integration formulas
// ===================
  public static double intSpline(double S[][],double a,double b)
  {
    //cubic spline integration
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y1,y2;
  double hk=0;
  double toplam=0;
  y1=0;y2=0;
  for(int k=0;k<(n-1);k++)
```

```
   { hk=(S[3][k+1]-S[3][k]);
     //System.out.println("x1="+S[3][k]+"x2="+S[3][k+1]+"a="+a+"b="+b);
    if(a>S[3][k+1])
    {toplam=0;//System.out.println("şart 1 toplam = "+toplam);
    }//şart1
            else if(S[3][k]<=a && a<=S[3][k+1] && b>S[3][k+1])//şart 2
     {
     xx1=(a-S[3][k]);
     xx2=(S[3][k+1]-a);
     y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
     xx1=hk;
     xx2=0;
     y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
     toplam+=(y2-y1);
     //System.out.println("şart 2toplam = "+toplam+"y1="+y1+"y2="+y2);
     }
     else if(S[3][k]<=a && a<=S[3][k+1] && S[3][k]<=b && b<=S[3][k+1])//şart 3
     {
     xx1=(a-S[3][k]);
     xx2=(S[3][k+1]-a);
     y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
     xx1=(b-S[3][k]);
     xx2=(S[3][k+1]-b);
     y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
     toplam+=(y2-y1);
     //System.out.println("şart 3 toplam = "+toplam+"y1="+y1+"y2="+y2);
     }
     else if(a<S[3][k] && b>=S[3][k+1]) //şart 4
     {
     xx1=0;
     xx2=hk;
     y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
     xx1=hk;
     xx2=0;
     y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
     toplam+=(y2-y1);
     //System.out.println("şart 4 toplam = "+toplam+"y1="+y1+"y2="+y2);
     }
     else if(S[3][k]<=b && b<=S[3][k+1])//şart 5
     {xx1=0;
      xx2=hk;
      y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      xx1=(b-S[3][k]);
      xx2=(S[3][k+1]-b);
      y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      toplam+=(y2-y1);
      //System.out.println("şart 5 toplam = "+toplam+"y1="+y1+"y2="+y2);
      }
      else break;
   }
return toplam;
}

public static double intSpline(double xi[],double yi[],double c0,double cn,double a,double b)
{double S[][]=cubic_spline(xi,yi,c0,cn);
 return intSpline(S,a,b);
}

public static void main(String args[]) throws IOException
```

```
  {
  double x[];
  double y[];
  String s1="a.txt";
  //JFileChooser fc=new JFileChooser();
  //if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file = fc.getSelectedFile();s1=file.getName();
}
  double a[][]=Text.readDoubleT(s1);
  x=a[0];
  y=a[1];
  int n=x.length;
  double S[][]=cubic_spline(x,y,2.0,2.0);
  System.out.println("s="+intSpline(S,0.0,10.0));
  }
  }
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" NA79
s=0.5654504564547622

> Terminated with exit code 0.
```

PROBLEM:

The spectral emmissivity of an opaque surface at T=1000 K is

ϵ_λ	$\lambda_1 \quad \mu m$	$\lambda_2 \quad \mu m$
0.1	0	0.5
0.5	0.5	6
0.7	6	15
0.8	15	∞

$$\varepsilon_\lambda(\lambda,T)=\frac{E_\lambda(\lambda,T)}{E_{\lambda,\mathrm{b}}(\lambda,T)}$$

$$\varepsilon\,(T)=\frac{\int_{\lambda=0}^{\infty}E_\lambda(\lambda,T)d\lambda}{E_\mathrm{b}(T)}=\frac{\int_{\lambda=0}^{\infty}\varepsilon_\lambda(\lambda,T)E_{\lambda,\mathrm{b}}(\lambda,T)d\lambda}{E_\mathrm{b}(T)}$$

$$\varepsilon\,(T)=\frac{15}{\pi^4\left(F_{(0\to\lambda 2)}(T)-F_{(0\to\lambda 1)}(T)\right)}\int_{\xi=\xi 1}^{\xi 2}\frac{\varepsilon(\xi)\xi^3}{[e^{\xi}-1]}d\xi \quad \text{where } \xi=\frac{C_2}{\mathrm{n}\lambda\mathrm{T}}$$

$$\varepsilon\,(T)=\sum_{i=0}^{\infty}\left(F_{(0\to\lambda\mathrm{i})}(T)-F_{(0\to\lambda\mathrm{i}+1)}(T)\right)\varepsilon_{\lambda\mathrm{i}}(\lambda\mathrm{i},T)$$

$$\varepsilon\,(T)=\sum_{i=0}^{\infty}dF_{(\lambda\mathrm{i}+1\to\lambda\mathrm{i})}(T)\varepsilon_{\lambda\mathrm{i}}(\lambda\mathrm{i},T)$$

```
public class F_radiation1C
{ public static double F(double x)
  {double C2=1.438769e4;
  double y=C2/x;
  double total=0;
  for(int n=1;n<80;n++)
  {total+=Math.exp(-n*y)/n*(y*y*y+3.0*y*y/n+6.0*y/(n*n)+6.0/(n*n*n));}
  double pi=Math.PI;
  total=15.0/(pi*pi*pi*pi)*total;
  return total;
  }

  public static void main(String arg[])
  {
  double lambda[]={0,0.5,6,15,1000000};
  double eps_lambda[]={0.1,0.5,0.7,0.8};
```

```
  int m=lambda.length;
  double x[]=new double[m];
  double F[]=new double[m];
  double dF[]=new double[m-1];
  double n=1.0; //refractive index for space(vacuum)
  double T=1000.0; //degree K
   F[0]=0.0;
  for(int i=1;i<m;i++)
  {x[i]=n*T*lambda[i];
   F[i]=F(x[i]);
   System.out.println("i="+i+"F["+i+"]="+F[i]);
  }
  System.out.println("_____");
  for(int i=1;i<m;i++)
  {dF[i-1]=F[i]-F[i-1];
  System.out.println("i="+(i-1)+"dF["+(i-1)+"]="+dF[i-1]);
  }
  System.out.println("_____");
  double eps=0;
  for(int i=0;i<m-1;i++)
  {eps+=eps_lambda[i]*dF[i];}
   System.out.println("eps="+eps);
  }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" F_radiation1C
i=1F[1]=1.2988973795691605E-9
i=2F[2]=0.7377922055755127
i=3F[3]=0.9689346652512938
i=4F[4]=0.9999993871050767
_____
i=0dF[0]=1.2988973795691605E-9
i=1dF[1]=0.7377922042766153
i=2dF[2]=0.23114245967578106
i=3dF[3]=0.03106472185378295
_____
eps=0.5555476015242705
> Terminated with exit code 0.
```

PROBLEM: Iron at T=1122 K emmisivity values are given below. Calculate total emmisivity by

a) Directly using the data (assumming in each step emmisivity remaining constatnt)
b) Apply least square curve fit and use finite difference
c) Apply least square curve fit and integration
d) Apply cubic spline interpolation and use cubic spline interpolation integration

T=1122 K	
λ	ε
0.99	0.324
1.18	0.264
1.41	0.264
1.57	0.244
1.77	0.24
1.99	0.226
2.16	0.218
2.55	0.205
2.76	0.194
2.96	0.188
3.45	0.175
3.97	0.169
4.48	0.166
4.92	0.164
5.42	0.147
5.97	0.134
6.38	0.135
6.87	0.134
7.41	0.135
7.91	0.126

8.45	0.123
8.89	0.122
9.51	0.117
9.84	0.117
10.20	0.115
10.90	0.115
11.30	0.114
11.80	0.112
12.40	0.105
13.40	0.107

a. Directly using the data (assumming in each step emmisivity remaining constatnt)

```
public class F_radiation1D
{ public static double F(double x)
  {double C2=1.438769e4;
  double y=C2/x;
  double total=0;
  for(int n=1;n<80;n++)
  {total+=Math.exp(-n*y)/n*(y*y*y+3.0*y*y/n+6.0*y/(n*n)+6.0/(n*n*n));}
  double pi=Math.PI;
  total=15.0/(pi*pi*pi*pi)*total;
  return total;
  }

  public static void main(String arg[])
  {
  double a[][]=Text.readDoubleT("eps_iron.txt");
  double lambda[]=a[0];
  double eps_lambda[]=a[1];
  int m=lambda.length+1;
  double x[]=new double[m];
  double F[]=new double[m];
  double dF[]=new double[m-1];
  double n=1.0; //refractive index for space(vacuum)
  double T=1000.0; //degree K
   F[0]=0.0;
  for(int i=1;i<m;i++)
  {x[i]=T*lambda[i-1];
   F[i]=F(x[i]);
   System.out.println("i="+i+" F["+i+"]="+F[i]);
  }
  System.out.println("_____");
  for(int i=1;i<m;i++)
  {dF[i-1]=F[i]-F[i-1];
  System.out.println("i="+(i-1)+" dF["+(i-1)+"]="+dF[i-1]+" eps_lamda["+(i-1)+"]="+eps_lambda[i-1]);
  }
  System.out.println("_____");
  double eps=0;
  for(int i=0;i<m-1;i++)
  {eps+=eps_lambda[i]*dF[i];}
   System.out.println("eps="+eps);
  }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" F_radiation1D
i=1 F[1]=2.8526248893405985E-4
i=2 F[2]=0.0018238051620682269
i=3 F[3]=0.008221015569791568
i=4 F[4]=0.017459193679708145
i=5 F[5]=0.03589222142733151
i=6 F[6]=0.06518999874988675
i=7 F[7]=0.0935888502791333
i=8 F[8]=0.17217081906532306
i=9 F[9]=0.21885503730244477
i=10 F[10]=0.2641735875120476
i=11 F[11]=0.37238144747912416
i=12 F[12]=0.47539422214216537
i=13 F[13]=0.5612479989413174
i=14 F[14]=0.6234979145476536
i=15 F[15]=0.6824814103928024
i=16 F[16]=0.7352314762392873
i=17 F[17]=0.7677486920436554
i=18 F[18]=0.8003947489865475
i=19 F[19]=0.8299889420413478
i=20 F[20]=0.8525984339103343
i=21 F[21]=0.8728787509000688
i=22 F[22]=0.8868179517857215
i=23 F[23]=0.9032854923387336
i=24 F[24]=0.9107912010194025
i=25 F[25]=0.9181377279325369
i=26 F[26]=0.9303063657486972
```

```
i=27 F[27]=0.9362094449944063
i=28 F[28]=0.9427006226303127
i=29 F[29]=0.9493929029462698
i=30 F[30]=0.9584321096731078
_____________________
i=0 dF[0]=2.8526248893405985E-4 eps_lamda[0]=0.324
i=1 dF[1]=0.001538542673134167 eps_lamda[1]=0.264
i=2 dF[2]=0.006397210407723342 eps_lamda[2]=0.264
i=3 dF[3]=0.009238178109916577 eps_lamda[3]=0.244
i=4 dF[4]=0.018433027747623367 eps_lamda[4]=0.24
i=5 dF[5]=0.029297777322555237 eps_lamda[5]=0.226
i=6 dF[6]=0.028398851529246552 eps_lamda[6]=0.218
i=7 dF[7]=0.07858196878618975 eps_lamda[7]=0.205
i=8 dF[8]=0.04668421823712171 eps_lamda[8]=0.194
i=9 dF[9]=0.04531855020960285 eps_lamda[9]=0.188
i=10 dF[10]=0.10820785996707655 eps_lamda[10]=0.175
i=11 dF[11]=0.1030127746630412 eps_lamda[11]=0.169
i=12 dF[12]=0.08585377679915202 eps_lamda[12]=0.166
i=13 dF[13]=0.06224991560633619 eps_lamda[13]=0.164
i=14 dF[14]=0.05898349584514884 eps_lamda[14]=0.147
i=15 dF[15]=0.052750065846484895 eps_lamda[15]=0.134
i=16 dF[16]=0.03251721580436806 eps_lamda[16]=0.135
i=17 dF[17]=0.03264605694289213 eps_lamda[17]=0.134
i=18 dF[18]=0.029594193054800266 eps_lamda[18]=0.135
i=19 dF[19]=0.02260949186898653 eps_lamda[19]=0.126
i=20 dF[20]=0.020280316989734537 eps_lamda[20]=0.123
i=21 dF[21]=0.013939200885652703 eps_lamda[21]=0.122
i=22 dF[22]=0.016467540553012028 eps_lamda[22]=0.117
i=23 dF[23]=0.00750570868066891 eps_lamda[23]=0.117
i=24 dF[24]=0.007346526913134421 eps_lamda[24]=0.115
i=25 dF[25]=0.012168637816160288 eps_lamda[25]=0.115
i=26 dF[26]=0.0059030792457090975 eps_lamda[26]=0.114
i=27 dF[27]=0.006491177635906453 eps_lamda[27]=0.112
i=28 dF[28]=0.006692280315957033 eps_lamda[28]=0.105
i=29 dF[29]=0.009039206726838 eps_lamda[29]=0.107
_____________________
eps=0.15983144378391959
> Terminated with exit code 0.
```

b,c) Least square curve fitting method

```
 /=====================================================
// Radiation heat transfer
// total emmissivity (wavelength based emmisisvity is given as
// data- partial continuous pulynomial curve fittin ids applied)
// Dr. Turhan Coban
// =====================================================
import java.io.*;
import javax.swing.*;
class fh extends f_xr
{   double func(double x,int n)
                { double xx=1.0;
     if(n==0) xx=1.0;
     else if(n==1) xx=x;
     else if(n==2) xx=x*x;
     else if(n==3) xx=x*x*x;
     else if(n==4) xx=x*x*x*x;
     else if(n==5) xx=x*x*x*x*x;
     else if(n==6) xx=x*x*x*x*x*x;
     else if(n==7) xx=x*x*x*x*x*x*x;
     else if(n==8) xx=x*x*x*x*x*x*x*x;
     else if(n==9) xx=x*x*x*x*x*x*x*x*x;
     else if(n==10) xx=x*x*x*x*x*x*x*x*x*x;
     else if(n==10) xx=x*x*x*x*x*x*x*x*x*x*x;
     else if(n==10) xx=x*x*x*x*x*x*x*x*x*x*x*x;
     else if(n==10) xx=x*x*x*x*x*x*x*x*x*x*x*x*x;
     else if(n==10) xx=x*x*x*x*x*x*x*x*x*x*x*x*x*x;
     else if(n==10) xx=x*x*x*x*x*x*x*x*x*x*x*x*x*x*x;
     else
     {            for(int i=1;i<=n;i++) xx*=x;}
     return xx;
                }
}

class f1 extends f_x
{ public double C1=3.741774892e8; //    W/(mikrometer)^4/(m^2sr)
   public double C2=14387.69;  //mikrometer K
  public double sigma=C1/(C2*C2*C2*C2)*PI*PI*PI*PI/15.0;
  double a[][]; //coefficients of partial continuous curve fitting
  double T;
  double xmin,xmax;
  double ymin,ymax;
  static char lambda='\u03BB';
  static char  epsilon='\u03B5';
  static char  ksi='\u03BE';
  static String s1=""+lambda+"*T";
```

```
  static String s2=""+epsilon+"_"+lambda+"*T";
  fh f2;
  public f1(double x[][],double Ti)
  {   T=Ti;
      f2=new fh();
      xmin=1.0e200;
      xmax=0;
                  for(int i=0;i<x[0].length;i++)
       {x[0][i]=T*x[0][i];
        if(x[0][i]<xmin) {xmin=x[0][i];ymin=x[1][i];}
        if(x[0][i]>xmax) {xmax=x[0][i];ymax=x[1][i];}
       }
       String label[]={s1,s2};
       Text.printT(x,label,"data read");
       a=partial_continuous.EKKgeneral(f2,x[0],x[1],15,6);
       Text.print(a,"a");
       partial_continuous.funcEKKPlot(f2,x[0],x[1],15,6);
  }
  public f1(String name,double Ti)
  {double x[][]=Text.readDoubleT(name);T=Ti;
   T=Ti;
      f2=new fh();
      xmin=1.0e200;
      xmax=0;
                  for(int i=0;i<x[0].length;i++)
       {x[0][i]=T*x[0][i];
        if(x[0][i]<xmin) {xmin=x[0][i];ymin=x[1][i];}
        if(x[0][i]>xmax) {xmax=x[0][i];ymax=x[1][i];}
       }
       String label[]={s1,s2};
       Text.printT(x,label,"data read");
       a=partial_continuous.EKKgeneral(f2,x[0],x[1],5,3);
       Text.print(a,"a");
        partial_continuous.funcEKKPlot(f2,x[0],x[1],5,3);
  }
  public f1(double Ti)
  { T=Ti;
    double x[][]=Text.readDoubleT("copper1.txt");T=Ti;
    T=Ti;
      f2=new fh();
      xmin=1.0e200;
      xmax=0;
                  for(int i=0;i<x[0].length;i++)
       {x[0][i]=T*x[0][i];
        if(x[0][i]<xmin) xmin=x[0][i];
        if(x[0][i]>xmax) xmax=x[0][i];
       }
       String label[]={s1,s2};
       Text.printT(x,label,"data read");
       a=partial_continuous.EKKgeneral(f2,x[0],x[1],5,3);
       Text.print(a,"a");
        partial_continuous.funcEKKPlot(f2,x[0],x[1],5,3);
  }
  public double func(double x)
  {//x ksi
    return eps(x)*15.0/(PI*PI*PI*PI)*x*x*x/(exp(x)-1.0);}

  public double eps(double x)
  {//x ksi
   //y lambda*T
   //z lambda
   double C2=14387.69;  //mikrometer K
   double y=C2/x;
   double z=y/T;
   double eps=0;
   if(y<xmin) eps=ymin;
   else if(y>xmax) eps=ymax;
   else eps=partial_continuous.funcEKKgeneral(f2,a,y);
   return eps;
  }
}
class F_radiation1E
{
  public static double F(double x)
  {
  if(x<=0) return 0;
   double C2=14387.69;  //mikrometer K
  double y=C2/x;
  double total=0;
  for(int n=1;n<80;n++)
  {total+=Math.exp(-n*y)/n*(y*y*y+3.0*y*y/n+6.0*y/(n*n)+6.0/(n*n*n));}
  double pi=Math.PI;
  total=15.0/(pi*pi*pi*pi)*total;
  return total;
  }
 public static double[][] gauss_legendre_coefficients(double x1,double x2,int n)
 {
                  //calculates legendre gauss-coefficients as coefficients of the integral
```

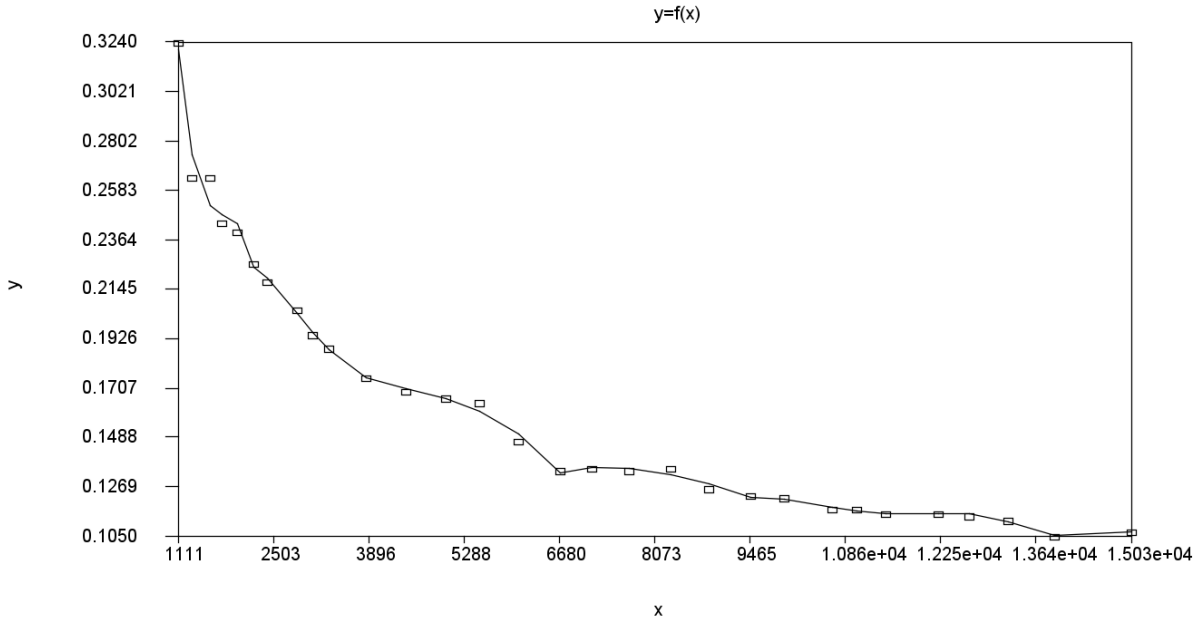
```
            //for n terms
            double EPS=3.0e-15;
            int m,j,i;
            double z1,z,xm,xl,pp,p3,p2,p1;
   //double x[]=new double[n];
   //double w[]=new double[n];
   double a[][]=new double[2][n];//a[0][i]=x[i]   a[1][i]=w[i]
            m=(n+1)/2;
            xm=0.5*(x2+x1);
            xl=0.5*(x2-x1);
            for (i=1;i<=m;i++)  {
                        z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                        do {
                                    p1=1.0;
                                    p2=0.0;
                                    for (j=1;j<=n;j++) {
                                                p3=p2;
                                                p2=p1;
                                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                                    }
                                    pp=n*(z*p1-p2)/(z*z-1.0);
                                    z1=z;
                                    z=z1-p1/pp;
                        } while (Math.abs(z-z1) > EPS);
                        a[0][i-1]=xm-xl*z;
                        a[0][n-i]=xm+xl*z;
                        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                        a[1][n-i]=a[1][i-1];
                                    }
return a;
}
public static double integral(f_x f_xnt,double x1,double x2,int n)
{
//n : number of integral coefficients
// this routine first generates gauss legendre coefficients
// for [x1,x2] band
// then calculates gauss legendre integral
double a[][]=new double[2][n];
a=gauss_legendre_coefficients(x1,x2,n);
double z=0;
for(int i=0;i<n;i++)
{ z+=a[1][i]*f_xnt.func(a[0][i]);}
return z;
}
public static double total_emmissivity(String name,double T,double lambda1,double lambda2)
{ // integration method
  f1 b1=new f1(name,T);
  double C2=14387.69;  //mikrometer K
  double x1=C2/(lambda1*T);
  double x2=C2/(lambda2*T);
  double dF=F(lambda2*T)-F(lambda1*T);
  double z=integral(b1,x2,x1,20)/dF;
  return z;
}
public static double total_emmissivity_FD(String name,double T,double lambda1,double lambda2)
{return total_emmissivity_FD(name,T,lambda1,lambda2,300);}
public static double total_emmissivity_FD(String name,double T,double lambda1,double lambda2,int n)
{ //finite difference addition (euler integration) method
  f1 b1=new f1(name,T);
  double dlambdaT=(lambda2-lambda1)*T/(double)(n-1);
  double z=0;
  double lambdaT=lambda1*T;
  double lambdaT1;
  double y;
  double eps;
  double C2=14387.69;  //mikrometer K
  double x1=C2/(lambda1*T);
  double x2=C2/(lambda2*T);
  for(int i=0;i<n;i++)
  {   y=C2/(lambdaT+dlambdaT/2.0);
                lambdaT1=lambdaT+dlambdaT;
                z+=b1.eps(y)*(F(lambdaT1)-F(lambdaT));
                lambdaT=lambdaT1;
  }
  double dF=F(lambda2*T)-F(lambda1*T);
  z/=dF;
  return z;
}
   public static double min(double a[])
   {  int n=a.length;
      double max1=a[0];
      for(int i=1;i<n;i++)
      {max1=Math.min(max1,a[i]);}
                  return max1;
   }
   public static double max(double a[])
   {  int n=a.length;
      double max1=a[0];
```

```
      for(int i=1;i<n;i++)
     {max1=Math.max(max1,a[i]);}
                    return max1;
   }
  public static void main(String args[])
  {  String data_name=JOptionPane.showInputDialog("data file name(eps_iron.txt) : ");
     double T=Double.parseDouble(JOptionPane.showInputDialog("Temperature(1122  K) : "));
     double a[][]=Text.readDoubleT(data_name);
     double lambda[]=a[0];
     double eps_lambda[]=a[1];
     double lambda_min=min(lambda);
     double lambda_max=max(lambda);
                 double eps=total_emmissivity(data_name,T,lambda_min,lambda_max);
     System.out.println("eps total (integral method)  = "+eps);
     double eps1=total_emmissivity_FD("a.txt",1000.0,0.01,100.0);
     System.out.println("eps1 total (Finite difference method) = "+eps1);
  }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" F_radiation1E
eps total (integral method)  = 0.17830279960933584
eps1 total (Finite difference method) = 0.16822785526819434
```

data read

λ*T	ε_λ*T
1110.78	0.324
1323.96	0.264
1582.02	0.264
1761.54	0.244
1985.94	0.24
2232.78	0.226
2423.52	0.218
2861.1	0.205
3096.72	0.194
3321.12	0.188
3870.9	0.175
4454.34	0.169
5026.56	0.166
5520.24	0.164
6081.24	0.147
6698.34	0.134
7158.36	0.135
7708.14	0.134
8314.02	0.135
8875.02	0.126
9480.9	0.123
9974.58	0.122
10670.22	0.117
11040.48	0.117
11444.4	0.115
12229.800000000001	0.115
12678.6	0.114
13239.6	0.112
13912.800000000001	0.105
15034.800000000001	0.107



d) Cubic spline interpoaltion:
if_x

```
import static java.lang.Math.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import static java.lang.Math.*;
  // single function single independent variable
  // example f=x*x
  // includes full set of derivatives
  // Reference :"Generation of Finite Difference Formulas on Arbitrary Spaced Grids",
  // Bength Fornberg, Mathematics of Computation, Volume 51, Number 184, October 1988
  // pages 699-706
  //interface (empty function) version
@FunctionalInterface
interface if_x extends iMathd
{  public double func(double x);
//first order derivative
default double dfunc(double x)
{double h=1.0e-4;
 int n=1;
 int M=10;
 return dfunc(x,n,M,h);
}
default double dfunc(double x,double h)
{
 int n=1;
 int M=10;
 return dfunc(x,n,M,h);
}
//second order derivative
default double dfunc2(double x)
{double h=1.0e-6;
 int n=2;
 int M=10;
 return dfunc(x,n,M,h);
}
default double dfunc(double x,int N,int Mi,double hi)
{// order of the maximum derivative
 // N order of derivative
 // M degree of difference formula
double c[][][];
double alpha[];
double h;
int M=Mi;
double a[]=new double[0];
h=0.01;
double x0=0;
double alphai[]={0,1,-1,2,-2,3,-3,4,-4,5,-5,6,-6,7,-7,8,-8,9,-9,10,-10,11,-11,12,-12,13,-13,14,-14,15,
-15,16,-16,17,-17,18,-18,19,-19,20,-20,21,-21,22,-22,23,-23,24,-24,25,-25,26,-26,27,-27,28,-28,29,-29,30,-30,
31,-31,32,-32,33,-33,34,-34,35,-35,36,-36,37,-37,38,-38,39,-39,40,-40,41,-41,42,-42,43,-43,44,-44,45,-45,46,
-46,47,-47,48,-48,49,-49,50,-50,51,-51,52,-52,53,-53,54,-54,55,-55,56,-56,57,-57,58,-58,59,-59,60,-60,
-61,61,-62,62,-63,63,-64,64,-65,65,-66,66,-67,67,-68,68,-70,70,-71,71,-72,72,-73,73,-74,74,-75,75,
-76,76,-77,77,-78,78,-79,79,-80,80,-81,81,-82,82,-83,83,-84,84,-85,85,-86,86,-87,87,
```

```
-88,88,-89,89,-90,90,-91,91,-92,92,-93,93,-94,94,-95,95,-96,96,-97,97,-98,98,-99,99,-100,100};
alpha=alphai;
int N1=alpha.length-1;
// M   degree of highest derivative
// N+1 number of coefficients
double delta[][][]=new double[N1+1][N1+1][M+1];
double c1,c2,c3;
delta[0][0][0]=1.0;
c1=1.0;
for(int n=1;n<=N1;n++)
{  c2=1;
   for(int nu=0;nu<=(n-1);nu++)
   {c3=alpha[n]-alpha[nu];
    c2=c2*c3;
    if(n<=M) delta[n-1][nu][n]=0.0;
    for(int m=0;m<=Math.min(n,M);m++)
    {
        if(m==0)
        { delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m])/c3;}
        else
        {delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m]-m*delta[n-1][nu][m-1])/c3;}
    }//next m
   }//next nu
   for(int m=0;m<=Math.min(n,M);m++)
   {   if(m==0)
          {delta[n][n][m]=c1/c2*(-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
          else
          {delta[n][n][m]=c1/c2*(m*delta[n-1][n-1][m-1]-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
   }//next m
   c1=c2;
}//next n
c=delta;
  if(Mi<N) M=N;
  else M=Mi;
  h=hi;
  double deriv=0;
  double h1=1/h;
  double h2=1;
  for(int j=0;j<N;j++)
  {h2*=h1;}
  for(int i=0;i<c[0].length;i++)
  {   deriv+=c[M][i][N]*func(x+alpha[i]*h);}
  return deriv*h2;}
default double dfunc(double x,int N)
{int M=30;double h=0.01*N*N;return dfunc(x,N,M,h);}
default double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{   //calculates legendre gauss-coefficients as coefficients of the integral
        //for n terms
        double EPS=3.0e-8;
        int m,j,i;
        double z1,z,xm,xl,pp,p3,p2,p1;
    //double x[]=new double[n];
    //double w[]=new double[n];
    double a[][]=new double[2][n];//a[0][i]=x[i]   a[1][i]=w[i]
        m=(n+1)/2;
        xm=0.5*(x2+x1);
        xl=0.5*(x2-x1);
        for (i=1;i<=m;i++)
        { z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                    do {
                                p1=1.0;
                                p2=0.0;
                                for (j=1;j<=n;j++) {
                                        p3=p2;
                                        p2=p1;
                                        p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                                }
                                pp=n*(z*p1-p2)/(z*z-1.0);
                                z1=z;
                                z=z1-p1/pp;
                    } while (Math.abs(z-z1) > EPS);
                    a[0][i-1]=xm-xl*z;
                    a[0][n-i]=xm+xl*z;
                    a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                    a[1][n-i]=a[1][i-1];
        }
return a;
}
default double integral(double x1,double x2)
{ int n=10;
  return integral(x1,x2,n);
}
default double integral(double x1,double x2,int n)
{
```

```
//n : number of integral coefficients
// this routine first generates gauss legendre coefficients
// for [x1,x2] band
// then calculates gauss legendre integral
double a[][]=new double[2][n];
a=gauss_legendre_coefficients(x1,x2,n);
double z=0;
for(int i=0;i<n;i++)
{ z+=a[1][i]*func(a[0][i]);
}
return z;
}
default double bisection(double a,double b)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   double eps=1.0e-6;
   int nmax=100;
   int i=1;
   while(Math.abs(func(r))>eps && i<nmax)
   {if(func(a)*func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;;
    i++;
   }
   if(i>=nmax) r=bisection(a,b1);
   return r;
  }
default double secant(double x)
{ double eps=1.0e-10;
  double y=1.0;
  double dy=0.0;
  int miter=100;
  int i=0;
  while(abs(y)>eps && i<miter)
  {y=func(x);
   dy=dfunc(x);
   x=x-y/dy;
   i++;
  }
  if(i>=miter)
    System.out.println("i="+i+"results may not be valid");
  return x;
}
default double illinois(double xa,double xu)
{
//Modified False position root finding: Illinois method
double test;
double p=0;
double es,ea;
double fa,fp,fu;
int maxit=100,iter=0;
es=0.000001;
ea=1.1*es;
double xold;
int ia=0,iu=0;
fa=func(xa);
fu=func(xu);
p=xu;
double xpold;
while(iter<maxit && ea>es)
{
xpold=p;
p=xu-fu*(xa-xu)/(fa-fu);
fp=func(p);
if(signum(fp)!=signum(fu))
{xa=xu;fa=fu;}
else
{fa=fa/2;}
xu=p;fu=fp;
iter++;
if(p!=0) ea=abs((p-xpold)/p)*100.0;
}
if(iter>=maxit) System.out.println("Maximum number of iteration is exceeded ");
return p;
}
default double newton_bisection(double x1, double x2)
{
double xacc=1.0e-10; //accuracy
double h=0.0001; //derivative function step
int MAXIT=100;  //Maximum number of iterations
int j;
double dfr,dx,dxold,fr,fh,fl;
double temp,b,a,r;
fl=func(x1);
```

```
fh=func(x2);
//Check if a root is existed in the given region
if ((fl > 0.0 && fh > 0.0) || (fl < 0.0 && fh < 0.0))
System.out.println("There are no root in the given region or double root?");
if (fl == 0.0) return x1;
if (fh == 0.0) return x2;
if (fl < 0.0)   { a=x1;b=x2;}
else           { b=x1;a=x2;}
r=0.5*(x1+x2); //midpoint value
dxold=Math.abs(x2-x1);
dx=dxold;
fr=func(r);   //function value at midpoint
dfr=dfunc(r); //derivative of function value at midpoint
for (j=1;j<=MAXIT;j++)
{
if ((((r-b)*dfr-fr)*((r-a)*dfr-fr) > 0.0) || (Math.abs(2.0*fr) > Math.abs(dxold*dfr)))
   { //bisection step
        dxold=dx;
     dx=0.5*(b-a);
     r=a+dx;
     if (a == r) return r; //çözüm!!!!
   }
else{ //Newton-Raphson (Secant) step
         dxold=dx;
     dx=fr/dfr;
     temp=r;
     r -= dx;
     if (temp == r) return r;  //solution!!!!!
   }
if (Math.abs(dx) < xacc) return r; //solution!!!!!
fr=func(r);
dfr=dfunc(r);
if (fr < 0.0)
a=r;
else
b=r;
}
System.out.println("Maximum number of iterations are exceeded");
return 0.0; //program should never reach here
          //dummy return
}
default double brent(double a,double b)
{
double test;
double p=0;
double es,ea;
double f1,f2,f3,fp;
int maxit=500,iter=0;
double tol=1.0e-15;
es=0.0000001;
double x1=a;f1=func(x1);
double x2=b;f2=func(x2);
double x3=(x1+x2)/2.0;f3=func(x3);
if(f1==0) return x1;
else if(f2==0) return x2;
else if(f3==0) return x3;
if(f1*f2>0) System.out.println("No root is existed in the given region");
p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
fp=func(p);
ea=Math.abs(x3-p);
while((ea>es)&&(iter<maxit))
{      if(Math.abs(f3)<tol) return x3;
   if((p<x1) && (p>x2))
   {p=(x1+x2)/2.0;
    if(p>x3) {x1=x3;f1=f3;x3=p;f3=fp;}
    else if(p<x3) {x2=x3;f2=f3;x3=p;f3=fp;}
   }
   else
   {
   if(p>x3) {x1=x3;f1=f3;x3=p;f3=fp;}
   else if(p<x3) {x2=x3;f2=f3;x3=p;f3=fp;}
   p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
   fp=func(p);
   ea=Math.abs(x3-p);
   }
   iter++;
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Warning : MAximum number of iteration is exceeded \n"+
  " result may not be valid","MAXIMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return p;
}
default double enlarge(double x0,double dx)
{  //enlarge region untill a root is existed
       double x1=x0;
```

```
        double x2=x1+dx;
        int NTRY=200;
        double a[]=new double[2];
        double FACTOR=1.001;
        int j;
        double f1,f2;
        if (x1 == x2) System.out.println("Giriş değerleri yanlış verildi");
        f1=func(x1);
        f2=func(x2);
        for (j=1;j<=NTRY;j++)
        { if (f1*f2 < 0.0) {break;}
          else {x2+=dx;f2=func(x2);}
        }
        return x2;
    }
    default void plot(double x1,double x2)
       {Plot p=new Plot(this,x1,x2);
        double x[]={x1,x2};
        double y[]={0,0};
        p.addData(x,y);
        p.plot();}
    }
```

```
public class f_eps implements if_x
{ double S[][];
  double lambda[];
  double x[];
  double eps_lambda[];
  double eps_min;
  double eps_max;
  double lambda_min;
  double lambda_max;

  public f_eps(String data_name,double T)
  {  double a[][]=Text.readDoubleT(data_name);
     double lambda1[]=a[0];
     lambda=lambda1;
     double eps_lambda1[]=a[1];
     eps_lambda=eps_lambda1;
     eps_min=dmin(eps_lambda);
     eps_max=dmax(eps_lambda);
     lambda_min=dmin(lambda);
     lambda_max=dmax(lambda);
     int m=lambda.length;
     double x1[]=new double[m];
     for(int i=0;i<m;i++)
     {x1[i]=T*lambda[i];
     }
     x=x1;
     double S1[][]=cubic_spline(lambda,eps_lambda,0.0,0.0);
     S=S1;
  }
  public double func(double xx)
  {return funcSpline(S,xx);}

  public static double [] thomas(double a[][],double r[])
  {
  //
  int n=a.length;
  double f[]=new double[n];
  double e[]=new double[n];
  double g[]=new double[n];
  double x[]=new double[n];
  for(int i=0;i<n;i++)     {f[i]=a[i][i];}
  for(int i=0;i<(n-1);i++) {g[i]=a[i][i+1];}
  for(int i=0;i<(n-1);i++) {e[i+1]=a[i+1][i];}
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
     }
  for(int k=1;k<n;k++)
     {r[k]=r[k]-e[k]*r[k-1];
     }
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
     {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
  return x;
  }

  public static double [] thomas(double f[],double e[],double g[],double r[])
  {
  int n=f.length;
  double x[]=new double[n];
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
```

```
    }
  for(int k=1;k<n;k++)
    {r[k]=r[k]-e[k]*r[k-1];
    }
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
    {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
  return x;
  }


  public static double [][] cubic_spline(double xi[],double yi[],double c0,double cn)
  {
  // c0 second derivative of the first point
  // cn second derivative of the last point
  int n=xi.length;
  double h[]=new double[n];
  double w[]=new double[n];
  double f[]=new double[n];
  double e[]=new double[n];
  double g[]=new double[n];
  double d[]=new double[n];
  double x[]=new double[n];
  double S[][]=new double[4][n];
  int k;
  for(k=0;k<(n-1);k++)
    {h[k]=xi[k+1]-xi[k];
     w[k]=(yi[k+1]-yi[k])/h[k];
                }
  d[0]=c0;
  d[n-1]=cn;
  for(k=1;k<(n-1);k++)
    {d[k]=6.0*(w[k]-w[k-1]);}
  f[0]=1.0;
  f[n-1]=1.0;
  g[0]=0.0;
  g[n-1]=0.0;
  e[0]=0.0;
  e[n-1]=0.0;
  for(k=1;k<(n-1);k++)
  {f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];}
  S[2]=thomas(f,e,g,d);
  S[3]=xi;
  for(k=0;k<(n-1);k++)
    {S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
     S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
    }
  return S;
  }

  public static double funcSpline(double S[][],double x)
  {
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y=0;
  double hk=0;
  for(int k=0;k<(n-1);k++)
    {if(S[3][k]<=x && x<=S[3][k+1])
     {hk=(S[3][k+1]-S[3][k]);
      xx1=(x-S[3][k]);
      xx2=(S[3][k+1]-x);
      y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
      break;
      }
    }
      if(y==0 && S[3][n-2]<=x )
      {
                int k=n-2;
                hk=(S[3][k+1]-S[3][k]);
      xx1=(x-S[3][k]);
      xx2=(S[3][k+1]-x);
      y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
      }
  return y;
  }

// Derivative formulations
// =================
  public static double dfSpline(double xi[],double yi[],double c0,double cn,double x)
  { //kübik şerit türev formülü
              double S[][]=cubic_spline(xi,yi,c0,cn);
    return dfSpline(S,x);
  }

  public static double dfSpline(double S[][],double x)
  {
    //derivative of the cubic spline formula
```

```
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y=0;
  double hk=0;
  for(int k=0;k<(n-1);k++)
    {if(S[3][k]<=x && x<=S[3][k+1])
     {hk=(S[3][k+1]-S[3][k]);
      xx1=(x-S[3][k]);
      xx2=(S[3][k+1]-x);
      y=S[0][k]-S[1][k]+(xx1*xx1*S[2][k+1]-xx2*xx2*S[2][k])/(2.0*hk);
      break;
      }
    }
      if(y==0 && S[3][n-2]<=x )
      {
                  int k=n-2;
                  hk=(S[3][k+1]-S[3][k]);
      xx1=(x-S[3][k]);
      xx2=(S[3][k+1]-x);
      y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
      }
  return y;
  }

 public static double  dfSpline(double xi[],double yi[],double x)
 { double S[][]=cubic_spline(xi,yi,0,0);
   return dfSpline(S,x);
 }
// Integration formulas
// ====================
  public static double intSpline(double S[][],double a,double b)
  {
    //cubic spline integration
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y1,y2;
  double hk=0;
  double toplam=0;
  y1=0;y2=0;
  for(int k=0;k<(n-1);k++)
    { hk=(S[3][k+1]-S[3][k]);
     if(a>S[3][k+1])
     {toplam=0;
     }//şart1
                 else if(S[3][k]<=a && a<=S[3][k+1] && b>S[3][k+1])//şart 2
      {
      xx1=(a-S[3][k]);
      xx2=(S[3][k+1]-a);
      y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      xx1=hk;
      xx2=0;
      y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      toplam+=(y2-y1);
      }
      else if(S[3][k]<=a && a<=S[3][k+1] && S[3][k]<=b && b<=S[3][k+1])//şart 3
      {
      xx1=(a-S[3][k]);
      xx2=(S[3][k+1]-a);
      y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      xx1=(b-S[3][k]);
      xx2=(S[3][k+1]-b);
      y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      toplam+=(y2-y1);
      }
      else if(a<S[3][k] && b>=S[3][k+1]) //şart 4
      {
      xx1=0;
      xx2=hk;
      y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      xx1=hk;
      xx2=0;
      y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      toplam+=(y2-y1);
      }
      else if(S[3][k]<=b && b<=S[3][k+1])//şart 5
      {xx1=0;
       xx2=hk;
       y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       xx1=(b-S[3][k]);
       xx2=(S[3][k+1]-b);
       y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       toplam+=(y2-y1);
       }
       else break;
    }
  return toplam;
```

```
  }

  public static double intSpline(double xi[],double yi[],double c0,double cn,double a,double b)
  {double S[][]=cubic_spline(xi,yi,c0,cn);
   return intSpline(S,a,b);
  }

   public static double dmin(double a[])
   {  int n=a.length;
      double max1=a[0];
      for(int i=1;i<n;i++)
      {max1=Math.min(max1,a[i]);}
                  return max1;
   }
   public static double dmax(double a[])
   {  int n=a.length;
      double max1=a[0];
      for(int i=1;i<n;i++)
      {max1=Math.max(max1,a[i]);}
                  return max1;
   }

   public void Plot()
   {Plot p=new Plot(this,lambda_min,lambda_max);
                  p.addData(lambda,eps_lambda);
                  p.setPlotType(1,20);
                  p.plot();
   }

  public static void main(String arg[])
  {   double T=1122.0;
      String data_name="eps_iron.txt";
                  f_eps f=new f_eps(data_name,T);
                  f.Plot();

  }
}
```

```
//=======================================================
// Radiation heat transfer
// total emmissivity (wavelength based emmisisvity is given as
// data- partial continuous pulynomial curve fittin ids applied)
// Dr. Turhan Coban
// =======================================================
import java.io.*;
import javax.swing.*;


class f1 implements if_x
{ public double C1=3.741774892e8; //     W/(mikrometer)^4/(m^2sr)
   public double C2=14387.69;  //mikrometer K
  public double sigma=C1/(C2*C2*C2*C2)*PI*PI*PI*PI/15.0;
  double a[][]; //coefficients of partial continuous curve fitting
  double T;
  double xmin,xmax;
  double ymin,ymax;
  static char lambda='\u03BB';
  static char  epsilon='\u03B5';
  static char  ksi='\u03BE';
  static String s1=""+lambda+"*T";
  static String s2=""+epsilon+"_"+lambda+"*T";
  f_eps f2;

  public f1(String name,double Ti)
  {double x[][]=Text.readDoubleT(name);T=Ti;
   T=Ti;
      f2=new f_eps(name,Ti);
      f2.Plot();
      xmin=1.0e200;
      xmax=0;
                  for(int i=0;i<x[0].length;i++)
        {x[0][i]=T*x[0][i];
         if(x[0][i]<xmin) {xmin=x[0][i];ymin=x[1][i];}
         if(x[0][i]>xmax) {xmax=x[0][i];ymax=x[1][i];}
        }
        String label[]={s1,s2};
        Text.printT(x,label,"data read");
  }

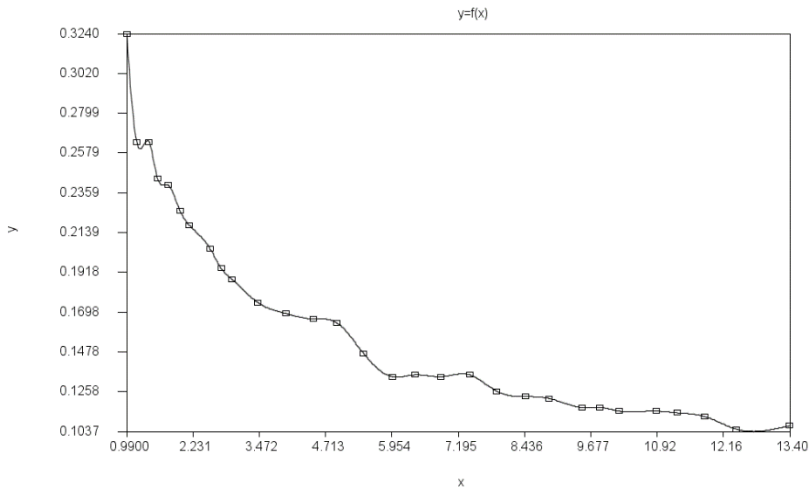
  public double func(double x)
  {//x ksi
    return eps(x)*15.0/(PI*PI*PI*PI)*x*x*x/(exp(x)-1.0);}

  public double eps(double x)
  {//x ksi
```

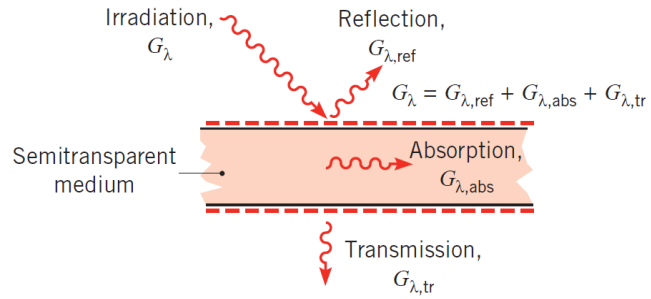
```
   //y lambda*T
   //z lambda
   double C2=14387.69;  //mikrometer K
   double y=C2/x;
   double z=y/T;
   double eps=0;
   if(y<xmin) eps=ymin;
   else if(y>xmax) eps=ymax;
   else eps=f2.func(x);
   return eps;
  }
}
class F_radiation1F
{
  public static double F(double x)
  {
  if(x<=0) return 0;
   double C2=14387.69;  //mikrometer K
  double y=C2/x;
  double total=0;
  for(int n=1;n<80;n++)
  {total+=Math.exp(-n*y)/n*(y*y*y+3.0*y*y/n+6.0*y/(n*n)+6.0/(n*n*n));}
  double pi=Math.PI;
  total=15.0/(pi*pi*pi*pi)*total;
  return total;
  }
public static double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{
            //calculates legendre gauss-coefficients as coefficients of the integral
            //for n terms
            double EPS=3.0e-15;
            int m,j,i;
            double z1,z,xm,xl,pp,p3,p2,p1;
   //double x[]=new double[n];
   //double w[]=new double[n];
   double a[][]=new double[2][n];//a[0][i]=x[i]   a[1][i]=w[i]
            m=(n+1)/2;
            xm=0.5*(x2+x1);
            xl=0.5*(x2-x1);
            for (i=1;i<=m;i++)  {
                        z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                        do {
                                    p1=1.0;
                                    p2=0.0;
                                    for (j=1;j<=n;j++) {
                                                p3=p2;
                                                p2=p1;
                                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                                    }
                                    pp=n*(z*p1-p2)/(z*z-1.0);
                                    z1=z;
                                    z=z1-p1/pp;
                        } while (Math.abs(z-z1) > EPS);
                        a[0][i-1]=xm-xl*z;
                        a[0][n-i]=xm+xl*z;
                        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                        a[1][n-i]=a[1][i-1];
                                    }
return a;
}
public static double integral(if_x f_xnt,double x1,double x2,int n)
{
//n : number of integral coefficients
// this routine first generates gauss legendre coefficients
// for [x1,x2] band
// then calculates gauss legendre integral
double a[][]=new double[2][n];
a=gauss_legendre_coefficients(x1,x2,n);
double z=0;
for(int i=0;i<n;i++)
{ z+=a[1][i]*f_xnt.func(a[0][i]);}
return z;
}
public static double total_emmissivity(String name,double T,double lambda1,double lambda2)
{ // integration method
  f1 b1=new f1(name,T);
  double C2=14387.69;  //mikrometer K
  double x1=C2/(lambda1*T);
  double x2=C2/(lambda2*T);
  double dF=F(lambda2*T)-F(lambda1*T);
  double z=integral(b1,x2,x1,20)/dF;
  return z;
}
public static double total_emmissivity_FD(String name,double T,double lambda1,double lambda2)
```

```
{return total_emmissivity_FD(name,T,lambda1,lambda2,300);}
public static double total_emmissivity_FD(String name,double T,double lambda1,double lambda2,int n)
{ //finite difference addition (euler integration) method
  f1 b1=new f1(name,T);
  double dlambdaT=(lambda2-lambda1)*T/(double)(n-1);
  double z=0;
  double lambdaT=lambda1*T;
  double lambdaT1;
  double y;
  double eps;
  double C2=14387.69;  //mikrometer K
  double x1=C2/(lambda1*T);
  double x2=C2/(lambda2*T);
  for(int i=0;i<n;i++)
  {   y=C2/(lambdaT+dlambdaT/2.0);
                lambdaT1=lambdaT+dlambdaT;
                z+=b1.eps(y)*(F(lambdaT1)-F(lambdaT));
                lambdaT=lambdaT1;
  }
  double dF=F(lambda2*T)-F(lambda1*T);
  z/=dF;
  return z;
}
   public static double min(double a[])
   {  int n=a.length;
      double max1=a[0];
      for(int i=1;i<n;i++)
      {max1=Math.min(max1,a[i]);}
                return max1;
   }
   public static double max(double a[])
   {  int n=a.length;
      double max1=a[0];
      for(int i=1;i<n;i++)
      {max1=Math.max(max1,a[i]);}
                return max1;
   }
  public static void main(String args[])
  {  String data_name=JOptionPane.showInputDialog("data file name(eps_iron.txt) : ");
     double T=Double.parseDouble(JOptionPane.showInputDialog("Temperature(1122  K) : "));
     double a[][]=Text.readDoubleT(data_name);
     double lambda[]=a[0];
     double eps_lambda[]=a[1];
     double lambda_min=min(lambda);
     double lambda_max=max(lambda);
               double eps=total_emmissivity(data_name,T,lambda_min,lambda_max);
     System.out.println("eps total (integral method)  = "+eps);
     double eps1=total_emmissivity_FD("a.txt",1000.0,0.01,100.0);
     System.out.println("eps1 total (Finite difference method) = "+eps1);
  }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" F_radiation1F
eps total (integral method)  = 0.1834149994763962
eps1 total (Finite difference method) = 0.17821932634920443
```



20 ABSORPTION, REFLECTION, AND TRANSMISSION BY REAL SURFACES



In previus section, it is seen that emission from real surface is associated with surface property emission. When emitted radiative enegies from sources of emitters came to another surface, some of it will be absorbed, some will be reflected and some will be transmitted through the body (assuming that it is transparent for the given wavelengths). Energy coming to the surface is called irradiation G and like emitted energy it depends on direction and wavelength and temperatures. From energy balance it can be written

$$G_\lambda(\theta,\phi) = G_{\lambda,\text{reflection}}(\theta,\phi) + G_{\lambda,\text{absorbtion}}(\theta,\phi) + G_{\lambda,\text{transmission}}(\theta,\phi)$$

Absorptivity:

Total absorbtivities of some materials

Metal
Aluminum:
Highly polished 0.10
Polished
Chromium, electroplated 0.40
Copper:
Highly polished 0.18
Clean 0.25
Tarnished 0.64
Oxidized 0.70
Galvanized iron 0.38
Gold, bright foil 0.29
Iron:
Ground with fine grit 0.36
Blued 0.55
Sandblasted 0.75
Magnesium, polished 0.19
Nickel:
Highly polished 0.15
Polished 0.36
Electrolytic 0.40
Platinum, bright 0.31
Silver:
Highly polished 0.07
Polished 0.13
Commercial sheet 0.30
Stainless steel #301, polished 0.37
Tungsten, highly polished 0.37
Dielectric
Aluminum oxide (Al2O3) 0.06–0.23
Asphalt pavement, dust-free 0.93
Brick, red 0.75
Clay 0.39
Concrete roofing tile:
Uncolored 0.73
Brown 0.91
Black 0.91
Earth, plowed field 0.75
Felt, black 0.82
Graphite 0.88
Grass 0.75–0.80
Gravel 0.29
Leaves, green 0.71–0.79
Magnesium oxide (MgO) 0.15
Marble, white 0.46

Paint:
Aluminum 0.55
Oil, zinc white 0.30
Oil, light green 0.50
Oil, light gray 0.75
Oil, black on galvanized iron 0.90
Silicone, flat black 0.95–0.97
Velostat black plastic 0.96
Paper, white 0.28
Slate, blue gray 0.88
Snow, clean 0.2–0.35
Soot, coal 0.95
Titanium dioxide (TiO2) 0.12
Zinc oxide 0.15
Zinc sulfide (ZnS) 0.21

Coating
Black coatings:
Anodize black 0.88
Carbon black paint NS-7 0.96
Ebanol C black 0.97
Martin black velvet paint 0.91
3M black velvet paint 0.97
Tedlar black plastic 0.94
Velostat black plastic 0.96
White coatings:
Barium sulfate with polyvinyl alcohol 0.06
Catalac white paint 0.23
Dow Corning white paint DC-007 0.19
Magnesium oxide white paint 0.09
Potassium fluorotitanate white paint 0.15
Tedlar white plastic 0.39
Titanium oxide white paint with methyl silicone 0.20
Zinc oxide with sodium silicate 0.15
Conversion coatings (values can vary significantly with coating thickness):
Alzac A-2 0.16
Black chrome 0.96
Black copper 0.98
Black irridite 0.62
Black nickel 0.91
Vapor-deposited coatings on glass substrates:
Aluminum 0.08
Chromium 0.56
Gold 0.19
Nickel 0.38
Silver 0.04
Titanium 0.52
Tungsten 0.60
Plastic films with metal backing:
Mylar film, 3-mil aluminum backing 0.17
Tedlar film, 1-mil gold backing 0.26
Teflon film, 2-mil aluminum backing 0.08
Teflon film, 1-mil gold backing 0.22
Teflon film, 2-mil silver backing 0.08

Directional spectral absorptivity $\alpha_\lambda(\theta,\phi,T)$

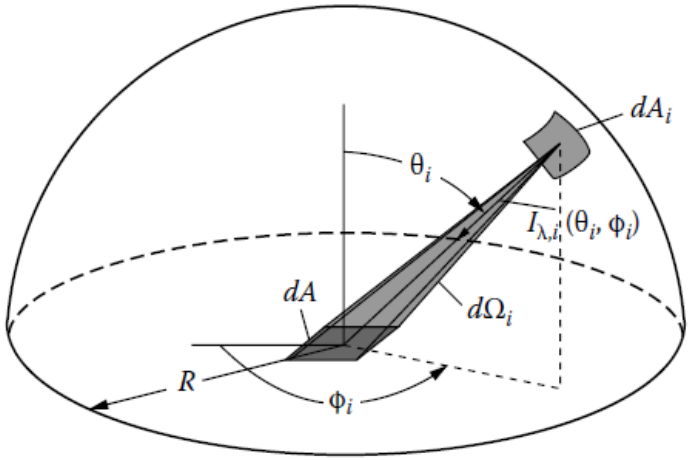


Figure illustrates energy incident on a surface element dA from (θ_i, ϕ_i) direction. The incident spectral intensity passing through dA_i is $I_{\lambda i}(\theta_i, \phi_i)$. This is the energy per unit area of the hemisphere, per unit solid angle $d\,\Omega$, per unit time, and per unit wavelength interval. The energy within the incident solid angle $d\,\Omega$ strikes the
area dA of the absorbing surface. Hence

$G_{\lambda i}(\theta_i, \phi_i)d\lambda = I_{\lambda i}(\theta_i, \phi_i)dA_i cos(\theta_i)d\Omega d\lambda = I_{\lambda i}(\theta_i, \phi_i)dA_i \frac{dAcos(\theta_i)}{R^2} d\lambda$ or

$G_{\lambda i}(\theta_i, \phi_i)d\lambda = I_{\lambda i}(\theta_i, \phi_i)dAcos(\theta_i)d\Omega_i d\lambda = I_{\lambda i}(\theta_i, \phi_i)dA \frac{dA_i cos(\theta_i)}{R^2} d\lambda$ where $d\Omega = \frac{dA}{R^2}$ and $d\Omega_i = \frac{dA_i}{R^2}$

So Directional spectral absorptivity $\alpha_\lambda(\theta,\phi,T)$

$\alpha_\lambda(\theta,\phi,T) = \frac{G_{\lambda a}(\theta_i,\phi_i,T)}{I_{\lambda i}(\theta_i,\phi_i)dAd\Omega_i d\lambda}$

Kirchoff's law: If we consider a black enclosure in thermal equilibrium, then relation $\alpha_\lambda(\theta,\phi,T) = \varepsilon_\lambda(\theta,\phi,T)$ will hold. This means in the enclosure absorbing and emmitting energies must be equal.

Directional total absorptivity $\alpha(\theta,\phi,T)$

The directional total absorptivity is the energy including all wavelengths that is absorbed from a given direction, divided by the energy incident from that direction.

Radiation coming in all wavelengths

$G_i(\theta_i, \phi_i) = dAcos(\theta_i)d\Omega_i \int_{\lambda=0}^{\infty} I_{\lambda i}(\theta_i, \phi_i)d\lambda$

Radiation absorbed in all wavelengths

$G_a(\theta_i, \phi_i, T_i) = dAcos(\theta_i)d\Omega_i \int_{\lambda=0}^{\infty} \alpha_\lambda(\theta_i, \phi_i, T) I_{\lambda i}(\theta_i, \phi_i)d\lambda$

Directional total absorptivity $\alpha(\theta_i, \phi_i, T_i) = \frac{G_a(\theta_i,\phi_i,T_i)}{G_i(\theta_i,\phi_i)} = \frac{\int_{\lambda=0}^{\infty} \alpha_\lambda(\theta_i,\phi_i,T) I_{\lambda i}(\theta_i,\phi_i)d\lambda}{\int_{\lambda=0}^{\infty} I_{\lambda i}(\theta_i,\phi_i)d\lambda}$

Hemispherical spectral absorptivity $\alpha_\lambda(T)$

The hemispherical spectral absorptivity is the fraction of the spectral energy that is absorbed from the spectral energy that is incident from all directions of a surrounding hemisphere

$G_{\lambda i}d\lambda = d\lambda dA \int_\Omega I_{\lambda i}(\theta_i, \phi_i)cos(\theta_i)d\Omega_i$

$G_{\lambda a}d\lambda = d\lambda dA \int_\Omega \alpha_\lambda(\theta_i, \phi_i, T) I_{\lambda i}(\theta_i, \phi_i)cos(\theta_i)d\Omega_i$

$$\alpha_\lambda(T) = \frac{G_{\lambda a}}{G_{\lambda i}} = \frac{\int_\Omega \alpha_\lambda(\theta_i, \phi_i, T) I_{\lambda i}(\theta_i, \phi_i)cos(\theta_i)d\Omega_i}{\int_\Omega I_{\lambda i}(\theta_i, \phi_i)cos(\theta_i)d\Omega_i}$$

Hemispherical total absorptivity $\alpha(T)$

The hemispherical total absorptivity represents the fraction of energy absorbed that is incident from all directions of the enclosing hemisphere and for all wavelengths.

$$G_i = dA \int_\Omega \left[\int_{\lambda=0}^{\infty} I_{\lambda i}(\theta_i, \phi_i)d\lambda \right] cos(\theta_i)d\,\Omega_i$$

$G_a = dA \int_\Omega \left[\int_{\lambda=0}^{\infty} \alpha_\lambda(\theta_i, \phi_i, T) I_{\lambda i}(\theta_i, \phi_i)d\lambda \right] cos(\theta_i)d\,\Omega_i$

$\alpha(T) = \frac{G_a}{G_i} = \frac{\int_\Omega \left[\int_{\lambda=0}^{\infty} \alpha_\lambda(\theta_i,\phi_i,T) I_{\lambda i}(\theta_i,\phi_i)d\lambda \right] cos(\theta_i)d\Omega_i}{\int_\Omega \left[\int_{\lambda=0}^{\infty} I_{\lambda i}(\theta_i,\phi_i)d\lambda \right] cos(\theta_i)d\Omega_i}$

Reflectivity

The reflective properties of a surface are more complicated to specify than the emissivity or absorptivity. This is because reflected energy depends not only on the angle at which the incident energy impinges on the surfaces but also on the direction being considered for the reflected energy.

Spectral Reflectivity of copper

reflectivity	
copper	
T=298	
λ	**ρ**
1	0.855
1.75	0.899
2	0.915
2.25	0.9
2.5	0.923
2.75	0.931
3	0.933
3.25	0.945
3.5	0.946
3.75	0.951
4.25	0.948
4.75	0.961
5	0.95
5.25	0.96
5.5	0.955
5.75	0.964
6	0.96
6.25	0.967
6.5	0.953
7	0.953
7.25	0.96
7.75	0.951
8	0.959
8.25	0.952
8.5	0.96
9	0.96
9.25	0.965
9.5	0.965
9.75	0.955
10	0.955
10.25	0.951
10.5	0.953
10.75	0.951
11	0.962
11.5	0.953
11.75	0.959
12	0.959
12.25	0.971
12.5	0.96
12.75	0.971
13.25	0.949
13.5	0.949
13.75	0.958

Reflectivity of copper as a function of angle

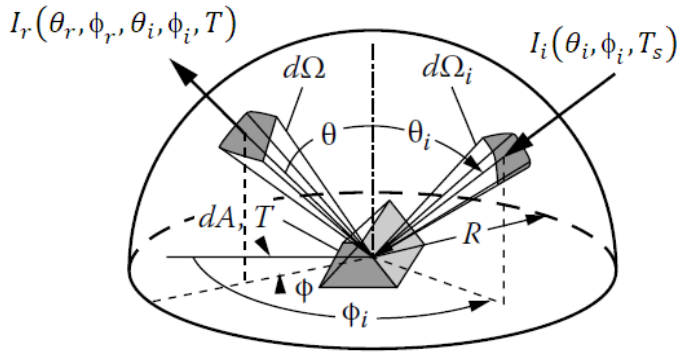
reflectivity		Highly polished	
copper		0.25 μm	
T=298K	**λ=1.2**	T=298K	**λ=1.8**
angle deg.	**ρ**	angle deg.	**ρ**
10	0.92	10	0.933
20	0.92	20	0.933
30	0.92	30	0.933
40	0.925	40	0.923
50	0.93	50	0.034
60	0.92	60	0.915
70	0.91	70	0.904
75	0.91	75	0.914

reflectivity		sanded with emery paper			
copper		1.3 μm			
T=298K	**λ=0.65**	T=298K	**λ=1.2**	T=298K	**λ=1.8**
angle deg.	**ρ**	angle deg.	**ρ**	angle deg.	**ρ**
10	0.739	10	0.83	10	0.833
20	0.739	20	0.824	20	0.822
30	0.734	30	0.82	30	0.809
40	0.725	40	0.811	40	0.808
50	0.715	50	0.799	50	0.808
60	0.711	60	0.781	60	0.799
70	0.709	70	0.77	70	0.794
75	0.709	75	0.765	75	0.784

Biderectional spectral reflectivity $\rho_\lambda(\theta_r, \phi_r, \theta_i, \phi_i)$

Coming radiation incident : $I_i(\theta_i, \phi_i, T_s)$

Reflecting radiation incident $I_r(\theta_r, \phi_r, \theta_i, \phi_i, T)$



$$G_{\lambda i}(\theta_i,\phi_i)d\lambda = I_{\lambda i}(\theta_i,\phi_i)dA\cos(\theta_i)d\Omega_i d\lambda = I_{\lambda i}(\theta_i,\phi_i)dA\frac{dA_i\cos(\theta_i)}{R^2}d\lambda$$

Biderectional spectral reflectivity $\rho_\lambda(\theta_r,\phi_r,\theta_i,\phi_i) = \frac{I\lambda_r(\theta_r,\phi_r,\theta_i,\phi_i,T)}{I_{\lambda i}(\theta_i,\phi_i)dA\cos(\theta_i)d\Omega_i}$

Directional spectral reflectivities $\rho_\lambda(\theta_i,\phi_i),\ \rho_\lambda(\theta_r,\phi_r)$

The **directional-hemispherical spectral reflectivity** is defined as the energy reflected into all solid angles divided by the energy incident from a particular direction (θ_i,ϕ_i)

$G_{\lambda i}(\theta_i,\phi_i)d\lambda = d\lambda dA\int_\Omega I_{\lambda i}(\theta_i,\phi_i)\cos(\theta_i)d\Omega_i$

$G_{\lambda r}(\theta_i,\phi_i)d\lambda = d\lambda dA\int_\Omega I_{\lambda i}(\theta_r,\phi_r,\theta_i,\phi_i)\cos(\theta_r)d\Omega_r =$

$d\lambda dA\, I_{\lambda i}(\theta_i,\phi_i)\int_\Omega \rho_\lambda(\theta_r,\phi_r,\theta_i,\phi_i)\cos(\theta_r)d\Omega_r$

$$\rho_\lambda(\theta_i,\phi_i) = \frac{G_{\lambda r}(\theta_i,\phi_i)}{= G_{\lambda i}(\theta_i,\phi_i)} = \int_\Omega \rho_\lambda(\theta_r,\phi_r,\theta_i,\phi_i)\cos(\theta_r)d\Omega_r$$

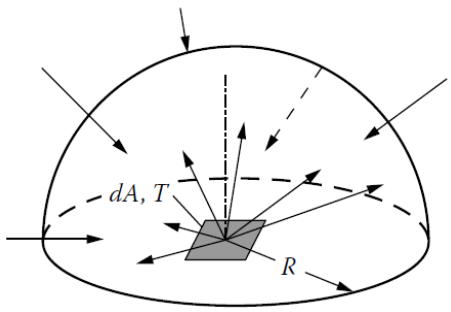
The **hemispherical -directional spectral reflectivity** is defined as the energy reflected into a particular direction (θ_r,ϕ_r)

$\rho_\lambda(\theta_r,\phi_r) = \frac{\int_\Omega \rho_\lambda(\theta_r,\phi_r,\theta_i,\phi_i)I_{\lambda i}(\theta_i,\phi_i)\cos(\theta_i)d\Omega_i}{\frac{1}{\pi}\int_\Omega I_{\lambda i}(\theta_i,\phi_i)\cos(\theta_i)d\Omega_i}$

If incident intensity is uniform over all incident directions

$$\rho_\lambda(\theta_r,\phi_r) = \int_\Omega \rho_\lambda(\theta_r,\phi_r,\theta_i,\phi_i)\cos(\theta_i)d\Omega_i$$

Hemispherical spherical reflectivity $\boldsymbol{\rho_\lambda}$



If the incident spectral radiation arrives from all angles of the hemisphere and teflects to all angles of the hemisphere

$$G_{\lambda i}d\lambda = d\lambda dA\int_\Omega I_{\lambda i}(\theta_i,\phi_i)\cos(\theta_i)d\Omega_i$$

$$G_{\lambda \mathrm{r}} d\lambda = \mathrm{d}\lambda \mathrm{dA} \int_{\Omega} \rho_{\lambda}(\theta_i, \phi_i) I_{\lambda \mathrm{i}}(\theta_i, \phi_i) cos(\theta_i) d\Omega_i$$

$$\rho_{\lambda} = \frac{G_{\lambda \mathrm{r}} d\lambda}{G_{\lambda \mathrm{i}} d\lambda} = \frac{\int_{\Omega} \rho_{\lambda}(\theta_i, \phi_i) I_{\lambda \mathrm{i}}(\theta_i, \phi_i) cos(\theta_i) d\Omega_i}{\int_{\Omega} I_{\lambda \mathrm{i}}(\theta_i, \phi_i) cos(\theta_i) d\Omega_i}$$

Bidirectional total reflectivity $\rho(\theta_r, \phi_r, \theta_i, \phi_i)$

$$\rho(\theta_r, \phi_r, \theta_i, \phi_i) = \frac{\int_{\lambda=0}^{\infty} I_{\lambda \mathrm{r}}(\theta_r, \phi_r, \theta_i, \phi_i) d\lambda}{\cos(\theta_i) d\Omega_i \int_{\lambda=0}^{\infty} I_{\lambda \mathrm{r}}(\theta_i, \phi_i) d\lambda} = \frac{I_{\mathrm{r}}(\theta_r, \phi_r, \theta_i, \phi_i)}{I_{\mathrm{i}}(\theta_i, \phi_i) cos(\theta_i) d\Omega_i}$$

$$\rho(\theta_r, \phi_r, \theta_i, \phi_i) = \frac{\int_{\lambda=0}^{\infty} \rho_{\lambda}(\theta_r, \phi_r, \theta_i, \phi_i) I_{\lambda \mathrm{r}}(\theta_i, \phi_i) d\lambda}{\int_{\lambda=0}^{\infty} I_{\lambda \mathrm{r}}(\theta_i, \phi_i) d\lambda}$$

Directional total reflectivity $\rho(\theta_i, \phi_i) = \frac{\int_{\lambda=0}^{\infty} \rho_{\lambda}(\theta_i, \phi_i) I_{\lambda \mathrm{r}}(\theta_i, \phi_i) d\lambda}{\int_{\lambda=0}^{\infty} I_{\lambda \mathrm{r}}(\theta_i, \phi_i) d\lambda}$

Hemispherical total reflectivity ρ

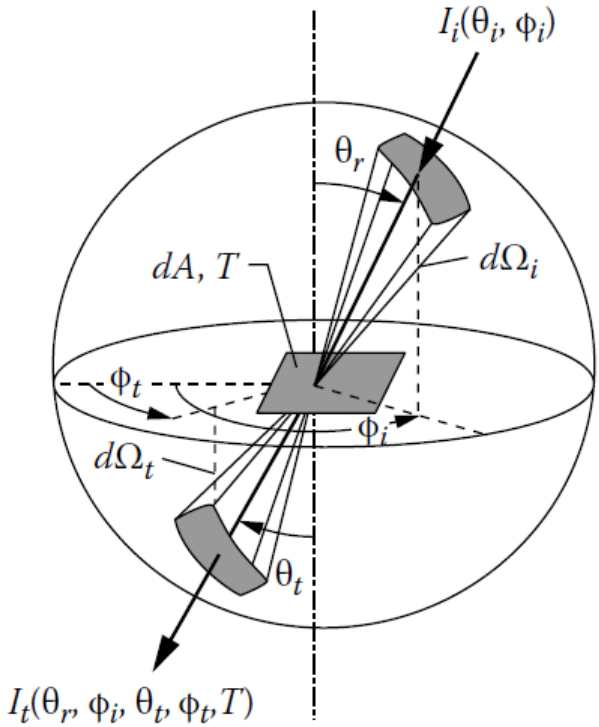
$$\rho = \frac{\int_{\lambda=0}^{\infty} \rho_{\lambda} I_{\lambda \mathrm{i}} d\lambda}{\int_{\lambda=0}^{\infty} I_{\lambda \mathrm{i}} d\lambda}$$

Transmissivities:

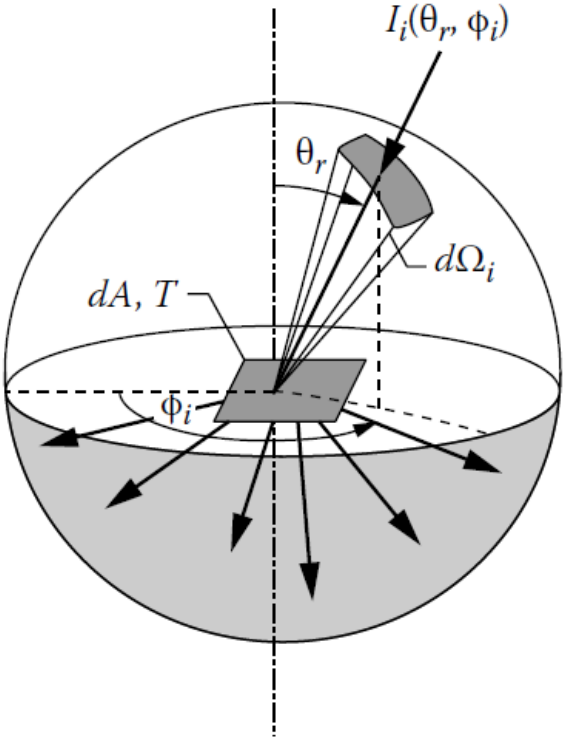
Bidirectional spectral transmissivity $\tau_{\lambda}(\theta_r, \phi_r, \theta_i, \phi_i)$

The ratio of intensity that is transmitted across an interface into the direction (θ_t, ϕ_t) to the incident spectral energy on the interface from a particular incident direction (θ_i, ϕ_i)

$$\tau_{\lambda}(\theta_t, \phi_t, \theta_i, \phi_i) = \frac{I_{\lambda \mathrm{t}}(\theta_t, \phi_t, \theta_i, \phi_i, T)}{I_{\mathrm{i}}(\theta_i, \phi_i) cos(\theta_i) d\Omega_i}$$



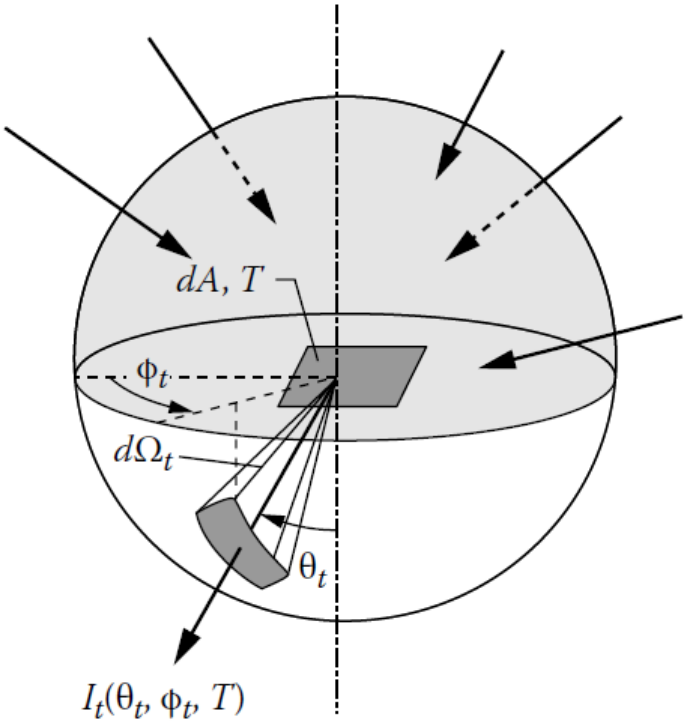
Directional hemispherical spectral transmissivity $\tau_\lambda(\theta_i, \phi_i)$



The directional-hemispherical spectral reflectivity is the fraction of the incident spectral energy that is transmitted across the interface into all directions

$$\tau_\lambda(\theta_i, \phi_i) = \frac{\int_\Omega I_{\lambda t}(\theta_t, \phi_t, \theta_i, \phi_i, T) cos(\theta_t) d\Omega_t}{I_i(\lambda, \theta_i, \phi_i) cos(\theta_i) d\Omega_i} = \int_\Omega \tau_\lambda(\theta_t, \phi_t, \theta_i, \phi_i u\iota) cos(\theta_t) d\Omega_t$$

Hemispherical directional spectral transmissivity $\tau_\lambda(\theta_i, \phi_i)$



$$\tau_\lambda(\theta_t, \phi_t) = \frac{\int_\Omega \tau_\lambda(\theta_t, \phi_t, \theta_i, \phi_i, T) I_{\lambda t}(\theta_t, \phi_t, \theta_i, \phi_i, T) cos(\theta_i) d\Omega_i}{\left(\frac{1}{\pi}\right) \int_\Omega I_{\lambda i}(\theta_i, \phi_i) cos(\theta_i) d\Omega_i}$$

Hemispherical-directional spectral transmissivity (for uniform incident intensity)

$$\tau_\lambda(\theta_t, \phi_t) = \int_\Omega \tau_\lambda(\theta_t, \phi_t, \theta_i, \phi_i) cos(\theta_i) d\Omega_i$$

Hemispherical spectral transmissivity τ_λ

The fraction of spectral energy incident on the surface from all directions that is transmitted across the surface into all directions

$$\tau_\lambda = \frac{\int_\Omega I_{\lambda t}(\theta_t, \phi_t) cos(\theta_t) d\Omega_t}{\int_\Omega I_{\lambda i}(\theta_i, \phi_i) cos(\theta_i) d\Omega_i}$$

Bidirectional total transmissivity $\tau(\theta_t, \phi_t, \theta_i, \phi_i)$

$$\tau(\theta_t, \phi_t, \theta_i, \phi_i, T) = \frac{I_\tau(\theta_t, \phi_t, \theta_i, \phi_i)}{I_\iota(\theta_i, \phi_i) \cos(\theta_i) d\Omega_i}$$

Directional total transmissivity $\tau(\theta_i, \phi_i)$

$$\tau_\lambda(\theta_i, \phi_i) = \frac{\int_\Omega I_{\lambda t}(\theta_t, \phi_t, \theta_i, \phi_i) cos(\theta_t) d\Omega_t}{I_{\lambda i}(\theta_i, \phi_i) cos(\theta_i) d\Omega_i}$$

$$\tau_\lambda(\theta_i, \phi_i) = \int_\Omega \tau_\lambda(\theta_t, \phi_t, \theta_i, \phi_i) cos(\theta_i) d\Omega_i$$

Hemispherical-Directional total transmissivity $\tau(\theta_t, \phi_t)$

$$\tau_\lambda(\theta_t, \phi_t) = \frac{\int_\Omega \tau(\theta_t, \phi_t, \theta_i, \phi_i) I_i(\theta_i, \phi_i) cos(\theta_i) d\Omega_i}{\frac{1}{\pi} I_{\lambda i}(\theta_i, \phi_i) cos(\theta_i) d\Omega_i}$$

$$\tau_\lambda(\theta_t,\phi_t) = \int_\Omega \tau(\theta_t,\phi_t,\theta_i,\phi_i)cos(\theta_i)d\Omega_i$$

Hemispherical total transmissivity τ

The fraction of total energy incident on the surface from all directions that is transmitted across the surface into all directions

$$\tau = \frac{\int_\Omega I_t(\theta_t,\phi_t)cos(\theta_i)d\Omega_i}{\int_\Omega I_i(\theta_i,\phi_i)cos(\theta_i)d\Omega_i}$$

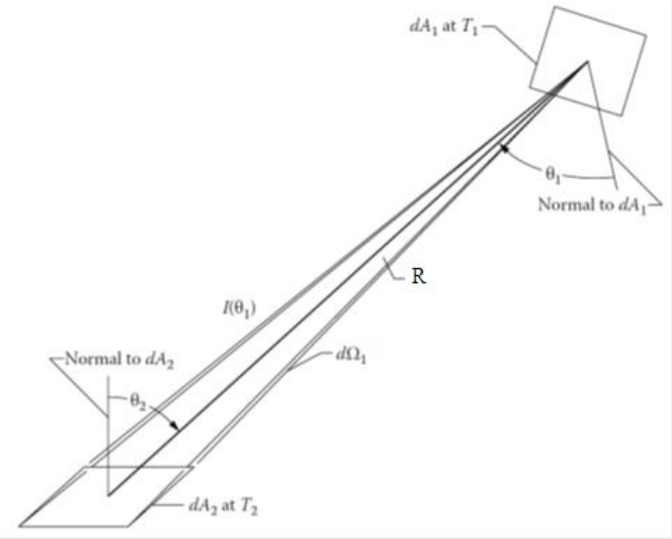
$$\tau = \frac{\int_\Omega \tau(\theta_i,\phi_i)I_i(\theta_i,\phi_i)cos(\theta_i)d\Omega_i}{\int_\Omega I_i(\theta_i,\phi_i)cos(\theta_i)d\Omega_i}$$

For uniform incident intensity:

$$\tau = \int_\Omega \tau(\theta_i,\phi_i)cos(\theta_i)d\Omega_i$$

21 SHAPE FACTOR (CONFUGURATION FACTOR) FOR SURFACES SEPERATED BY A TRANSPARENT MEDIUM

Consider two diferentially small surfaces in space seperated with a medium that does not interact with the radiation.



Energy transfer from one differential surface to the other can be expressed as

$$dQ_{dA_1-dA_2} = I_1 dA_1 \cos(\theta_1) d\Omega_1$$

Radiant energy in wavelength interval

$$dQ_{dA_1-dA_2}d\lambda = I_{\lambda,1}d\lambda dA_1 \cos(\theta_1) d\Omega_1$$

Total radiant energy by integrating for all wavelengths

$$dQ_{dA_1-dA_2} = dA_1 \cos(\theta_1) d\Omega_1 \int_{\lambda=0}^{\infty} I_{\lambda,1}d\lambda$$

The solid angle $d\Omega_1$ is related to the projected area of dA_2 and the distance R between the differential elements by

$$d\Omega_1 = \frac{dA_2 \cos(\theta_2)}{R^2}$$

Substituting into the heat transfer equation gives

$$dQ_{dA_1-dA_2} = I_1 dA_1 \cos(\theta_1) \frac{dA_2 \cos(\theta_2)}{R^2}$$

Analogously heat transfer from surface dA_2 to surface dA_1 can be written as

$$dQ_{dA_2-dA_1} = I_2 dA_2 \cos(\theta_2) \frac{dA_1 \cos(\theta_1)}{R^2}$$

Fraction of energy leaving surface dA_1 that arrives at element dA_2 is defined as shape factpr or configuration factor

$$dF_{dA_1-dA_2} = \frac{I_1 dA_1 \cos(\theta_1) \frac{dA_2 \cos(\theta_2)}{R^2}}{\pi I_1 dA_1} = \frac{\cos(\theta_1)\cos(\theta_2)}{\pi R^2} dA_2 = \frac{\cos(\theta_1)}{\pi} \, d\Omega_1$$

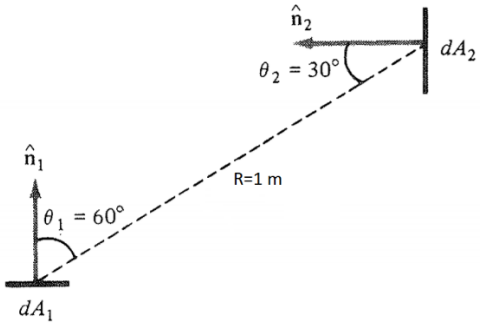
Analogously form from surface dA_2 to surface dA_1 can be written as

$$dF_{dA_2-dA_1} = \frac{I_2 dA_2 \cos(\theta_2) \frac{dA_1 \cos(\theta_1)}{R^2}}{\pi I_2 dA_2} = \frac{\cos(\theta_1)\cos(\theta_2)}{\pi R^2} dA_1 = \frac{\cos(\theta_2)}{\pi} \, d\Omega_2$$

From these two equation, it can be written that

$$dF_{dA_1-dA_2} dA_1 = dF_{dA_2-dA_1} dA_2 = \frac{\cos(\theta_1)\cos(\theta_2)}{\pi R^2} dA_1 dA_2$$

PROBLEM: Two surfaces $dA_1 = 0.05$ m and $dA_2 = 0.08$ m are separated by R=1 m distance and angles of $\theta_1 = 60°$ and $\theta_2 = 30°$ as shown in the figure. Calculate the view factors between surfaces



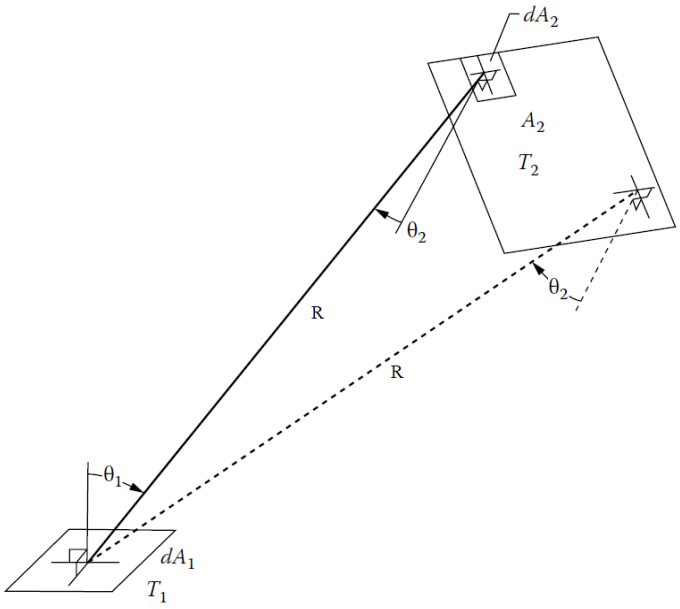
Areas are relatively small so, they can be assumed to be differential areas

$$dF_{dA_1-dA_2} dA_1 = dF_{dA_2-dA_1} dA_2 = \frac{\cos(\theta_1)\cos(\theta_2)}{\pi R^2} dA_1 dA_2$$

$$dF_{dA_1-dA_2} = \frac{\cos(\theta_1)\cos(\theta_2)}{\pi R^2} dA_2 = \frac{\cos(\pi/3)\cos(\pi/6)}{\pi 1^2} 0.08 = 0.011027$$

$$dF_{dA_2-dA_1} = dF_{dA_1-dA_2} \frac{dA_1}{dA_2} = 0.011027 \frac{0.05}{0.08} = 0.00689$$

If the surface is a real surface in one side and radiative energy be transferred from dA_1 to A_2 above equation can be integrated for the area



$$F_{dA_1-A_2} = \int_{A_2} \frac{\cos(\theta_1)\cos(\theta_2)}{\pi R^2} dA_2 = \int_{A_2} dF_{dA_1-dA_2}$$

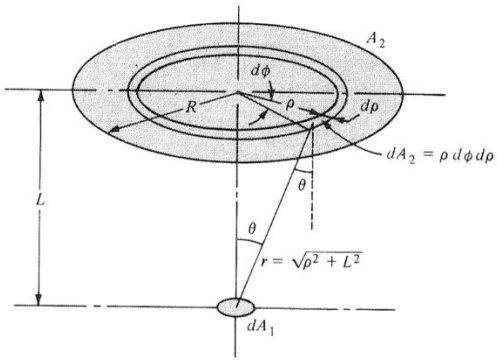
Inverse relation can also be written (configuration factor from A_2 to dA_1)

$$F_{A_2-dA_1} = \frac{dA_1}{A_2} \int_{A_2} \frac{\cos(\theta_1)\cos(\theta_2)}{\pi R^2} dA_2 = \frac{dA_1}{A_2} \int_{A_2} dF_{dA_1-dA_2}$$

or

$$A_2 F_{A_2-dA_1} = dA_1 F_{dA_1-A_2}$$

PROBLEM: a small surface dA_1=0.05 m^2 is located parallel to around plate of radius R=1 m. The distance between plates are L=1 m. Calculate configuration factor $F_{dA_1-A_2}$ between surfaces



This problem can be solved by analytical integration or numerical integration

Analytical solution: Problem can be considered in cylindirical coordinate system.

$$dA_2 = \rho d\phi d\rho$$

$$r = \sqrt{\rho^2 + L^2}$$

$$\cos(\theta) = \frac{L}{r} = \frac{L}{\sqrt{\rho^2 + L^2}}$$

$$F_{dA_1-A_2} = \int_{A_2} \frac{\cos(\theta_1)\cos(\theta_2)}{\pi r^2} dA_2$$

$$F_{dA_1-A_2} = \frac{1}{\pi}\int_{\rho=0}^{R}\int_{\phi=0}^{2\pi}\frac{\frac{L}{\sqrt{\rho^2+L^2}}\frac{L}{\sqrt{\rho^2+L^2}}}{\sqrt{\rho^2+L^2}\sqrt{\rho^2+L^2}}\rho d\phi d\rho = \frac{1}{\pi}\int_{\rho=0}^{R}\int_{\phi=0}^{2\pi}\frac{L^2}{(\rho^2+L^2)^2}\rho d\phi d\rho$$

$$F_{dA_1-A_2} = \frac{2\pi}{\pi}\int_{\rho=0}^{R}\frac{L^2}{(\rho^2+L^2)^2}\rho d\rho = \frac{R^2}{R^2+L^2} = \frac{1}{1+1} = 0.5$$

Numerical solution: (2 dimensional Gauss-Legendre integration is used to solve configuration factor)

```
//=====================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Gauss-Legendre integration with two variables
// variable number of coefficients and roots
// calculated by Legendre polynomials
// Dr. Turhan Coban
// =====================================================
import java.io.*;

abstract class fdi
{
public double a,b;
public fdi(double ai,double bi)
{a=ai;b=bi;}
public fdi()
{a=0;b=1;}
public void setab(double ai,double bi)
{a=ai;b=bi;}
abstract public  double c(double x);
abstract public  double d(double x);
abstract public  double f(double x[]);
}

class f1 extends fdi
{  public double L;
  public f1(double ai,double bi,double Li)
  {super(ai,bi);L=Li;}
  //lower limit of inner integration
  public double c(double x)
  {return 0.0;}
  //upper limit of inner integration
  public double d(double x)
  {return 2.0*Math.PI;}

  public double f(double x[])
  {
  //çözümü istenen fonksiyon
  double ff;
  double x1=(x[0]*x[0]+L*L);
  ff=1.0/Math.PI*L*L/(x1*x1)*x[0];
  return ff;
  }
}

class NA76
{
public static double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{
        //calculates legendre gauss-coefficients as coefficients of the integral
        //for n terms
        double EPS=3.0e-15;
        int m,j,i;
        double z1,z,xm,xl,pp,p3,p2,p1;
        double a[][]=new double[2][n];
        m=(n+1)/2;
        xm=0.5*(x2+x1);
        xl=0.5*(x2-x1);
        for (i=1;i<=m;i++)  {
                z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                do {
                        p1=1.0;
                        p2=0.0;
                        for (j=1;j<=n;j++) {
```

```
                                        p3=p2;
                                        p2=p1;
                                        p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                                }
                                pp=n*(z*p1-p2)/(z*z-1.0);
                                z1=z;
                                z=z1-p1/pp;
                        } while (Math.abs(z-z1) > EPS);
                        a[0][i-1]=xm-xl*z;
                        a[0][n-i]=xm+xl*z;
                        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                        a[1][n-i]=a[1][i-1];
                                }
return a;
}

public static double I(fdi f,int n,int m)
{
//integral f(x,y)dxdy
//integral of a function by using gauss-legendre quadrature
//coefficients are pre-calculated for n,m terms for [-1,1]
//band then utilises variable transform
 int i,j;
 double rn[],cn[];
 double rm[],cm[];
 rn=new double[n];
 rm=new double[m];
 cn=new double[n];
 double a[][]=new double[2][n];
 a=gauss_legendre_coefficients(-1.0,1.0,n);
 rn=a[0];
 cn=a[1];
 a=gauss_legendre_coefficients(-1.0,1.0,m);
 rm=a[0];
 cm=a[1];
double x[],h1,h2,J,a1,b1,c1,d1,k1,k2,JX,Q;
x=new double[2];
a1=f.a;
b1=f.b;
h1=(b1-a1)/2.0;
h2=(b1+a1)/2.0;
J=0;
for(i=0;i<m;i++)
{
JX=0;
x[0]=h1*rm[i]+h2;
d1=f.d(x[0]);
c1=f.c(x[0]);
k1=(d1-c1)/2.0;
k2=(d1+c1)/2.0;
//4=========================
   for(j=0;j<n;j++)
   {
            x[1]=k1*rn[j]+k2;
            Q=f.f(x);
            JX+=cn[j]*Q;
            }
//System.out.println("JX="+JX);
//5=========================
J=J+cm[i]*k1*JX;
}
J=h1*J;
return J;
}


public static double I(fdi f,int n)
{return I(f,n,n);}

public static double I(fdi f)
{return I(f,60,60);}

  public static void main(String args[]) throws IOException
  {
              double R=1.0;
  f1 ff=new f1(0.0,R,R);
  System.out.println("integral of class        f1 : "+I(ff));
```

```
}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" NA76
integral of class       f1 : 0.4999999999999986

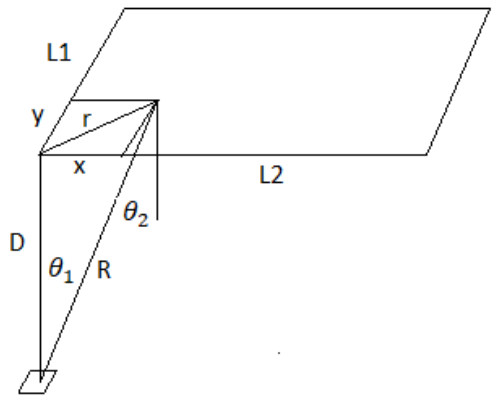
> Terminated with exit code 0.
```

As an alternative integration method, Montecarlo integration can be used
monte-Carlo integration method is relativelly simple compare to Gauss-Legendre integration

```
public class Radiation_Shape3
{public static double F1(double L,double R)
{return R*R/(R*R+L*L);
}
public static double F(double R,double L)
{
int total_ray=0;
int N=10000000;
double teta=Math.atan(R/L);
double tetamax=Math.PI/2;
double teta1=0;
for(int i=0;i<N;i++)
{teta1=tetamax*Math.random();
 if(teta1<=teta) total_ray++;
}
return (double)total_ray/(double)N;
}
public static void main(String arg[])
{
double R=1;
double L=1;
double F1=F(R,L);
double F2=F1(R,L);
System.out.println("F1="+F1+" F2="+F2);
}}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" Radiation_Shape3
F1=0.5002186 F2=0.5
> Terminated with exit code 0.
```

PROBLEM: a small surface dA_1 is located parallel to a rectangular plate of size L_1=1 m and L_2=1 m The distance between plates are D=1 m. Calculate configuration factor $F_{dA_1-A_2}$ between surfaces



$$r = \sqrt{x^2 + y^2}$$
$$R^2 = D^2 + r^2 = D^2 + x^2 + y^2$$
$$R = \sqrt{D^2 + x^2 + y^2}$$
$$\cos(\theta) = \cos(\theta_1) = \cos(\theta_2) = \frac{D}{R}$$

$$\cos(\theta) = \frac{D}{\sqrt{D^2 + x^2 + y^2}}$$

$$F_{dA_1 - A_2} = \int_{A_2} \frac{\cos(\theta_1)\cos(\theta_2)}{\pi R^2} dA_2$$

$$F_{dA_1 - A_2} = \int_{x=0}^{L_2} \int_{y=0}^{L_1} \frac{D}{\sqrt{D^2 + x^2 + y^2}} \frac{D}{\sqrt{D^2 + x^2 + y^2}} \frac{}{\pi(D^2 + x^2 + y^2)} dxdy$$

$$F_{dA_1 - A_2} = \frac{1}{\pi} \int_{x=0}^{L_2} \int_{y=0}^{L_1} \frac{D^2}{(D^2 + x^2 + y^2)^2} dxdy$$

Analytical solution of this problem yields:

$X = \frac{L_1}{D} \quad Y = \frac{L_2}{D}$

$$F_{dA_1 - A_2} = \frac{1}{2\pi} \left(\frac{X}{\sqrt{1 + X^2}} tan^{-1} \left(\frac{Y}{\sqrt{1 + X^2}} \right) + \frac{Y}{\sqrt{1 + Y^2}} tan^{-1} \left(\frac{X}{\sqrt{1 + X^2}} \right) \right)$$

```
//=====================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Gauss-Legendre integration with two variables
// variable number of coefficients and roots
// calculated by Legendre polynomials
// Dr. Turhan Coban
// =====================================================
import java.io.*;

abstract class fdi
{
public double a,b;
public fdi(double ai,double bi)
{a=ai;b=bi;}
public fdi()
{a=0;b=1;}
public void setab(double ai,double bi)
{a=ai;b=bi;}
abstract public  double c(double x);
abstract public  double d(double x);
abstract public  double f(double x[]);
}

class f1 extends fdi
{  public double L1,L2,D;
  public f1(double ai,double bi,double L1i,double L2i,double Di)
  {super(ai,bi);L1=L1i;L2=L2i;D=Di;}

  public double c(double x)
  {return 0.0;}

  public double d(double x)
  {return L1;}

  public double f(double x[])
  {
  //çözümü istenen fonksiyon
  double ff;
  double x1=(x[0]*x[0]+x[1]*x[1]+D*D);
  ff=1.0/Math.PI*D*D/(x1*x1);
  return ff;
  }
}

class NA76C
{
public static double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{
        //calculates legendre gauss-coefficients as coefficients of the integral
```

```
        //for n terms
        double EPS=3.0e-15;
        int m,j,i;
        double z1,z,xm,xl,pp,p3,p2,p1;
        double a[][]=new double[2][n];
        m=(n+1)/2;
        xm=0.5*(x2+x1);
        xl=0.5*(x2-x1);
        for (i=1;i<=m;i++)  {
                z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                do {
                        p1=1.0;
                        p2=0.0;
                        for (j=1;j<=n;j++) {
                                p3=p2;
                                p2=p1;
                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                        }
                        pp=n*(z*p1-p2)/(z*z-1.0);
                        z1=z;
                        z=z1-p1/pp;
                } while (Math.abs(z-z1) > EPS);
                a[0][i-1]=xm-xl*z;
                a[0][n-i]=xm+xl*z;
                a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                a[1][n-i]=a[1][i-1];
                        }
return a;
}

public static double I(fdi f,int n,int m)
{
//integral f(x,y)dxdy
//integral of a function by using gauss-legendre quadrature
//coefficients are pre-calculated for n,m terms for [-1,1]
//band then utilises variable transform
 int i,j;
 double rn[],cn[];
 double rm[],cm[];
 rn=new double[n];
 rm=new double[m];
 cn=new double[n];
 double a[][]=new double[2][n];
 a=gauss_legendre_coefficients(-1.0,1.0,n);
 rn=a[0];
 cn=a[1];
 a=gauss_legendre_coefficients(-1.0,1.0,m);
 rm=a[0];
 cm=a[1];
double x[],h1,h2,J,a1,b1,c1,d1,k1,k2,JX,Q;
x=new double[2];
a1=f.a;
b1=f.b;
h1=(b1-a1)/2.0;
h2=(b1+a1)/2.0;
J=0;
for(i=0;i<m;i++)
{
JX=0;
x[0]=h1*rm[i]+h2;
d1=f.d(x[0]);
c1=f.c(x[0]);
k1=(d1-c1)/2.0;
k2=(d1+c1)/2.0;
//4=========================
   for(j=0;j<n;j++)
   {
        x[1]=k1*rn[j]+k2;
        Q=f.f(x);
        JX+=cn[j]*Q;
        }
//System.out.println("JX="+JX);
//5=========================
J=J+cm[i]*k1*JX;
}
J=h1*J;
return J;
```

```
}
public static double Fa(double L1,double L2,double D)
{double X=L1/D;
 double Y=L2/D;
 double X1=X/(Math.sqrt(1.0+X*X));
 double X2=Y/(Math.sqrt(1.0+X*X));
 double Y1=Y/(Math.sqrt(1.0+Y*Y));
 double Y2=X/(Math.sqrt(1.0+Y*Y));
 double F1=1.0/(2.0*Math.PI)*(X1*Math.atan(X2)+Y1*Math.atan(Y2));
 return F1;
}

public static double I(fdi f,int n)
{return I(f,n,n);}

public static double I(fdi f)
{return I(f,60,60);}

  public static void main(String args[]) throws IOException
  {
                double L1=1.0;
                double L2=1.0;
                double D=1.0;
  f1 ff=new f1(0.0,L2,L1,L2,D);
  System.out.println("F dA1-A2 integral result "+I(ff));
  System.out.println("F dA1-A2_analytical formula="+Fa(L1,L2,D));
}
}
```

---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" NA76C
F dA1-A2 integral result 0.13853160599489256
F dA1-A2_analytical formula=0.13853160599489298

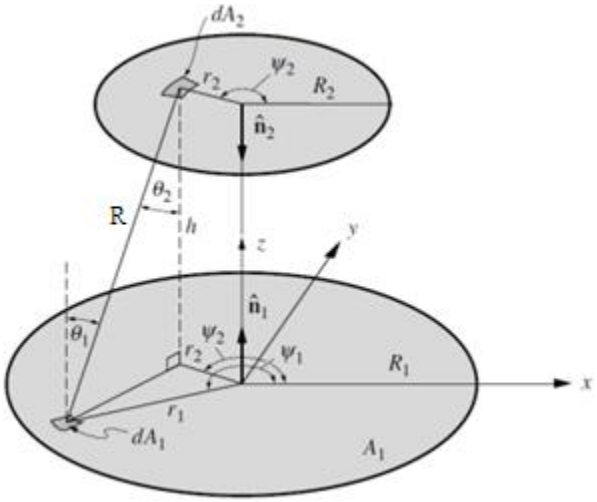
> Terminated with exit code 0.

Configuration factor between two finite area involves multidimensional integration

$$F_{A_1-A_2} = \frac{1}{A_1}\int_{A_1}\int_{A_2}\frac{\cos(\theta_1)\cos(\theta_2)}{\pi R^2}\,dA_2\,dA_1$$

Integrals can be calculated analytically or numerically.

PROBLEM: Two parallel round plates are located as in the figure. Configuration factor of two parallel round plate will be calculated.



Radiation shape factor is the percentage of radiative energy reached to surface A2, emitted from surface A1. If the Radius of plates are R1 and R2, and the distance in between is h, the radiation shape factor equation is given as:

$$F_{A1-A2} = \frac{1}{\pi^2 R_1^2} \int_{r1=0}^{R1} \int_{r1=0}^{R2} \int_{\psi_1=0}^{2\pi} \int_{\psi_2=0}^{2\pi} \frac{h^2 r_1 r_2 d\psi_2 d\psi_1 dr_2 dr_1}{\left[h^2 + r_1^2 + r_2^2 - 2r_1 r_2 \cos(\psi_1 - \psi_2)\right]}$$

This equation has an analytical solution as:

$F_{A1-A2} = \frac{R_2}{R_1}\left(X - \sqrt{X^2 - 1}\right)$ $\qquad X = \frac{h^2 + R_1^2 + R_2^2}{2R_1 R_2}$. In the following program numerical solution by Gauss-Legendre integration is obtained and compared to the analytical solution.

four dimensional Gauss-Legendre integration

```
 //=================================================
// Numerical Analysis package in java
// 4 dimensional Gauss-Legendre integration
// written by: Hamza TAŞ
// =================================================
import java.io.*;
abstract class fdi4
{
public double a,b;
public fdi4(double ai,double bi)
{a=ai;b=bi;}
public fdi4()
{a=0;b=1;}
public void setab(double ai,double bi)
{a=ai;b=bi;}
abstract public  double c(double x);
abstract public  double d(double x);
abstract public  double alfa(double x[]);
abstract public  double beta(double x[]);
abstract public  double gama(double x[]);
abstract public  double teta(double x[]);
abstract public  double f(double x[]);
}

class f1 extends fdi4
{ double h,R1,R2;
  public f1(double hi,double R1i, double R2i)
  {h=hi;R1=R1i;R2=R2i;a=0;b=R1;}

  public double c(double x)
  {return 0;}

  public double d(double x)
  {return R2;}

   public double alfa(double x[])
  {return 0;}

  public double beta(double x[])
  {return 2.0*Math.PI;}

  public double gama(double x[])
  {return 0.0;}

  public double teta(double x[])
  {return 2.0*Math.PI;}

  public double f(double x[])
  {
  //çözümü istenen fonksiyon
```

```
  double ff;
  double z=(h*h+x[0]*x[0]+x[1]*x[1]-2.0*x[0]*x[1]*Math.cos(x[2]-x[3]));
  ff=1.0/(Math.PI*Math.PI*R1*R1)*h*h*x[0]*x[1]/(z*z);
  return ff;
  }
  public double  F12(double h,double R1,double R2)
  {double X=(h*h+R1*R1+R2*R2)/(2.0*R1*R2);
   double F=R2/R1*(X-Math.sqrt(X*X-1));
   return F;
  }
}

class NA78A
{
public static double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{
        //calculates legendre gauss-coefficients as coefficients of the integral for n terms
        double EPS=3.0e-15;
        int m,j,i;
        double z1,z,xm,xl,pp,p3,p2,p1;
    double a[][]=new double[2][n];//a[0][i]=x[i]   a[1][i]=w[i]
        m=(n+1)/2;
        xm=0.5*(x2+x1);
        xl=0.5*(x2-x1);
        for (i=1;i<=m;i++)  {
                z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                do {
                        p1=1.0;
                        p2=0.0;
                        for (j=1;j<=n;j++) {
                                p3=p2;
                                p2=p1;
                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                        }
                        pp=n*(z*p1-p2)/(z*z-1.0);
                        z1=z;
                        z=z1-p1/pp;
                } while (Math.abs(z-z1) > EPS);
                a[0][i-1]=xm-xl*z;
                a[0][n-i]=xm+xl*z;
                a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                a[1][n-i]=a[1][i-1];
                        }
return a;
}
public static double I(fdi4 f,int n,int m,int p,int s)
{
 int i,j,k,h;
 double rn[],cn[];
 double rm[],cm[];
 double rp[],cp[];
 double rs[],cs[];
 rn=new double[n];
 rm=new double[m];
 rp=new double[p];
 rs=new double[s];
 cn=new double[n];
 cm=new double[m];
 cp=new double[p];
 cs=new double[s];
 double a[][]=new double[2][n];
 a=gauss_legendre_coefficients(-1.0,1.0,n);
 rn=a[0];
 cn=a[1];
 a=gauss_legendre_coefficients(-1.0,1.0,m);
 rm=a[0];
 cm=a[1];
 a=gauss_legendre_coefficients(-1.0,1.0,p);
```

```
 rp=a[0];
 cp=a[1];
 a=gauss_legendre_coefficients(-1.0,1.0,s);
 rs=a[0];
 cs=a[1];
double x[],h1,h2,J,a1,b1,c1,d1,k1,k2,JX,JY,JZ,Q,l1,l2,alfa1,beta1,gama1,teta1,m1,m2;
x=new double[4];
a1=f.a;
b1=f.b;
h1=(b1-a1)/2.0;
h2=(b1+a1)/2.0;
J=0;
for(i=0;i<m;i++)
{
JX=0;
x[0]=h1*rm[i]+h2;
d1=f.d(x[0]);
c1=f.c(x[0]);
k1=(d1-c1)/2.0;
k2=(d1+c1)/2.0;
     for(j=0;j<n;j++)
     {
              JY=0;
     x[1]=k1*rn[j]+k2;
     beta1=f.beta(x);
     alfa1=f.alfa(x);
     l1=(beta1-alfa1)/2.0;
     l2=(beta1+alfa1)/2.0;
          for(k=0;k<p;k++)
          {
                   JZ=0;
                   x[2]=l1*rp[k]+l2;
                   teta1=f.teta(x);
                   gama1=f.gama(x);
                   m1=(teta1-gama1)/2.0;
                   m2=(teta1+gama1)/2.0;
                        for (h=0;h<s;h++)
                        {
                                 x[3]=m1*rs[h]+m2;
                        Q=f.f(x);
                        JZ+=cs[h]*Q;
                }
        JY+=cp[k]*m1*JZ;
        }
    JX+=cn[j]*l1*JY;
     }
J=J+cm[i]*k1*JX;
}
J=h1*J;
return J;
}
public static double I(fdi4 f,int n)
{return I(f,n,n,n,n);}
public static double I(fdi4 f)
{return I(f,60,60,60,60);}
  public static void main(String args[]) throws IOException
  {
  double h=1.0;
  double R1=1.0;
  double R2=1.0;
  f1 ff=new f1(h,R1,R2);
  System.out.println("FA1-A2= "+I(ff)+" four dimensional auss Legendre integration ");
  System.out.println("FA1-A2= "+ff.F12(h,R1,R2)+ " analytical solution ");
  }
}
```

```
 ---------- Capture Output ----------
> "c:\java\bin\javaw.exe" NA78A
```

```
FA1-A2= 0.3819660112501026 four dimensional Gauss-Legendre integration
FA1-A2= 0.3819660112501051 analytical solution

> Terminated with exit code 0.
```

Area Integration Method for Configuration factor calculations:
General definition for the configuration integration in cartesian coordinate system

$$F_{A_i-A_j} = \frac{1}{A_1}\int_{A_1}\int_{A_6}\frac{\cos(\theta_i)\cos(\theta_j)}{\pi R^2}\,dA_i\,dA_j$$

$$R^2 = (x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2$$

$$R = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}$$

Vector pointing from original to an arbitrary pont (x,y,z): $r = \left[x\vec{\imath} + y\vec{\jmath} + z\vec{k}\right]$

Surface normal for area i: $n_i = \left[l_i\vec{\imath} + m_i\vec{\jmath} + n_i\vec{k}\right] = \left[\cos(\alpha)_i\vec{\imath} + \cos(\beta)_i\vec{\jmath} + \cos(\gamma)_i\vec{k}\right]$

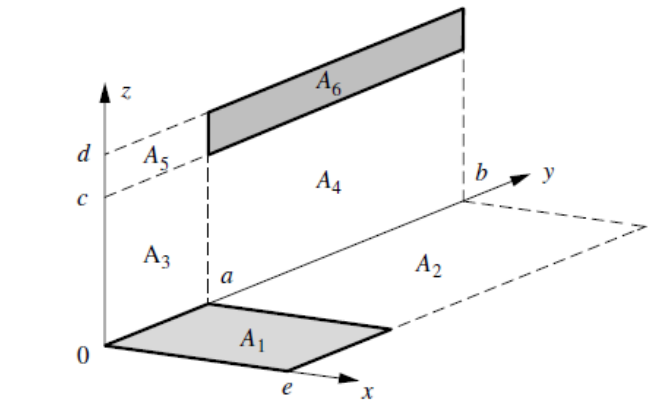
Surface normal for area j: $n_j = \left[l_j\vec{\imath} + m_j\vec{\jmath} + n_j\vec{k}\right] = \left[\cos(\alpha)_j\vec{\imath} + \cos(\beta)_j\vec{\jmath} + \cos(\gamma)_j\vec{k}\right]$

Where l,m,n are direction cosines of the surface normal vector

$$\cos(\theta_i) = \left[(x_j - x_i)l_i + (y_j - y_i)m_i + (z_j - z_i)n_i\right]/R$$

$$\cos(\theta_j) = \left[(x_i - x_j)l_j + (y_i - y_j)m_j + (z_i - z_j)n_j\right]/R$$

PROBLEM:



Calculate shape factor $F_{A_1-A_6}$ for the given configuration

$$F_{A_1-A_6} = \frac{1}{A_1}\int_{A_1}\int_{A_6}\frac{\cos(\theta_1)\cos(\theta_6)}{\pi R^2}\,dA_6\,dA_1$$

$$R^2 = (x_6 - x_1)^2 + (y_6 - y_1)^2 + (z_6 - z_1)^2$$

$x_6 = 0 \quad z_1 = 0$

$$R^2 = x_1^2 + (y_6 - y_1)^2 + z_6^2$$

$\overrightarrow{n_1} = \vec{k} \quad \overrightarrow{n_6} = \vec{\imath}$

$\cos(\theta_1) = \frac{z_6 - z_1}{R} = \frac{z_6 - 0}{R} = \frac{z_6}{R}$

$\cos(\theta_6) = \frac{x_1 - x_6}{R} = \frac{x_1 - 0}{R} = \frac{x_1}{R}$

$A_1F_{A_1-A_6} =$

$\int_{x_1=0}^{e}\int_{y_1=0}^{a}\int_{y6=a}^{b}\int_{z_6=c}^{d}\frac{x_1 z_6\,dz_6\,dy_6\,dy_1\,dx_1}{\pi\left[x_1^2+(y_6-y_1)^2+z_6^2\right]^2}$

General definition in numerical 4 dimensional integration formula

$$I = \int_a^b\int_{c(x)}^{d(x)}\int_{\alpha(x,y)}^{\beta(x,y)}\int_{\gamma(x,y,z)}^{\theta(x,y,z)} f(x,y,z,t)\,dt\,dz\,dy\,dx$$

```
//=====================================================
// Numerical Analysis package in java
// Gauss integration of four dimensional function
// =====================================================
import java.io.*;
abstract class fdi4
{
public double a,b;
public fdi4(double ai,double bi)
{a=ai;b=bi;}
public fdi4()
{a=0;b=1;}
public void setab(double ai,double bi)
{a=ai;b=bi;}
```

```
abstract public  double c(double x);
abstract public  double d(double x);
abstract public  double alfa(double x[]);
abstract public  double beta(double x[]);
abstract public  double gama(double x[]);
abstract public  double teta(double x[]);
abstract public  double f(double x[]);
}

class f2 extends fdi4
{ double a,b,c,d,e;
  public f2(double ai,double bi, double ci,double di,double ei)
  {a=ai;b=bi;c=ci;d=di;e=ei;}

  public double c(double x)
  {return 0;}

  public double d(double x)
  {return a;}

   public double alfa(double x[])
  {return a;}

  public double beta(double x[])
  {return b;}

  public double gama(double x[])
  {return c;}

  public double teta(double x[])
  {return d;}

  public double f(double x[])
  {
  //function to be integrated
  double ff;
  // x1=x[0] y1=x[1] y6=x[2] z6=x[3]
  double R2=x[0]*x[0]+(x[1]-x[2])*(x[1]-x[2])+x[3]*x[3];
  double z=(x[0]*x[3])/(R2*R2);
  return z;
  }}
class NA78B
{
public static double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{
            //calculates legendre gauss-coefficients as coefficients of the integral for n terms
            double EPS=3.0e-15;
            int m,j,i;
            double z1,z,xm,xl,pp,p3,p2,p1;
    double a[][]=new double[2][n];//a[0][i]=x[i]   a[1][i]=w[i]
            m=(n+1)/2;
            xm=0.5*(x2+x1);
            xl=0.5*(x2-x1);
            for (i=1;i<=m;i++)  {
                        z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                        do {
                                    p1=1.0;
                                    p2=0.0;
                                    for (j=1;j<=n;j++) {
                                                p3=p2;
                                                p2=p1;
                                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                                    }
                                    pp=n*(z*p1-p2)/(z*z-1.0);
                                    z1=z;
                                    z=z1-p1/pp;
                        } while (Math.abs(z-z1) > EPS);
                        a[0][i-1]=xm-xl*z;
                        a[0][n-i]=xm+xl*z;
                        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                        a[1][n-i]=a[1][i-1];
                                    }
return a;
}
public static double I(fdi4 f,int n,int m,int p,int s)
{
 int i,j,k,h;
```

```
 double rn[],cn[];
 double rm[],cm[];
 double rp[],cp[];
 double rs[],cs[];
 rn=new double[n];
 rm=new double[m];
 rp=new double[p];
 rs=new double[s];
 cn=new double[n];
 cm=new double[m];
 cp=new double[p];
 cs=new double[s];
 double a[][]=new double[2][n];
 a=gauss_legendre_coefficients(-1.0,1.0,n);
 rn=a[0];
 cn=a[1];
 a=gauss_legendre_coefficients(-1.0,1.0,m);
 rm=a[0];
 cm=a[1];
 a=gauss_legendre_coefficients(-1.0,1.0,p);
 rp=a[0];
 cp=a[1];
 a=gauss_legendre_coefficients(-1.0,1.0,s);
 rs=a[0];
 cs=a[1];
double x[],h1,h2,J,a1,b1,c1,d1,k1,k2,JX,JY,JZ,Q,l1,l2,alfa1,beta1,gama1,teta1,m1,m2;
x=new double[4];
a1=f.a;
b1=f.b;
h1=(b1-a1)/2.0;
h2=(b1+a1)/2.0;
J=0;
for(i=0;i<m;i++)
{
JX=0;
x[0]=h1*rm[i]+h2;
d1=f.d(x[0]);
c1=f.c(x[0]);
k1=(d1-c1)/2.0;
k2=(d1+c1)/2.0;
     for(j=0;j<n;j++)
     {
               JY=0;
     x[1]=k1*rn[j]+k2;
     beta1=f.beta(x);
     alfa1=f.alfa(x);
     l1=(beta1-alfa1)/2.0;
     l2=(beta1+alfa1)/2.0;
          for(k=0;k<p;k++)
          {
                    JZ=0;
                    x[2]=l1*rp[k]+l2;
                    teta1=f.teta(x);
                    gama1=f.gama(x);
                    m1=(teta1-gama1)/2.0;
                    m2=(teta1+gama1)/2.0;
                         for (h=0;h<s;h++)
                         {
                                   x[3]=m1*rs[h]+m2;
                         Q=f.f(x);
                         JZ+=cs[h]*Q;
               }
        JY+=cp[k]*m1*JZ;
        }
    JX+=cn[j]*l1*JY;
     }
J=J+cm[i]*k1*JX;
}
J=h1*J;
return J;
}
public static double I(fdi4 f,int n)
{return I(f,n,n,n,n);}
public static double I(fdi4 f)
{return I(f,60,60,60,60);}
  public static void main(String args[]) throws IOException
  {
```

```
double a=1.0;
double b=3.0;
double c=1.0;
double d=2.0;
double e=2.0;
f2 ff=new f2(a,b,c,d,e);
System.out.println("FA1-A6= "+I(ff,100)+" numeric four dimensional gauss-legendre integral");
}}
```

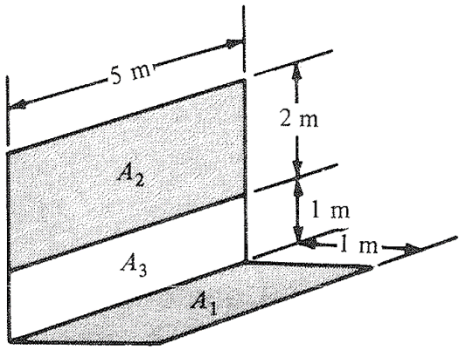
```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" NA78B
FA1-A6= 0.0778761490540283 numeric four dimensional gauss-legendre integral
> Terminated with exit code 0.
```

For an enclosure it can be written that $\sum_{j=1}^{N} F_{A_i-A_j} = 1$ and $A_i F_{A_i-A_j} = A_j F_{A_j-A_i}$

For sub-surfaces $A_j = \sum_{k=1}^{K} A_{(j,k)}$ $F_{A_i-A_j} = \sum_{k=1}^{K} F_{A_i-A_{(j,k)}}$

This properties can be used to calculate shape factor of the unknown parts.

PROBLEM :



Determine configuration factor $F_{A_1-A_2}$ for the given shape, assuming that shape factor formula for two perpendicular touching surface is known($F_{A_1-A_3}$ and $F_{A_1-A_3+A_2}$)

Solution:

$$F_{A_1-A_2} + F_{A_1-A_3} = F_{A_1-A_3+A_2}$$
$$F_{A_1-A_2} = F_{A_1-A_3+A_2} - F_{A_1-A_3}$$

Contour Integration

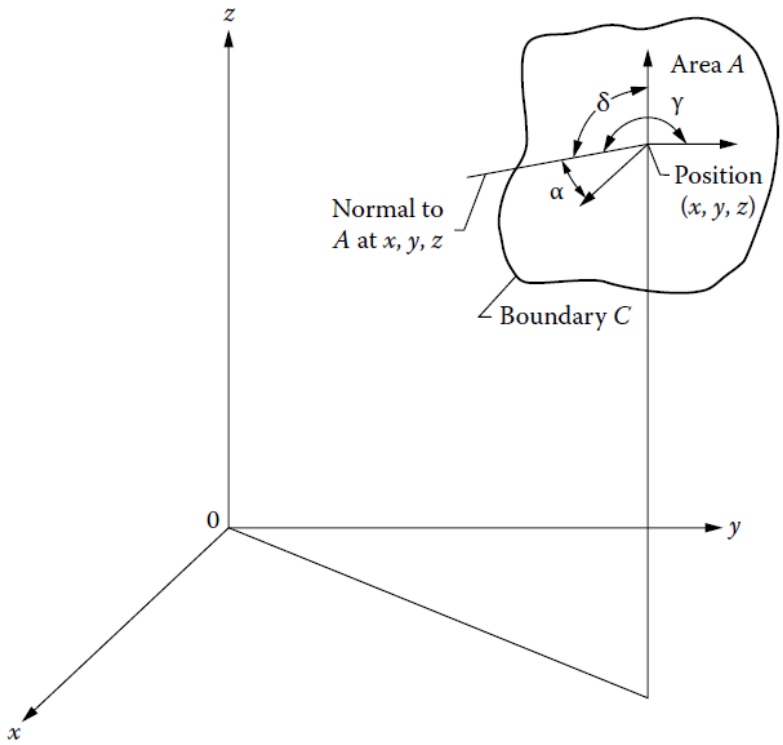
Stoke's theorem gives a way to reduce multiple integration to single integration. Consider the area A in the figüre with countour boundary C. It is assumed that boundary is continuous or picewise- continuous. Letus also consider an arbitrary point on the surface as P(x,y,z). Normal to this point can be written as

Vector pointing from original to an arbitrary point $P(\mathrm{x},\mathrm{y},\mathrm{z}) = \left[x\vec{\imath} + y\vec{\jmath} + z\vec{k}\right]$

Surface normal for area: $n = \left[l\vec{\imath} + m\vec{\jmath} + n\vec{k}\right] = \left[\cos(\alpha)\vec{\imath} + \cos(\beta)\vec{\jmath} + \cos(\gamma)\vec{k}\right]$

Where l,m,n are direction cosines of the surface normal vector, α, β and γ are angles of normal with x,y and z axeses respectively. And also let P,Q, and W be any twice differentiable functions of (x,y,z). Stokes' theorem in three dimensions states that

$$\oint_C (Pdx + Qdy + Wdz) = \int_A \left[\left(\frac{\partial W}{\partial y} - \frac{\partial Q}{\partial z}\right) l + \left(\frac{\partial P}{\partial z} - \frac{\partial W}{\partial x}\right) m + \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) n\right] dA$$



Now this theory can be apply to solve configuration factor problems. Let us consider **configuration factors between a differential area and a finite area**

$$\cos(\theta_i) = \left[(x_j - x_i)l_i + (y_j - y_i)m_i + (z_j - z_i)n_i\right]/R$$
$$\cos(\theta_j) = \left[(x_i - x_j)l_j + (y_i - y_j)m_j + (z_i - z_j)n_j\right]/R$$

And

$$R^2 = (x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2$$
$$R = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}$$
$$F_{dA_i - A_j} = \int_{A_2} \frac{\cos(\theta_i)\cos(\theta_j)}{\pi R^2} dA_j$$

$$F_{dA_i - A_j}$$
$$= \frac{1}{\pi} \int_{A_j} \frac{\left[(x_j - x_i)l_i + (y_j - y_i)m_i + (z_j - z_i)n_i\right]\left[(x_i - x_j)l_j + (y_i - y_j)m_j + (z_i - z_j)n_j\right]}{R^4} dA_j$$

Now let $f = \frac{\left[(x_j - x_i)l_i + (y_j - y_i)m_i + (z_j - z_i)n_i\right]}{\pi R^4}$ then equation becomes

$$F_{dA_i - A_j} = \int_{A_j} \left[(x_i - x_j)fl_j + (y_i - y_j)fm_j + (z_i - z_j)fn_j\right] dA_j$$

Comparison of this equation with Stokes' equation gives

$$\left(\frac{\partial W}{\partial y} - \frac{\partial Q}{\partial z}\right) = (x_i - x_j)f$$
$$\left(\frac{\partial P}{\partial z} - \frac{\partial W}{\partial x}\right) = (y_i - y_j)f$$
$$\left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) = (z_i - z_j)f$$

Useful solution of these equations[98,99] are:

$$P = \frac{-m_i(z_j - z_i) + n_i(y_j - y_i)}{2\pi R^2}$$

$$Q = \frac{l_i(z_j - z_i) - n_i(x_j - x_i)}{2\pi R^2}$$

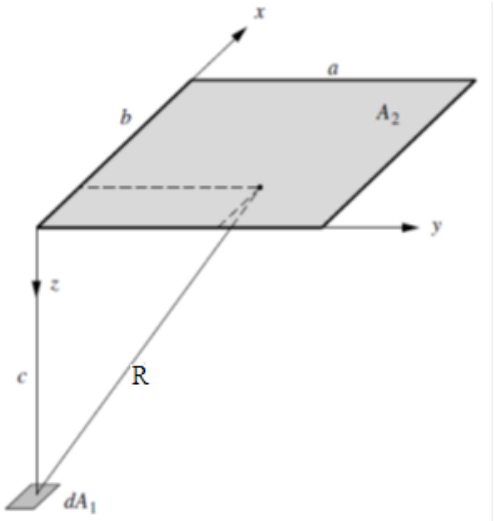
$$W = \frac{-l_i(y_j - y_i) + m_i(x_j - x_i)}{2\pi R^2}$$

$$F_{dA_i - A_j} = \oint_{C_j} (P dx_j + Q dy_j + W dz_j)$$

$$F_{dA_i - A_j} = \oint_{C_j} \left(\frac{-m_i(z_j - z_i) + n_i(y_j - y_i)}{2\pi R^2} dx_j + \frac{l_i(z_j - z_i) - n_i(x_j - x_i)}{2\pi R^2} dy_j + \frac{-l_i(y_j - y_i) + m_i(x_j - x_i)}{2\pi R^2} dz_j \right)$$

$$F_{dA_i - A_j} = \frac{l_i}{2\pi} \oint_{C_j} \frac{(z_j - z_i)dy_j - (y_j - y_i)dz_j}{R^2} + \frac{m_i}{2\pi} \oint_{C_j} \frac{(x_j - x_i)dz_j - (z_j - z_i)dx_j}{R^2} + \frac{n_i}{2\pi} \oint_{C_j} \frac{(y_j - y_i)dx_j - (x_j - x_i)dy_j}{R^2}$$

PROBLEM:



The shape in the left side is given. By using contour integration method find the shape factor analytically and numerically

$l_1 = m_1 = 0\ \overrightarrow{n_1} = -\vec{k} \quad n_1 = -1 \quad F_{dA_1 - A_2} = \frac{-1}{2\pi} \oint_{C_2} \frac{ydx - xdy}{R^2} \quad R^2 = x^2 + y^2 + c^2$

$$F_{dA_1 - A_2} = \frac{-1}{2\pi} \left[\int_{x=0}^{b} \frac{y}{R^2} dx \Big|_{y=0} + \int_{y=0}^{a} \frac{-x}{R^2} dy \Big|_{x=b} + \int_{x=b}^{0} \frac{y}{R^2} dx \Big|_{y=a} + \int_{x=a}^{0} \frac{y}{R^2} dy \Big|_{x=0} \right]$$

For the configuration $F_{dA_1 - A_2}$ given as

$$F_{dA_1 - A_2} = \frac{1}{2\pi} \left(\int_{y=0}^{a} \frac{bdy}{b^2 + y^2 + c^2} + \int_{x=0}^{b} \frac{adx}{x^2 + a^2 + c^2} \right)$$

Analytical solution gives

$F_{dA_1 - A_2} = \frac{1}{2\pi} \left(\frac{b}{\sqrt{b^2 + c^2}} tan^{-1} \left(\frac{a}{\sqrt{b^2 + c^2}} \right) + \frac{a}{\sqrt{a^2 + c^2}} tan^{-1} \left(\frac{b}{\sqrt{a^2 + c^2}} \right) \right)$

Calculate first equation by using numerical integration and compare it with the analytical result

a= 1 m b= 1 m and c= 1 m

```
//=====================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Gander and Gautschi adaptive Gauss-Lobatto integral
// Dr. Turhan Coban
// =====================================================
import java.io.*;

class fx extends f_x
{ double a,b,c;
  public fx(double ai,double bi,double ci)
  {a=ai;b=bi;c=ci;}
  public double func(double x)
  {return 1.0/(2.0*Math.PI)*a/(x*x+a*a+c*c);}
}
class fy extends f_x
{ double a,b,c;
  public fy(double ai,double bi,double ci)
  {a=ai;b=bi;c=ci;}
  public double func(double y)
  {return 1.0/(2.0*Math.PI)*b/(b*b+y*y+c*c);}
}

class NA71
{
public static double adaptive_Lobatto_integral(f_x ff,double a,double b)
{   int i;
            double m=(a+b)/2.0; double h=(b-a)/2.0;
    double alpha=Math.sqrt(2.0/3.0);
    double beta=1.0/Math.sqrt(5.0);
    double x1,x2,x3;
    x1=0.942882415695479719056351758431857202 32;
    x2=0.641853342345781305781235541329031883 54;
    x3=0.236383199662149880282223773492052925 99;
    double A,B,C,D,E,F,G;
    A=0.015827191973480183087169986733305510591;
    B=0.094273840218850045531282505077108171960;
    C=0.155071987336585396253635979802102986 80;
    D=0.188821573960182454420005339372971671 25;
    E=0.199773405226858526792068022066488402 46;
    F=0.224926465333339527016017687996395080 76;
    G=0.242611071901407733799640957903256352 33;
            double x[]={a,(m-x1*h),(m-alpha*h),(m-x2*h),(m-beta*h),(m-x3*h),m,(m+x3*h),(m+beta*h),(m+x2*h),(m+alpha*h),(m+x1*h),b};
            int n=x.length;
            double y[]=new double[n];
            for(i=0;i<n;i++) y[i]=ff.func(x[i]);
            double fa=y[0];
            double fb=y[12];
    double i2=(h/6.0)*(y[0]+y[12]+5.0*(y[4]+y[8]));
    double i1=(h/1470.0)*(77.0*(y[0]+y[12])+432.0*(y[2]+y[10])+
     625.0*(y[4]+y[8])+672.0*y[6]);
    double is=h*(A*(y[0]+y[12])+B*(y[1]+y[11])+C*(y[2]+y[10])+D*(y[3]+y[9])+E*(y[4]+y[8])+F*(y[5]+y[7])+G*y[6]);
    int s=(int)(is/Math.abs(is));
    if(s==0) s=1;
    double erri1=Math.abs(i1-is);
    double erri2=Math.abs(i2-is);
    double R;
    R=erri1/erri2;
    //if(R>0 && R<1) tol=tol/R;
    //is=s*Math.abs(is)*tol/eps;
    if(is==0) is=b-a;
  double Q=0;
  Q=Lobatto(ff,a,b,fa,fb,is);
  return Q;
}

public static double Lobatto(f_x ff,double a,double b,double fa,double fb,double is)
{
  double h=(b-a)/2;
  double m=(a+b)/2;
  double alpha=Math.sqrt(2.0/3.0);
  double beta=1.0/Math.sqrt(5.0);
  double mll=m-alpha*h;
  double ml=m-beta*h;
```

```
  double mr=m+beta*h;
  double mrr=m+alpha*h;
  double x[]={mll,ml,m,mr,mrr};
  int n=x.length;
  double y[]=new double[n];
  for(int i=0;i<n;i++) y[i]=ff.func(x[i]);
  double fmll=y[0];
  double fml=y[1];
  double fm=y[2];
  double fmr=y[3];
  double fmrr=y[4];
  double Q;
  double i2=(h/6)*(fa+fb+5*(fml+fmr));
  double i1=(h/1470)*(77*(fa+fb)+432*(fmll+fmrr)+625*(fml+fmr)+672*fm);
  boolean c1,c2,c3;
  double w1=i1-i2;;
  double w2=is;
  double w3=w2+w1;
  c1=(w3==w2);
  c2=(mll<=a);
  c3=(b<=mrr);
  if( c1|| c2 ||c3 )
  {
   if (((m <= a) || (b<=m)))
   {
      System.out.println("istenilen toleransa ulaşılamadı");
   }
   Q=i1;
  }
  else
  {
    Q=Lobatto(ff,a,mll,fa,fmll,is) + Lobatto(ff,mll,ml,fmll,fml,is)+
      Lobatto(ff,ml,m,fml,fm,is)   + Lobatto(ff,m,mr,fm,fmr,is)+
      Lobatto(ff,mr,mrr,fmr,fmrr,is)+Lobatto(ff,mrr,b,fmrr,fb,is);
  }
  return Q;
}
  public static double F(double a,double b, double c)
  {   double A=b/Math.sqrt(b*b+c*c);
      double B=a/Math.sqrt(b*b+c*c);
      double C=a/Math.sqrt(a*a+c*c);
      double D=b/Math.sqrt(a*a+c*c);
                double f=1.0/(2.0*Math.PI)*(A*Math.atan(B)+C*Math.atan(D));
  return f;
  }
  public static void main(String args[]) throws IOException
  {double a=1;
  double b=1;
  double c=1;
  fx x1=new fx(a,b,c);
  fy y1=new fy(a,b,c);
  double I=adaptive_Lobatto_integral(y1,0.0,a)+adaptive_Lobatto_integral(x1,0.0,b);
  System.out.println("F_dA1_A2 numeric : "+I);
  System.out.println("F_dA1_A2 analytical : "+F(a,b,c));
  }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" NA71
F_dA1_A2 numeric : 0.13853160599489303
F_dA1_A2 analytical : 0.13853160599489298

> Terminated with exit code 0.
```

Contour integration for configuration factor between finite areas

In order to calculate contour integration for finite areas, the previously find contour integration between differential and finite area eqution integrated for area

$$A_i F_{A_i - A_j} = A_j F_{A_j - A_i} = \int_{A_i} F_{dA_i - A_j} \, dA_i$$

$$A_i F_{A_i-A_j} = A_j F_{A_j-A_i} = \frac{1}{2\pi}\oint_{C_j}\int_{A_i}\left[\frac{-m_i(z_j-z_i)+n_i(y_j-y_i)}{R^2}dx_j + \frac{l_i(z_j-z_i)-n_i(x_j-x_i)}{R^2}dy_j + \frac{-l_i(y_j-y_i)+m_i(x_j-x_i)}{R^2}dz_j)\right]dA_i$$

$$A_i F_{A_i-A_j} = A_j F_{A_j-A_i} = \frac{1}{2\pi}\oint_{C_j}\left[\left(\int_{A_i}\frac{-m_i(z_j-z_i)+n_i(y_j-y_i)}{R^2}dA_i\right)dx_j + \left(\int_{A_i}\frac{l_i(z_j-z_i)-n_i(x_j-x_i)}{R^2}dA_i\right)dy_j + \left(\int_{A_i}\frac{-l_i(y_j-y_i)+m_i(x_j-x_i)}{R^2}dA_i\right)dz_j)\right]$$

where the integrals have been rearranged, and dx_j, dy_j, and dz_j have been factored out since they are independent of the area integration over $A1$. Stokes' theorem is applied in turn to each of the three area integrals. Consider area integral

$$\int_{A_i}\frac{-m_i(z_j-z_i)+n_i(y_j-y_i)}{R^2}dA_i$$

Integral in stokes theorem gives

$$\left(\frac{\partial W}{\partial y}-\frac{\partial Q}{\partial z}\right)=0$$

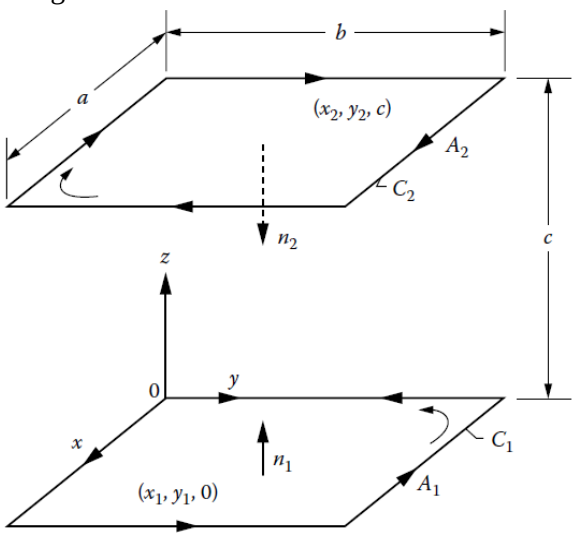
$$\left(\frac{\partial P}{\partial z}-\frac{\partial W}{\partial x}\right)=\frac{-(z_j-z_i)}{R^2}$$

$$\left(\frac{\partial Q}{\partial x}-\frac{\partial P}{\partial y}\right)=\frac{(y_j-y_i)}{R^2}$$

Solution of this set of partial differential equations is P=ln(R) Q=0 and W=0 and

$$A_i F_{A_i-A_j} = A_j F_{A_j-A_i} = \frac{1}{2\pi}\oint_{C_1}\oint_{C_2}\left[\ln(R)\,dx_j dx_i + \ln(R)\,dy_j dy_i + \ln(R)\,dz_j dz_i\right]$$

PROBLEM: Find configuration factor by using contour integration for two parallel rectangular shown in the figure



$$F_{A_1-A_2} = F_{A_2-A_i} = \frac{1}{2\pi ab}\oint_{C_1}\oint_{C_2}\begin{bmatrix}\ln\left([(x_2-x_1)^2+(y_2-y_1)^2+c^2]^{\frac{1}{2}}\right)dx_2dx_1 + \\ \ln\left([(x_2-x_1)^2+(y_2-y_1)^2+c^2]^{1/2}\right)dy_2dy_1\end{bmatrix}$$

Opening contour integral for area 2

$F_{A_1-A_2} = F_{A_2-A_i}$

$$= \frac{1}{2\pi ab} \oint_{C_1} \left[\left(\int_{x_2=0}^{a} \ln\left([(x_2-x_1)^2 + (b-y_1)^2 + c^2]^{\frac{1}{2}} \right) dx_2 \right) dx_1 + \left(\int_{x_2=a}^{0} \ln\left([(x_2-x_1)^2 + (0-y_1)^2 + c^2]^{\frac{1}{2}} \right) dx_2 \right) dx_1 + \left(\int_{y_2=0}^{b} \ln\left([(0-x_1)^2 + (y_2-y_1)^2 + c^2]^{1/2} \right) dy_2 \right) dy_1 + \left(\int_{y_2=b}^{0} \ln\left([(a-x_1)^2 + (y_2-y_1)^2 + c^2]^{1/2} \right) dy_2 \right) dy_1 \right]$$

Opening contour integral for area 2

$F_{A_1-A_2} = F_{A_2-A_1}$

$$= \frac{1}{2\pi ab} \left[\int_{x_1=0}^{a} \left(\int_{x_2=0}^{a} \ln\left([(x_2-x_1)^2 + (b-0)^2 + c^2]^{\frac{1}{2}} \right) dx_2 \right) dx_1 + \int_{x_1=0}^{a} \left(\int_{x_2=a}^{0} \ln\left([(x_2-x_1)^2 + (0-0)^2 + c^2]^{\frac{1}{2}} \right) dx_2 \right) dx_1 + \int_{x_1=a}^{0} \left(\int_{x_2=0}^{a} \ln\left([(x_2-x_1)^2 + (b-b)^2 + c^2]^{\frac{1}{2}} \right) dx_2 \right) dx_1 + \int_{x_1=a}^{0} \left(\int_{x_2=a}^{0} \ln\left([(x_2-x_1)^2 + (0-b)^2 + c^2]^{\frac{1}{2}} \right) dx_2 \right) dx_1 + \int_{y_1=0}^{b} \left(\int_{y_2=0}^{b} \ln\left([(0-a)^2 + (y_2-y_1)^2 + c^2]^{\frac{1}{2}} \right) dy_2 \right) dy_1 + \int_{y_1=0}^{b} \left(\int_{y_2=b}^{0} \ln\left([(a-a)^2 + (y_2-y_1)^2 + c^2]^{\frac{1}{2}} \right) dy_2 \right) dy_1 + \int_{y_1=b}^{0} \left(\int_{y_2=0}^{b} \ln\left([(0-0)^2 + (y_2-y_1)^2 + c^2]^{1/2} \right) dy_2 \right) dy_1 + \int_{y_1=b}^{0} \left(\int_{y_2=b}^{0} \ln\left([(a-0)^2 + (y_2-y_1)^2 + c^2]^{1/2} \right) dy_2 \right) dy_1 \right]$$

$$F_{A_1-A_2} = F_{A_2-A_1} = \frac{1}{\pi ab} \left[\int_{x_1=0}^{a} \left(\int_{x_2=0}^{a} \ln\left(\frac{[(x_2-x_1)^2 + b^2 + c^2]^{\frac{1}{2}}}{[(x_2-x_1)^2 + c^2]^{\frac{1}{2}}} \right) dx_2 \right) dx_1 + \int_{y_1=0}^{b} \left(\int_{y_2=0}^{b} \ln\left(\frac{[a^2 + (y_2-y_1)^2 + c^2]^{\frac{1}{2}}}{[(y_2-y_1)^2 + c^2]^{1/2}} \right) dx_2 \right) dx_1 \right]$$

Analytical solution of the equation:

$X = \frac{a}{c} \quad Y = \frac{b}{c}$

$$F_{A_1-A_2} = F_{A_2-A_1} = \frac{2}{\pi XY} \left\{ ln\left[\frac{(1+X^2)(1+Y^2)}{(1+X^2+Y^2)} \right]^{1/2} + X\sqrt{1+Y^2}\, tan^{-1}\left(\frac{X}{\sqrt{1+Y^2}} \right) + Y\sqrt{1+X^2}\, tan^{-1}\left(\frac{Y}{\sqrt{1+X^2}} \right) - Xtan^{-1}(X) - Ytan^{-1}(Y) \right\}$$

In order to solve numerical integration Gauss-Legendre integration formula is used.

QUADRATURES: GAUSS-LEGENDRE

A problem known from antique times, is to create a rectangle with the same area with a polynomial. IT is called a quadrature problem.

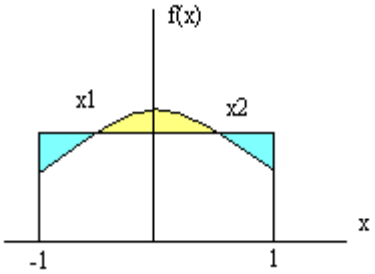


Figure a rectangle with the same area with a polynomial

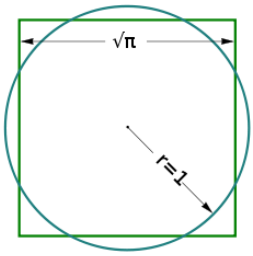


Figure A rectangle with the same area with a circle

If point x1 and x2 points intersecting the polynomial and rectangle is known, area calculation of the polynomial can be interchange with the area calculation of the rectangle. Or in more general means instead of integration process of summation can be substituted. As a general definition:

$$I_w(-1,1) \cong \int_{-1}^{1} f(x)w(x)dx \cong \sum_{k=1}^{n} c_k f(x_k)$$

Can be written. In this equation w(x) is the weight factor. In the first of a such formulation, Gauss-Legendre integral formulation, weight factor can be taken as 1.

$$I_w(-1,1) \cong \int_{-1}^{1} f(x)dx \cong \sum_{k=1}^{n} c_k f(x_k)$$

If this integration is solved for a general polynamial equation. Following relation is found:

$$\int_{-1}^{1} x^k dx \cong \frac{1-(-1)^{k+1}}{k+1}$$

This equation can be written in open form as:

$c_1 x_1^k + c_2 x_2^k + ... + c_n x_n^k = \frac{1-(-1)^{k+1}}{k+1}, \quad 0 \le k \le 2n$ For a special case of n=2 equation becomes :

$f(x)=a_0+a_1x+a_2x^2+a_3x^3$

$$I_w(-1,1) \cong \int_{-1}^{1} f(x)dx \cong c_1 f(x_1) + c_2 f(x_2) = \int_{-1}^{1} (a_0 + a_1 x + a_2 x^2 + a_3 x^3)dx$$

$$c_1(a_0 + a_1x_1 + a_2x_1^2 + a_3x_1^3) + c_2 f(a_0 + a_1x_2 + a_2x_2^2 + a_3x_2^3) = a_0\int_{-1}^{1} dx + a_1\int_{-1}^{1} x dx + a_2\int_{-1}^{1} x^2 dx + a_3\int_{-1}^{1}$$

$c_1+c_2=2$

$c_1 x_1+c_2 x_2=0$

$c_1 x_1^2+c_2 x_2^2=2/3$

$c_1 x_1^3+c_2 x_2^3=0$

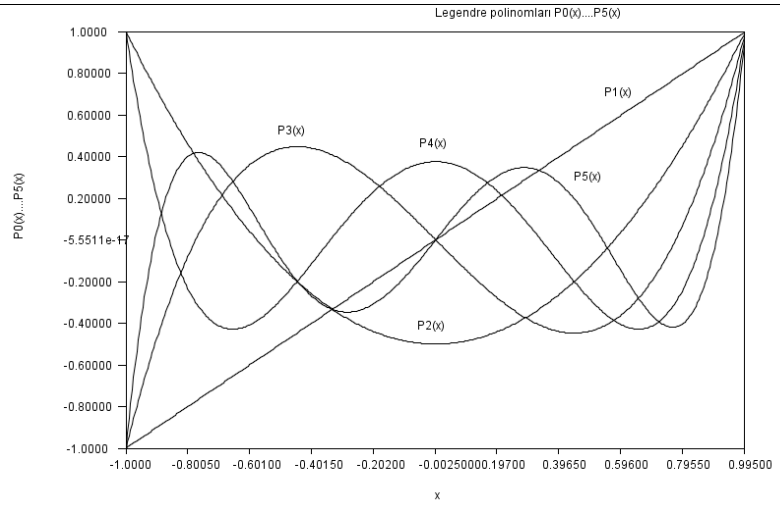
If this system of equation is solved roots and coefficients can be found as:

Roots: $x_{1,2} = \pm 1/\sqrt{3}$ and Coefficients : $c_{1,2}=1$ bulunur.

The general solution of this problem can be defined with the Legendre polynomials. Legendre polynomials has a general definition as:

$(k+1)P_{k+1}(x) = (2k+1)xP_k(x) - kP_{k-1}(x), \quad k \ge 1$ and $P_0(x)=1$,$P_1(x)=x$

The first 6 Legendre polynomials are giving in graphic form.



The roots of Legendre Polynomials are the roots of the Gauss-Legendre integration Formula. For example if $P_2(x)$ value is calculated from the above general form:

$P_2(x)=(3x^2-1)/2$. The root of this is equal to $x_{1,2}=\pm 1/\sqrt{3}$. In general roots can be calculated from Legendre polynomials as follows:

$$\theta_{n,k}=\frac{n-k+3/4}{n+1/2}\pi$$

$$x_{n,k}=\left[1-\frac{1}{8n^2}+\frac{1}{8n^3}-\frac{1}{384n^4}\left(39-\frac{28}{\sin^2\theta_{n,k}}\right)\right]\cos\theta_{n,k}+E(n^{-5})$$

This equation contains an error level of $\mathbf{n^{-5}}$. In order to decrease the error Newton-Raphson root finding method can be usefull.

$$x_{\nu+1}=x_\nu-\frac{1-x_\nu^2}{n}\frac{P_n(x_\nu)}{P_{n-1}(x_\nu)-x_\nu P_n(x_\nu)}$$

After finding the roots coeficients can be calculated as

$$c_1x_1^{2j}+c_2x_2^{2j}+....+c_nx_n^{2j}=\frac{2}{2j+1},\quad 0\le j\le n \text{ or}$$

$$c_k=\frac{2(1-x_k^2)}{[nP_{n-1}(x_k)]^2}$$

As a last concept, consider that Gauss-Legendre integration limits are -1 and 1
Region x=[-1,1] can be converted to z=[a,b] by changing the variables

$$z=\left(\frac{b-a}{2}\right)x+\left(\frac{b+a}{2}\right)=\alpha x+\beta$$

$$\alpha=\left(\frac{b-a}{2}\right)\quad \beta=\left(\frac{b+a}{2}\right)$$

In this case, integration Formula will became :

$$I_w(a,b)\cong\int_a^b f(z)dx\cong\alpha\sum_{k=1}^{n}c_k\,f(\alpha x_k+\beta)$$

First 10 Gauss-Legendre integration formulas are given in the table

Table Gauss – Legendre integration roots and coefficients

N	x_k	c_k
2	-0.577350269189625	1.000000000000000
	0.577350269189625	1.000000000000000
3	-0.774596669241483	0.555555555555552

	0.000000000000000	**0.888888888888888**
	0.774596669241483	**Format long**
4	**-0.861136311594052**	**0.347854845137447**
	-0.339981043584856	**0.652145154862546**
	0.339981043584856	**0.652145154862546**
	0.861136311594052	**0.347854845137447**
5	**-0.906179845938664**	**0.236926885056181**
	-0.538469310105683	**0.478628670499366**
	0.000000000000000	**0.568888888888888**
	0.538469310105683	**0.478628670499366**
	0.906179845938664	**0.236926885056181**
6	**-0.932469514203152**	**0.171324492379162**
	-0.661209386466264	**0.360761573048138**
	-0.238619186083196	**0.467913934572691**
	0.238619186083196	**0.467913934572691**
	0.661209386466264	**0.360761573048138**
	0.932469514203152	**0.171324492379162**
7	**-0.949107912342758**	**0.129484966168862**
	-0.741531185599394	**0.279705391489276**
	-0.405845151377397	**0.381830050505119**
	0.000000000000000	**0.417959183673469**
	0.405845151377397	**0.381830050505119**
	0.741531185599394	**0.279705391489276**
	0.949107912342758	**0.129484966168862**
8	**-0.960289856497536**	**0.101228536290369**
	-0.796666477413626	**0.222381034453374**
	-0.525532409916329	**0.313706645877887**
	-0.183434642495649	**0.362683783378362**
	0.183434642495649	**0.362683783378362**
	0.525532409916329	**0.313706645877887**
	0.796666477413626	**0.222381034453374**
	0.960289856497536	**0.101228536290369**
9	**-0.968160239507626**	**0.081274388361569**
	-0.836031107326635	**0.180648160694857**
	-0.613371432700590	**0.260610696402935**
	-0.324253423403808	**0.312347077040002**
	0.000000000000000	**0.330239355001259**
	0.324253423403808	**0.312347077040002**
	0.613371432700590	**0.260610696402935**
	0.836031107326635	**0.180648160694857**
	0.968160239507626	**0.081274388361569**
10	**-0.973906528517171**	**0.066671344308684**
	-0.865063366688984	**0.149451349150580**
	-0.679409568299024	**0.219086362515982**
	-0.433395394129247	**0.269266719309996**
	-0.148874338981631	**0.295524224714752**
	0.148874338981631	**0.295524224714752**
	0.433395394129247	**0.269266719309996**
	0.679409568299024	**0.219086362515982**
	0.865063366688984	**0.149451349150580**
	0.973906528517171	**0.066671344308684**

MULTIDIMENSIONAL INTEGRATION BY PARAMETRIC METHODS

A two dimensional integration

$$\mathrm{I(D)} = \int\int_D f(x,y)dydx$$

Where D is a region in xy plane. If this region is defined as $a \le x \le b$ and $c(x) \le y \le d(x)$, one way two calculate two dimensional integration is to repeat one dimensional integration twice. In the first step intehration function g(x) is calculated as an integration in y dimension and then function g(x) is solved to get integration result.

$$I = \int_a^b \int_{c(x)}^{d(x)} f(x,y)dydx$$

$$g(x) = \int_{c(x)}^{d(x)} f(x,y)dy, \quad a \le x \le b$$

$$I(D) = \int_{a}^{b} g(x)dx$$

Any integration formulation can be used for this purpose. All it has to be done is to evaluate integration Formula twice onece for each independent variable. As our first example Gauss-Legendre formulation will be used to evaluate double integration. Two code is defined. In the first one Gauss-Legendre coefficients are defined as constants, in the second one they are calculated through Legendre polynomials. In the two dimensional integration x dimesion limits are still given as constant values, but y dimension limits c(x) and d(x) should be given as functions for proper evaluation of the integral.

$$I = \int_{a}^{b} \int_{c(x)}^{d(x)} f(x,y)dy = \int_{a}^{b} \frac{d(x)-c(x)}{2} \sum_{J=1}^{n} c_{n,j} f(x, \frac{(d(x)-c(x))r_{n,j} + d(x) + c(x)}{2})dx$$

In this equation $r_{n,j}$ root values and $c_{n,j}$ are equation coefficients. These values can be calculated from Legendre polynomials.

```
//=======================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Gauss-Legendre integration with two variables
// variable number of coefficients and roots
// calculated by Legendre polynomials
// Dr. Turhan Coban
// =======================================================
import java.io.*;

abstract class fdi
{
public double a,b;
public fdi(double ai,double bi)
{a=ai;b=bi;}
public fdi()
{a=0;b=1;}
public void setab(double ai,double bi)
{a=ai;b=bi;}
abstract public  double c(double x);
abstract public  double d(double x);
abstract public  double f(double x[]);
}

class fx extends fdi
{ double a1,b1,c1;
  public fx(double ai,double bi,double a1i,double b1i,double c1i)
  {super(ai,bi);a1=a1i;b1=b1i;c1=c1i;}

  public double c(double x)
  {return 0.0;}

  public double d(double x)
  {return b1;}

  public double f(double x[])
  {
  //çözümü istenen fonksiyon
  double ff;
  double xx=x[1]-x[0];
  double xx2=xx*xx;
  ff=1.0/(Math.PI*a1*b1)*Math.log(Math.sqrt(xx2+b1*b1+c1*c1)/Math.sqrt(xx2+c1*c1));
  return ff;
  }
}
```

```
class fy extends fdi
{ double a1,b1,c1;
  public fy(double ai,double bi,double a1i,double b1i,double c1i)
  {super(ai,bi);a1=a1i;b1=b1i;c1=c1i;}

  public double c(double y)
  {return 0.0;}

  public double d(double y)
  {return a1;}

  public double f(double y[])
  {
  //çözümü istenen fonksiyon
  double ff;
  double yy=y[1]-y[0];
  double yy2=yy*yy;
  ff=1.0/(Math.PI*a1*b1)*Math.log(Math.sqrt(a1*a1+yy2+c1*c1)/Math.sqrt(yy2+c1*c1));
  return ff;
  }
}
class NA76
{ public static double two_parallel_rect_shape_factor(double x[])
  {double X=x[0]/x[2];
  double Y=x[1]/x[2];
  double x1=Math.sqrt(((1+X*X)*(1+Y*Y))/(1+X*X+Y*Y));
  double x2=X*Math.sqrt(1+Y*Y);
  double x3=Math.sqrt(1+Y*Y);
  double x4=Y*Math.sqrt(1+X*X);
  double x5=Math.sqrt(1+X*X);
  double z=2.0/(Math.PI*X*Y)*(Math.log(x1)+x2*Math.atan2(X,x3)+x4*Math.atan2(Y,x5)
  -X*Math.atan(X)-Y*Math.atan(Y));
  return z;
  }
public static double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{
        //calculates legendre gauss-coefficients as coefficients of the integral
        //for n terms
        double EPS=3.0e-15;
        int m,j,i;
        double z1,z,xm,xl,pp,p3,p2,p1;
        double a[][]=new double[2][n];
        m=(n+1)/2;
        xm=0.5*(x2+x1);
        xl=0.5*(x2-x1);
        for (i=1;i<=m;i++)  {
                z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                do {
                        p1=1.0;
                        p2=0.0;
                        for (j=1;j<=n;j++) {
                                p3=p2;
                                p2=p1;
                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                        }
                        pp=n*(z*p1-p2)/(z*z-1.0);
                        z1=z;
                        z=z1-p1/pp;
                } while (Math.abs(z-z1) > EPS);
                a[0][i-1]=xm-xl*z;
                a[0][n-i]=xm+xl*z;
                a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                a[1][n-i]=a[1][i-1];
                        }
return a;
}

public static double I(fdi f,int n,int m)
{
//integral f(x,y)dxdy
//integral of a function by using gauss-legendre quadrature
//coefficients are pre-calculated for n,m terms for [-1,1]
//band then utilises variable transform
 int i,j;
 double rn[],cn[];
 double rm[],cm[];
 rn=new double[n];
```

```
 rm=new double[m];
 cn=new double[n];
 double a[][]=new double[2][n];
 a=gauss_legendre_coefficients(-1.0,1.0,n);
 rn=a[0];
 cn=a[1];
 a=gauss_legendre_coefficients(-1.0,1.0,m);
 rm=a[0];
 cm=a[1];
double x[],h1,h2,J,a1,b1,c1,d1,k1,k2,JX,Q;
x=new double[2];
a1=f.a;
b1=f.b;
h1=(b1-a1)/2.0;
h2=(b1+a1)/2.0;
J=0;
for(i=0;i<m;i++)
{
JX=0;
x[0]=h1*rm[i]+h2;
d1=f.d(x[0]);
c1=f.c(x[0]);
k1=(d1-c1)/2.0;
k2=(d1+c1)/2.0;
//4=========================
    for(j=0;j<n;j++)
    {
                x[1]=k1*rn[j]+k2;
                Q=f.f(x);
                JX+=cn[j]*Q;
                }
//System.out.println("JX="+JX);
//5=========================
J=J+cm[i]*k1*JX;
}
J=h1*J;
return J;
}

public static double I(fdi f,int n)
{return I(f,n,n);}

public static double I(fdi f)
{return I(f,60,60);}

  public static void main(String args[]) throws IOException
  {
  double a,b,c;
  a=1;b=1;c=1;
  fx fx1=new fx(0.0,b,a,b,c);
  fy fy1=new fy(0.0,a,a,b,c);
  double Ix=I(fx1);
  double Iy=I(fy1);
  double I=Ix+Iy;
  double x[]={a,b,c};
  System.out.println("a="+a+" m");
  System.out.println("b="+b+" m");
  System.out.println("c="+c+" m");
  double F=two_parallel_rect_shape_factor(x);
  System.out.println("analytical solution F="+F);
  System.out.println("numerical integration solution F="+I);
}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" NA76
a=1.0 m
b=1.0 m
c=1.0 m
analytical solution F=0.19982489569838746
numerical integration solution F=0.19982489569838668

> Terminated with exit code 0.
```

```
---------- Capture Output ----------
```

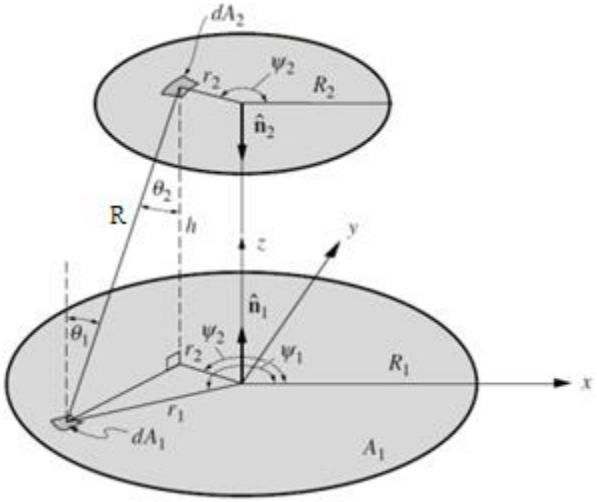
```
> "D:\co\java\bin\java.exe" NA76
a=0.5 m
b=0.5 m
c=1.0 m
analytical solution F=0.0685895888185526
numerical integration solution F=0.0685895888185522

> Terminated with exit code 0.
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" NA76
a=1.0 m
b=1.0 m
c=0.5 m
analytical solution F=0.4152532835771469
numerical integration solution F=0.41525328357714536

> Terminated with exit code 0.
```

PROBLEM:



Consider configuration factor between two parallel co-axial disks of radius R_1 and R_2, respectively. Use contour integration to calculate it.

$ds = dx\vec{\imath} + dy\vec{j} + dz\vec{k}$

$ds_1 = R_1 d\,\psi_1(-sin\psi_1\vec{\imath} + cos\psi_1\vec{j})$

$ds_2 = R_2 d\,\psi_2(-sin\psi_2\vec{\imath} + cos\psi_2\vec{j})$

$ds_1 ds_2 = R_1 R_2 d\psi_1 d\psi_2 (sin\psi_1 sin\psi_2 + cos\psi_1 cos\psi_2)$

$ds_1 ds_2 = R_1 R_2 cos(\psi_1 - \psi_2) d\psi_1 d\psi_2$

$R = [h^2 + R_1^2 + R_2^{-2} - 2R_1 R_2 cos(\psi_1 - \psi_2)]^{1/2}$

$F_{A_1-A_2} = \frac{R_1 R_2}{2\pi(\pi R_1^2)} \int_{\psi_1=0}^{2\pi} \int_{\psi_2=0}^{-2\pi} ln[h^2 + R_1^2 + R_2^2 - 2R_1 R_2 cos(\psi_1 - \psi_2)]^{1/2} cos(\psi_1 - \psi_2) d\psi_1 d\psi_2$

Integrating by parts

$F_{A_1-A_2} = -\frac{R_2}{\pi R_1}\left[sin\psi ln[h^2 + R_1^2 + R_2^2 - 2R_1 R_2 cos\psi]^{1/2}\Big|_{\psi=0}^{2\pi} - R_1 R_2 \int_{\psi=0}^{2\pi} \frac{sin^2\psi}{h^2+R_1^2+R_2^{-2}-2R_1 R_2 cos\psi} d\,\psi\right]$

$F_{A_1-A_2} = \frac{R_2}{\pi R_1} \int_{\psi=0}^{2\pi} \frac{sin^2\psi}{X - cos\psi} d\,\psi$ where $X = \frac{h^2+R_1^2+R_2^2}{2R_1 R_2}$

Analytical solution of the integral:

$F_{A_1-A_2} = \frac{R_2}{R_1}\left(X - \sqrt{X^2 - 1}\right)$

Lets now evaluate the double integral and analytical function and compare the results obtained.

For numerical calculations take $R_1 = 1$ $R_2 = 1$ and h=1

```
//=====================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
```

```
// Gauss-Legendre integration with two variables
// variable number of coefficients and roots
// calculated by Legendre polynomials
// Dr. Turhan Coban
// ======================================================
import java.io.*;

abstract class fdi
{
public double a,b;
public fdi(double ai,double bi)
{a=ai;b=bi;}
public fdi()
{a=0;b=1;}
public void setab(double ai,double bi)
{a=ai;b=bi;}
abstract public  double c(double x);
abstract public  double d(double x);
abstract public  double f(double x[]);
}

class fx extends fdi
{ public double a,b,c,d,h,R1,R2;
  public fx(double ai,double bi,double ci,double di,double hi,double R1i,double R2i)
  {super(ai,bi);a=ai;b=bi;c=ci;d=di;h=hi;R1=R1i;R2=R2i;}

  public double c(double x)
  {return c;}

  public double d(double x)
  {return d;}

  public double f(double x[])
  {
  //çözümü istenen fonksiyon
  double ff;
  double fi1=x[0];double fi2=x[1];
  double R=Math.sqrt(h*h+R1*R1+R2*R2-2.0*R1*R2*Math.cos(fi1-fi2));
  ff=R1*R2/(2.0*Math.PI)/(Math.PI*R1*R1)*Math.log(R)*Math.cos(fi1-fi2);
  return ff;
  }
}

class NA76C
{ public static double two_parallel_disk_shape_factor(double h,double R1,double R2)
  {double X=(h*h+R1*R1+R2*R2)/(2.0*R1*R2);
  double z=R2/R1*(X-Math.sqrt(X*X-1));
  return z;
  }
public static double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{
        //calculates legendre gauss-coefficients as coefficients of the integral
        //for n terms
        double EPS=3.0e-15;
        int m,j,i;
        double z1,z,xm,xl,pp,p3,p2,p1;
        double a[][]=new double[2][n];
        m=(n+1)/2;
        xm=0.5*(x2+x1);
        xl=0.5*(x2-x1);
        for (i=1;i<=m;i++)  {
                z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                do {
                        p1=1.0;
                        p2=0.0;
                        for (j=1;j<=n;j++) {
                                p3=p2;
                                p2=p1;
                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                        }
                        pp=n*(z*p1-p2)/(z*z-1.0);
                        z1=z;
                        z=z1-p1/pp;
                } while (Math.abs(z-z1) > EPS);
                a[0][i-1]=xm-xl*z;
                a[0][n-i]=xm+xl*z;
                a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
```

```
                                a[1][n-i]=a[1][i-1];
                                        }
return a;
}

public static double I(fdi f,int n,int m)
{
//integral f(x,y)dxdy
//integral of a function by using gauss-legendre quadrature
//coefficients are pre-calculated for n,m terms for [-1,1]
//band then utilises variable transform
 int i,j;
 double rn[],cn[];
 double rm[],cm[];
 rn=new double[n];
 rm=new double[m];
 cn=new double[n];
 double a[][]=new double[2][n];
 a=gauss_legendre_coefficients(-1.0,1.0,n);
 rn=a[0];
 cn=a[1];
 a=gauss_legendre_coefficients(-1.0,1.0,m);
 rm=a[0];
 cm=a[1];
double x[],h1,h2,J,a1,b1,c1,d1,k1,k2,JX,Q;
x=new double[2];
a1=f.a;
b1=f.b;
h1=(b1-a1)/2.0;
h2=(b1+a1)/2.0;
J=0;
for(i=0;i<m;i++)
{
JX=0;
x[0]=h1*rm[i]+h2;
d1=f.d(x[0]);
c1=f.c(x[0]);
k1=(d1-c1)/2.0;
k2=(d1+c1)/2.0;
//4=========================
    for(j=0;j<n;j++)
    {
                x[1]=k1*rn[j]+k2;
                Q=f.f(x);
                JX+=cn[j]*Q;
                }
//System.out.println("JX="+JX);
//5=========================
J=J+cm[i]*k1*JX;
}
J=h1*J;
return J;
}


public static double I(fdi f,int n)
{return I(f,n,n);}

public static double I(fdi f)
{return I(f,60,60);}

  public static void main(String args[]) throws IOException
  {
  double h,R1,R2;
  h=1;R1=1;R2=1;
  fx fx1=new fx(0.0,2.0*Math.PI,0.0,-2.0*Math.PI,h,R1,R2);
  double I=I(fx1);
  System.out.println("h="+h+" m");
  System.out.println("R1="+R1+" m");
  System.out.println("R2="+R2+" m");
  double F=two_parallel_disk_shape_factor(h,R1,R2);
  System.out.println("analytical solution F="+F);
  System.out.println("numerical integration solution F="+I);
  }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" NA76C
h=1.0 m
R1=1.0 m
R2=1.0 m
analytical solution F=0.3819660112501051
numerical integration solution F=0.38196601125010377

> Terminated with exit code 0.
```

A general Configuration factor example ShapeFactor.java

```
// equation source:
// http://www.thermalradiation.net/tablecon.html
// Radiation Heat Transfer 6th Edition John R. Howell, Robert Siegel,M. Pınar Mengüç
// Programmed by: M. Turhan ÇOBAN
import javax.swing.*;
import java.util.*;
import javax.imageio.ImageIO;
import java.net.URL;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.image.BufferedImage;
import java.io.*;
public class shapeFactor
{ String folder="images\\";
  input6 i1;
  char g[]=gr();
  char Alpha=g[0];
  char Beta=g[1];
  char Eta=g[6];
  char Ksi=g[5];
  char Teta=g[7];
  char Fi='\u03A6';
  char Delta=g[3];
  char Lambda=g[10];
  char pi=g[15];
  double PI=Math.PI;
  public String[] namedd;
  public String[] namedf;
  public String[] nameff;
public static double ct(double x)
{return Math.cos(x);}
public static double st(double x)
{return Math.sin(x);}
public static double tt(double x)
{return Math.tan(x);}
public static double att(double x)
{return Math.atan(x);}
public static double act(double x)
{return Math.acos(x);}
public static double ast(double x)
{return Math.asin(x);}
public static double sqrt(double x)
{return Math.sqrt(x);}
public static double pow(double x,double y)
{return Math.pow(x,y);}
public static double ln(double x)
{return ln(x);}

public static char[] gr()
{char p[]=new char[25];
 int i=0;
 String s[]=new String[50];
 for(char c='\u03B1';c<='\u03C9';c++)
 {p[i]=c;s[i]="i="+i+"c="+c;i++;}
 //Text.printT(s);
 return p;
}
// Differential-Differential
// Factors Between Differential Elements
// Section A
//=======================================
public double[] input_dd(int no)
{ String s=name_dd(no)+"\n     enter :  ";
  String s1[]=varible_dd(no);
```

```
  for(int i=0;i<s1.length;i++) {s+=s1[i]+" ";}
  i1=new input6(s,"Confiuration factor differential area to differential area",folder+"dd"+no+".jpg");
  double a[]=i1.vericiktisi();
  return a;
 }
public String name_dd(int n)
{String x[]={"","Two elemental areas in arbitrary configuration",
"Two elemental areas lying on parallel lines",
"Elemental area of any length z to infinitely long parallel strip of differential width; plane containing element does not intersect strip",
"Corner element on end of square channel to sectional element on channel wall",
"Plane element to a differential coaxial ring parallel to the element",
"Plane element on and normal to axis to inside of coaxial differential circular ring",
"Element on interior surface of right circular cylinder to coaxial differential ring in cylinder base",
"Element on interior surface of right circular cone to coaxial differential ring in cone base",
"Differential element on interior of hemisphere to a coaxial differential ring element on the base",
"Parallel differential strip elements in intersecting planes",
"Strip element on outer surface of circular semicylinder to parallel strip element in plane passing through cylinder axis",
"Infinitely long differential bounding strip to opposed parallel strip",
"Strip of finite length and of differential width, to differential strip of same length on parallel generating line",
"Square strip element to opposed square strip element",
"1x2 rectangular strip element to opposed 1x2 strip element",
"Differential ring element to opposed ring element on coaxial disk",
"Differential ring element on circular disk to opposed coaxial ring element on coaxial disk separated by coaxial cylinder",
"Circumferential ring element on interior of right circular cylinder to coaxial ring element on base. (Corrected from original reference.)",
"Ring element on exterior of tube to coaxial annular element on circular fin",
"Two ring elements on the interior of a right circular cylinder",
"Differential ring on surface of right circular cylinder to differential ring on parallel cylinder of the same diameter",
"Differential ring on surface of right circular cylinder to differential ring on parallel cylinder of the same diameter. Cylinders are connected
through axes by thin parallel plate of width 2l",
"Ring element on interior of right circular cylinder to ring element on interior of coaxial right circular cone",
"Coaxial ring element to second coaxial ring element on the interior of a cone",
"Coaxial ring element on interior of right circular cone to coaxial ring element on the interior of a butted second cone",
"Differential Ring Element on the Surface of a Sphere to Differential Ring Element on Interior of Coaxial Cylinder",
"Differential Ring Element on the Surface of a Sphere to a Differential Ring Element in the Base of a Coaxial Cylinder",
"Differential Ring Element on Base of Cylinder to Ring Element on Interior of Cylinder with Blockage by Coaxial Sphere Within
Cylinder",
"Differential areas on the inside of a spherical cavity",
"Bounding element on wall of tapered duct to element further along duct"};
namedd=x;
return x[n];
}
public String[] varible_dd(int no)
{  String y[]={""};
   char gr[]=gr();
   if(no==1)
   {String y1[]={""+Teta+"1",""+Teta+"2","S","dA2"};y=y1;}
   else if(no==2)
   {String y1[]={""+Beta,"d"+Beta,"a","x","dx"};y=y1;}
   else if(no==3)
   {String y1[]={""+Beta,"d"+Beta};y=y1;}
   else if(no==4)
   {String y1[]={"a","x","dx"};y=y1;}
   else if(no==5)
   {String y1[]={"h","r","dr"};y=y1;}
   else if(no==6)
   {String y1[]={"r","x","dx"};y=y1;}
   else if(no==30)
   {String y1[]={"a","b","z","h","d"+Ksi+"d"+Eta};y=y1;}
   else if(no==7)
   {String y1[]={"z","r1","r1","dr"};y=y1;}
   else if(no==8)
   {String y1[]={"z","r1","r1","dr",""+Alpha};y=y1;}
   else if(no==9)
   {String y1[]={"z","r1","r2","dr"};y=y1;}
   else if(no==10)
   {String y1[]={"x","y","dy",""+Fi};y=y1;}
   else if(no==11)
   {String y1[]={"r","x",""+Delta};y=y1;}
   else if(no==12)
   {String y1[]={"y","h","dy2"};y=y1;}
   else if(no==13)
   {String y1[]={"b","S",""+Fi,"d"+Fi};y=y1;}
   else if(no==14)
   {String y1[]={"h","y","dy2"};y=y1;}
   else if(no==15)
   {String y1[]={"h","y","dy2"};y=y1;}
   else if(no==16)
```

```
   {String y1[]={"h","r1","r2","dr"};y=y1;}
   else if(no==17)
   {String y1[]={"h","r","r1","r2","dr2"};y=y1;}
   else if(no==18)
   {String y1[]={"x","r1","r2","dr"};y=y1;}
   else if(no==19)
   {String y1[]={"h","r1","r2","dr"};y=y1;}
   else if(no==20)
   {String y1[]={"x","r","dx2"};y=y1;}
   else if(no==21)
   {String y1[]={"z","l","dz"};y=y1;}
   else if(no==22)
   {String y1[]={"z","l","dz"};y=y1;}
   else if(no==23)
   {String y1[]={"h","dx","r1","r2",""+Alpha};y=y1;}
   else if(no==24)
   {String y1[]={"x",""+Ksi,"d"+Ksi,""+Teta};y=y1;}
   else if(no==25)
   {String y1[]={"x","h","r1","r2",""+Alpha+"1",""+Alpha+"2","dx"};y=y1;}
   else if(no==26)
   {String y1[]={"r","S","z","dz",""+Beta};y=y1;}
   else if(no==27)
   {String y1[]={"r1","r2","dr2",""+Beta};y=y1;}
   else if(no==28)
   {String y1[]={"r","r1","r2","z","dz"};y=y1;}
   else if(no==29)
   {String y1[]={"r","dA2"};y=y1;}
     return y;
}
public double F_dd(int no,double x[])
{ double z=0;
 if(no==1) //teta1=x[0] teta2=x[1]
 {z=ct(x[0])*ct(x[1])/(PI*x[2]*x[2])*x[3];}
 else if(no==2)
 {   double X=x[3]/x[2]; //  x/a
     double beta=x[0];
    double dbeta=x[1]; //  dbeta
     double dX=x[4]/x[2]; //  dX
              z=ct(beta)*dbeta*dX/(PI*(1.0+X*X)*(1.0+X*X));
  }
  else if(no==3)
  {   double X=x[3]/x[2]; //  x/a
     double beta=x[0]; //beta
    double dbeta=x[1]; //  dbeta
              z=ct(beta)*dbeta/2.0;
  }
  else if(no==4)
  {   double X=x[1]/x[0]; //  x/a
     double dX=x[2]/x[0];
     double x1=1.0+X*X;
     double x2=sqrt(x1);
     double x3=2.0+X*X;
               z=X/(PI*x2*x2*x2)*(x2/x3+att(1.0/x2))*dX;
  }
  else if(no==5)
  {   double R=x[1]/x[0]; //  r/h
     double dR=x[2]/x[0];
     double x1=1.0+R*R;
               z=2.0*R*dR/(x1*x1);
  }
  else if(no==6)
  { double X=x[1]/x[0]; //  r/h
     double dX=x[2]/x[0];
     double x1=1.0+X*X;
               z=2.0*X*dX/sqrt(x1);
  }
  else if(no==7)
  {   double zz=x[0];
     double r1=x[1];
     double r2=x[2];
     double R=r2/r1;
     double Z=zz/r1;
     double dr=x[3];
     double dR=dr/r1;
               double X=1.0+Z*Z+R*R; //  r/h
               z=2.0*Z*(X-2.0*R*R)*R*dR/(X*X-4.0*R*R);
  }
```

```
else if(no==8)
{   double zz=x[0];
    double r1=x[1];
    double r2=x[2];
    double R=r2/r1;
    double Z=zz/r1;
    double dr=x[3];
            double X=1.0+Z*Z+R*R; //  r/h
            double alpha=x[4];
    double dR=dr/r1;
    double A=1.0+Z*tt(alpha);
            z=2.0*Z*ct(alpha)*(A*X-2.0*R*R)*R*dR/pow((X*X-4.0*R*R),1.5);
}
else if(no==9)
{   double zz=x[0];
    double r1=x[1];
    double r2=x[2];
    double R=r2/r1;
    double Z=zz/r1;
    double dr=x[3];
            double X=1.0+Z*Z+R*R; //  r/h
    double dR=dr/r1;
    double x1=Z*Z+1.0;
    double x2=Z*Z-1.0;
    double x3=pow((X*X-4.0*R*R),1.5);
            z=2.0*Z*(x1*x1+R*R*x2)*R*dR/sqrt(x1)/x3;
}
else if(no==10)
{   double xx=x[0];
    double y=x[1];
    double dy=x[2];
    double fi=x[3];
    double Y=y/xx;
    double x1=1.0+Y*Y-2.0*Y*ct(fi);
    double x2=st(fi);
    z=Y*x2*x2/2.0/pow(x1,1.5);
}
else if(no==11)
{   double r=x[0];
    double xx=x[1];
    double delta=x[2];
    double X=xx/r;
    double x1=st(delta)*((1.0+X)*ct(delta)-1.0);
    double x2=X*X+2.0*(X+1.0)*(1.0-ct(delta));
    z=x1/2.0/pow(x2,1.5);
}
else if(no==12)
{   double y=x[0];
    double h=x[1];
    double dy2=x[2];
    double Y=y/h;
    double dY2=dy2/h;
    double x1=Y*Y+1.0;
    z=dY2/2.0/pow(x1,1.5);
}
else if(no==13)
{   double b=x[0];
    double S=x[1];
    double fi=x[2];
    double dfi=x[3];
    double B=b/S;
    z=ct(fi)*dfi/PI*att(B);
}
else if(no==14)
{   double h=x[0];
    double y=x[1];
    double dy2=x[2];
    double dY2=dy2/h;
    double Y=y/h;
    double A=Y*Y+1.0;
    z=dY2/PI*(A*sqrt(A)*att(1.0/sqrt(A)+0.5*ln(A*A/(A+1.0)/Y/Y)));
}
else if(no==15)
{   double h=x[0];
    double y=x[1];
    double dy2=x[2];
    double dY2=dy2/h;
```

```
    double Y=y/h;
    double A=Y*Y+1.0;
    double B=Y*Y+4.0;
    double A3_2=pow(A,1.5);
    double B3_2=pow(B,1.5);
    double A0_5=sqrt(A);
    double B0_5=sqrt(B);
    double x1=4.0/B3_2*att(1.0/B0_5);
    double x2=2.0/A3_2*att(2.0/A0_5);
    double x3=ln(A*B/(B+1.0)/Y/Y);
    z=dY2/3.0/PI*(x1+x2+x3)*dY2;
  }
  else if(no==16)
  {   double h=x[0];
    double r1=x[1];
    double r2=x[2];
    double dr=x[3];
    double R=r2/r1;
    double dR=dr/r1;
    double H=h/r1;
    double Y=H*H+R*R+1.0;
    z=2.0*R*H*H*Y/pow((Y*Y-4.0*R*R),1.5)*dR;
  }
  else if(no==17)
  {   double h=x[0];
    double r=x[1];
    double r1=x[2];
    double r2=x[3];
    double dr2=x[4];
    double R1=r1/r;
    double R2=r2/r;
    double dR=dr2/r;
    double dR2=dr2/r;
    double H=h/r;
    double X=R1*R1+R2*R2+H*H;
    double FI=act(1.0/R1)+act(1.0/R2);
    double x1=4.0*H*H*R2*dR2/PI/(X*X-4.0*R1*R1*R2*R2);
    double x2=R1*R1*st(FI)/(X-2.0*R1*R2*ct(FI));
    double x3=X/sqrt(X*X-4.0*R1*R1);
    double x4=sqrt((X+2.0*R1*R1)/(X-2.0*R1*R1));
    z=x1*(x2+x3*att(x4*tt(FI/2.0)));
  }
  else if(no==18)
  {   double xx=x[0];
    double r1=x[1];
    double r2=x[2];
    double dr2=x[3];
    double X=xx/r1;
    double R=r2/r1;
    double dR=dr2/r1;
    double x1=2.0*X*R-(1.0+X*X-R*R)*dR;
    double x2=(1.0+X*X+R*R);
    double x3=pow((x2*x2-4.0*R*R),1.5);
    z=x1/x3;
  }
  else if(no==19)
  {   double h=x[0];
    double r1=x[1];
    double r2=x[2];
    double dr=x[3];
    double H=h/r1;
    double R=r2/r1;
    double dR=dr/r1;
    double Z=1.0+R*R+H*H;
    double x1=2.0*H*R*dR/PI/(Z*Z-4.0*R*R);
    double x2=sqrt(R*R-1.0);
    double x3=2.0*(R*R-H*H-1.0)/sqrt(Z*Z-4.0*R*R);
    double x4=sqrt((Z+2.0*R)/(Z-2.0*R)*(R-1.0)/(R+1.0));
    z=x1*(x2+x3*att(x4));
  }
  else if(no==20)
  {   double xx=x[0];
    double r=x[1];
    double dx2=x[2];
    double X=xx/2.0/r;
    double dX2=dx2/2.0/r;
    double x1=2.0*X*X*X+3.0*X;
```

```
    double x2=2.0*pow((X*X+1.0),1.5);
    z=(1-x1/x2)*dX2;
  }
  else if(no==21)
  {   double l=x[0];
    double r=x[1];
    double dz=x[2];
    double L=l/r;
    double dZ=dz/r;
    double x1=0.025155638551629955+0.546105936898297*L-2.4628095207264016*L*L+2.984308445284705*L*L*L;
    double x2=0.06679989474230022-0.02530874640300365*L+0.00413283227337468*L*L-2.179672160149773E-4*L*L*L;
    if(L>=0.01 && L<=0.5) z=x1*dZ;
    else                  z=x2*dZ;
  }
  else if(no==22)
  {   double l=x[0];
    double r=x[1];
    double dz=x[2];
    double L=l/r;
    double dZ=dz/r;
    double x1=0.023657655445323677+0.4863850035806015*L-2.4070201828509528*L*L+3.0006738006841465*L*L*L;
    double x2=0.05027657788789108-0.023820905295369482*L+0.0042955709371557 3*L*L-2.369026836015907E-4*L*L*L;
    if(L>=0.01 && L<=0.5) z=x1*dZ;
    else                  z=x2*dZ;
  }
  else if(no==23)
  {   double h=x[0];
    double dx=x[1];
    double r1=x[2];
    double r2=x[3];
    double alpha=x[4];
    double R1=r1/h;
    double R2=r2/h;
    double dX=dx/h;
    double Y1=1.0+R1*R1+R2*R2;
    double Y2=Y1*Y1-4.0*R1*R1*R2*R2;
    double Z=ct(alpha)-R2*st(alpha);
    double x1=dX/2.0/R1;
    double x2=1.0-Y1*ct(alpha)+2.0*Z/(sqrt(Y2)*ct(alpha));
    double x3=Y1*(2.0*Y1*Z+4.0*R1*R1*R2*st(alpha))/pow(Y2,1.5)/ct(alpha);
    z=x1*(x2+x3);
  }
  else if(no==24)
  {   double xx=x[0];
    double fi=x[1];
    double dfi=x[2];
    double teta=x[3];
    double X=fi/xx;
    double dX=dfi/xx;
    double cteta=ct(teta/2.0);
    double steta=st(teta/2.0);
    double x1=cteta*cteta/(2.0*steta*steta)*dX;
    double x2=(X-1.0)*(X-1.0)+6.0*X*steta*steta;
    double x3=(X-1.0)*(X-1.0)+4.0*X*steta*steta;
    double x4=pow(x3,1.5);
    double x5=Math.abs(X-1.0);
    z=x1*(1.0-x5*x2/x4);
  }
  else if(no==25)
  {   double xx=x[0];
    double h=x[1];
    double r1=x[2];
    double r2=x[3];
    double alpha1=x[4];
    double alpha2=x[5];
    double dx=x[6];
    double dX=dx/h;
    double X=xx/h;
    double R1=r1/h;
    double R2=r2/h;
    double Y1=1.0+R1*R1+R2*R2;
    double Y2=Y1*Y1-4.0*R1*R1*R2*R2;
    double Z1=ct(alpha1)-R2*st(alpha1);
    double Z2=ct(alpha2)-R2*st(alpha2);
    double salpha1=st(alpha1);
    double salpha2=st(alpha2);
    double calpha1=ct(alpha1);
```

```
    double calpha2=ct(alpha2);
    double x1=dX/(2.0*R1);
    double x2=2.0*Z2*Z1+Y1*calpha1*calpha2-4.0*R1*R2*salpha1*salpha2;
    double x3=sqrt(Y2)*calpha2;
    double x4=(Y1*Z1+R1*R2*R2*salpha1)*(2.0*Y1*Z1+4.0*R1*R1*R2*salpha2);
    z=x1*(calpha1-x2/x3+x4/x3);
  }
  else if(no==26)
  {   double r=x[0];
    double S=x[1];
    double zz=x[2];
    double dz=x[3];
    double beta=x[4];
    double fi=S/r;
    double sbeta=st(beta);
    double cbeta=ct(beta);
    double alpha=Math.asin(zz*cbeta+sbeta*sbeta/fi/sbeta);
    double a=fi*sbeta*sbeta;
    double b=(fi*fi+sbeta*sbeta+zz*cbeta)*sbeta;
    double c=fi*sbeta*sbeta-zz*fi*cbeta;
    double e=fi*fi+zz*zz+sbeta*sbeta;
    double h=-2.0*fi*sbeta;
    double A=a/h;
    double B=b/h;
    double C=c/h;
    double E=e/h;
    double Z=z/h;
    double dZ=dz/h;
    double eps=sqrt(E*E-1.0);
    double calpha=ct(alpha);
    double salpha=st(alpha);
    double x1=fi*dZ/PI;
    double x2=(PI-2.0*alpha)*A;
    double x3=(A+(A-C-E*B)/(eps*eps))*2.0*calpha/(E+salpha);
    double x4=2.0/eps*((E*(A-C)-B)/(eps*eps)-A*E);
    double x5=(1.0+E/tt(alpha/2.0))/eps;
    double x6=(1.0+E*tt(alpha/2.0))/eps;
    z=x1*(x2-x3+x4*(att(x5)-att(x6)));
  }
  else if(no==27)
  {   double r1=x[0];
    double r2=x[1];
     double dr2=x[2];
    double beta=x[3];
    double R=r1/r2;
    double sbeta=st(beta);
    double cbeta=ct(beta);
    double a=-2.0*R-sbeta;
    double b=R*R+2.0*(1.0-cbeta);
    double e=(1.0-cbeta)*(1.0-cbeta);
    double g=R*sbeta*(1.0-cbeta);
    double fi=Math.asin(tt(beta/a)/R);
    double B=b/a;
    double E=e/a;
    double G=g/a;
    double eps=sqrt(B*B-1.0);
    double dR=dr2/r1;
    double x1=R*dR/PI;
    double x2=2.0*(E+G*B)/(eps*eps*(B+st(fi)));
    double x3=4.0*(G+B*E)/(eps*eps);
    double x4=(B-1.0)*tt(PI/4.0-fi/2.0);

    z=x1*(x2-x3*att(x4));
  }
  else if(no==28)
  {   double r=x[0];
            double r1=x[1];
    double r2=x[2];
    double zz=x[3];
     double dz=x[4];
    double R=r1/r;
    double fi=r2/r;
    double Z=zz/r;
    double dZ=dz/r;
    double a=fi*fi+Z*Z+R*R;
    double b=-2.0*R*fi;
    double mu=act((Z-R*sqrt(fi*fi+Z*Z-2.0*Z))/(R*fi));
```

```
      double A=a/b;
      double eps=sqrt(A*A-1.0);
      double x1=2.0*Z*fi*dZ/(PI*eps*eps);
      double x2=2.0*(A*fi+R)/eps;
      double x3=tt((A-1.0)*tt(mu/2.0)/eps);
      double x4=(fi+A*R)*st(mu)/(A+ct(mu));
      z=x1*(x2*x3-x4);
   }
   else if(no==29)
   {   double r=x[0];
      double dA2=x[1];
      z=dA2/(4.0*PI*r*r);
   }
 return z;
}
// Differential-Finite
// Factors Between Differential Area to Finite Area
// Section B
//=======================================
public double[] input_df(int no)
{ String s=name_df(no)+"\n     enter : ";
  String s1[]=varible_df(no);
  for(int i=0;i<s1.length;i++) {s+=s1[i]+" ";}
  i1=new input6(s,"Confiuration factor differential area to finite area",folder+"df"+no+".jpg");
  double a[]=i1.vericiktisi();
  return a;
}

public String name_df(int n)
{String x[]=
{"","1.  Differential element of any length to semi-infinite plane. Plane containing element and receiving semi-infinite plane intersect at
angle ? at edge of semiinfinite plane.",
"2.  Differential planar element touching any convex one-, two-, or three-dimensional surface at tangent angle ?.  ",
"3.  Differential planar element to finite parallel rectangle. Normal to element passes through corner of rectangle.",
"4.  Differential planar element to rectangle in plane 90° to plane of element and perpendicular to corner of plane. ",
"5.  From differential element tilted at arbitrary angle to a finite rectangle.  ",
"6.  Planar element to rectangle with right triangle added. Vertex of added triangle at end of rectangle nearest element.",
"7.  Planar element to rectangle with right triangle added. Vertex of added triangle at end of rectangle farthest from element.",
"8.  Differential planar element to plane rectangle. Planes containing dA1 and A2 intersect at angle 0 < "+Fi+" < "+pi+" Element dA1 lies
on normal to line of intersection between planes with origin at one corner of rectangle.",
"9.  Planar element to right triangle in plane parallel to plane containing element. Normal to element passes through vertex of triangle.  ",
"10.  Planar element to isosceles triangle in plane parallel to plane containing element. Normal to element passes through vertex of
triangle.",
"11.  Planar element dA1 to regular n-sided polygon A2 lying in parallel plane. Normal to element passes through center of polygon. ",
"12.  Planar element dA1 to circular disk A2 in parallel plane. Normal to element passes through center of disk.  ",
"13.  Differential tilted planar element dA1 to disk A2. Element lies on normal to disk passing through disk center.",
"14.  Planar element dA1 to a circular disk A2 in a parallel plane. Element is offset from normal to disk center by distance a.",
"15.  Planar element dA1 to a circular disk A2 in a perpendicular plane.  ",
"16.  Planar element dA1 to circular disk A2. Element is rotated at arbitrary angle q in x-z plane.",
"17.  From differential element at coordinate system origin tilted at arbitrary angle to a disk bisected by the y-z plane. ",
"18.  Planar element to elliptical plate in plane parallel to element. Normal to element passes through center of plate.  ",
"19.  Differential planar element dA1 to segment A2 of disk in parallel plane. Element lies on normal to disk passing through disk center.",
"20.  Differential element to parallel disk segment.  ",
"21.  Plane element to sector of circular disk parallel to element.",
"22.  Differential element or ring on disk 1 to coaxial parallel disk 2.",
"23.  Differential sector on disk 1 to coaxial parallel disk 2.  ",
"24.  Plane element to ring sector on circular disk parallel to element.",
"25.  Plane element to ring area in plane perpendicular to element.  ",
"26.  Element on coaxial annular disk to second annular disk separated by coaxial cylinder.",
"27.  Plane element to annular disk with conical blockage of view.  ",
"28.  Plane element to interior of coaxial right circular cylinder.  ",
"29.  Planar or coaxial ring element in plane perpendicular to cylinder axis to right circular cylinder of finite length.",
"30.  Differential element on annulus between coaxial cylinders to interior of outer cylinder.  ",
"31.  Plane element to right circular cylinder of finite length and radius, normal to element passes through one end of cylinder and is
perpendicular to cylinder axis.",
"32.  Element on plane to exterior of right circular cylinder of finite length. Plane does not intersect cylinder.  ",
"33.  Front face of plane vertical element to circular cylinder tilted toward the element. ",
"34.  Vertical or horizontal element to base of a tilted cylinder. ",
"35.  Plane element on a ring to an inverted cone; ring and cone have the same axis and plane of ring does not intersect cone.",
"36.  Plane element on a ring to an inverted cone; ring and cone have the same axis and plane of ring intersects cone. ",
"37.  Plane element to exterior of right circular truncated cone; element and cone base are in the same plane. ",
"38.  Differential element tilted in y-z plane to inverted cone. ",
"39.  Plane element to sphere; normal to center of element passes through center of sphere.",
"40.  Plane element to sphere; tangent to element passes through center of sphere. ",
"41.  Differential planar element to sphere; element plane does not intersect sphere.",
"42.  Differential planar element to sphere; element plane intersects sphere. ",
"43.  Arbitrarily oriented differential planar element to a sphere  ",
```

"44. Area element to sphere; element lies in plane perpendicular to sphere axis.",
"45. Differential element on non-coaxial disk ring to sphere; plane of element does not intersect sphere. ",
"46. Plane element to sphere. ",
"47. Differential element parallel to sphere axis to sphere. ",
"48. From differential element at coordinate system origin tilted at arbitrrary angle to a hemisphere with base in the x-z plane and a center at height z=h. ",
"49. Differential element on plane to standing standard person facing plane.",
"50. Differential element on vertical plane to seated standard person facing plane.",
"51. Element on longitudinal strip inside cylinder to inside cylinder surface.",
"52. Element on longitudinal strip on inside of right circular cylinder to base of cylinder.",
"53. Differential element on surface of right circular cylinder to disk on base of cylinder, r2 <rr.",
"54. Element on interior of coaxial right circular cylinder sitting atop a second cylinder of larger radius to base of lower cylinder. ",
"55. Element at end of outer cylinder to inside of outer coaxial cylinder. ",
"56. Element on strip inside cylinder to entire interior surface of outer concentric right circular cylinder. ",
"57. Element on parallel strip on interior of outer cylinder to exterior of concentric smaller right circular cylinder.",
"58. Differential element on interior of right circular cylinder of finite length to annular end enclosing space between coaxial cylinders.",
"59. Element on strip on exterior surface of inner coaxial right circular cylinder to inner surface of outer cylinder. ",
"60. Element on strip on exterior surface of inner coaxial cylinder to annular end enclosing space between cylinders.",
"61. Element on surface of cylinder at longitude a to sphere; axis of sphere bisects and is normal to cylinder axis. ",
"62. Element on internal surface of right circular cone to base of cone.",
"63. Element on interior of right circular cone to coaxial disk on base. ",
"64. Any differential area on the interior of a hemisphere to the entire base.",
"65. Element on interior of hemisphere to section from base to height l. ",
"66. Differential element or ring on interior of hemisphere to a coaxial disk in base.",
"67. Element or coaxial ring on surface of sphere to sphere. ",
"68. Differential planar element on ceiling, floor or any wall to cow. ",
"69. Differential area on the inside of a spherical cavity to finite area on interior of sphere.",
"70. Infinite strip to parallel infinite plane of finite width. Plane and plane containing strip intersect at arbitrary angle. ",
"71. Differential element of any length to surface generated by a line of infinite length parallel to the plane of the element, and moved parallel to itself. Plane of element does not intersect surface.",
"72. Infinite differential bounding strip elements to infinitely long opposed strip. ",
"73. Infinitely long strip of differential width to parallel semi-cylinder. Strip element and cylinder axis are in the same plane. ",
"74. Infinitely long strip element to infinitely long parallel cylinder. ",
"75. Strip element to rectangle in plane parallel to strip; strip is opposite one edge of rectangle.",
"76. Strip element to rectangle in plane 90o to plane of strip.",
"77. Strip of finite length to opposed rectangle of infinite width. Plane containing strip intersects rectangle at angle ? .",
"78. A strip element to a rectangular plane intersecting at an angle ? . ",
"79. Strip element to parallel opposed right circular cylinder of same length as strip. ",
"80. Strip element on plane to exterior of right circular cylinder of finite length. Strip and cylinder are parallel, and of equal length. Plane does not intersect cylinder. ",
"81. Vertical differential strip on plane contained in cylinder with axis parallel to strip. Strip and cylinder of equal length. Factor from strip to entire inner surface of cylinder. ",
"82. Square strip element to opposed square. ",
"83. 1 x 2 rectangular strip to 1 x 2 opposed rectangle. ",
"84. Element of any length on exterior of cylinder to plane of infinite length and width. ",
"85. Infinitely long differential strip element on tube to fin. ",
"86. Longitudinal strip element on inside of right circular cylinder to entire inside cylinder surface. ",
"87. Longitudinal strip element on inside of right circular cylinder to base of cylinder. ",
"88. Strip on interior of outer cylinder to entire interior of outer right circular coaxial cylinder. ",
"89. Strip on interior of outer cylinder to exterior of inner opposed right circular coaxial cylinder.",
"90. Strip on interior of outer right circular cylinder of finite length to annular end enclosing space between coaxial cylinders. ",
"91. Differential ring element on surface of disk to coaxial sphere. ",
"92. Sector of differential ring element on surface of disk to coaxial sphere. ",
"93. Ring element on interior of right circular cylinder to circular disk at end of cylinder.",
"94. Ring element on base of right circular cylinder to finite circumferential ring on interior of cylinder.",
"95. Ring element on interior of right circular cone to end of frustum on cone. ",
"96. Ring element on interior of right circular cone to coaxial disk. ",
"97. Conical ring element to coaxial body of revolution (cone, paraboloid, or ellipsoid). ",
"98. Exterior of differential conical ring to coaxial sphere, Case I. ",
"99. Exterior of differential conical ring to coaxial sphere, Case II.",
"100. Differential ring on surface of right circular cylinder to finite area on surface of parallel cylinder of the same diameter. End of finite area is opposite the ring. ",
"101. Differential ring on surface of right circular cylinder to finite area on surface of parallel cylinder of the same diameter. End of finite area is opposite the ring. Axes of cylinders are connected by thin plate of width 2l. ",
"102. Differential ring element on base of cylinder to finite ring on interior of cylinder with blockage by coaxial sphere within cylinder.",
"103. Differential ring on interior of hemisphere to disk on base. ",
"104. Differential ring element on the surface of a sphere to finite ring on interior of coaxial cylinder. ",
"105. Differential ring sector to spherical segment.",
"106. Spherical point source to plane rectangle. Point source lies on corner of adjacent rectangle with common side and intersecting A2 at angle ø ",
"107. Spherical point source to sphere. ",
"108. Element on surface of hemisphere to second hemisphere in contact.",
"109 3a. Differential planar element to finite parallel rectangle. Normal to element passes through center of rectangle. Rectangle is partially shaded by rectangle in parallel plane.",
"110 3e. Differential planar element to finite parallel rectangle. Normal to element passes through center of rectangle. Rectangle is partially shaded by circular disk in parallel plane.",
"111 11a. Bounding element on wall of tapered duct to end plane of duct. ",

```
"112 21a.  Plane element to coaxial circular disk. Disk is partially shaded by disk in parallel plane.",
"113 21b . Plane element to coaxial circular disk. Disk is partially shaded by rectangle in parallel plane.",
"114 34a. Vertical or horizontal element to side of a tilted cylinder; element in plane of cylinder base.",
"115 93a. Ring element on exterior of right circular cylinder to coaxial circular disk displaced from cylinder. "};
namedf=x;
return x[n];
}
public String[] varible_df(int no)
{  String y[]={""};
   if(no==1)
   {String y1[]={""+Fi};y=y1;}
   else if(no==2)
   {String y1[]={""+Lambda};y=y1;}
   else if(no==3)
   {String y1[]={"a","b","c"};y=y1;}
   else if(no==108)
   {String y1[]={"a","b","c"};y=y1;}
   else if(no==109)
   {String y1[]={"a","b","c"};y=y1;}
   else if(no==4)
   {String y1[]={"a","b","c"};y=y1;}
   else if(no==5)
   {String y1[]={"a","b","c",Teta+"i",Teta+"j",Teta+"k"};y=y1;}
   else if(no==6)
   {String y1[]={"a","b","c",""+Teta,""+Fi};y=y1;}
   else if(no==7)
   {String y1[]={"a","b","c",""+Teta,""+Fi};y=y1;}
   else if(no==8)
   {String y1[]={"a","b","c",""+Fi};y=y1;}
   else if(no==9)
   {String y1[]={"d","h",""+Fi};y=y1;}
   else if(no==10)
   {String y1[]={"d","h","r"};y=y1;}
   else if(no==11)
   {String y1[]={"d","h","r","n"};y=y1;}
   else if(no==12)
   {String y1[]={"h","r"};y=y1;}
   else if(no==13)
   {String y1[]={"h","r",""+Teta};y=y1;}
   else if(no==14)
   {String y1[]={"h","r","a"};y=y1;}
   else if(no==15)
   {String y1[]={"h","r","a"};y=y1;}
   else if(no==16)
   {String y1[]={"h","r","a"+""+Teta};y=y1;}
   else if(no==17)
   {String y1[]={"h","r","a"+""+Teta+"i",""+Teta+"j",""+Teta+"k"};y=y1;}
   else if(no==18)
   {String y1[]={"a","b","h"};y=y1;}
   else if(no==19)
   {String y1[]={"h","r","d"};y=y1;}
   else if(no==20)
   {String y1[]={"x","r","h",""+Teta};y=y1;}
   return y;
}
public double F_df(int no,double x[])
{ double z=0;
 if(no==1) {z=0.5*(1.0+ct(x[0]));}
 else if(no==2) {z=0.5*(1.0+ct(x[0]));}
 else if(no==3)
 { double A=x[0]/x[2];double B=x[1]/x[2];double A1=sqrt(1+A*A);double B1=sqrt(1+B*B);
 z=1.0/(2.0*PI)*(A/A1*att(B/A1)+B/B1*att(A/B1));}
 else if(no==4)
 { double A=x[0]/x[1];double C=x[2]/x[1];double Y=sqrt(A*A+C*C);
   z=1.0/(2.0*PI)*(att(1.0/C)-C/Y*att(1.0/Y)); }
 else if(no==5)
 { double A=x[0]/x[2];double B=x[1]/x[2];double ti=x[3];double tj=x[4];double tk=x[5];double AA=sqrt(1.0+A*A);double
BB=sqrt(1.0+B*B);
   z=1.0/(2.0*PI)*(att(B)*ct(ti)+att(A)*ct(tj)+(A*ct(tk)-ct(ti))/AA*att(B/AA)+(B*ct(tk)-ct(tj))/BB*att(A/BB)); }
 else if(no==6)
 { double A=x[0]/x[1];double C=x[2]/x[1];double t=x[3];double f=x[4];double k=tt(t);double sf=st(f);double cf=ct(f);double Z=A-
C*cf;double X=sqrt((k*k+1.0)*C*C*sf*sf+Z*Z);double Y=sqrt(1.0+C*C*sf*sf);
   z=1.0/(2.0*PI)*(att(1.0/C)+(A*cf-C)/X*(att(((k*k+1.0)+k*Z)/X)-att(k*Z/X))+cf/Y*(att(C*cf/Y)+att(Z+k)/Y)); }
 else if(no==7)
 { double A=x[0]/x[1];double C=x[2]/x[1];double t=x[3];double f=x[4];double k=tt(t);double sf=st(f);double cf=ct(f);double Z=A-
C*cf;double X=sqrt((k*k+1.0)*C*C*sf*sf+Z*Z);double Y=sqrt(1.0+C*C*sf*sf);
   z=1.0/(2.0*PI)*(att(1.0/C)+cf/Y*(att(Z/Y)+att(C*cf/Y))+((A+k)*cf-C)/X*(att((1.0-k*Z)/X)+att(k*(Z+k)/X))); }
```

```
 else if(no==8)
 { double A=x[0]/x[1];double C=x[2]/x[1];double f=x[4];double sf=st(f);double cf=ct(f);double X=sqrt(A*A+C*C*cf);double
Y=sqrt(1.0+C*C*sf*sf);
   z=1.0/(2.0*PI)*(att(1.0/C)+X*(A*cf-C)*att(X)+cf/Y*(att((A-C*cf)/Y)+att(C*cf/Y))); }
 else if(no==9)
 { double D=x[0]/x[1];double A=sqrt(1.0+D*D);double f=x[2];
   z=D/(2.0*PI)*att(D*tt(f)/A); }
 else if(no==10)
 { double D=x[0]/x[1];double R=x[2]/x[1];double A=sqrt((R*R-D*D)/(1.0+D*D));
   z=D/(PI*sqrt(1.0+D*D))*att(A); }
 else if(no==11)
 { double D=x[0]/x[1];double R=x[2]/x[1];double n=x[3];double A=sqrt((R*R-D*D)/(1.0+D*D));
   z=D*n/(PI*sqrt(1.0+D*D))*att(A); }
 else if(no==12)
 { double H=x[0]/x[1];z=1.0/(H*H+1); }
 else if(no==13)
 { double t=x[2];double H=x[0]/x[1];double tt=1.0/tt(t);double ct=ct(t);double st=ct(t);double cot=att(H);double X=sqrt(1.0-H*H/(tt*tt));
     if(t<=cot) {z=1.0/(1.0+H*H)*ct;}
     else{z=-H*X*st/PI/(1.0+H*H)+1.0/PI*att(X*st/H)+ct/PI/(1+H*H)*(PI-act(H/tt(t)));}
 }
 else if(no==14)
 { double t=x[2];double H=x[0]/x[2];double R=x[1]/x[2];double Z=1.0+R*R+H*H;
   z=0.5*(1.0-(Z-2.0*R*R)/sqrt(Z*Z-4.0*R*R));
 }
 else if(no==15)
 { double H=x[0]/x[2];double R=x[1]/x[2];double Z=1.0+R*R+H*H;
   z=H/2.0*(Z/sqrt(Z*Z-4.0*R*R)-1.0);
 }
 else if(no==16)
 {double H=x[0]/x[1];double A=x[2]/x[1];double t=x[3];double st=st(t);double ct=ct(t);double t1=att(H/(A+1.0));double t2=att(H/(A-1.0));
  double X=1.0+A*A+H*H;double Z=sqrt(X*X-4.0*A*A);double Y=H/tt(t);
  if(t>=t1 && t<=t2)
  {z=0.5*(((1.0-A*A-H*H)/Z+1.0)*ct+H/A*(1.0-X/Z)*st);}
  else if((t>=0 && t<=PI) || (t>=t2 && t<=t1))
  {z=((1.0-A*A-H*H)*ct-X*H*st/A)*att(sqrt((X+2.0*A)/(X-2.0*A))/tt(act(Y)/2.0))+
   1.0/(2.0*PI)*(ct+H/A*st)*(PI-act(Y))+1.0/PI*att(st/H*sqrt(1.0-Y*Y));}
 }
 else if(no==17)
 {double H=x[0]/x[2];double R=x[1]/x[2];double Z=1.0+H*H+R*R;double P=sqrt(Z*Z-4.0*R*R);double ti=x[3];double tj=x[4];double
tk=x[5];double cti=ct(ti);double ctj=ct(tj);double ctk=ct(tk);
  z=0.5*((ctk-H*cti)+(H*Z*ctj/P)-(1.0+H*H-R*R)/P*ctk);
  }
 else if(no==18)
 {double A=x[0]/x[2];double B=x[1]/x[2];
  z=A*B/sqrt((1+A*A)*(1+B*B));
  }
 else if(no==19)
 {double R=x[1]/x[0];double D=x[2]/x[0];
  z=1.0/PI*(R*R/(1.0+R*R)*act(D/R)-D/(1.0+D*D)*att(sqrt((R*R-D*D)/(1.0+D*D))));
  }
 //else if(no==20)
 //{double X=x[0]/x[1];double H=x[2]/x[1];double M=(1.0+X*X+H*H)-4.0*X*X;double t=x[3];double st=st(t
 // z=1.0/PI*(R*R/(1.0+R*R)*act(D/R)-D/(1.0+D*D)*att(sqrt((R*R-D*D)/(1.0+D*D))));
  //}
 return z;
}
// Finite-Finite
// Factors Between Finite Area to Finite Area
// Section C
//=======================================
    public double[] input_ff(int no)
    { String s=name_ff(no)+"\n    enter : ";
      String s1[]=varible_ff(no);
      for(int i=0;i<s1.length;i++) {s+=s1[i]+" ";}
      i1=new input6(s,"Configuration Factor finite area to finite area",folder+"ff"+no+".jpg");
      double a[]=i1.vericiktisi();
      return a;
    }
public String name_ff(int n)
{String x[]=
{
"","Two infinitely long, directly opposed parallel plates of the same finite width.",
"Two infinitely long parallel plates of different widths; centerlines of plates are connected by perpendicular between plates.",
"Two infinitely long plates of unequal widths h and w, having one common edge, and at an angle of 90o to each other.",
"Two infinitely long plates of equal widths having one common edge and an included angle a.",
"Two infinitely long plates of unequal width having a common edge with an included angle a.",
"Infinitely long enclosure formed by three planar or convex surfaces.",
```

"Infinite plane to row of parallel cylinders, or rows of in-line cylinders",
"Infinite plane to first, second, and first plus second rows of infinitely long parallel tubes of equal diameter in equilateral triangular array.",
"Top surface of finite rectangle tilted relative to an infinite plane.",
"Rectangles having a common edge and forming an arbitrary angle; one rectangle infinitely long.",
"Identical, parallel, directly opposed rectangles.",
"Coaxial parallel squares of different edge length.",
"Rectangle to rectangle in a parallel plane; all boundaries are parallel or perpendicular to x and x boundaries.",
"Two finite rectangles of same length, having one common edge, and at an angle of 90o to each other.",
"Rectangle to disk in perpendicular plane. Disk is symmetrically centered with rectangle, and touches rectangle center",
"Two rectangles with one common edge and included angle of o.",
"Rectangle 1 to rectangle 2 in a plane intersecting the plane containing rectangle 1. All rectangle edges are parallel or perpendicular to the line of intersection of the containing planes.",
"Parallel areas on interior ends of tapered duct",
"Right triangle to perpendicular rectangle with common side.",
"Isosceles triangle to perpendicular rectangle: Base of triangle and rectangle have common edge.",
"Right triangle with side of length one-half that of perpendicular rectangle to rectangle.",
"Perpendicular right triangles with one equal edge.",
"Right triangle to perpendicular right triangle with common edge; apexes at common point.",
"Right triangle to perpendicular right triangle with common edge; apexes at opposite ends.",
"Triangle to perpendicular rectangle.",
"Right triangle to perpendicular right triangle of unequal size; apexes at opposite ends of shared edge.",
"Right triangle to perpendicular right triangle of unequal size; apexes at common point.",
"Right isosceles triangle to adjacent congruent perpendicular triangle connected along short side.",
"Parallel directly opposed right triangles.",
"Parallel directly opposed rectangles with triangular extensions.",
"Floor to end wall with triangular extension.",
"Side wall to end wall with triangular extension.",
"Between all combinations of surfaces in a hexagonal prism.",
"Between parallel regular polygons.",
"One side of rectangle to one quarter of parallel cylinder of same length as rectangle; cylinder is bisected longitudinally by plane containing rectangle.",
"Rectangle to perpendicular circular segment.",
"Horizontal panel to adjacent vertical panel with circular segment on top.",
"Side wall to end wall with circular segment extension.",
"Circular disk to parallel right triangle; normal from center of circle passes through one acute vertex.",
"Disk to parallel coaxial disk of same radius.",
"Disk to parallel coaxial disk of unequal radius.",
"Disk to second coaxial disk inside cone.",
"Nonintersecting disks with intersecting axes; axes intersect between disks; and disks can be inscribed in sphere of radius r3 (i.e. r12 + h12 = r22 + h22 = r32)",
"Nonintersecting disks with intersecting axes; axes do not intersect between disks. Disks can be inscribed in sphere of radius r3 (i.e. r12 + h12 = r22 + h22 = r32)",
"Sector of circular disk to sector of parallel circular disk.",
"Parallel opposed circular segments.",
"Disk to coaxial annular ring on parallel disk.",
"Disk to coaxial cone.",
"Annular Disk to coaxial truncated cone; cone can be convergent (+a) or divergent (-a).",
"Disk to a coaxial paraboloid.",
"Disk to coaxial ellipsoid.",
"Ring to parallel coaxial ring.",
"Disk to coaxial disk of equal radius with intervening coaxial cylinder",
"Coaxial annular rings separated by coaxial cylinder.",
"Annular ring between two concentric cylinders to inside of outer cylinder; inner radius of ring is equal to radius of inner cylinder.",
"Ring on annulus between coaxial cylinders to inner surface of outer cylinder.",
"Annular ring around base of hemisphere to hemisphere.",
"Ring around base of hemisphere to section of hemisphere.",
"Annular ring to attached coaxial paraboloid.",
"Annular ring to attached coaxial ellipsoid.",
"Exterior of infinitely long cylinder to symmetrically placed infinitely long parallel rectangle.",
"Exterior of infinitely long cylinder to unsymmetrically placed infinitely long parallel plate.",
"Concentric cylinders of infinite length.",
"Exterior of infinitely long cylinder to interior of concentric semicylinder.",
"Interior of infinitely long semicylinder to itself when concentric coaxial cylinder is present.",
"Interior of infinitely long semicylinder 1 to interior of semicylinder 2 when concentric parallel cylinder 3 is present.",
"Infinitely long cylinder to non-concentric cylindrical enclosure.",
"Infinitely long parallel cylinders of the same diameter.",
"Infinite parallel cylinders of different radius.",
"Infinitely long facing parallel semicylinders of equal radius.",
"Infinitely long semicylinder to parallel infinitely long semicylinder of same radius.",
"Infinitely long cylinder to other cylinders in square array of parallel cylinders of equal diameter.",
"Infinitely long cylinder to other cylinders in equilateral triangular array of parallel cylinders of equal diameter.",
"Finite-length cylinder to rectangle with two edges parallel to cylinder axis and of length equal to cylinder.",
"Finite cylinder to finite rectangle of same length.",
"Outside surface of cylinder to perpendicular right triangle; triangle is in plane of cylinder base with one vertex of triangle at center of base.",
"Outer surface of cylinder to annular disk at end of cylinder.",
"Inner surface of right circular cylinder to itself.",

```
"Base of right circular cylinder to inside surface of cylinder.",
"Disk in cylinder base or top to inside surface of right circular cylinder."
};
nameff=x;
return x[n];
}
public double F_ff(int no,double x[])
{ double z=0;
 if(no==1) //h=x[0] w=x[1]
 { double H=x[0]/x[1];z=sqrt(1.0+H*H)-H;}
 else if(no==2) //a=x[0] b=x[1] c=x[2]
 {double B=x[1]/x[0];double C=x[2]/x[0];
 z=(1/(2*B))*(sqrt((B+C)*(B+C)+4)-sqrt((C-B)*(C-B)+4));}
 else if(no==3) //h=x[0] w=x[1]
 { double H=x[0]/x[1];z=0.5*(1+H-sqrt(1+H*H));}
 else if(no==4) //alpha=x[0]
 { z=1-st(x[0]/2);}
  else if(no==5) //a=x[0] b=x[1] alpha=x[1]
  {double A=x[0]/x[1];z=(A+1-sqrt(A*A+1-2*A*ct(x[2])))/2;}
  else if(no==6) //A1=x[0] A2=x[1] A3=x[2]
  {double A1=x[0];double A2=x[1];double A3=x[2];z=(A1+A2-A3)/(2*A1);}
  else if(no==7) //d=x[0] b=x[1] n=x[2]
  {double D=x[0]/x[1];z= 1-sqrt(1-D*D)+D*att(sqrt((1-D*D)/D*D));
   z=1.0-pow((1.0-z),x[2]);}
  else if(no==8)
  {double R=x[0]/x[1];
   z= R;}
  else if(no==9)
  { z=(1-ct(x[0]))/2;}
  else if(no==10)
  { double A=x[0]/x[1];
  z=A;
  }
  else if(no==11)
  {double X=x[0]/x[2];
  double Y=x[1]/x[2];
  double x1=sqrt(((1+X*X)*(1+Y*Y))/(1+X*X+Y*Y));
  double x2=X*sqrt(1+Y*Y);
  double x3=sqrt(1+Y*Y);
  double x4=Y*sqrt(1+X*X);
  double x5=sqrt(1+X*X);
  z=2.0/(PI*X*Y)*(ln(x1)+x2*Math.atan2(X,x3)+x4*Math.atan2(Y,x5)
  -X*att(X)-Y*att(Y));}
  else if(no==12)
  {double A=x[0]/x[2];
   double B=x[1]/x[0];
   double X=A*(1+B);
   double Y=A*(1-B);
  z=(1/PI*A*A)*(ln(pow((A*A*(1+B*B)+2),2)/((Y*Y+2)*(X*X+2)))+sqrt(Y*Y+4)*(Y*att(Y/sqrt(Y*Y+4)-
X*att(X/sqrt(Y*Y+4)))+sqrt(X*X+4)*(X*att(X/sqrt(X*X+4)-Y*att(Y/sqrt(X*X+4))))));
  }
  else if(no==13)
  {
  z=0.00;
  }
  else if(no==14)
  {double H=x[0]/x[2];
  double W=x[1]/x[2];
  z=(1/W*PI)*(W*att(1/W)+H*att(1/H)-sqrt(H*H-
W*W)*att(sqrt(1/(H*H+W*W)))+(1/4)*ln((1+W*W)*(1+H*H)/(1+W*W+H*H)*pow(W*W*(1+W*W+H*H)/(1+W*W)/(W*W+H*H),
W*W)*pow(H*H*(1+W*W+H*H)/(1+H*H)/(W*W+H*H),H*H)));
  }
  else if(no==15)
  { double B=x[0];
              double L=x[1];

z=(0.5974*pow(0.5*L+L*L,0.4976)*pow(0.05*B+B*B,0.2091)+0.5974*pow(0.1*B*L+(B*L*B*L),0.0613))/(pow(L+L*L*L*L,0.4685)+
pow(B+B*B*B*B,0.30775)+pow(B*L+(B*L*B*L),0.7457));
  }
  else if(no==16)
  {double A=x[0]/x[2];
   double B=x[1]/x[2];
   double C=A*A+B*B-2*A*B*ct(x[3]);
   double D=sqrt(1+A*A*st(x[3]));
   double q=x[3];
  z=-st(x[3])/4*PI*B*(A*B*st(x[3])+(PI/2-x[3])*(A*A+B*B)+B*B*att((A-B*ct(x[3]))/(B*st(x[3])))+A*A*att((B-A*ct(x[3]))/A*st(x[3])));
  }
```

```
else if(no==17)
{ z=0.00;}
else if(no==18)
{ double A=x[0]/x[3];
  double B=x[1]/x[3];
  double H=x[2]/x[3];
            z=(x[3]/PI/A)*(ln(((B+A)*(B+A)+4)*((B-A)*(B-A)+4*(1+H*H))/((B-A)*(B-A)+4)*((B+1)*(B+1)+4*(1+H*H))));
            } //eksik
else if(no==19)
{ double L=x[0]/x[1];
  double W=x[2]/x[1];
z=0.00;}
 else if(no==20)
{ double L=x[0]/x[1];
  double W=x[2]/x[1];
z=0.00;}
 else if(no==21)
{ double L=x[0]/x[1];
  double W=x[2]/x[1];
z=0.00;}
 else  if(no==22)
{ double L=x[0]/x[1];
  double W=x[2]/x[1];
z=0.00;}
 else if(no==23)
{ double L=x[0]/x[1];
  double W=x[2]/x[1];
z=0.00;}
 else if(no==24)
{ double L=x[0]/x[1];
  double W=x[2]/x[1];
z=0.00;}
  else if(no==25)
{ double L=x[0]/x[1];
  double W=x[2]/x[1];
z=0.00;}
  else  if(no==26)
{ double L=x[0]/x[1];
  double W=x[2]/x[1];
z=0.00;}
 else if(no==27)
{ double L=x[0]/x[1];
  double W=x[2]/x[1];
z=0.00;}
 else  if(no==28)
{z=(1/2)*(1-(1/sqrt(3)));}
 else if(no==29)
{ double C=x[0]/x[1];
z=0.00;}
 else   if(no==30)
{ double A=x[0]/x[1];
  double C=x[2]/x[1];
z=0.00;}
 else if(no==31)
{ double A=x[0]/x[1];
  double C=x[2]/x[1];
z=0.00;}
 else if(no==32)
{ double A=x[0]/x[1];
  double C=x[2]/x[1];
z=0.00;}
 else if(no==33)
{ double L=x[0]/x[1];
z=0.00;}
else if(no==34)
{ double L=x[0]/x[1];
z=0.00;}
 else if(no==35)
{ double S=x[0]/x[3];
            double L=x[1]/x[3];
            double W=x[2]/x[3];
z=0.00;}
 else if(no==36)
{ double C=x[0]/x[1];
z=0.00;}
 else if(no==37)
{ double A=x[0]/x[1];
```

```
  double C=x[2]/x[1];
z=0.00;}
 else if(no==38)
{ double A=x[0]/x[1];
  double C=x[2]/x[1];
z=0.00;}
 else if(no==39)
{ double X=x[0]/x[1];
  double Y=x[1]/x[2];
  double Z=x[1]/x[3];
z=0.00;}
 else if(no==40)
{ double R=x[0]/x[1];double X=(2.0*R*R+1)/(R*R);  z=0.5*(X-sqrt(X*X-4.0));}
 else if(no==41)
{ double R1=x[2];double R2=x[3];double X=1+(1+x[3]*x[3])/(x[2]*x[2]);
  z=1/2*(X-sqrt(X*X-4*(R2*R2/R1*R1)));}
 else if(no==42)
{ double X=x[0]/x[1];
            double Q=Math.toRadians(x[2]);
z=(1+X*X-2*X*ct(Q/2)-(1-X)*sqrt((X+1)*(X+1)-4*X*ct(Q/2)))/(2*X*X*st(Q/2));}
 else if(no==43)
{ double H1=x[0]/x[3];
            double H2=x[1]/x[3];
            double R=x[2]/x[3];
z=((1-H1)*(1-H2)/(R*R));}
 else if(no==44)
{ double H1=x[0]/x[3];
            double H2=x[1]/x[3];
            double R=x[2]/x[3];
z=((1-H1)*(1-H2)/(R*R));}
 else if(no==45)
{ double R=x[0]/x[1];
            double Q=x[2];
            double B=x[3];
z=0.00;}
 else if(no==46)
{ double C=x[0]/x[1];
            double Q=x[2];
z=0.00;}
 else if(no==47)
{ double H=x[0]/x[1];
            double R2=x[2]/x[1];
            double R3=x[3]/x[1];
z=(1/2)*(R3*R3-R2*R2-sqrt(pow(1+R3*R3+H*H,2)-4*R3*R3)+sqrt(pow(1+R2*R2+H*H,2)-4*R2*R2));}
 else if(no==48)
{ double S=x[0]/x[1];
            double R=x[2]/x[1];
            double a=Math.toRadians(x[3]);
            double X=S+(R*ct(a)/st(a));
            double A=sqrt(X*X+pow(1+R,2));
            double B=sqrt(X*X+pow(1-R,2));
            double C=sqrt(ct(a)+S*st(a));
            double D=sqrt(ct(a)-S*st(a));
            double E=R*ct(a)/st(a)-S;
z=1/2*sqrt(R*R+X*X+1-pow(1+R*R+X*X,2)-4*R*R);}
if(no==49)
{ double H=x[0]/x[1];
            double R=x[2]/x[1];
            double a=Math.toRadians(x[3]);
            double B=sqrt(H*H+pow(1-R,2));
            double C=sqrt(ct(a)+R*st(a));
            z=1/2*sqrt(R*R+H*H+1-pow(1+R*R+H*H,2)-4*B*C);}
if(no==50)
{ double E=x[0]/x[1];
            double S=x[2]/x[1];
            double L=x[3]/x[1];
            z=0.00;}
if(no==51)
{ double A=x[0]/x[1];
            double S=x[2]/x[1];
            double R=x[3]/x[1];
            z=0.00;}
if(no==52)
{ double H=x[0]/x[1];
            double R2=x[2]/x[1];
            double R3=x[3]/x[1];
            double R4=x[4]/x[1];
```

```
            z=(1/2*(R2*R2-1))*(sqrt(pow(R2*R2+R3*R3+H*H,2)-pow(2*R3*R2,2))-sqrt(pow(R2*R2+R4*R4+H*H,2)-
pow(2*R2*R4,2)));}
 if(no==53)
 { double R1=x[0]/x[1];
            double R2=x[2]/x[1];
            double L=x[3]/x[1];
            z=(1-(pow(R2*L-R1,2)/pow(R1*(L-1),2)))*(1-(L/R1)*(sqrt(pow(L/(2*L),2)+1)-L/(2*R1)));}
 if(no==54)
 { double P1=x[0]/x[1];
            double P2=x[2]/x[1];
            double R1=x[3]/x[1];
            double R2=x[4]/x[1];
            double R=x[5]/x[1];
            z=0.00;}
 if(no==55)
 { double R1=x[0]/x[1];
            double R2=x[2]/x[1];
            double Rc=x[3]/x[1];
            double A=R1*R1-Rc*Rc;
            double B=R2*R2-Rc*Rc;
            double C=R2+R1;
            double D=R2-R1;
            double Y=sqrt(A)+sqrt(B);
            z=0.00;}
 if(no==56)
 { double H=x[0]/x[2];
            double R1=x[1]/x[2];
            double P1=x[3]/x[2];
            double P2=x[4]/x[2];
            double Y=(P2-R1)/(1-R1);
            z=0.00;}
 if(no==57)
 { double R=x[1]/x[0];
            double Z=sqrt(R*R-1);
            z=(1/(Z*Z*PI))*(-(PI/2)-Z+R*R*att(1/Z)+2*att(Z));}
 if(no==58)
 { double R=x[1]/x[0];
            double H=x[2]/x[0];
            double alfa=Math.asin(H);
            z=0.00;}
 if(no==59)
 { double L=x[1]/x[0];
            double E=x[2]/x[0];
            double H=x[3]/x[0];
            z=0.00;}
 if(no==60)
 { double A=x[0]/x[1];
            double D=x[2]/x[1];
            double R=x[3]/x[1];
            z=0.00;}
 if(no==61)
 { double B=x[1]/x[0];
            z=(1/PI)*att(B);}
 if(no==62)
 { double B1=x[1]/x[0];
   double B2=x[2]/x[0];
            z=(1/(2*PI))*(att(B1)-att(B2));}
 if(no==63)
 { double D1=x[0];
   double D2=x[1];
            z=1-D1/D2;}
 if(no==64)
 { double r1=x[0];
   double r2=x[1];
            z=0.5;}
 if(no==65)
 { double R=x[0]/x[1];
            z=1-(2/PI)*(sqrt(1-R*R)+R*Math.asin(R));}
 if(no==66)
 { double R=x[0]/x[1];
            z=(2/PI)*(sqrt(1-R*R)+R*Math.asin(R))-R;}
 if(no==67)
 { double E=x[0]/x[1];
            double alfa1=Math.toRadians(x[2]);
            double alfa2=Math.toRadians(x[3]);
            z=(1/(2*PI))*((alfa2-alfa1)/2+att(((1+E)/(1-E)*tt(alfa2/2)))-att((1+E)/(1-E)*tt(alfa1/2)));}
 if(no==68)
```

```
  { double X=1+x[0]/(2*x[1]);
              z=(1/(2*PI)*(sqrt(X*X-1)+Math.asin(1/X-X)));}
  if(no==69)
  { double R=x[0]/x[1];
              double S=x[2]/x[1];
              double C=1+R+S;
              z=(C/(PI)*Math.asin(1/C)*Math.asin(R/C));}
  if(no==70)
  { double X=1+x[0]/(2*x[1]);
              z=(2/(PI)*(sqrt(X*X-1)+((PI/2)-act(1/X)-X)));}
  if(no==71)
  { double R=x[0]/x[1];
              z=1-(2/(PI))-(4/R*(PI)*(1-sqrt(R+1)+R/2*act(R/(R+2))));}
  if(no==72)
  { double P=x[0]/x[1];
              z=0.00;}
  if(no==73)
  { double P=x[0]/x[1];
              z=0.00;}
  if(no==74)
  { double A=x[0]/x[1];
              double B=x[2]/x[1];
              double C=x[3]/x[1];
              z=0.00;}
  if(no==75)
  { double L=x[0]/x[1];
              double N=x[2]/x[1];
              double Z=x[3]/x[1];
              z=0.00;}
  if(no==76)
  { double W=x[0]/x[1];
              double U=x[0]/x[2];
              double V=x[0]/x[3];
              z=0.00;}
  if(no==77)
  { double R=x[0]/x[1];
              double H=x[2]/x[1];
              double A=H*H+R*R-1;
              double B=H*H-R*R+1;
              z=B/(8*R*H)+1/(2*(PI))*(act(A/B)-1/(2*H)*(sqrt(pow(A+2,2)/(R*R)-4)))*(ct(A*R/B)-(A/(2*R*H)*Math.asin(R)));}
  if(no==78)
  { double H=x[0]/(2*x[1]);
              z=(1+H)-sqrt(1+H*H);}
  if(no==79)
  { double H=x[0]/(2*x[1]);
              z=2*H*(sqrt(1+H*H)-H);}
  if(no==80)
  { double R=x[0]/x[1];
              double H=x[2]/x[1];
              z=(1/2)*(1-R*R-H*H+sqrt(pow(1+R*R+H*H,2)-4*R*R));}

  return z;

 }

public String[] varible_ff(int no)
{  String y[]={""};
   if(no==1)
   {String y1[]={"h","w"};y=y1;}
   else if(no==2)
   {String y1[]={"a,b,c"};y=y1;}
   else if(no==3)
   {String y1[]={"h","w"};y=y1;}
   else if(no==4)
   {String y1[]={""+'\u03B1'};y=y1;}
   else if(no==5)
   {String y1[]={"a,b",","+'\u03B1'};y=y1;}
   else if(no==6)
   {String y1[]={"A1, A2, A3"};y=y1;}
   else if(no==7)
   {String y1[]={"d, b, n"};y=y1;}
   else if(no==8)
   {String y1[]={"p,d"};y=y1;}
   else if(no==9)
   {String y1[]={""+'\u03B7'};y=y1;}
   else if(no==10)
   {String y1[]={"a,b",","+'\u03A6'};y=y1;}
```

```
else if(no==11)
{String y1[]={"a,b,c"};y=y1;}
else if(no==12)
{String y1[]={"a,b,c"};y=y1;}
else if(no==13)
{String y1[]={"a,b,c"};y=y1;}
else if(no==14)
{String y1[]={"h,w,l"};y=y1;}
else if(no==15)
{String y1[]={"B,L"};y=y1;}
else if(no==16)
{String y1[]={"a,b,c",","+'\u03A6'};y=y1;}
else if(no==17)
{String y1[]={"a,b,c"};y=y1;}
else if(no==18)
{String y1[]={"a,b,h,z"};y=y1;}
else if(no==19)
{String y1[]={"l,d,w"};y=y1;}
else if(no==20)
{String y1[]={"l,d,w"};y=y1;}
else if(no==21)
{String y1[]={"l,d,w"};y=y1;}
else if(no==22)
{String y1[]={"l,d,w"};y=y1;}
else if(no==23)
{String y1[]={"l,d,w"};y=y1;}
else if(no==24)
{String y1[]={"l,d,w"};y=y1;}
else if(no==25)
{String y1[]={"l,d,w"};y=y1;}
else if(no==26)
{String y1[]={"l,d,w"};y=y1;}
else if(no==27)
{String y1[]={"l,d,w"};y=y1;}
else if(no==28)
{String y1[]={"h"};y=y1;}
else if(no==29)
{String y1[]={"c,b"};y=y1;}
else if(no==30)
{String y1[]={"a,b,c"};y=y1;}
else if(no==31)
{String y1[]={"a,b,c"};y=y1;}
else if(no==32)
{String y1[]={"a,b,c"};y=y1;}
else if(no==33)
{String y1[]={"l,h"};y=y1;}
else if(no==34)
{String y1[]={"l,h"};y=y1;}
else if(no==35)
{String y1[]={"s,l,w,r"};y=y1;}
else if(no==36)
{String y1[]={"c,b"};y=y1;}
else if(no==37)
{String y1[]={"a,b,c"};y=y1;}
else if(no==38)
{String y1[]={"a,b,c"};y=y1;}
else if(no==39)
{String y1[]={"r,d,l1,l2"};y=y1;}
else if(no==40)
{String y1[]={"r,a"};y=y1;}
else if(no==41)
{String y1[]={"R1,R2"};y=y1;}
else if(no==42)
{String y1[]={"\u03B6,x,Q"};y=y1;}
else if(no==43)
{String y1[]={"h1,h2,r1,r3"};y=y1;}
else if(no==44)
{String y1[]={"h1,h2,r1,r3"};y=y1;}
else if(no==45)
{String y1[]={"r,h,\u03F4,\u03B2"};y=y1;}
else if(no==46)
{String y1[]={"c,b,\u03F4"};y=y1;}
else if(no==47)
{String y1[]={"a,r1,r2,r3"};y=y1;}
else if(no==48)
{String y1[]={"s,r1,r2,\u03B1"};y=y1;}
else if(no==49)
```

```
   {String y1[]={"h,r1,r2,\u03B1"};y=y1;}
   else if(no==50)
   {String y1[]={"e,r,s,l"};y=y1;}
   else if(no==51)
   {String y1[]={"a,b,s,r"};y=y1;}
   else if(no==52)
   {String y1[]={"a,r1,r2,r3,r4"};y=y1;}
   else if(no==53)
   {String y1[]={"r1,z,r2,l"};y=y1;}
   else if(no==54)
   {String y1[]={"p1,h,p2,r1,r2,r"};y=y1;}
   else if(no==55)
   {String y1[]={"r1,h,r2,rc"};y=y1;}
   else if(no==56)
   {String y1[]={"h,r1,r2,p1,p2"};y=y1;}
   else if(no==57)
   {String y1[]={"r1,r2"};y=y1;}
   else if(no==58)
   {String y1[]={"r1,r2"};y=y1;}
   else if(no==59)
   {String y1[]={"r,l,e,h"};y=y1;}
   else if(no==60)
   {String y1[]={"a,b,d,r"};y=y1;}
   else if(no==61)
   {String y1[]={"a,b"};y=y1;}
   else if(no==62)
   {String y1[]={"a,b1,b2"};y=y1;}
   else if(no==63)
   {String y1[]={"D1,D2"};y=y1;}
   else if(no==64)
   {String y1[]={"r1,r2"};y=y1;}
   else if(no==65)
   {String y1[]={"r1,r2"};y=y1;}
   else if(no==66)
   {String y1[]={"r1,r2"};y=y1;}
   else if(no==67)
   {String y1[]={"e,r,\u03B11,\u03B12"};y=y1;}
   else if(no==68)
   {String y1[]={"s,r"};y=y1;}
   else if(no==69)
   {String y1[]={"r2,r1,s"};y=y1;}
   else if(no==70)
   {String y1[]={"s,r"};y=y1;}
   else if(no==71)
   {String y1[]={"r,l"};y=y1;}
   else if(no==72)
   {String y1[]={"p,d"};y=y1;}
   else if(no==73)
   {String y1[]={"p,d"};y=y1;}
   else if(no==74)
   {String y1[]={"a,r,b,c"};y=y1;}
   else if(no==75)
   {String y1[]={"l,d,n,z"};y=y1;}
   else if(no==76)
   {String y1[]={"h,r,l1,l2"};y=y1;}
   else if(no==77)
   {String y1[]={"r1,r2,h"};y=y1;}
   else if(no==78)
   {String y1[]={"h,r"};y=y1;}
   else if(no==79)
   {String y1[]={"h,r"};y=y1;}
   else if(no==80)
   {String y1[]={"r2,r1,h"};y=y1;}
 return y;
 }
 public static void main(String arg[])
 { shapeFactor sf=new shapeFactor();
   String shapenames[];
 ImageIcon picture=new ImageIcon("shape.jpg");
   Object[] o=new Object[]{"diferential_to_diferential","diferential_to_finite","finite_to_finite"};
   String name=(String)JOptionPane.showInputDialog(null,"Please select configuration(shape) factor type : ",
     "Configuration factor",JOptionPane.QUESTION_MESSAGE,picture,o,"diferential to diferential configuration(shape) factors");
   int selection=0;String selection_name="";
   if(name.equals("diferential_to_diferential"))
   {selection=0;selection_name="diferential_to_diferential";sf.name_dd(0);}
   else if(name.equals("diferential_to_finite"))
   {selection=1;selection_name="diferential_to_finite";sf.name_df(0);}
```

```
  else
 {selection=2;selection_name="finite_to_finite";sf.name_ff(0);}
  if(selection==0)
    {shapenames=sf.namedd;
     int n=shapenames.length;
     String sn[][]=new String[2][n];
     for(int ii=1;ii<n;ii++)
     {sn[1][ii]=shapenames[ii];sn[0][ii]=""+ii;}
     String head[]={"no","name"};
     Text.printT(sn,head,selection_name);
     int no=Integer.parseInt(JOptionPane.showInputDialog("dd no = "));
     double x[]=sf.input_dd(no);
     sf.i1.jt1.setText("F dd  = "+sf.F_dd(no,x));
     }
   else if(selection==1)
     {shapenames=sf.namedf;
     int n=shapenames.length;
     String sn[][]=new String[2][n];
     for(int ii=1;ii<n;ii++)
     {sn[1][ii]=shapenames[ii];sn[0][ii]=""+ii;}
     String head[]={"no","name"};
     Text.printT(sn,head,selection_name);
     int no=Integer.parseInt(JOptionPane.showInputDialog("df no = "));
     double x[]=sf.input_df(no);
     sf.i1.jt1.setText("F df  = "+sf.F_df(no,x));
     }
   else
     {shapenames=sf.nameff;
     int n=shapenames.length;
        System.out.println("selection="+selection+"n="+n);
     String sn[][]=new String[2][n];
     for(int ii=1;ii<n;ii++)
     {sn[1][ii]=shapenames[ii];sn[0][ii]=""+ii;}
     String head[]={"no","name"};
     Text.printT(sn,head,selection_name);
     int no=Integer.parseInt(JOptionPane.showInputDialog("ff no = "));
     System.out.println("no="+no);
     double x[]=sf.input_ff(no);
     sf.i1.jt1.setText("F ff  = "+sf.F_ff(no,x));}
 }
}
```

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import javax.swing.event.*;
import java.io.*;
import java.util.*;


public class input6 extends Thread implements ActionListener
{    Container c;
                public String s=" ";
                JPanel jp;
                String s1="";
                JTextField jt;
                JTextField jt1;
                JLabel jl;
                JButton bb;
                double a[];
                boolean basildi;
                ImageIcon image;

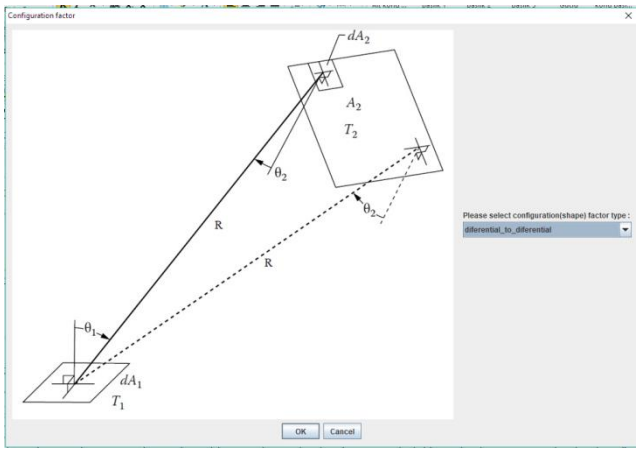
                public input6(String isim,String head,String im)
                { jp=new JPanel();
                  jp.setLayout(new BorderLayout());
                  java.net.URL imgURL = getClass().getResource(im);
     image=new ImageIcon(imgURL,head);
    a=new double[2];
    jl=new JLabel(isim);
                jt=new JTextField(s);
                jt1=new JTextField();
                jt.setFont(new Font("SansSerif",Font.PLAIN,20));
                jt1.setFont(new Font("SansSerif",Font.PLAIN,20));
                bb=new JButton(image);
```

```
                JScrollPane jsp=new JScrollPane(jt);
    JFrame cerceve=new JFrame(head);
    cerceve.addWindowListener(new BasicWindowMonitor());
    c=cerceve.getContentPane();
    c.setLayout(new BorderLayout());
    c.add(bb,BorderLayout.NORTH);
    jp.add(jsp,BorderLayout.CENTER);
    jp.add(jl,BorderLayout.NORTH);
    jp.add(jt1,BorderLayout.SOUTH);
    c.add(jp,BorderLayout.SOUTH);
    jt.addActionListener(this);
    cerceve.pack();
    cerceve.setVisible(true);
    basildi=false;
    }

    public double[] vericiktisi()
    {
      while(!basildi)
      {
      try {Thread.sleep(100);}
      catch(InterruptedException e)  {System.err.println(e.toString());}
      }
              return a;
            }

    protected void finalize() throws Throwable
    {super.finalize();}

public void actionPerformed( ActionEvent e)
{
 if(e.getSource()==jt)
 { s1=jt.getText();
       StringTokenizer token=new StringTokenizer(s1);
     int n=token.countTokens()-1;
     int m=n+1;
     double b[]=new double[m];
     int j=0;
     while(token.hasMoreTokens())
     { Double ax=new Double(token.nextToken());
      b[j++]=ax.doubleValue();
     }
     a=b;
  basildi=true;
 }
}
public static void main(String arg[])
    {
      input6 i1=new input6("ali veli 49 elli","baslik","sec.png");
      double a[]=i1.vericiktisi();
      Text.print(a);
    }
}
```



diferential_to_diferential

no	name
1	Two elemental areas in arbitrary con...
2	Two elemental areas lying on parallel...
3	Elemental area of any length z to infi...
4	Corner element on end of square cha...
5	Plane element to a differential coaxial...
6	Plane element on and normal to axis ...
7	Element on interior surface of right c...
8	Element on interior surface of right c...
9	Differential element on interior of he...
10	Parallel differential strip elements in i...
11	Strip element on outer surface of cir...
12	Infinitely long differential bounding s...
13	Strip of finite length and of differenti...
14	Square strip element to opposed squ...
15	1x2 rectangular strip element to opp...
16	Differential ring element to opposed ...
17	Differential ring element on circular ...
18	Circumferential ring element on interi...
19	Ring element on exterior of tube to c...
20	Two ring elements on the interior of ...
21	Differential ring on surface of right c...
22	Differential ring on surface of right c...
23	Ring element on interior of right circ...
24	Coaxial ring element to second coaxi...

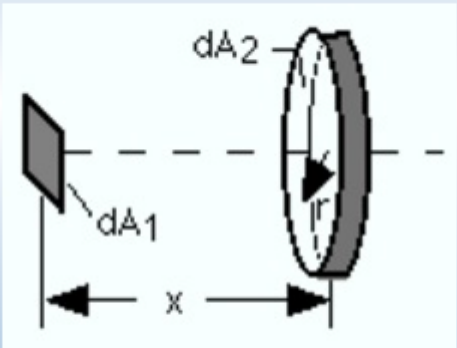
Input

dd no =

6

OK Cancel

Confiuration factor differential area to differential area



Plane element on and normal to axis to inside of coaxial differential circular ring enter : r x dx

1 1 0.1

F dd = 0.1414213562373095

diferential_to_finite

no	name
1	1. Differential element of any length...
2	2. Differential planar element touchi...
3	3. Differential planar element to finit...
4	4. Differential planar element to rect...
5	5. From differential element tilted at ...
6	6. Planar element to rectangle with r...
7	7. Planar element to rectangle with r...
8	8. Differential planar element to plan...
9	9. Planar element to right triangle in ...
10	10. Planar element to isosceles trian...
11	11. Planar element dA1 to regular n-...
12	12. Planar element dA1 to circular d...
13	13. Differential tilted planar element ...
14	14. Planar element dA1 to a circular...
15	15. Planar element dA1 to a circular...
16	16. Planar element dA1 to circular d...
17	17. From differential element at coo...
18	18. Planar element to elliptical plate i...
19	19. Differential planar element dA1 t...
20	20. Differential element to parallel di...
21	21. Plane element to sector of circul...
22	22. Differential element or ring on di...
23	23. Differential sector on disk 1 to c...
24	24. Plane element to ring sector on ...

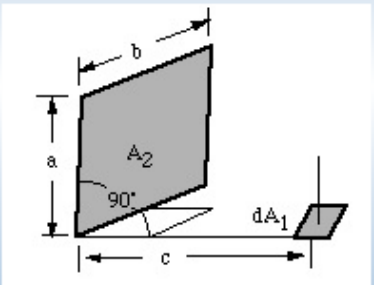
Input

df no =

4

OK Cancel

Confiuration factor differential area to finite area



4. Differential planar element to rectangle in plane 90° to plane of element and perpendicular to corner of plane. enter : a b c

1 1 1

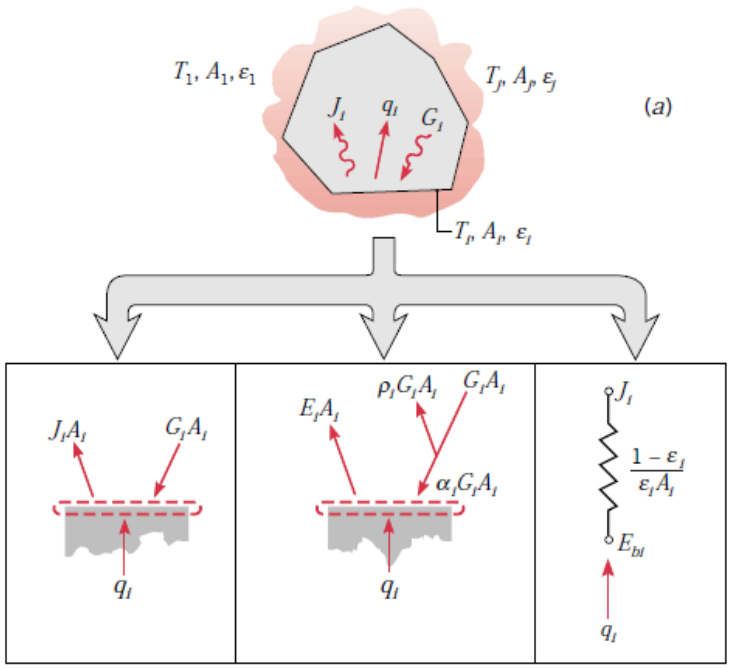
F df = 0.05573419700255351

22 SURFACE EXCHANGE– NO PARTICIPATING MEDIUM

We are assuming gray diffuse surfaces

G_i : Irradiation total energy coming to surface i.

J_i : Radiocity total energy coming from the surface i (emissivity + reflectivity from the surface)
E_{bi} (blackbody radiation from surface i)



Net radiation exchange at one surface:
$Q_i = A_i(J_i - G_i)$ but $J_i = \varepsilon_i E_{bi} + \rho_i G_i$ Assuming that $\rho_i = 1 - \alpha_i = 1 - \varepsilon_i$ $J_i = \varepsilon_i E_{bi} + (1-\varepsilon_i)G_i$
$G_i = \frac{J_i - \varepsilon_i E_{bi}}{(1-\varepsilon_i)}$ and $Q_i = A_i(J_i - \frac{J_i - \varepsilon_i E_{bi}}{(1-\varepsilon_i)})$ $Q_i = \frac{E_{bi} - J_i}{(1-\varepsilon_i)/(\varepsilon_i A_i)}$
Heat transfer between surface i and j

$$Q_{i-j} = -Q_{j-i} = \overline{s_i}\overline{s_j}(J_i - J_j) = A_i F_{A_i - A_j}(J_i - J_j)$$

$$\overline{s_i}\overline{s_j} = A_i F_{A_i - A_j} = \int_{A_i}\int_{A_j} \frac{\cos(\theta_i)\cos(\theta_j)}{\pi S_{ij}^2}\, dA_i\, dA_j$$

S_{ij} : distance between surface i and j
$\overline{s_i}\overline{s_j} = A_i F_{A_i - A_j}$: Shape factor multiplied by area
$\left(\frac{Q}{A}\right)_i = q_i = (J_i - G_i) = J_i - \sum_{j=1}^{N} \overline{s_i}\overline{s_j} J_j$
Where J is radiocity (emissivity + reflectivity from the surface)
G is irradiation (radiation coming to the surface)

$$q_i = \frac{\varepsilon_i}{(1-\varepsilon_i)}(E_{bi} - J_i)$$

For the system of N grey diffuse surfaces:

$$\sum_{j=1}^{N}\left(\frac{\delta_{ij}}{\varepsilon_j} - \frac{\varepsilon_i}{(1-\varepsilon_i)}F_{A_i - A_j}\right)J_j = \sum_{j=1}^{N} E_{bj} = \sum_{j=1}^{N}\sigma T_j^4$$

If q is given for some of the diffuse grey surfaces:

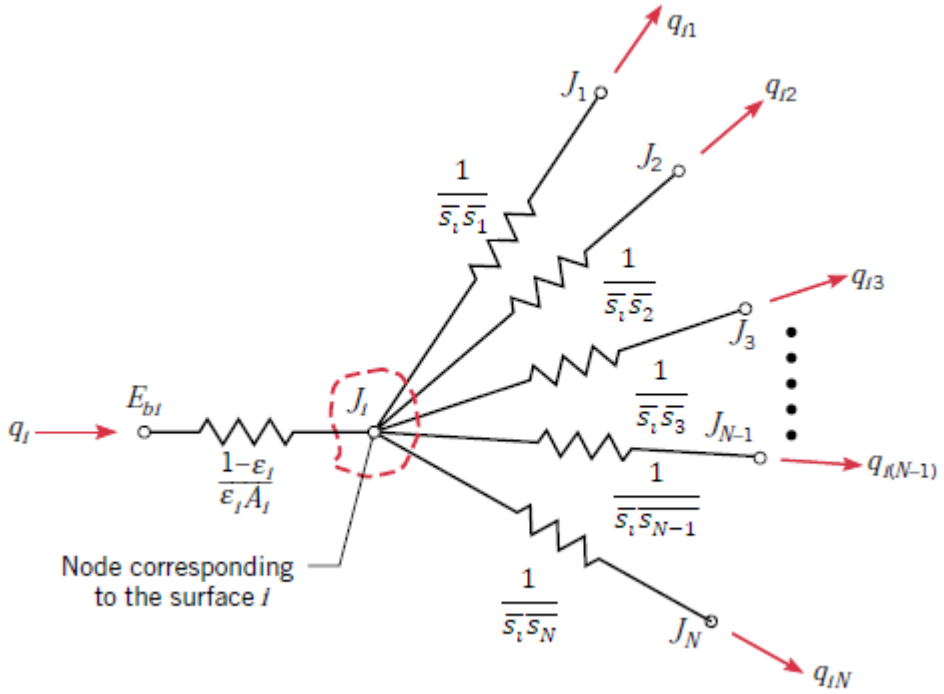
$$\sum_{j=1}^{d}\left(\frac{\delta_{ij}}{\varepsilon_j} - \frac{(1-\varepsilon_i)}{\varepsilon_i}F_{A_i - A_j}\right)J_j = \sum_{j=1}^{d} E_{bj} = \sum_{j=1}^{N}\sigma T_j^4$$

$$\sum_{j=d}^{N}\left(\delta_{ij} - F_{A_i - A_j}\right)J_j = \sum_{j=d}^{N} q_j$$

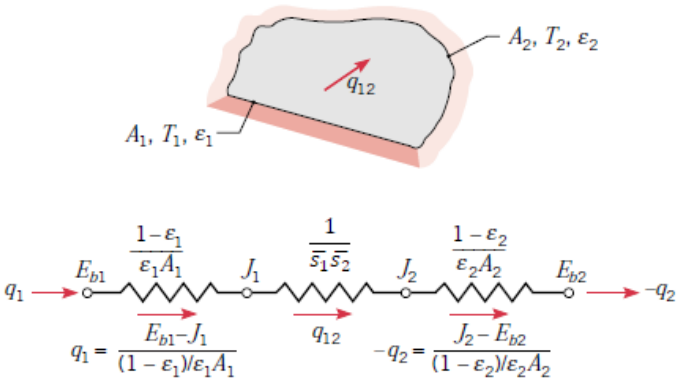
If q is unknown to be solved directly, cavity equation becomes:

$$\sum_{j=1}^{N}\left(\frac{\delta_{ij}}{\varepsilon_j}-\frac{(1-\varepsilon_i)}{\varepsilon_i}F_{A_i-A_j}\right)q_j=\sum_{j=1}^{N}F_{A_i-A_j}\sigma(T_i^4-T_j^4)$$

Where δ_{ij} is Dirac delta function (δ_{ij}=1 if i=j δ_{ij}=0 if $i \neq j$)



For only two surfaces: equation resuces to:



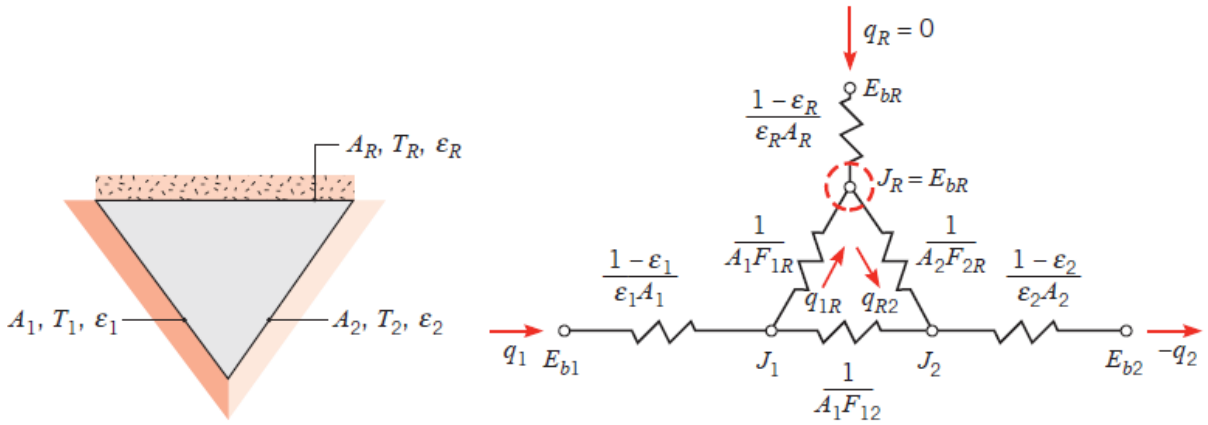
Two surface enclosure:

$$q_{12}=\frac{\sigma\left(T_1^4-T_2^4\right)}{\frac{1-\varepsilon_1}{\varepsilon_1 A_1}+\frac{1}{\overline{s_1 s_2}}+\frac{1-\varepsilon_2}{\varepsilon_2 A_2}}=\frac{\sigma\left(T_1^4-T_2^4\right)}{\frac{1-\varepsilon_1}{\varepsilon_1 A_1}+\frac{1}{A_1 F_{A_1-A_2}}+\frac{1-\varepsilon_2}{\varepsilon_2 A_2}}$$

$$\overline{s_1 s_2}=A_1 F_{A_1-A_2}$$

Re-radiating surfaces:

In some applications, a thermally insulated zone can be utilised. As radiation heat transfer concern such a surface emits as much energy as it receives by radiation from the surrounding. Such a surface is called re-radiating surface. Consider of a three enclosure zone

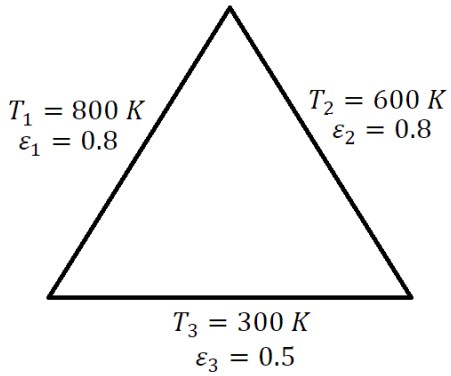


For this problem

$$q_{12} = \frac{\sigma(T_1^4 - T_2^4)}{\frac{1-\varepsilon_1}{\varepsilon_1 A_1} + \frac{1}{\overline{S_1 S_2} + \left(\frac{1}{\overline{S_1 S_R}} + \frac{1}{\overline{S_1 S_R}}\right)^{-1}} + \frac{1-\varepsilon_2}{\varepsilon_2 A_2}}$$

$$q_{12} = \frac{\sigma(T_1^4 - T_2^4)}{\frac{1-\varepsilon_1}{\varepsilon_1 A_1} + \frac{1}{A_1 F_{A_1-A_2} + \left(\frac{1}{A_1 F_{A_1-A_R}} + \frac{1}{A_2 F_{A_2-A_R}}\right)^{-1}} + \frac{1-\varepsilon_2}{\varepsilon_2 A_2}}$$

PROBLEM: The configuration of a furnace can be approximated as an equilateral triangular duct which is sufficiently long that the end effects are negligable. The hot wall is maintanined at T_1=800 K and has an emissivity ε_1=0.8. The second wall is maintained at T_2=600 K and has an emissivity ε_2=0.8. The third wall is maintained at T_2=300 K and has an emissivity ε_2=0.5. Calculate the Net Radiation heat transfer from the walls.




```
public class radiation_exchange_DS1
{
        //diffuse and specular reflection in an enclosure
public double t[];  // Temperature degree K
double A[];  //area m^2
public double T[][];
public double Tinv[][]; //inverse of T tensor
public double SS[];  //Total exchange area
```

```
public double S[][];
public double ss[][];  //direct exchange area
public double h[];
public double G[];    //irradiation
public double eb[];   // blackbody radiation per unit area
public double Eb[];   //total blackbody radiation
public double q[];   //heat transfer per unit area
public double Q[];   //heat transfer
public double eps[]; // emmissivity
public double ro[];  // reflectivity
public double F[][]; // shape factor
public double J[];    //radiosities
public double sigma=5.6703744192E-08;

public radiation_exchange_DS1(double Ai[],double Ti[],double qi[],double epsi[],double roi[],double Fi[][])
{  //Ai area m^2
   //Ti surface temperature K
   //d number of diffuse reflected surfaces
int d=Ti.length;
int n=d+qi.length;  //number of surfaces
t=new double[n];  //surface temperature K
eps=new double[n];
ro=new double[n];
q=new double[n];
Q=new double[n];
F=new double[n][n];
ss=new double[n][n];
SS=new double[n];
T=new double[n][n];
J=new double[n];
eb=new double[n];
Eb=new double[n];
G=new double[n];
A=new double[n];
for(int k=0;k<n;k++)
  {t[k]=Ti[k];
   eps[k]=epsi[k];
   ro[k]=roi[k];
   A[k]=Ai[k];
   for(int j=0;j<n;j++)
    {F[k][j]=Fi[k][j];
     ss[k][j]=A[k]*F[k][j];
    }
  }
for(int k=0;k<n;k++)
{   for(int j=0;j<d;j++)
    { T[k][j]=1.0/eps[k]*dirac_delta(k,j)-ro[k]/eps[k]*F[k][j];}
    for(int j=d;j<n;j++)
    { T[k][j]=dirac_delta(k,j)-F[k][j];}

    SS[k]=eb[k];
}
for(int k=0;k<d;k++)
  {eb[k]=sigma*t[k]*t[k]*t[k]*t[k];SS[k]=eb[k];}
for(int k=d;k<n;k++)
  {SS[k]=qi[k-d];}
J=NA28.AXB(T,SS);
for(int k=0;k<n;k++)
{for(int j=0;j<n;j++)
 {G[k]+=J[j]*F[k][j];}
}
for(int k=0;k<n;k++)
{q[k]=(J[k]-G[k]);
Q[k]=A[k]*q[k];}
}
public static double[] ro(double eps[])
{ int n=eps.length;
  double ro[]=new double[n];
  for(int i=0;i<n;i++)
  {ro[i]=1.0-eps[i];}
  return ro;
}

  public int dirac_delta(int i, int j)
  {
   int k=0;
   if(i==j) k=1;
```

```
   return k;
  }

//==================
public static double F12(double alpha,double a,double b)
{double f=0.5*(1.0+b/a-Math.sqrt(1.0-2.0*b/a*Math.cos(alpha)+(b/a)*(b/a)));
 return f;
}

public static double[][] Fij(int m,double alpha,double a,double b)
{   double F[][]=new double[m][m];
              for(int i=0;i<m;i++)
                {for(int j=0;j<m;j++)
                 {if(i==j) F[i][j]=0.0;
                 else F[i][j]=F12(alpha,a,b); }
     }
     return F;
}

//=====================================
public static void main(String arg[])
{
double F[][]=Fij(3,Math.PI/3.0,1.0,1.0);
Text.print(F,"F Configuration(shape) factor");
double eps[]={0.8,0.8,0.5};
double ro[]=ro(eps);
double Ti[]={800,600,300};
double qi[]={};
double Ai[]={1,1,1};
radiation_exchange_DS1 sisj=new radiation_exchange_DS1(Ai,Ti,qi,eps,ro,F);
Text.print(sisj.J,"J Radiocity total energy coming from the surface i (emissivity + reflectivity from the surface)");
Text.print(sisj.q,"q radiative heat transfer ");
Text.print(sisj.G,"G Irradiation total energy coming to surface i");
Text.print(sisj.eb,"Eb blackbody radiation from surface i");
}

}
```

```
import java.io.*;
import java.text.*;
import java.util.Locale;

class NA28
{
  public static void pivot(double a[][],int o[],double s[],int k)
  {// GaussLU  pivoting process
    int n=a.length;
    int dummy1=0;
    double buyuk;
    double dummy=0;
    //pivotlama
              int p=k;
              buyuk=Math.abs(a[o[k]][k]);
              for(int ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[o[ii]][k]/s[o[ii]]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              dummy1=o[p];
              o[p]=o[k];
              o[k]=dummy1;
  }

  public static int[] pivot_index(int n)
  {   // indexed the pivoted matrix
                int o[]=new int[n];
                for(int i=0;i<n;i++) {o[i]=i;}
                return o;
  }

  public static double[][] gaussLU(double ai[][],int o[])
  { //Gauss LU with pivoting
  int n=ai.length;
  double a[][]=new double[n][n];
  double s[]=new double[n];
  double carpan=0;
  double sum=0;
```

```
double absaij;

//gauss elimination
int i,j,k,p,ii,jj;
for(i=0;i<n;i++)
{for(j=0;j<n;j++){a[i][j]=ai[i][j];}}
for(i=0;i<n;i++)
{ o[i]=i;s[i]=Math.abs(a[i][0]);
  for(j=1;j<n;j++)
    { absaij=Math.abs(a[i][j]);
      if(absaij>s[i]) {s[i]=absaij;}
    }
}
for(k=0;k<(n-1);k++)
{ pivot(a,o,s,k);
              //change pivort reference
              //solve gauss elimination
  for(i=k+1;i<n;i++)
  {  carpan=a[o[i]][k]/a[o[k]][k];
     a[o[i]][k]=carpan;
     for(j=k+1;j<n;j++)
     {   a[o[i]][j]-=carpan*a[o[k]][j]; }
              }
}
return a;
}

public static double[] back_substiution(double ai[][],double bi[])
{   int n=ai.length;
              return back_substiution(ai,bi,pivot_index(n));
}

public static double[][] back_substiution(double ai[][],double bi[][])
{   int n=ai.length;
              return back_substiution(ai,bi,pivot_index(n));
}

public static double[] back_substiution(double ai[][],double bi[],int o[])
{
  int n=ai.length;
  double sum;
  double x[]=new double[n];
  double a[][]=new double[n][n];
  double b[]=new double[n];
  int i,j;
  for(i=0;i<n;i++)
    {for(j=0;j<n;j++){a[i][j]=ai[i][j];};b[i]=bi[i];}
for(i=1;i<n;i++)
    {sum=b[o[i]];for(j=0;j<i;j++){sum-=a[o[i]][j]*b[o[j]];};b[o[i]]=sum;}
x[n-1]=b[o[n-1]]/a[o[n-1]][n-1];
for(i=(n-2);i>=0;i--)
    {sum=0;
     for(j=i+1;j<n;j++)
      {sum+=a[o[i]][j]*x[j];};
      x[i]=(b[o[i]]-sum)/a[o[i]][i];
    }
return x;
}

public static double[][] back_substiution(double ai[][],double bi[][],int o[])
{
  int n=ai.length;
  int m=bi[0].length;
  double sum;
  double x[][]=new double[n][m];
  double a[][]=new double[n][n];
  double b[][]=new double[n][m];
  int i,j,k;
for(i=0;i<n;i++)
    {for(j=0;j<n;j++){a[i][j]=ai[i][j];};for(k=0;k<m;k++) {b[i][k]=bi[i][k];}}
for(k=0;k<m;k++)
{
for(i=1;i<n;i++)
    {sum=b[o[i]][k];for(j=0;j<i;j++){sum-=a[o[i]][j]*b[o[j]][k];};b[o[i]][k]=sum;}
x[n-1][k]=b[o[n-1]][k]/a[o[n-1]][n-1];
for(i=(n-2);i>=0;i--)
    {sum=0;
```

```
      for(j=i+1;j<n;j++)
       {sum+=a[o[i]][j]*x[j][k];};
       x[i][k]=(b[o[i]][k]-sum)/a[o[i]][i];
     }
}
return x;
}

public static double [] AXB(double a[][],double b[])
{
//solution of the system of equation
int n=a.length;
int q[]=new int[n];
double c[][]=gaussLU(a,q);
double s[]=back_substiution(c,b,q);
return s;
}

public static double [][] AXB(double a[][],double b[][])
{//solution of the set of the system  of equations
int n=a.length;
int q[]=new int[n];
double c[][]=gaussLU(a,q);
double s[][]=back_substiution(c,b,q);
return s;
}

public static double[][] I(int n)
{
//Inverse matrix
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
  b[i][i]=1.0;
return b;
}

public static double[][] inverse(double a[][])
{
//Inverse matrix
int n=a.length;
int q[]=new int[n];
double c[][]=gaussLU(a,q);
double s[][]=back_substiution(c,I(n),q);
return s;
}

// Matrix output classes
public static String toString(double[][] left)
{
//return a string representation of a matrix
return toString(left,25,15);
}
public static String toString(double[][] left,int w,int d)
{
//return a string representation of a matrix
int n,m;
String b;
b="";
n=left.length;
m=left[0].length;
for(int i=0;i<n;i++)
{
for(int j=0;j<m;j++)
  {
  b=b+toString(left[i][j],w,d);
  }
  b=b+"\n";
}
return b;
}

public static String toString(double left, int w, int d)
// converts a double to a string with given width and decimals.
{
            Locale us=new Locale("US");
            String s="%"+w+"."+d+"f";
```

```
            String s1=String.format(us,s,left);
    return s1;
  }

  public static void main(String args[]) throws IOException
  {
  double a[][]={{8,2,3},{1,-9,2},{2,3,6}};
  System.out.println("Matrix a : \n"+toString(a));
  System.out.println("Inverse Matrix of a : \n"+toString(inverse(a)));
  }
}
```

Eb blackbody radiation from surface i

0	1	2
23225.8536210432	7348.805247283201	459.3003279552001

G Irradiation total energy coming to surface i

0	1	2
8039.408334969189	13812.880470881915	14415.09242791567

J Radiocity total energy coming from the surface i (e...

0	1	2
20188.564563828397	8641.620292002943	7437.1963779354355

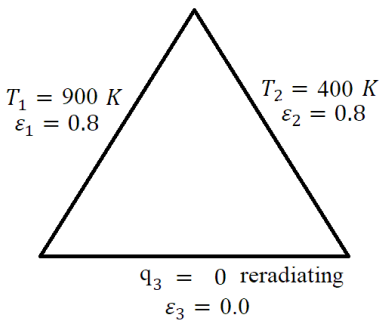
q radiative heat transfer

0	1	2
12149.156228859207	-5171.2601788789725	-6977.896049980234

F Configuration(shape) factor

0	1	2
0.0	0.5	0.5
0.5	0.0	0.5
0.5	0.5	0.0

PROBLEM: The configuration of a furnace can be approximated as an equilateral triangular duct which is sufficiently long that the end effects are negligable. The hot wall is maintanined at T_1=900 K and has an emissivity ε_1=0.8. The second wall is maintained at T_2=400 K and has an emissivity ε_2=0.8. The third wall is reradiating zone for which q_3=0(has an emissivity ε_3=0.0). Calculate the Net Radiation heat transfer from the walls.



$$\bar{s_1}\bar{s_2} = A_1 F_{A_1-A_2} = 0.5$$

$$\bar{s_1}\bar{s_3} = A_1 F_{A_1-A_3} = 0.5$$

$$\overline{s_2}\overline{s_3} = A_2 F_{A_2-A_3} = 0.5$$

$$\sigma = \left[\left(\frac{C_1}{C_2^4}\right)\frac{\pi^4}{15}\right] = 5.6703744192\text{x}10^{-8} \quad \text{W/m}^2\text{K}$$

$$R = \frac{1-\varepsilon_1}{\varepsilon_1 A_1} + \frac{1}{\overline{s_1}\overline{s_2} + \left(\frac{1}{\overline{s_1}\overline{s_R}} + \frac{1}{\overline{s_1}\overline{s_R}}\right)^{-1}} + \frac{1-\varepsilon_2}{\varepsilon_2 A_2}$$

$$R = \frac{1-0.8}{0.8} + \frac{1}{0.5 + \left(\frac{1}{0.5} + \frac{1}{0.5}\right)^{-1}} + \frac{1-0.8}{0.8} = 1.8333$$

$$q_{12} = \frac{\sigma(T_1^4 - T_2^4)}{R} = \frac{5.6703744192\text{x}10^{-8}(900^4 - 400^4)}{1.8333}$$

$$= 19500.93\ W/m^2$$

σ	5.67037E-08			
R	0.25	1.333333	0.25	1.833333
T1	900			
T2	400			
q12	19500.93312	W/m2		

Using computer program to solve it:

```
public class radiation_exchange_DS1
{
        //diffuse and specular reflection in an enclosure
public double t[];  // Temperature degree K
double A[];  //area m^2
public double T[][];
public double Tinv[][]; //inverse of T tensor
public double SS[];  //Total exchange area
public double S[][];
public double ss[][];  //direct exchange area
public double h[];
public double G[];    //irradiation
public double eb[];   // blackbody radiation per unit area
public double Eb[];   //total blackbody radiation
public double q[];   //heat transfer per unit area
public double Q[];   //heat transfer
public double eps[]; // emmissivity
public double ro[];  // reflectivity
public double F[][]; // shape factor
public double J[];    //radiosities
public double sigma=5.6703744192E-08;

public radiation_exchange_DS1(double Ai[],double Ti[],double qi[],double epsi[],double roi[],double Fi[][])
{
int d=Ti.length;
int n=d+qi.length;  //number of surfaces
t=new double[d];  //surface temperature K
eps=new double[d];
ro=new double[d];
q=new double[n];
Q=new double[n];
F=new double[n][n];
SS=new double[n];
T=new double[n][n];
J=new double[n];
eb=new double[d];
Eb=new double[n];
G=new double[n];
```

```
A=new double[n];
for(int k=0;k<n;k++)
{for(int j=0;j<n;j++)
 {F[k][j]=Fi[k][j];}
}
for(int k=0;k<d;k++)
  {t[k]=Ti[k];
   eps[k]=epsi[k];
   ro[k]=roi[k];
  }
for(int k=d;k<n;k++)
   {A[k]=Ai[k];}

for(int k=0;k<d;k++)
{  for(int j=0;j<n;j++){ T[k][j]=1.0/eps[k]*dirac_delta(k,j)-ro[k]/eps[k]*F[k][j];}}

for(int k=d;k<n;k++)
{   for(int j=0;j<n;j++){ T[k][j]=dirac_delta(k,j)-F[k][j];}}

for(int k=0;k<d;k++)
  {eb[k]=sigma*t[k]*t[k]*t[k]*t[k];SS[k]=eb[k];}
for(int k=d;k<n;k++) {SS[k]=qi[k-d];}
J=NA28.AXB(T,SS);
for(int k=0;k<n;k++)
{for(int j=0;j<n;j++)
 {G[k]+=J[j]*F[k][j];}
}
for(int k=0;k<n;k++)
{q[k]=(J[k]-G[k]);
Q[k]=A[k]*q[k];}
}
public static double[] ro(double eps[])
{ int n=eps.length;
  double ro[]=new double[n];
  for(int i=0;i<n;i++)
  {ro[i]=1.0-eps[i];}
  return ro;
}
  public int dirac_delta(int i, int j)
  {
   int k=0;
   if(i==j) k=1;
   return k;
  }

//==================
public static double F12(double alpha,double a,double b)
{double f=0.5*(1.0+b/a-Math.sqrt(1.0-2.0*b/a*Math.cos(alpha)+(b/a)*(b/a)));
 return f;
}

public static double[][] Fij(int m,double alpha,double a,double b)
{   double F[][]=new double[m][m];
                for(int i=0;i<m;i++)
                  {for(int j=0;j<m;j++)
                   {if(i==j) F[i][j]=0.0;
                   else F[i][j]=F12(alpha,a,b); }
     }
     return F;
}

//=======================================
public static void main(String arg[])
{
double F[][]=Fij(3,Math.PI/3.0,1.0,1.0);
Text.print(F,"F Configuration(shape) factor");
double eps[]={0.8,0.8,0.5};
double ro[]=ro(eps);
double Ti[]={900,400};
double qi[]={0};
double Ai[]={1,1,1};
radiation_exchange_DS1 sisj=new radiation_exchange_DS1(Ai,Ti,qi,eps,ro,F);
Text.print(sisj.J,"J Radiocity total energy coming from the surface i (emissivity + reflectivity from the surface)");
Text.print(sisj.q,"q radiative heat transfer ");
Text.print(sisj.G,"G Irradiation total energy coming to surface i");
Text.print(sisj.eb,"Eb blackbody radiation from surface i");
}
```

```
}
```

Eb blackbody radiation from surface i

0	1
37203.3265643712	1451.6158513152

G Irradiation total energy coming to surface i

0	1	2
12827.160169105746	25827.782246580653	19327.4712078432

q radiative heat transfer

0	1	2
19500.933116212364	-19500.933116212364	3.637978807091713E-12

J Radiocity total energy coming from the surface i (e...

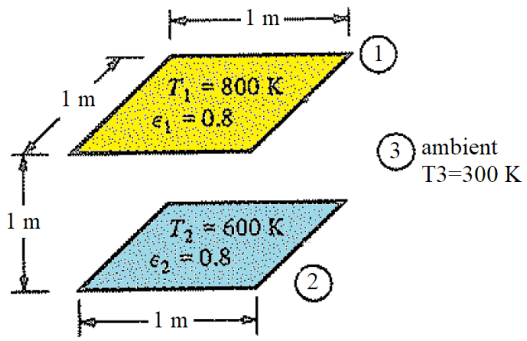
0	1	2
32328.093285318108	6326.849130368289	19327.471207843202

F Configuration(shape) factor

0	1	2
0.0	0.5	0.5
0.5	0.0	0.5
0.5	0.5	0.0

PROBLEM

Two square plates each 1m by 1 m are parallel to and directly opposite to each other at a distance 1 m. The hot plate is T_1=800 K and has an emissivity ε_1=0.8. The cold plate is T_2=600 K and has an emissivity ε_2=0.8. The radiation heat transfer takes place also with the ambient which T_3=300 K. Note that emissivity of the ambient can be assumed to approximate as black body(has an emissivity ε_2=1.0). Calculate the heat transfer between plates and ambient.



Note A_3 is very big compare to A_1=1 m^2 and A_2=1 m^2 so in order to solve problem as a close enclosure a very big number A_3=100000 m^2.

```
 public class radiation_exchangeSiSj_1
{
public double t[];  // Temperature degree K
double A[];  //area m^2
public double T[][];
public double Tinv[][]; //inverst of T tensor
public double SS[][];  //Total exchange area
public double S[][];
public double ss[][];  //direct exchange area
public double h[];
public double G[];    //irradiation
public double G1[];
public double eb[];   // blackbody radiation per unit area
public double Eb[];   //total blackbody radiation
public double q[];   //heat transfer per unit area
```

```
public double Q[];   //heat transfer
public double Q1[];   //heat transfer
public double eps[]; // emmissivity
public double ro[];  // reflectivity
public double F[][]; // shape factor
public double J[];    //radiosities
public double sigma=5.67e-8;
public radiation_exchangeSiSj_1(double Ai[],double Ti[],double epsi[],double roi[],double Fi[][])
{  //Ai area m^2
   //Ti surface temperature K
int n=Ti.length;  //number of surfaces
t=new double[n];  //surface temperature K
eps=new double[n];
ro=new double[n];
q=new double[n];
Q=new double[n];
Q1=new double[n];
F=new double[n][n];
ss=new double[n][n];
SS=new double[n][n];
T=new double[n][n];
S=new double[n][n];
J=new double[n];
eb=new double[n];
Eb=new double[n];
h=new double[n];
G=new double[n];
G1=new double[n];
A=new double[n];
for(int i=0;i<n;i++)
  {t[i]=Ti[i];
   eps[i]=epsi[i];
   ro[i]=roi[i];
   A[i]=Ai[i];
   for(int j=0;j<n;j++)
    {F[i][j]=Fi[i][j];
     ss[i][j]=A[i]*F[i][j];
    }
  }
for(int i=0;i<n;i++)
{
                for(int j=0;j<n;j++)
   { T[i][j]=1.0/eps[j]*dirac_delta(i,j)-ro[j]*ss[i][j]/eps[j]/A[j];
     S[i][j]=ss[i][j]*eps[j];
   }
   eb[i]=sigma*t[i]*t[i]*t[i]*t[i];
}
J=NA28.AXB(T,eb);
SS=NA28.AXB(T,S);
for(int i=0;i<n;i++)
{   double toplam1=0;
    double toplam2=eps[i]*A[i];
                for(int j=0;j<n;j++)
   { toplam1+=SS[i][j];}
}

for(int i=0;i<n;i++)
{   h[i]=0;
    q[i]=0;
                for(int j=0;j<n;j++)
    {h[i]+=SS[i][j]*eb[j];
     q[i]+=SS[i][j]*(eb[i]-eb[j]);
    }
}
for(int i=0;i<n;i++)
{G[i]=h[i]/(A[i]*eps[i]); }
}
public static double[] ro(double eps[])
{ int n=eps.length;
  double ro[]=new double[n];
  for(int i=0;i<n;i++)
  {ro[i]=1.0-eps[i];}
  return ro;
}
  public int dirac_delta(int i, int j)
  {
   int k=0;
```

```
   if(i==j) k=1;
   return k;
  }
//==================
public static double F_parallel_rectangles(double X,double Y, double Z)
{// Z distance between plates
 // X ,Y plate size
                double x=X/Z;
 double y=Y/Z;
 double x1=1+x*x;
 double x2=Math.sqrt(x1);
 double y1=1+y*y;
 double y2=Math.sqrt(y1);
 double Fij=2.0/(Math.PI*x*y)*(Math.log(Math.sqrt(x1*y1/(1+x*x+y*y)))+
 x*y2*Math.atan(x/y2)+
 y*x2*Math.atan(y/x2)-x*Math.atan(x)-y*Math.atan(y));
 return Fij;
}

public static double F_perpendicular_rectangles(double X,double Y, double Z)
{ // X,Y size of plate 1
  // X,Z size of plate 2
  //Plate 1 and plate 2 are perpendicular to each other
 double H=Z/X;
 double W=Y/X;
 double x1=W*Math.atan(1.0/W)+H*Math.atan(1.0/H);
 double y1=Math.sqrt(H*H+W*W);
 double x2=y1*Math.atan(1.0/y1);
 double y2=(1+W*W)*(1+H*H);
 double y3=1+W*W+H*H;
 double y4=(1.0+H*H)*(H*H+W*W);
 double x3=y2/y3*Math.pow((W*W*y3/y2),W*W)*Math.pow((H*H*y3/y4),H*H);
 double x4=1.0/(Math.PI*W)*(x1-x2+0.25*Math.log(x3));
 double Fij=x4;
 return Fij;
}

public static double[][] Fij( double X,double Y,double Z,double Ai[])
{   double F[][]=new double[3][3];
int m=3;
                for(int i=0;i<m;i++)
                  {F[i][i]=0;}
                  F[0][1]=F_parallel_rectangles(X,Y,Z);
                  F[0][2]=4.0*F_perpendicular_rectangles(X,Y,Z);
                  F[1][2]=4.0*F_perpendicular_rectangles(X,Y,Z);
                  F[2][1]=Ai[1]*F[1][2]/Ai[2];
                  F[2][0]=Ai[0]*F[0][2]/Ai[2];
                  F[1][0]=Ai[0]*F[0][1]/Ai[1];
     return F;
}
//==================
public static void main(String arg[])
{


double eps[]={0.8,0.8,1.0};
double ro[]=ro(eps);
double Ti[]={800,600,300};
double Ai[]={1,1,100000};
double F[][]=Fij(1.0,1.0,1.0,Ai);
Text.print(F,"F Configuration(shape) factor");
radiation_exchangeSiSj_1 sisj=new radiation_exchangeSiSj_1(Ai,Ti,eps,ro,F);
Text.print(sisj.J,"J Radiocity total energy coming from the surface i (emissivity + reflectivity from the surface)");
Text.print(sisj.q,"q radiative heat transfer ");
Text.print(sisj.G,"G Irradiation total energy coming to surface i");
Text.print(sisj.eb,"Eb blackbody radiation from surface i");
}
}
```

F Configuration(shape) factor

0	1	2
0.0	0.19982489569838746	0.8001751043016126
0.19982489569838746	0.0	0.8001751043016126
8.001751043016127E-6	8.001751043016127E-6	0.0

Eb blackbody radiation from surface i

0	1	2
23224.32	7348.319999999999	459.2699999999999

G Irradiation total energy coming to surface i

0	1	2
1707.9887223178184	4148.394011143617	0.2050799865353606

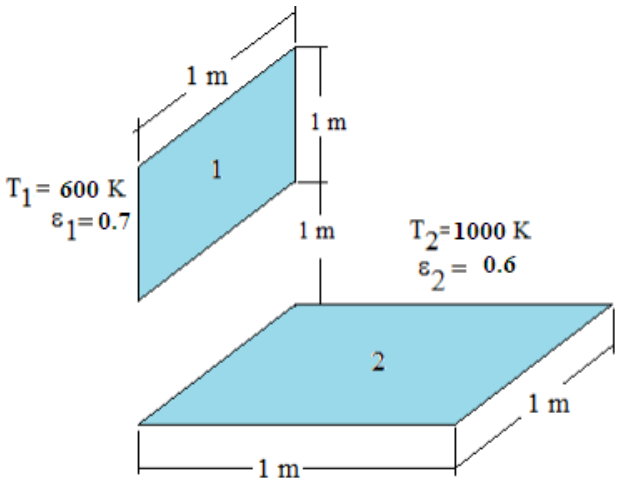
J Radiocity total energy coming from the surface i (e...

0	1	2
18844.494783140824	6631.775840905983	5555.63937595319

q radiative heat transfer

0	1	2
17213.065022145747	2559.940791085107	-19773.005813230855

PROBLEM Two gray surfaces with emmissivities ε_1=0.7 and ε_2=0.6 are located as shown in the figure. The temperature of the plate 1 T1=600 K and temperature of the plate 2 is T2=1000 K. Calculate the radiation heat transfer between the surfaces. Ignore the heat transfer between surfaces and the surrounding air.

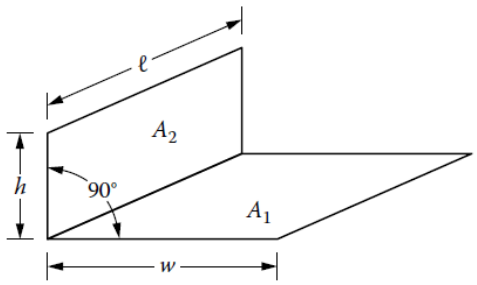


For an enclosure it can be written that $\sum_{j=1}^{N} F_{A_i-A_j} = 1$ and

$A_i F_{A_i-A_j} = A_j F_{A_j-A_i}$

For sub-surfaces $A_j = \sum_{k=1}^{K} A_{(j,k)}$ $\quad F_{A_i-A_j} = \sum_{k=1}^{K} F_{A_i-A_{(j,k)}}$

8



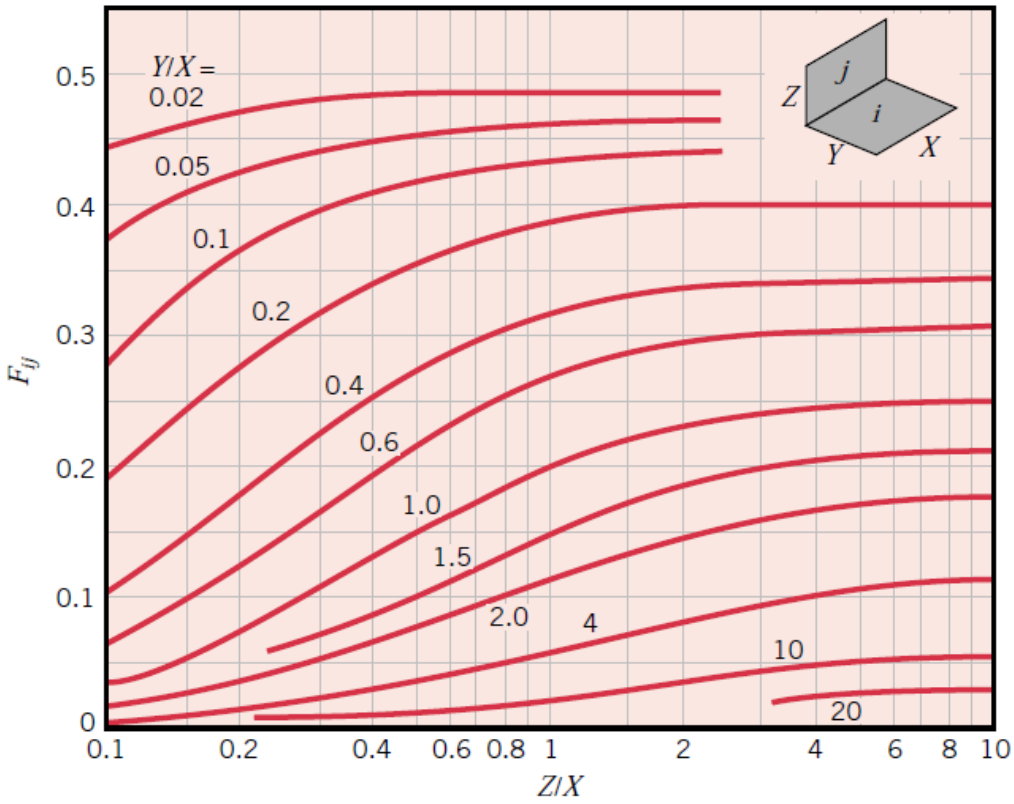
Two finite rectangles of same length, having one common edge and having an angle of 90° to each other.

$$H = \frac{h}{l}; \qquad W = \frac{w}{l}$$

$$F_{1-2} = \frac{1}{\pi W}\left(W \tan^{-1}\frac{1}{W} + H \tan^{-1}\frac{1}{H} \right.$$

$$-\sqrt{H^2+W^2}\,\tan^{-1}\frac{1}{\sqrt{H^2+W^2}} + \frac{1}{4}\ln\left\{ \frac{(1+W^2)(1+H^2)}{1+W^2+H^2} \right.$$

$$\left.\left. \times \left[\frac{W^2(1+W^2+H^2)}{(1+W^2)(W^2+H^2)} \right]^{W^2} \times \left[\frac{H^2(1+H^2+W^2)}{(1+H^2)(H^2+W^2)} \right]^{H^2} \right\} \right)$$




```
public class radiation_exchange_DS3
{
//==================
//==================
public static double F_parallel_rectangles(double X,double Y, double Z)
{// Z distance between plates
 // X ,Y plate size
              double x=X/Z;
 double y=Y/Z;
 double x1=1+x*x;
 double x2=Math.sqrt(x1);
 double y1=1+y*y;
 double y2=Math.sqrt(y1);
 double Fij=2.0/(Math.PI*x*y)*(Math.log(Math.sqrt(x1*y1/(1+x*x+y*y)))+
 x*y2*Math.atan(x/y2)+
 y*x2*Math.atan(y/x2)-x*Math.atan(x)-y*Math.atan(y));
```

```
 return Fij;
}

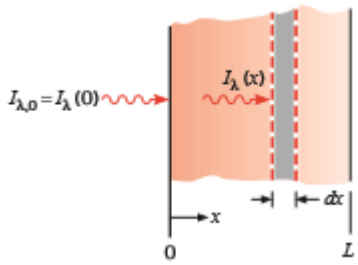
public static double F_perpendicular_rectangles(double X,double Y, double Z)
{ // X,Y size of plate 1
  // X,Z size of plate 2
  //Plate 1 and plate 2 are perpendicular to each other
 double H=Z/X;
 double W=Y/X;
 double x1=W*Math.atan(1.0/W)+H*Math.atan(1.0/H);
 double y1=Math.sqrt(H*H+W*W);
 double x2=y1*Math.atan(1.0/y1);
 double y2=(1+W*W)*(1+H*H);
 double y3=1+W*W+H*H;
 double y4=(1.0+H*H)*(H*H+W*W);
 double x3=y2/y3*Math.pow((W*W*y3/y2),W*W)*Math.pow((H*H*y3/y4),H*H);
 double x4=1.0/(Math.PI*W)*(x1-x2+0.25*Math.log(x3));
 double Fij=x4;
 return Fij;
}
public static double[][] Fij(double Ai[])
{
int m=2;
double F[][]=new double[m][m];
                for(int i=0;i<m;i++)
                  {F[i][i]=0;}
                  double F1=F_perpendicular_rectangles(1,1,1);
                  double F2=F_perpendicular_rectangles(1,1,2);
                  double F3=F2-F1;
                  System.out.println("F1="+F1+"F2="+F2+"F3="+F3);
                  F[0][1]=F3;
                  F[1][0]=Ai[0]*F[0][1]/Ai[1];
     return F;
}
public static double[] ro(double eps[])
{ int n=eps.length;
  double ro[]=new double[n];
  for(int i=0;i<n;i++)
  {ro[i]=1.0-eps[i];}
  return ro;
}
//=====================================
public static void main(String arg[])
{
double A[]={1,1};
double F[][]=Fij(A);
double eps[]={0.7,0.6};
double ro[]=ro(eps);
double T[]={600,1000};
double R=ro[0]/eps[0]/A[0]+1/(F[0][1]*A[0])+ro[1]/eps[1]/A[1];
double sigma=5.6703744192E-08;
double q=sigma*(T[1]*T[1]*T[1]*T[1]-T[0]*T[0]*T[0]*T[0])/R;
System.out.println("R="+R+"q="+q);
}

}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" radiation_exchange_DS3
F1=0.20004377607540316F2=0.2328526027953619F3=0.032808826719958745
R=31.574840689245683 q=1563.1096742643892
> Terminated with exit code 0.
```

23 SURFACE EXCHANGE –PARTICIPATING MEDIUM

Radiation in gases differs fundamentaly from the fact that gaseous radiation is concentrated in specific wavelength intervals (called bands), and it is not is not a surface phenomenon like radiation in solids, but is instead a volumetric phenomenon. For a gas column of length L



$$dI_\lambda(x) = -\kappa_\lambda I_\lambda(x)dx$$

Where κ_λ is absorbtion coefficient, L is thickness of the gas column, $I_\lambda(x)$ is the radition intensity. Integration of the equation gives:

$$\int_{I_\lambda(0)}^{I_\lambda(L)} \frac{dI_\lambda(x)}{I_\lambda(x)} = -\kappa_\lambda \int_{x=0}^{L} dx$$

$$\tau_\lambda = \frac{I_\lambda(L)}{I_\lambda(0)} = \exp(-\kappa_\lambda L)$$

$$\alpha_\lambda = 1-\tau_\lambda = 1-\exp(-\kappa_\lambda L)$$

If Kirchhoff's law is assumed to be valid, $\alpha_\lambda = \varepsilon_\lambda = 1-\tau_\lambda = 1-\exp(-\kappa_\lambda L)$

We should be noted that this equation is given spectral dependent value. And in actual heat transfer calculations total emmissivity data should be needed.

$$\varepsilon = \frac{\int_{\lambda=0}^{\infty} I_{b\lambda}\varepsilon_\lambda d\lambda}{\int_{\lambda=0}^{\infty} I_{b\lambda} d\lambda} = \frac{\int_{\lambda=0}^{\infty} I_{b\lambda}\left[1-\exp(-\kappa_\lambda L)\right]d\lambda}{\int_{\lambda=0}^{\infty} I_{b\lambda} d\lambda}$$

There are several way to approach to this solution, one way is taken data directly. An experimental data for CO_2 water wapor is developed by Leckner [xx], which (for temperatures above 400K) have a maximum error of 5% for water vapor and 10% for CO_2

$$\varepsilon = \varepsilon(p_a L, p, T_g)$$

$$\varepsilon = \alpha(p_a L, p, T_g, T_s) \approx \left(\frac{T_g}{T_s}\right)^{1/2} \varepsilon(p_a L \frac{T_s}{T_g}, p, T_s)$$

Where T_g is the gas temperature and T_s is the temperature of an external surface, p_a is the partial pressure of the gas and p is the total pressure. Emissivities of gases are calculated from:

$$\varepsilon_0(p_a L, p = 1bar, T_g) = \exp\left[\sum_{i=0}^{M}\sum_{j=0}^{N} c_{ji}\left(\frac{T_g}{T_0}\right)^j \left(\log_{10}\frac{p_a L}{(p_a L)_0}\right)^i\right]$$

$T_0 = 1000K \qquad (p_a L)_0 = 1\ barcm$

And c_{ij} are correlation contants for carbondioxide an water vapor. The values are given in the table below.

Gas	Water Vapor	Carbon Dioxide
M,N	2,2	2,3
$c_{00} \quad \dots \quad c_{N0}$ $\vdots \quad \dots \quad \vdots$ $c_{0M} \quad \dots \quad c_{NM}$	−2.2118 −1.1987 0.035596 0.85667 0.93048 −0.14391 −0.10838 −0.17156 0.045915	−3.9893 2.7669 −2.1081 0.39163 1.2710 −1.1090 1.0195 −0.21897 −0.23678 0.19731 −0.19544 0.044644
P_E	$(p+2.56p_a/\sqrt{t})/p_0$	$(p+0.28p_a)/p_0$
$(p_aL)_m/(p_aL)_0$	$1.32/t^{1.4}$	0.23
a	$2.144, \quad t<0.75$ $1.888-2.053\log_{10} t \quad t>0.75$	$1+0.1/t^{1.45}$
b	$1.1/t^{1.4}$	0.23
c	0.5	1.47
T_0=1000 K, p_0=1 bar, t=T/T_0, $(p_aL)_0$=1 barcm		

The emissivity for different pressure conditions is then found from:

$$\frac{\varepsilon(p_aL,\,p,T_g)}{\varepsilon_0(p_aL,\,p=1bar,T_g)}=1-\frac{(a-1)(1-P_E)}{a+b-1+P_E}\exp\left(-c\left[\log_{10}\frac{(p_aL)_m}{p_aL}\right]^2\right)$$

Where P_E is an effective pressure, and a,b,c, and $(p_aL)_m$ are correlation parameters given in the table above. In a mixture that contains both carbondioxide and water vapor, the bands partially overlap and another correction factor must be introduced

$$\Delta\varepsilon=\left[\frac{\varsigma}{10.7+101\varsigma}-0.0089\varsigma^{10.4}\right]\left(\log_{10}\frac{\left(p_{H_2O}+p_{CO_2}\right)L}{\left(p_aL\right)_0}\right)^{2.76}$$

$$\varsigma=\frac{p_{H_2O}}{p_{H_2O}+p_{CO_2}}$$

So the total emissivity and absorptivity of gases containing CO_2 and water vapor can be calculated as:

$$\varepsilon_i(p_aL,\,p,T_g)=\varepsilon_{0i}(p_aL,\,p=1bar,T_g)\left(\frac{\varepsilon}{\varepsilon_0}\right)_i(p_aL,\,p,T_g)$$

$$\alpha_i(p_aL,\,p,T_g,T_s)=\left(\frac{T_g}{T_s}\right)^{1/2}\varepsilon_{0i}(p_iL\frac{T_s}{T_g},\,p,T_s)$$

$$\varepsilon_{CO_2+H_2O}=\varepsilon_{CO_2}+\varepsilon_{H_2O}-\Delta\varepsilon(p_{H_2O}L,\,p_{CO_2}L,)$$

$$\alpha_{CO_2+H_2O}=\alpha_{CO_2}+\alpha_{H_2O}-\Delta\varepsilon(p_{H_2O}L\frac{T_s}{T_g},\,p_{CO_2}L\frac{T_s}{T_g})$$

Solutions of these equations are presented in computer form below:

Main program is solving the following problem: Consider a 1m thick layer of a gas mixture at 1000K and 5bar that consists of 10% carbon dioxide, 20% water vapor, and 70% nitrogen. What is the total normal intensity escaping from this layer? Normal intensity can be calculated as:

$$I = \int_0^\infty I_{b\lambda}(1-e^{-\kappa_\lambda L})d\lambda = \int_0^\infty I_{b\lambda}\varepsilon_\lambda d\lambda = \frac{\varepsilon\sigma T^4}{\pi}$$

```
//SAYISAL ISI IŞINIMI-SEMİH YILMAZ- HAFTA 8 ÖDEV-24.11.2016
//Modest, Radiative Heat Transfer, Sayfa 366-367

import java.text.DecimalFormat;

public class gasemissivity_odev8
{ gasemissivityDATA g1,g2,g1A,g2A;
  public double eps,alpha;
public double gasemissivity(double pi,double pa_h2oi,double pa_co2i,double Ti,double Tsi,double Li)
{ g1=new gasemissivityDATA("water",pi,pa_h2oi,Ti,Li);
  g2=new gasemissivityDATA("carbondioxide",pi,pa_co2i,Ti,Li);
  double eps1=g1.eps;
  //System.out.println("eps1="+eps1);
  double eps2=g2.eps;
  double fi=pa_h2oi/(pa_h2oi+pa_co2i);
  double deps=(fi/(10.7+101.0*fi)-0.0089*Math.pow(fi,10.4));
  deps*=Math.pow((Math.log10((pa_h2oi+pa_co2i)*Li)),2.76);
  eps=eps1+eps2-deps;
  g1A=new gasemissivityDATA("water",pi,pa_h2oi,Ti,Li*Tsi/Ti);
  g2A=new gasemissivityDATA("carbondioxide",pi,pa_co2i,Ti,Li*Tsi/Ti);
  double eps1A=g1A.eps;
  double eps2A=g2A.eps;
  double alpha1=eps1*Math.sqrt(Ti/Tsi);
  double alpha2=eps1*Math.sqrt(Ti/Tsi);
  double depsA=(fi/(10.7+101.0*fi)-0.0089*Math.pow(fi,10.4));
  depsA*=Math.pow((Math.log10((pa_h2oi+pa_co2i)*Li*(Ti/Tsi))),2.76);
  System.out.println(" eps1A="+ eps1A);
  System.out.println(" eps2A="+ eps2A);
  System.out.println("depsA="+ depsA);
  alpha=alpha1+alpha2-depsA;
  return eps;
}

public static void main(String args[])
{       double L=100.0; // cm gaz tabakası kalınlığı
            double pt=5.0; //bar
            double pa_h2o=0.2*pt; //h2o kısmi basınç
            double pa_co2=0.1*pt;//co2 kısmi basınç
            double Tg=1000.0; //K gaz sıcaklığı
            double Ts=1000.0;
            double sigma=5.67e-8;  // W/(m2 K4)
            gasemissivity_odev8 xx=new gasemissivity_odev8();
double eps_sonuc=xx.gasemissivity(pt,pa_h2o,pa_co2,Tg,Ts,L);
System.out.println("eps_sonuc = "+eps_sonuc);
double I=eps_sonuc*sigma*Ts*Ts*Ts*Ts/Math.PI;
DecimalFormat df = new DecimalFormat("0.00");
System.out.println("Intensity I="+I+" kW/m^2.sr");
}}
```

```
 public class gasemissivityDATA
{ public String gasname;
  public int M,N;     //number of rows .M+1 number of columns N+1
  public double c[][]; //coefficient c
  public double p,pa,p0,PE;    // pressure bar p total pressure pa.partial pressure
  public double T,T0;  // temperature K
  public double t; //temperature ratio T/T0
  public double a,b,cc;
  public double paLratio,paLratiom;
  public double L;
  public double eps;
  public gasemissivityDATA(String gasnamei,double pi,double pai,double Ti,double Li)
  {gasname=gasnamei;
  p=pi;
  pa=pai;
  T=Ti;
  T0=1000.0; //degree K;
  t=T/T0;
  p0=1.0; //bar
  L=Li;  //gas thickness cm
```

```
   if(gasname.equals("water"))
   {M=2;N=2;
    double ci[][]={{-2.2118, -1.1987,  0.035596},
                   { 0.85667, 0.93048,-0.14391},
                   {-0.10838,-0.17156, 0.045915}};
   c=ci;
   PE=(p+2.56*pa/Math.sqrt(t))/p0;

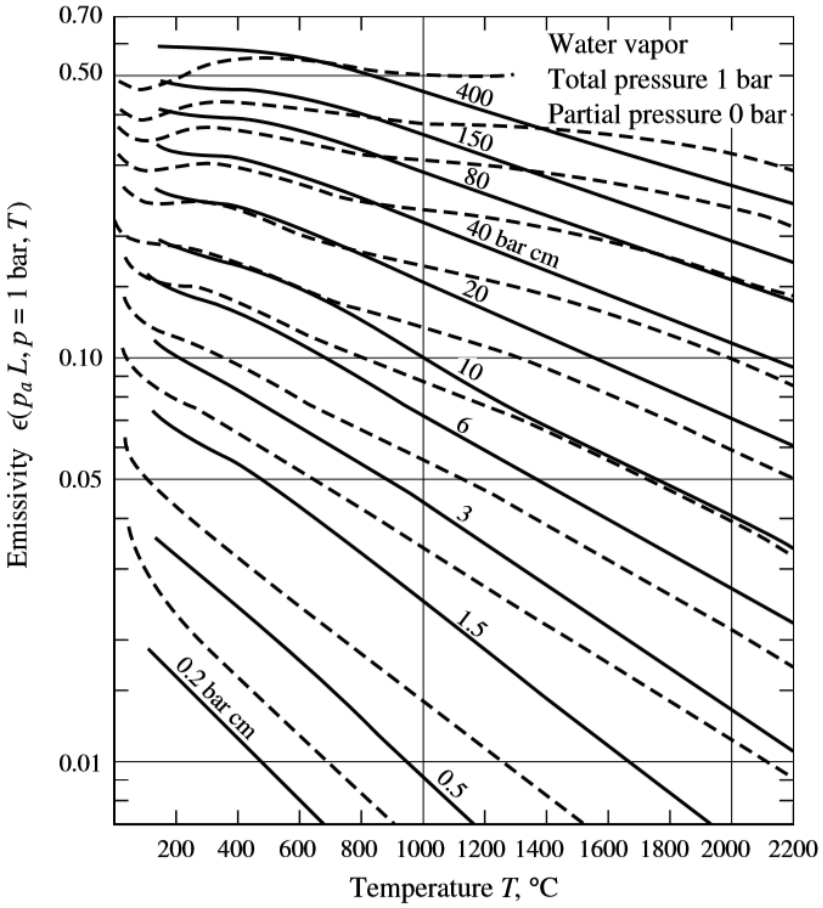

   paLratiom=13.2*t*t;
   if(t<0.75) a=2144.0;
   else a=1.888-2.053*Math.log10(t);
   b=1.1/Math.pow(t,1.4);
   cc=0.5;
   }
   else if(gasname.equals("carbondioxide"))
   {M=2;N=3;
    double ci[][]={{-3.9893,2.7669,-2.1081,0.39163},
                   {1.2710,-1.1090,1.0195,-0.21897},
                   {-0.23678,0.19731,-0.19544,0.044644}};
   c=ci;
   PE=(p+0.28*pa)/p0;
   if(t<0.7) paLratiom=0.054/(t*t);
   else paLratiom=0.225*t*t;
   a=1.0+0.1/Math.pow(t,1.45);
   b=.23;
   cc=1.47;
   }
  double eps0=0;
  double tratio=1.0;
  double pratio=1.0;
  for(int i=0;i<=M;i++)
  { tratio=1.0;
   for(int j=0;j<=N;j++)
   {   eps0+=c[i][j]*tratio*pratio;
                tratio*=t;
   }
   pratio*=Math.log10(pa*L);
  }
  eps0=Math.exp(eps0);
  double logpal=Math.log10(paLratiom/(pa*L));
  eps=eps0*(1.0-(a-1.0)*(1.0-PE)/(a+b-1.0+PE)*Math.exp(-cc*logpal*logpal));

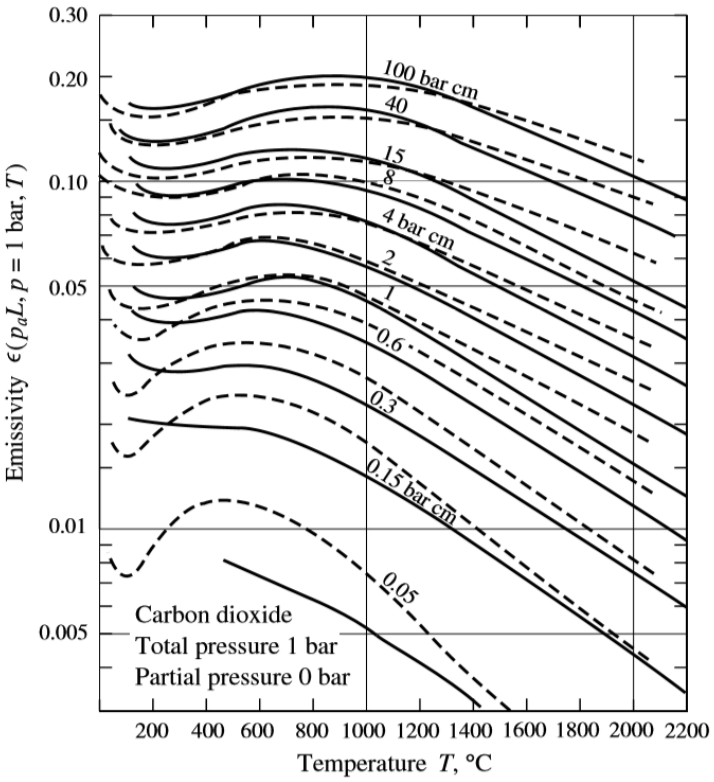
  }
}
```

```
 ---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" gasemissivity_odev8
 eps1A=0.5077387872915612
 eps2A=0.15680693133724524
depsA=0.07192742875261586
eps_sonuc = 0.5926182898761906
Intensity I=10695.675964732329 W/m^2.sr

> Terminated with exit code 0.
```

Values can be determined also from the charts below.

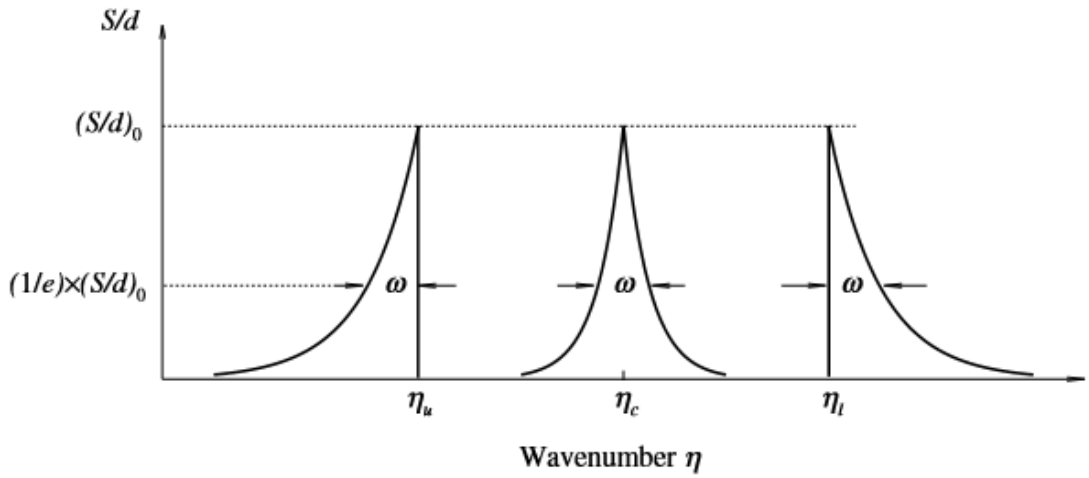




Wide band model

Another way of calculating emmissivity values of gases is using wide band model. It was mentioned that gas emmissivities are behaved as bands of emmission, instead of continious emmision curves observed in solid surfaces. In the exponential wide box model bands are modelled as exponential curves

as seen in the figure below. Where S/d is called smoothed absorbtion coefficient and Wavenumber, (measured in 1/cm) $\frac{1}{\lambda} = \eta$



It is assumed that there will be three kind of behavior of exponential band: with upper limit head band, symmetric band and with lower limit head band

```
public class WBM
{   //Reference :  Michael F. Modest. Radiative heat transfer 2013
    //C Euler-Mascheroni constant
            public static double C=0.57721566490153286060651209008240243l;
            public int m;
   public double s; //gas path m
   public double M; //molecular weight kg/kmol
   public int J;
   public String name; //gas name
   public double eta[];
  public double u[];
  public double u0[];
  public double band[];
  public double inlet_band;
  public int selected_band;
  public int gk[];
  public int delta[][];
  public int v0[][];
  public double P0,P,Pa,Pe,T0,T;
  double n,b,betastar0,w0;
  double alfa0[];
  double etac;
  // constructor method
  public WBM(String namei,double Ti,double Pi,double Pai,double si,int select)
  {name=namei;
   s=si;
   T=Ti; //degree K
   inlet_band=2898/T;
   P=Pi;  //atm
   Pa=Pai; //atm
   P0=1.0; //atm
   T0=100.0;
   //steam water
   if(name.equals("h2o") || name.equals("H2O") || name.equals("steam"))
   {double band1[]={71,6.3,2.7,1.87,1.38};band=band1;
    selected_band= select_band(select);
    double eta1[]={3652,1595,3756};eta=eta1;
    m=eta.length;
    M=18.015;  //kg/kmol
    u=new double[eta.length];
    u0=new double[eta.length];
    for(int k=0;k<eta.length;k++) {u[k]=1.4388*eta[k]/T;u0[k]=1.4388*eta[k]/T0;}
   if(selected_band==0)
   {etac=140.0;n=1;b=8.6*Math.sqrt(T0/T)+0.5;
   double alfa01[]={5.455};alfa0=alfa01;
   betastar0=0.143;w0=69.3;int delta1[][]={{0,0,0}};delta=delta1;
   }
   else if(selected_band==1) {etac=1600.0;n=1;b=8.6*Math.sqrt(T0/T)+0.5;
```

```
double alfa01[]={41.2};alfa0=alfa01;
betastar0=0.094;w0=56.4;int delta1[][]={{0,1,0}};delta=delta1;}
else if(selected_band==2) {etac=3760.0;n=1;b=8.6*Math.sqrt(T0/T)+0.5;
double alfa01[]={0.2,2.3,23.4};alfa0=alfa01;
betastar0=0.132;w0=60.0;int delta1[][]={{0,2,0},{1,0,0},{0,0,1}};delta=delta1;}
else if(selected_band==3) {etac=5350.0;n=1;b=8.6*Math.sqrt(T0/T)+0.5;
double alfa01[]={3};alfa0=alfa01;
betastar0=0.082;w0=43.1;int delta1[][]={{0,1,1}};delta=delta1;}
else if(selected_band==4) {etac=7250.0;n=1;b=8.6*Math.sqrt(T0/T)+0.5;
double alfa01[]={2.5};alfa0=alfa01;
betastar0=0.116;w0=32.0;int delta1[][]={{1,0,1}};delta=delta1;}
int gk1[]={1,1,1};gk=gk1;
}
// carbondioxide
if(name.equals("co2") || name.equals("CO2") || name.equals("carbondioxide"))
{double band1[]={15,10.4,9.4,4.3,2.7,2.0};band=band1;
selected_band= select_band(select);
 double eta1[]={1351,666,2396};eta=eta1;
 m=eta.length;
 M=44.01;
 u=new double[eta.length];
 u0=new double[eta.length];
 for(int k=0;k<eta.length;k++)
 {u[k]=1.4388*eta[k]/T;u0[k]=1.4388*eta[k]/T0;}
if(selected_band==0)
{etac=667;n=0.7;b=1.3;
double alfa01[]={19};alfa0=alfa01;
betastar0=0.06157;w0=12.7;int delta1[][]={{0,1,0}};delta=delta1;
}
else if(selected_band==1) {etac=960.0;n=0.8;b=1.3;
double alfa01[]={2.47e-9};alfa0=alfa01;
betastar0=0.04017;w0=13.4;int delta1[][]={{-1,0,1}};delta=delta1;}
else if(selected_band==2) {etac=1060.0;n=0.8;b=1.3;
double alfa01[]={2.48e-9};alfa0=alfa01;
betastar0=0.11888;w0=10.1;int delta1[][]={{0,-2,1}};delta=delta1;}
else if(selected_band==3) {etac=2410.0;n=0.8;b=1.3;
double alfa01[]={110};alfa0=alfa01;
betastar0=0.24723;w0=11.2;int delta1[][]={{0,0,1}};delta=delta1;}
else if(selected_band==4) {etac=3660.0;n=0.65;b=1.3;
double alfa01[]={4};alfa0=alfa01;
betastar0=0.13341;w0=23.5;int delta1[][]={{1,0,1}};delta=delta1;}
else if(selected_band==5) {etac=5200.0;n=0.65;b=1.3;
double alfa01[]={0.06};alfa0=alfa01;
betastar0=0.39305;w0=34.5;int delta1[][]={{2,0,1}};delta=delta1;}
int gk1[]={1,2,1};gk=gk1;
}
  // carbonmonoxide
if(name.equals("co") || name.equals("CO") || name.equals("carbonmonoxide"))
{double band1[]={4.7,2.35};band=band1;
selected_band= select_band(select);
 double eta1[]={2143};eta=eta1;
 m=eta.length;
 M=28.011;
 u=new double[eta.length];
 u0=new double[eta.length];
 for(int k=0;k<eta.length;k++)
 {u[k]=1.4388*eta[k]/T;u0[k]=1.4388*eta[k]/T0;}
if(selected_band==0)
{etac=2143;n=0.8;b=1.1;
double alfa01[]={20.9};alfa0=alfa01;
betastar0=0.07506;w0=25.5;int delta1[][]={{1}};delta=delta1;
}
else if(selected_band==1) {etac=4260;n=0.8;b=1.0;
double alfa01[]={0.14};alfa0=alfa01;
betastar0=0.16758;w0=20.0;int delta1[][]={{2}};delta=delta1;}
int gk1[]={1};gk=gk1;
}
  // methane
if(name.equals("ch4") || name.equals("CH4") || name.equals("methane"))
{double band1[]={7.7,3.3,2.4,1.7};band=band1;
selected_band= select_band(select);
 double eta1[]={2914,1526,3020,1306};eta=eta1;
 m=eta.length;
 M=16.043;
 u=new double[eta.length];
 u0=new double[eta.length];
 for(int k=0;k<eta.length;k++)
```

```
 {u[k]=1.4388*eta[k]/T;u0[k]=1.4388*eta[k]/T0;}
if(selected_band==0)
{etac=1310;n=0.8;b=1.3;
double alfa01[]={28};alfa0=alfa01;
betastar0=0.08698;w0=21.0;int delta1[][]={{0,0,0,1}};delta=delta1;
}
else if(selected_band==1)
{etac=3020;n=0.8;b=1.3;
double alfa01[]={46};alfa0=alfa01;
betastar0=0.06973;w0=56.0;int delta1[][]={{0,0,1,0}};delta=delta1;
}
else if(selected_band==2)
{etac=4220;n=0.8;b=1.3;
double alfa01[]={2.9};alfa0=alfa01;
betastar0=0.35249;w0=60.0;int delta1[][]={{1,0,0,1}};delta=delta1;
}
else if(selected_band==3)
{etac=5861;n=0.8;b=1.3;
double alfa01[]={0.42};alfa0=alfa01;
betastar0=0.68598;w0=45.0;int delta1[][]={{1,1,0,1}};delta=delta1;
}
int gk1[]={1,2,3,3};gk=gk1;
}
   // nitrousoxide
if(name.equals("no") || name.equals("NO") || name.equals(" nitrousoxide"))
{double band1[]={5.3};band=band1;
selected_band= select_band(select);
 double eta1[]={1876};eta=eta1;
 m=eta.length;
 M=30.006;
 u=new double[eta.length];
 u0=new double[eta.length];
 for(int k=0;k<eta.length;k++)
 {u[k]=1.4388*eta[k]/T;u0[k]=1.4388*eta[k]/T0;}
if(selected_band==0)
{etac=1876;n=0.65;b=1.0;
double alfa01[]={9};alfa0=alfa01;
betastar0=0.18050;w0=20.0;int delta1[][]={{1}};delta=delta1;
}
int gk1[]={1};gk=gk1;
}
// sulfurdioxide
if(name.equals("so2") || name.equals("sO2") || name.equals("sulfurdioxide"))
{double band1[]={19.3,8.7,7.3,4.3,4.0};band=band1;
selected_band= select_band(select);
 double eta1[]={1151,519,1361};eta=eta1;
 m=eta.length;
 M=44.01;
 u=new double[eta.length];
 u0=new double[eta.length];
 for(int k=0;k<eta.length;k++)
 {u[k]=1.4388*eta[k]/T;u0[k]=1.4388*eta[k]/T0;}
if(selected_band==0)
{etac=519;n=0.7;b=1.28;
double alfa01[]={4.22};alfa0=alfa01;
betastar0=0.05291;w0=33.1;int delta1[][]={{0,1,0}};delta=delta1;
}
else if(selected_band==1)
{etac=1151;n=0.7;b=1.28;
double alfa01[]={3.67};alfa0=alfa01;
betastar0=0.05952;w0=24.8;int delta1[][]={{1,0,0}};delta=delta1;
}
else if(selected_band==2)
{etac=1361;n=0.65;b=1.28;
double alfa01[]={29.97};alfa0=alfa01;
betastar0=0.49299;w0=8.8;int delta1[][]={{0,0,1}};delta=delta1;
}
else if(selected_band==3)
{etac=2350;n=0.6;b=1.28;
double alfa01[]={0.423};alfa0=alfa01;
betastar0=0.47513;w0=16.5;int delta1[][]={{2,0,0}};delta=delta1;
}
else if(selected_band==4)
{etac=2512;n=0.6;b=1.28;
double alfa01[]={0.346};alfa0=alfa01;
betastar0=0.58937;w0=10.9;int delta1[][]={{1,0,1}};delta=delta1;
}
```

```
 int gk1[]={1,1,1};gk=gk1;
 }
 J=delta.length;
 v0=new int[J][m];
 for(int jj=0;jj<J;jj++)
 {for(int k=0;k<m;k++)
  { if(delta[jj][k]>=0) v0[jj][k]=0;
    else v0[jj][k]=Math.abs(delta[jj][k]);
  }
 }
 Pe=Math.pow((P/P0*(1.0+(b-1.0)*Pa/P)),n);
}

public int select_band(int select)
{if(select<0 || select >= band.length)
 {
 double dband=1.0e30;
 double db=0;
 int k=0;
 for(int i=0;i<band.length;i++)
 {   db=Math.abs(inlet_band-band[i]);
              if(db<dband) {dband=db;k=i;}
 }
 return k;
}
else
{return select;}
}
 public static double factorial(int nn)
 {   if(nn==0) return 1;
     else if(nn==1) return 1;
              double fact=1.0;
              for(int i=1;i<=nn;i++) fact*=i;
              return fact;
 }
public double fi(int jj)
{double f1=1.0,f2=1.0;
 double total1=0,total2=0;
 for(int k=0;k<m;k++)
 {   double x1,x2,x3,x4;
     int xx1,xx2;
              for(int vk=v0[jj][k];vk<20;vk++)
    { xx1=vk+gk[k]+Math.abs(delta[jj][k])-1;
      xx2=gk[k]-1;
               x1=factorial(xx1);
      x2=factorial(xx2);
      x3=factorial(vk);
      x4=x1/x2/x3*Math.exp(-u[k]*vk);
               total1+=Math.sqrt(x4);
               total2+=x4;
              }
              f1*=total1;
              f2*=total2;
 }
 return f1*f1/f2;
}
public double fi0(int jj)
{double f1=1.0,f2=1.0;
 double total1=0,total2=0;
 for(int k=0;k<m;k++)
 {   double x1,x2,x3,x4;
     int xx1,xx2;
              for(int vk=v0[jj][k];vk<20;vk++)
    { xx1=vk+gk[k]+Math.abs(delta[jj][k])-1;
      xx2=gk[k]-1;
               x1=factorial(xx1);
      x2=factorial(xx2);
      x3=factorial(vk);
      x4=x1/x2/x3*Math.exp(-u0[k]*vk);
               total1+=Math.sqrt(x4);
               total2+=x4;
              }
              f1*=total1;
              f2*=total2;
 }
 return f1*f1/f2;
}
```

```
public double psi(int jj)
{double f1=1.0,f2=1.0;
 double total1=0,total2=0,total3=0;
 for(int k=0;k<m;k++)
 {   double x1,x2,x3,x4;
     int xx1,xx2;
                for(int vk=v0[jj][k];vk<20;vk++)
    { xx1=vk+gk[k]+Math.abs(delta[jj][k])-1;
      xx2=gk[k]-1;
                 x1=factorial(xx1);
      x2=factorial(xx2);
      x3=factorial(vk);
      x4=x1/x2/x3*Math.exp(-u[k]*vk);
                 total1+=x4;
                }
                for(int vk=0;vk<20;vk++)
    { xx1=vk+gk[k]-1;
      xx2=gk[k]-1;
                 x1=factorial(xx1);
      x2=factorial(xx2);
      x3=factorial(vk);
      x4=x1/x2/x3*Math.exp(-u[k]*vk);
                 total2+=x4;
                }
               total3+=u[k]*delta[jj][k];
               f1*=total1;
               f2*=total1;
    }
 return f1/f2*(1.0-Math.exp(-total3));
 }
public double psi0(int jj)
{double f1=1.0,f2=1.0;
 double total1=0,total2=0,total3=0;
 for(int k=0;k<m;k++)
 {   double x1,x2,x3,x4;
     int xx1,xx2;
                for(int vk=v0[jj][k];vk<20;vk++)
    { xx1=vk+gk[k]+delta[jj][k]-1;
      xx2=gk[k]-1;
                 x1=factorial(xx1);
      x2=factorial(xx2);
      x3=factorial(vk);
      x4=x1/x2/x3*Math.exp(-u0[k]*vk);
                 total1+=x4;
                }
                for(int vk=0;vk<20;vk++)
    { xx1=vk+gk[k]-1;
      xx2=gk[k]-1;
                 x1=factorial(xx1);
      x2=factorial(xx2);
      x3=factorial(vk);
      x4=x1/x2/x3*Math.exp(-u0[k]*vk);
                 total2+=x4;
                }
                 total3+=u[k]*delta[jj][k];
               f1*=total1;
               f2*=total1;
    }
 return f1/f2*(1.0-Math.exp(-total3));
 }
public double alpha(int jj)
{ double alfa=alfa0[jj]*psi(jj)/psi0(jj);
  return alfa;
}
public double alpha()
{ double total=0;
 for(int jj=0;jj<J;jj++)
 {total+=alpha(jj);}
 return total;
}
public double roa() //density of the gas
{double R=8314.5/M;
 double ro=Pa*1.0132e5*1.0e3/(R*T);
 return ro;  //gr/m^3
}
public double beta(int jj)
```

```
 { double bb=betastar0*Math.sqrt(T0/T)*fi(jj)/fi0(jj)*Pe;
   return bb;
 }
 public double beta()
 { double total=0;
  for(int jj=0;jj<J;jj++)
  {total+=Math.sqrt(alpha(jj)*beta(jj));}
  return total*total/alpha();
 }

 public double betastar(int jj)
 { return beta(jj)/Pe;}
 public double betastarratio(int jj)
 { return beta(jj)/Pe/betastar0;}
  public double betastar()
 { return beta()/Pe;}
 public double betastarratio()
 { return beta()/Pe/betastar0;}
 public double X() //density of the gas
 {return roa()*s;  //gr/m^2
 }
 public double w()
 { double bb=w0*Math.sqrt(T/T0); return bb;}

 public double to()
 {return alpha()*X()/w();}
 // Edwards and Menard model
 public double Astar()
 { double As=0.0;
            double beta=beta();
   double to=to();
   System.out.println("to="+to+"1.0/beta="+1.0/beta);
   if(beta<=1.0)
   {
   if(to>=0 && to<=beta) {As=to;}
   else if(to>=beta && to<=1.0/beta) {As=2.0*Math.sqrt(to*beta)-beta;}
   else if(to>=1.0/beta) {System.out.println("true");As=Math.log(to*beta)+2.0-beta;}
   }
   else
   {if(to>=0 && to<=1) {As=to;}
    else if(to>=1.0) {As=Math.log(to)+1.0;}
   }
   return As;
   }
   // Felske and Tien model
   public double Astar1()
   { double As=0.0;
            double beta=beta();
   double to=to();
   double ww=to/Math.sqrt(1.0+to/beta);
   As=2.0*Ein(ww)+Ein(ww/(2.0*beta))-Ein((1.0/(2.0*beta)+1.0)*ww);
   return As;
   }
   // Wang model
   public double Astar2()
   { double As=0.0;
            double beta=beta();
   double to=to();
   double ww=beta/2.0*(Math.sqrt(1.0+4.0*to/beta)-1.0);
   As=Math.exp(beta)*(E1(beta+ww)-E1(beta))+Math.log(1.0+ww/beta)+Ein(ww);
   return As;
   }
    public double A()
    {return Astar()*w();}

    public double A1()
    {return Astar1()*w();}

    public double A2()
    {return Astar2()*w();}

 //Exponential functions
            //exponential integral section
            public static double AAS_Ei(double x)
{
    double ei[] = {
    1.915047433355013959531e2,  4.403798995348382689974e2,
```

```
      1.037878290717089587658e3,  2.492228976241877759138e3,
      6.071406374098611507965e3,  1.495953266639752885229e4,
      3.719768849068903560439e4,  9.319251363396537129882e4,
      2.349558524907683035782e5,  5.955609986708370018502e5,
      1.516637894042516884433e6,  3.877904330597443502996e6,
      9.950907251046844760026e6,  2.561565266405658882048e7,
      6.612718635548492136250e7,  1.711446713003636684975e8,
      4.439663698302712208698e8,  1.154115391849182948287e9,
      3.005950906525548689841e9,  7.842940991898186370453e9,
      2.049649711988081236484e10, 5.364511859231469415605e10,
      1.405991957584069047340e11, 3.689732094072741970640e11,
      9.694555759683939661662e11, 2.550043566357786926147e12,
      6.714640184076497558707e12, 1.769803724411626854310e13,
      4.669055014466159544500e13, 1.232852079912097685431e14,
      3.257988998672263996790e14, 8.616388199965786544948e14,
      2.280446200301902595341e15, 6.039718263611241578359e15,
      1.600664914324504111070e16, 4.244796092136850759368e16,
      1.126348290166966760275e17, 2.990444718632336675058e17,
      7.943916035704453771510e17, 2.111342388647824195000e18,
      5.614329680810343111535e18, 1.493630213112993142255e19,
      3.975442747903744836007e19, 1.058563689713169096306e20
    };
    int  k = (int) (x + 0.5);
    int  j = 0;
    double xx = (double) k;
    double dx = x - xx;
    double xxj = xx;
    double edx = Math.exp(dx);
    double Sm = 1.0;
    double Sn = (edx - 1.0) / xxj;
    double max_double=1.7976931348623157e+308;
    double eps=1e-20;
    double term = max_double;
    double fact = 1.0;
    double dxj = 1.0;

    while (Math.abs(term) > eps * Math.abs(Sn) ) {
      j++;
      fact *= (double) j;
      xxj *= xx;
      dxj *= (-dx);
      Sm += (dxj / fact);
      term = ( fact * (edx * Sm - 1.0) ) / xxj;
      Sn += term;
    }

    return ei[k-7] + Sn * Math.exp(xx);
  }
  //Power series Ei
  public static double PS_Ei( double x )
  {  double max_double=1.7976931348623157e+308;
    double eps=1e-20;
    double xn = -x;
    double Sn = -x;
    double Sm1 = 0.0;
    double hsum = 1.0;
    double g = C;//Euler number ;
    double y = 1.0;
    double fact = 1.0;
    if ( x == 0.0 ) return  -max_double;

    while ( Math.abs(Sn - Sm1) > eps * Math.abs(Sm1) ) {
      Sm1 = Sn;
      y += 1.0;
      xn *= (-x);
      fact *= y;
      hsum += (1.0 / y);
      Sn += hsum * xn / fact;
    }
    return (g + Math.log(Math.abs(x)) - Math.exp(x) * Sn);
  }
  //Continued Faraction Ei
  public static double CF_Ei( double x )
  {  double max_double=1.7976931348623157e+308;
    double eps=1e-20;
    double Am1 = 1.0;
    double A0 = 0.0;
```

```
    double Bm1 = 0.0;
    double B0 = 1.0;
    double a = Math.exp(x);
    double b = -x + 1.0;
    double Ap1 = b * A0 + a * Am1;
    double Bp1 = b * B0 + a * Bm1;
    int j = 1;
    a = 1.0;
    while ( Math.abs(Ap1 * B0 - A0 * Bp1) > eps * Math.abs(A0 * Bp1) ) {
      if ( Math.abs(Bp1) > 1.0) {
        Am1 = A0 / Bp1;
        A0 = Ap1 / Bp1;
        Bm1 = B0 / Bp1;
        B0 = 1.0;
      } else {
        Am1 = A0;
        A0 = Ap1;
        Bm1 = B0;
        B0 = Bp1;
      }
      a = -j * j;
      b += 2.0;
      Ap1 = b * A0 + a * Am1;
      Bp1 = b * B0 + a * Bm1;
      j += 1;
    }
    return (-Ap1 / Bp1);
  }
  public static double Ei( double x )
  {  double max_double=1.7976931348623157e+308;
    if ( x < -5.0 ) return CF_Ei(x);
    if ( x == 0.0 ) return -max_double;
    if ( x < 6.8 )  return PS_Ei(x);
    if ( x < 50.0 ) return AAS_Ei(x);
    return CF_Ei(x);
  }
  public static double E1( double x )
  {return -Ei(-x);}
  public static double Ein( double x )
  {return E1(x)+Math.log(x)+C;}
  //end of exponential functions
    public static void exampleH2O()
    {WBM ww=new WBM("h2o",600.0,3.0,0.05*3.0,0.1,-1);
    int ks=ww.selected_band;
    System.out.println("ks="+ks+"inlet_band="+ww.inlet_band+"J="+ww.J);
    System.out.println("betastarratio="+ww.betastarratio()+"betastar0="+ww.betastar0+"b="+ww.b+"Pe="+ww.Pe);
    System.out.println("beta="+ww.beta());
    System.out.println("w="+ww.w());
    System.out.println("alpha = "+ww.alpha()+"beta="+ww.beta());
    System.out.println("ro="+ww.roa());
    System.out.println("X="+ww.X());
    System.out.println("to="+ww.to());
    System.out.println("As="+ww.Astar());
    System.out.println("A="+ww.A());
    System.out.println("As1="+ww.Astar1());
    System.out.println("A1="+ww.A1());
    System.out.println("As2="+ww.Astar2());
    System.out.println("A2="+ww.A2());
    }
    public static void exampleCO()
    {WBM ww=new WBM("co",800.0,1.0,1.0,0.2,-1);
    int ks=ww.selected_band;
    System.out.println("ks="+ks+"inlet_band="+ww.inlet_band+"J="+ww.J);
    System.out.println("betastarratio="+ww.betastarratio()+"betastar0="+ww.betastar0+"b="+ww.b+"Pe="+ww.Pe);
    System.out.println("w="+ww.w());
    System.out.println("alpha = "+ww.alpha()+"beta="+ww.beta());
    System.out.println("ro="+ww.roa());
    System.out.println("X="+ww.X());
    System.out.println("to="+ww.to());
    System.out.println("As="+ww.Astar());
    System.out.println("A="+ww.A());
    System.out.println("As1="+ww.Astar1());
    System.out.println("A1="+ww.A1());
    System.out.println("As2="+ww.Astar2());
    System.out.println("A2="+ww.A2());
    System.out.println("M="+ww.M);}
   public static double F(double x)
```

```
  {
  if(x<=0) return 0;
  double C2=1.438769e4;
  double y=C2/x;
  double total=0;
  for(int n=1;n<80;n++)
  {total+=Math.exp(-n*y)/n*(y*y*y+3.0*y*y/n+6.0*y/(n*n)+6.0/(n*n*n));}
  double pi=Math.PI;
  total=15.0/(pi*pi*pi*pi)*total;
  return total;
  }
public static double Eb(double nu,double T)
{   //lambda in micrometers Eb_nu/T^3
              double  k=1.3807e-23; // J/K
    double  h=6.626e-34;  // Js
    double  c0=2.998e8;   //m/s
    double  C1=2.0*Math.PI*h*c0*c0; //    Wm^2
    double  C2=h*c0/k*1e2;  //mikrometer K
    double nuoT=nu/T;
             double e=C1*nuoT*nuoT*nuoT*1e8/(Math.exp(C2*nuoT)-1.0);
    return e;
}

public static double eps(String name,double T,double P,double Pa,double s)
{   double PI=Math.PI;
  double  k=1.3807e-23; // J/K
  double  h=6.626e-34;  // Js
  double  c0=2.998e8;   //m/s
  double  C1=2.0*PI*h*c0*c0; //    W/(mikrometer)^4/(m^2sr)
  double C2=h*c0/k;  //mikrometer K

  double sigma=C1/(C2*C2*C2*C2)*PI*PI*PI*PI/15.0;
  WBM ww=new WBM(name,T,P,Pa,s,-1);
  int n=ww.band.length;
  WBM w1[]=new WBM[n];
  double eps[]=new double[n];
  double nu[]=new double[n];
  double A[]=new double[n];
  double ep=0;
  for(int i=0;i<n;i++)
  {w1[i]=new WBM(name,T,P,Pa,s,i);
   A[i]=w1[i].A();
   nu[i]=w1[i].etac;
   eps[i]=Eb(nu[i],T)*A[i]/(sigma*T);
   System.out.println("i="+i+"nu="+nu[i]+"eps="+eps[i]+"A="+A[i]);
   ep+=eps[i];
  }
   System.out.println("eps total ="+ep);
  return ep;
}
public static double eps1(String name,double T,double P,double Pa,double s)
{   double PI=Math.PI;
  double  k=1.3807e-23; // J/K
  double  h=6.626e-34;  // Js
  double  c0=2.998e8;   //m/s
  double  C1=2.0*PI*h*c0*c0; //    W/(mikrometer)^4/(m^2sr)
  double C2=h*c0/k;  //mikrometer K

  double sigma=C1/(C2*C2*C2*C2)*PI*PI*PI*PI/15.0;
  WBM ww=new WBM(name,T,P,Pa,s,-1);
  int n=ww.band.length;
  WBM w1[]=new WBM[n];
  double eps[]=new double[n];
  double nu[]=new double[n];
  double A[]=new double[n];
  double ep=0;
  for(int i=0;i<n;i++)
  {w1[i]=new WBM(name,T,P,Pa,s,i);
   A[i]=w1[i].A1();
   nu[i]=w1[i].etac;
   eps[i]=Eb(nu[i],T)*A[i]/(sigma*T);
   ep+=eps[i];
  }
  return ep;
}
public static double eps2(String name,double T,double P,double Pa,double s)
{   double PI=Math.PI;
```

```
  double  k=1.3807e-23; // J/K
  double  h=6.626e-34;  // Js
  double  c0=2.998e8;   //m/s
  double  C1=2.0*PI*h*c0*c0; //    W/(mikrometer)^4/(m^2sr)
  double C2=h*c0/k;  //mikrometer K

  double sigma=C1/(C2*C2*C2*C2)*PI*PI*PI*PI/15.0;
  WBM ww=new WBM(name,T,P,Pa,s,-1);
  int n=ww.band.length;
  WBM w1[]=new WBM[n];
  double eps[]=new double[n];
  double nu[]=new double[n];
  double A[]=new double[n];
  double ep=0;
  for(int i=0;i<n;i++)
  {w1[i]=new WBM(name,T,P,Pa,s,i);
   A[i]=w1[i].A2();
   nu[i]=w1[i].etac;
   eps[i]=Eb(nu[i],T)*A[i]/(sigma*T);
   ep+=eps[i];
  }
  return ep;
 }

  public static void main(String arg[])
  { //exampleH2O();
    //exampleCO();
    System.out.println("eps="+eps("co",800.0,1.0,1.0,0.2));
    System.out.println("eps1="+eps1("co",800.0,1.0,1.0,0.2));
    System.out.println("eps2="+eps2("co",800.0,1.0,1.0,0.2));
  }
 }
```

```
 public class WBM_DATA
 { public int m;
   public int J;
   public String name;
   public double eta[];
   public double u[];
   public double u0[];
   public double band[];
   public double inlet_band;
   public int selected_band;
   public int gk[];
   public int delta[][];
   public int v0[][];
   public double P0,P,Pa,Pe,T0,T;
   double n,b,gamma0,w0;
   double alfa0[];
   double etac;
   public int select_band()
   {double dband=1.0e30;
    double db=0;
    int k=0;
    for(int i=0;i<band.length;i++)
    {   db=Math.abs(inlet_band-band[i]);
                if(db<dband) {dband=db;k=i;}
    }
    return k;
   }
    public static double factorial(int nn)
    {   double fact=1.0;
                for(int i=1;i<=nn;i++) fact*=i;
                return fact;
    }
   public double fi(int jj)
   {double f1=1.0,f2=1.0;
    double total1=0,total2=0;
    for(int k=0;k<m;k++)
    {   double x1,x2,x3,x4;
        int xx1,xx2;
                for(int vk=v0[jj][k];vk<20;vk++)
       { xx1=vk+gk[k]+delta[jj][k]-1;
         xx2=gk[k]-1;
                  x1=factorial(xx1);
         x2=factorial(xx2);
```

```
        x3=factorial(vk);
        x4=x1/x2/x3*Math.exp(-u[k]*vk);
                    total1+=Math.sqrt(x4);
                    total2+=x4;
                   }
                   f1*=total1;
                   f2*=total2;
  }
  return f1*f1/f2;
 }
 public double fi0(int jj)
 {double f1=1.0,f2=1.0;
  double total1=0,total2=0;
  for(int k=0;k<m;k++)
  {   double x1,x2,x3,x4;
      int xx1,xx2;
                for(int vk=v0[jj][k];vk<20;vk++)
     { xx1=vk+gk[k]+delta[jj][k]-1;
       xx2=gk[k]-1;
                    x1=factorial(xx1);
       x2=factorial(xx2);
       x3=factorial(vk);
       x4=x1/x2/x3*Math.exp(-u0[k]*vk);
                    total1+=Math.sqrt(x4);
                    total2+=x4;
                   }
                   f1*=total1;
                   f2*=total2;
  }
  return f1*f1/f2;
 }
 public double psi(int jj)
 {double f1=1.0,f2=1.0;
  double total1=0,total2=0;
  for(int k=0;k<m;k++)
  {   double x1,x2,x3,x4;
      int xx1,xx2;
                for(int vk=v0[jj][k];vk<20;vk++)
     { xx1=vk+gk[k]+delta[jj][k]-1;
       xx2=gk[k]-1;
                    x1=factorial(xx1);
       x2=factorial(xx2);
       x3=factorial(vk);
       x4=x1/x2/x3*Math.exp(-u[k]*vk);
                    total1+=x4;
                   }
                   for(int vk=0;vk<20;vk++)
     { xx1=vk+gk[k]-1;
       xx2=gk[k]-1;
                    x1=factorial(xx1);
       x2=factorial(xx2);
       x3=factorial(vk);
       x4=x1/x2/x3*Math.exp(-u[k]*vk);
                    total2+=x4;
                   }
                   f1*=total1;
                   f2*=total1;
     }
  return f1/f2;
  }
 public double alpha(int jj)
 {double total=0;
 double total0=0;
  for(int k=0;k<m;k++)
  {total+=u[k]*delta[jj][k];
   total0+=u0[k]*delta[jj][k];
  }
  double bb=1.0-Math.exp(-total);
  double bb0=1.0-Math.exp(-total0);
  return alfa0[jj]*bb*psi(jj)/(bb0*psi0(jj));
 }
 public double alpha()
 { double total=0;
  for(int jj=0;jj<J;jj++)
  {total+=alpha(jj);}
  return total;
 }
```

```
public double psi0(int jj)
{double f1=1.0,f2=1.0;
 double total1=0,total2=0;
 for(int k=0;k<m;k++)
 {   double x1,x2,x3,x4;
     int xx1,xx2;
              for(int vk=v0[jj][k];vk<20;vk++)
    { xx1=vk+gk[k]+delta[jj][k]-1;
      xx2=gk[k]-1;
               x1=factorial(xx1);
      x2=factorial(xx2);
      x3=factorial(vk);
      x4=x1/x2/x3*Math.exp(-u0[k]*vk);
               total1+=x4;
              }
              for(int vk=0;vk<20;vk++)
    { xx1=vk+gk[k]-1;
      xx2=gk[k]-1;
               x1=factorial(xx1);
      x2=factorial(xx2);
      x3=factorial(vk);
      x4=x1/x2/x3*Math.exp(-u0[k]*vk);
               total2+=x4;
              }
              f1*=total1;
              f2*=total1;
    }
 return f1/f2;
 }
public double beta(int jj)
{ double bb=gamma0*Math.sqrt(T0/T)*fi(jj)/fi0(jj)*Pe;
  return bb;
}
public double beta()
{ double total=0;
 for(int jj=0;jj<J;jj++)
 {total+=beta(jj);}
 return total;
}
public double w()
{ double bb=w0*Math.sqrt(T0/T);
  return bb;
}
public WBM_DATA(String namei,double Ti,double Pi,double Pai)
{name=namei;
 T=Ti; //degree K
 inlet_band=2898/T;
 P=Pi;  //atm
 Pa=Pai; //atm
 P0=1.0; //atm
 T0=100.0;
 if(name.equals("h2o") || name.equals("H2O") || name.equals("steam"))
 {double band1[]={71,6.3,2.7,1.87,1.38};band=band1;
 selected_band= select_band();
  double eta1[]={3652,1595,3756};eta=eta1;
  u=new double[eta.length];
  u0=new double[eta.length];
  for(int k=0;k<eta.length;k++) {u[k]=1.4388*eta[k]/T;u0[k]=1.4388*eta[k]/T0;}
 if(selected_band==0) {etac=140.0;n=1;b=8.6*Math.sqrt(T0/T)+0.5;
 double alfa01[]={44205};alfa0=alfa01;
 gamma0=0.14311;w0=69.3;int delta1[][]={{0,0,0}};delta=delta1;}
 else if(selected_band==1) {etac=1600.0;n=1;b=8.6*Math.sqrt(T0/T)+0.5;
 double alfa01[]={41.2};alfa0=alfa01;
 gamma0=0.09427;w0=56.4;int delta1[][]={{0,0,0}};delta=delta1;}
 else if(selected_band==2) {etac=3760.0;n=1;b=8.6*Math.sqrt(T0/T)+0.5;
 double alfa01[]={0.2,2.3,23.4};alfa0=alfa01;
 gamma0=0.13219;w0=60.0;int delta1[][]={{0,2,0},{1,0,0},{0,0,1}};delta=delta1;}
 else if(selected_band==3) {etac=5350.0;n=1;b=8.6*Math.sqrt(T0/T)+1.5;
 double alfa01[]={3};alfa0=alfa01;
 gamma0=0.08169;w0=43.1;int delta1[][]={{0,1,1}};delta=delta1;}
 else if(selected_band==4) {etac=7250.0;n=1;b=8.6*Math.sqrt(T0/T)+1.5;
 double alfa01[]={2.5};alfa0=alfa01;
 gamma0=0.11628;w0=32.0;int delta1[][]={{1,0,1}};delta=delta1;}
 int gk1[]={1,1,1};gk=gk1;
 }
```

```
   J=delta.length;
   Pe=Math.pow((P/P0*(1.0+(b-1.0)*Pa/P)),n);
  }
  public static void main(String arg[])
  {WBM_DATA ww=new WBM_DATA("h2o",600.0,3.0,0.05*3.0);
   int ks=ww.selected_band;
   System.out.println("ks="+ks+"inlet_band="+ww.inlet_band+"J="+ww.J);
  }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" WBM
to=24.7279878035179481.0/beta=23.32659519453048
true
i=0nu=2143.0eps=0.04990125289203836A=145.365703631297
to=0.211193580044399731.0/beta=10.39406576817441
i=1nu=4260.0eps=6.265997937345691E-4A=10.68457542973929
eps total =0.05052785268577293
eps=0.05052785268577293
eps1=0.03877879029360184
eps2=0.03713943910412516

> Terminated with exit code 0.
```

REFERENCES

1. **Barr, D.I.H.,** "Solutions of the Colebrook-White functions for resistance to uniform turbulent flows.", Proc. Inst. Civil. Engrs. Part 2. 71,1981.
2. **Chen, N.H.,** "An Explicit Equation for Friction factor in Pipe", Ind. Eng. Chem. Fundam., Vol. 18, No. 3, 296-297, 1979.
3. **Churchill, S.W.,** "Friction factor equations spans all fluid-flow ranges.", Chem. Eng., 91,1977.
4. **Colebrook, C.F. and White, C.M.,** "Experiments with Fluid friction roughened pipes.",Proc. R.Soc.(A), 161,1937.
5. **Haaland, S.E.,** "Simple and Explicit formulas for friction factor in turbulent pipe flow.", Trans. ASME, JFE, 105, 1983.
6. **Liou, C.P.,** "Limiations and proper use of the Hazen-Williams equations.", J. Hydr., Eng., 124(9), 951-954, 1998.
7. **Manadilli, G.,** "Replace implicit equations with sigmoidal functions.", Chem.Eng. Journal, 104(8), 1997.
8. **McKeon, B.J., Swanson, C.J., Zagarola, M.V., Donnelly, R.J. and Smits, A.J.,** "Friction factors for smooth pipe flow.", J.Fluid Mechanics, Vol.541, 41-44, 2004.
9. **Moody, L.F.,** "Friction factors for pipe flows.", Trans. ASME, 66,641,1944.
10. **Nikuradse, J.** "Stroemungsgesetze in rauhen Rohren." Ver. Dtsch. Ing. Forsch., 361, 1933.
11. **Romeo, E., Royo, C., and Monzon, A**., ''Improved explicit equations for estimation of the friction factor in rough and smooth pipes.'' *Chem. Eng. J.*, 86, 369–374, 2002.
12. **Round, G.F.,** "An explicit approximation for the friction factor-Reynolds number relation for rough and smooth pipes.", Can. J. Chem. Eng., 58,122-123,1980.
13. **Schlichting, H.,** "Boundary-Layer Theory" ,McGraw–Hill, New York, 1979..
14. **Swamee, P.K. and Jain, A.K.,** "Explicit equation for pipe flow problems.", J. Hydr. Div., ASCE, 102(5), 657-664, 1976.
15. **U.S. Bureau of Reclamation.**, "Friction factors for large conduit flowing full." Engineering Monograph, No. 7, U.S. Dept. of Interior, Washington, D.C, 1965.
16. **Von Bernuth, R. D., and Wilson, T.**, "Friction factors for small diameter plastic pipes." J. Hydraul. Eng., 115(2), 183–192, 1989. kg/m^3

17. **Wesseling, J., and Homma, F**., "Hydraulic resistance of drain pipes." Neth. J. Agric. Sci., 15, 183–197, 1967.
18. **Wood, D.J.,** "An Explicit friction factor relationship.", Civil Eng., 60-61,1966.
19. **Zagarola, M. V.,** ''Mean-flow Scaling of Turbulent Pipe Flow,'' Ph.D.thesis, Princeton University, USA, 1996.
20. **Zigrang, D.J. and Sylvester, N.D.,** "Explicit approximations to the Colebrook's friction factor.", AICHE J. 28, 3, 514, 1982.
21. **Goudar, C.T. and Sonnad, J.R. ,** "Comparison of the iterative approximations of the Colebrook-White equation", Hydrocarbon Processing, August 2008, pp 79-83
22. **Serghides, T.K.,** "Estimate friction factor accurately", Chem. Eng. 91, 1984, pp. 63-64
23. **White, Frank M.,** "Fluid Mechanics", Fourth Edition, McGrawHill, 1998, ISBN 0-07-069716-7
24. **Dittus, F.W. and L.M Boelter,** University of CaliforniaPublications on Engineering, Vol 2 p 443, Berkeley 1930
25. **Sieder E. N., and G.E. Tate**, Ind. Eng. Chem., 28,1429,1936
26. **Hausen, H.**, Z. VDI Beih. Verfahrenstech., 4,91,1943
27. **Zhukauskas, A.** "Heat tranfer from Tubes in Cross Flow," in J.P. Harnett and T. F. Irvine, Jr, Eds, Advances in Heat Transfer, Vol 8, Academic Press, New York 1972
28. **Whiteaker, S**, AIChe J., 18, 361, 1972.
29. **Zhukauskas, A. And A. B. Ambrazyavichyus,** Int. J. Heat Mass Transfer, 3 305, 1961***, S.W., and M. Bernstein** , J. Heat Transfer, Vol 8, Academic Press, New York 1972.
31. **Hilpert, R**. Forch. Geb. Ingenieurwes., 4,215,1933
32. **Petukhov**,B.S. in T.F. Irvine and J.P. Harnett, Eds., Advances in Heat Transfer, Vol. &., Academic Press, New York, 1970
33. **Gnielinski**, New Equations for heat and mass transfer in turbulent pipe and channel flow, V., Int. Chem. Eng. , 16, 359-367, 1976
34. **Abraham, J.P., Sparrow, E.M., Tong,J.C.K**, Heat transfer in all pipe flow regimes: Laminar, transitional/intermittent and turbulent, International Journal of Heat and Mass Transfer 52 (2009) 557-563
35. **Incropere F. P., De Witt D.**,Fundamentals of Heat and Mass Transfer, Third EfitionWilley publication 1990
36. **Churchill, S.W. and H.H.S. Chu**, Correlating Equations for Laminar and Turbulent Free Convection from a vertical plate, Int. J. Heat Mass Transfer, 18, 1323, 1975
37. **Morgan, V. T**. The Overall Convective Heat Transfer from Smooth Circular Cylinders, in T.F. Irvine and J.P. Harnett, Eds., Advances in Heat Transfer Vol 11, Academic Press, New York, 1975, pp. 199-264
38. **Churchill, S.W. and H.H.S. Chu**, Int. J. Heat Mass Transfer, 18, 1049, 1975
39. **Madhusudana, C.V**., Thermal Contact Conductance, Springer, 1995, ISBN 13:978-1-4612-8457-4
40. **Mikhailov, M.D., Özışık**, M. N., Unified analysis and solutions of heat and mass diffusion, Dover publications,1984, ISBN 0-486-67876-8
41. **Özışık, Necati**, Heat Conduction, Second edition, John Wiley and Sons, Inc. ISBN 0-471-53256
42. **Özışık, Necati**, Heat transfer, a basic approach, Mc Graw Hill Book Company, ISBN 0-07-047982-8
43**. Holman**, J. P. Heat Transfer, 10th edition, Mc Graw Hill Book Company, ISBN 978-0-07-352936-3
44**. Modest**, Michael, Radiative heat transfer,second edition, Academic press, ISBN 0-12-503163-7
45. Rohsenow, W.M. Trans. ASME, 74, 969, 1952
46. **Pioro, I.L.**, Experimental evaluation of constants for the Rohsenow pool boiling correlation, International Journal of Heat and Mass Transfer , 42(1999) 2003-2013
47. **Chen,** J.C. A correlation for boiling heat transferto saturated fluids in convective flow, Int End. Chem. Process Des. Develop. 5:322 (1966)

48. **Gungor K.E. and Winterton R.H.S**., A general correlation for flow boiling in tubes and annuli, Int. J. Heat Mass Transfer Vol. 2, No 3, pp 351-358, 1986
49. **Kandlikar, S.G.,** A general correlation for saturated two-phase flow boiling heat transfer inside horizontal and vertical tubes. ASME Journal of Heat Transfer, February 1990, Vol 112, pp 219-228
50. **Turgut, Oguz Emrah, Asker, Mustafa, Coban, Mustafa Turhan,** Saturated Flow Boiling Heat Transfer Correlation for Small Channels Based on R134a Experimental Data, Arab J Sci Eng (2016) 41:1921–1939 DOI 10.1007/s13369-016-2038-1
51**. Gorenflo, D**.: Pool Boiling. VDI Heat Atlas. VDI Verlag, Dusseldorf (1993)
52. **Rohsenow, W. M.,** Heat Transfer and temperature distributionin laminar film condensation, Trans. ASME, 78, 1645, 1956
53. **Kutateladze, S. S,** Fundamentals of heat transfer, Academeic press, New york, 1963
54. **Labuntsov, D. A.**, Heat transfer in film condensation of püre steam on vertical surfaces and horizontal
tubes, Teploenergetica, 4, 72, 1957
55. **Turgut, Oguz Emrah, Coban, Mustafa Turhan, Asker, Mustafa,** Comparison of Flow Boiling pressure drop correlations for smooth macrotubes, Heat transfer engineering37:6,487-506
56. **Traviss, D.P., Rohsenow, W.M. and Baron A.B.,** Forced convection condensation inside tubes:a heat trasnfer equation for condenser design, ASHRAE Transactions, 79, part1, 157-165 (1973)
57. **Chato J.C. J.** ASHRAE,4, 52, 1962
58. **Elenbaas**, W, Physica. 9,1,1942
59. **Bar-Cohen, A., and W. M. Rohsen**ow, *J. Heat Transfer,* **106,** 116, 1984.
60. **Azevedo, L. F. A., and E. M. Sparrow**, *J. Heat Transfer,***107,** 893, 1985.
61. **Globe, S., and D. Dropkin,** *J. Heat Transfer,* **81C,** 24, 1959.
62. **Catton, I.,** "Natural Convection in Enclosures," *Proc. 6th Int. Heat Transfer Conf.*, Toronto, Canada, 1978, Vol.6,pp. 13–31.
63. **MacGregor, R. K**., and A. P. Emery, *J. Heat Transfer,* **91,** 391, 1969.
64. **Hollands, K. G. T., S. E. Unny, G. D. Raithby, and L. Konicek,** *J. Heat Transfer,* **98,** 189, 1976.
65. **Arnold, J. N., I. Catton, and D. K. Edwards**, Experimental Investigation of Natural Convection in Inclined Rectangular Regions of Differing Aspect Ratios, ASME Paper 75-HT-62, 1975.
66. **Ayyaswamy, P. S., and I. Catton**, *J. Heat Transfer,* **95,** 543, 1973.
67. **Shah, M.M.,** "Chart correlations for saturated boiling heat transfer: Equations and further study," ASHRAE Transactions, vol. 88, part 1, pp. 185-196, 1982
68. **Steiner, D., and Taborek, J.,** "Flow boiling heat transfer in vertical tubes correlated by an asymptotic
model," Heat Transfer Engineering, vol. 13, no.2, 1992
69. **Cooper, M.G.,** 1984, Heat Flow Rates in Saturated Nucleate Pool Boiling-A Wide Ranging Examination using Reduced Properties, Advances in Heat Transfer, Eds. Hartnett and IrvineAcademic
Press, Princeton, Vol. 16, pp. 157–239
70. **S. Kakaç, A.E Bergles, F. Mayinger and H. Yüncü**, Heat transfer Enhancement of Heat Exchangers,
NATO ASI Series, Kluwer Academic Publishers. 1998, ISBN 0-7923-5637-3, Page 141
71. **Çoban, M. Turhan,** Dairesel düz Sürekli ve dalgalı sürekli kanatlı yoğuşturucuların sonlu farklar yöntemiyle modellenmesi, ULIBTAK'15 Ulusal ısı bilimi ve tekniği kongresi 2-5 eylül 2015, Balıkesir
72. **Akturk, F, Gulben G., Aradag, S. Uzol, N.S. Kakac, Sadik,** Experimental Investigation of the Characteristics of a Chevron Type GasketedPlate Heat Exchanger, 6th ınternational advanced Technologies symposium, (IATS'11), 16-18 May 2011, Elazığ, Turkey

73. Dong-Hyouck Han, Kyu-Jung Lee, Yoon-Ho Kim, The Characteristics of Condensation in Brazed
Plate Heat Exchangers, with Different Chevron Angles; Journal of the Korean Physical Society, Vol.
43, No. 1, July 2003, pp. 66 -73
74. **Muthuraman, S.** The Characteristics of Brazed Plate Heat Exchangers with Different Chevron Angles,
Global journal of researchers in engineering mechanical and mechanics engineering, ISSN 2249-4596,
volume 11, ıssue 7, version 1 december 2011
75. **W.W. Focke, J. Zacharides, I. Oliver**, "The effect of the corrugation inclination angle on the thermohydraulic performance of the plate heat exchangers", *International Journal of Heat and Mass Transfer* vol. 28, pp. 1469 – 1479, 1985.
76. K. Okada, M. Ono, T. Tomimura, T. Okuma, H. Konno, S. Ohtani, "Design and Heat Transfer Characteristics of a New Plate Heat Exchanger", *Heat Transfer Japanese Research* vol. 1, pp. 90 – 95,
1972.
77. **S. Kakaç, H. Liu,** *Heat Exchangers Selection, Rating and Thermal Design*, Florida, CRC Press, 2002.
78. **Hausen, H. (1943).** Darstellung des W¨armeauberganges in R¨ohren durch verallgemeinerte Potenzbeziehungen, *Z.VDI,* 4, 91–95.
79. **Sleicher, C. A., and Rouse, M. W. (1975).** A Convenient Correlation for Heat Transfer to Constant and Variable Property Fluids in Turbulent Pipe Flow, *Int.J .Heat Mass Transfer,* 18, 677–684.
80. **Muley, A & Manglik, R.M. (1999)** ,Experimental Study of Turbulent Flow Heat Transfer and Pressure
Drop in a Plate Heat Exchanger With Chevron Plates, ASME Journal of Heat transfer, 121(1), 110.
81. Lewis, G.K., Shape factors in conduction heat flow for circular bars and slabs with various internal geometries, Int. J. Heat Mass Transfer. Vol. 11, pp. 985-992. Pergamon Press. 196882.
82. Steiner, D., 1993. Heat Transfer to boiling saturated liquids (VDI Heat Atlas). Dusseldorf. VDI-Gessellschaft Verfahrenstechnik und Chemieingenieurwesen (GCV), Germany.
83. **Rouhani, S.Z., Axelsson, E., 1970**. Calculation of void volume fraction in the subcooled and quality
boiling regions. Int. J. Heat Mass Transfer 13 (2), 383_393.
84. **Lockhart, R., Martinelli, R., 1949**. Proposed correlation of data for isothermal two-phase,two-component flow in pipes. Chem. Eng. Prog. 45 (1), 39_48.
85. Friedel, L., 1979. Improved friction pressure drop correlations for horizontal and vertical two phase pipe flow (Paper E2). European Two Phase Flow Group Meeting, Ispra, Italy.
86. Müller-Steinhagen, H., Heck, K., 1986. A simple friction pressure drop correlation for twophase flow
in pipes. Chem. Eng. Process. 20 (6), 297_308.
87. Boussinesq, J. (1877), "Essai sur la théorie des eaux courantes", Mémoires présentés par divers savants à l'Académie des Sciences **23** (1): 1-680
88. **L. Prandtl, Z. Angew**. Math. Mech., 5:136 (1925)
89. T. Von Karman, Proc. 3d Intern. Congr. Appl. Mech., Stockholm, 1930, pt. I, p. 85; NACA Memo 611,1931
90. Deisler, R.G. , Investigation of turbulent flow and heat transfer in smooth tubes, including the effects
of variable fluid properties, Trans. ASME. 73 101-105 (1951)
91. V. M. Falkner and S. W. Skan, Some Approximate Solutions of the boundary layer equations, Philos.
Mug. 12 (1931), 865-896.

92. S. Goldstein,A note on Boundary Layer Equations, Proc. Cambridge Phil. Soc., 35:388 (1939)
93. Touloukian Y.S, Powel R.W, Ho C. Y., Klemens P.G. Thermal Conductivity Metallic Elements and
alloys, Thermophysical properties of materials volume 2 ISBN 978-1-4615-9602-6
94. Touloukian Y.S, Powel R.W, Ho C. Y., Klemens P.G. Thermal Conductivity Non Metallic solids, Thermophysical properties of materials volume 2 ISBN 306-67022-4
95. Briggs, D.E. and Young, E.H. , Convection heat transfer and pressure drop of air flowing across triangular pitch banks of finned tubes, Chem. Eng. Prog. Symp. Ser., 59, 1-10 (1963)
96. Robinson, K. K., and Briggs, D. E., Pressure drop of air flowing across triangualar pitch banks of finned tubes, , Chem. Eng. Prog. Symp. Ser., 62, 177-184 (1966)
97. Kuppan, T. , Heat Exchanger design Handbook,ISBN 0-8247-9787-6, Marcel Dekker Inc. 2000
98. Howell, John R., Menguc M. Pınar, Siegel, Robert, Thermal Radiation Heat Transfer, 5th Edition,
CRC process, 2010, ISBN 978-1-4398-0533-6
99. Sparrow, E. M.: On the Calculation of Radiant Interchange between Surfaces, in W. Ibele (ed.), *Modern Developments in Heat Transfer*, pp. 181–212, Academic Press, New York, 1963b.
100. **Lewis, R. W,Nithiarasu, P., Seetharamu, K. N.;**Fundamentals of the Finite Element Method for Heat and Fluid Flow; Wilwe, 2004, ISBN 0-470-84789-1
101. Robert W. Serth and Thomas Lestina, Process Heat Transfer, Principles, Applications and Rules of
Thumb, Yaer 2014, Academic Press, ISBN 978-0-12-397195-1
102. **Shah, M.M.** (1979). A General Correlation for Heat Transfer during Film Condensation inside of Pipes, Int. J. Heat Mass Transfer, Vol. 22, pp. 547-556.
103. **El Hajal, J., Thome, J.R. and Cavallini, A.** (2003). Condensation in Horizontal Tubes, Part 1: Two-
Phase Flow Pattern Map, Int. J. Heat Mass Transfer, Vol. 46, pp. 3349-3363.
Condensation in Horizontal Tubes, Part 2: Two Phase Flow Pattern Map, Int. J. Heat Mass Transfer,
Vol. 46, pp. 3365-3387.
104. **D. Biberg,** An explicit approximation for the wetted angle in two-phase stratified pipe flow, Canadian
J. Chemical Engineering 77 (1999) 1221–1224.
105. **O. Zurcher, J.R. Thome, D. Favrat,** Evaporation of € ammonia in a smooth horizontal tube: heat transfer measurements and predictions, J. Heat Transfer 121 (1999) 89–101.
106. **Shah, M.M.** (2009). An Improved and Extended general Corroletaion for heat Transfer During Condensation in Plain Tubes, AHRAE HVAC&R Research, Vol. 15, No. 5, September 2009
107. **Shah, M.M.** (2013). General Correlation For Heat Transfer During Condensation in Plain Tubes: Further Development and verification, AHRAE Transaction, Volume 119, Part 2
108. **Dobson, M.K. and Chato, J.C.** (1998). Condensation in Smooth Horizontal Tubes, J. Heat Transfer,
Vol. 120, No. 1, pp. 193-213.
109. **Zivi, S.M. (1964).** Estimation of Steady-State Steam Void-Fraction by Means of the Principle of Minimum Entropy Generation, J. Heat Transfer, Vol. 86, pp. 247-252.
110. Soliman, H.M. (1982). On the Annular-to-Wavy Flow Pattern Transition during Condensation inside
Horizontal Tubes, The Canadian Journal of Chemical Engineering, Vol. 60, pp. 475-481.
111. Ripert, V.G., Sereda V. V., Condensation inside smooth horizontal tubes Part 1. Survey of Methods
of Heat Exchange Prediction, Thermal Science, Year 2015, Vol. 19, No 5. Pp. 1769-1789

112. Thome, J. R. , Condensation in Plain horizontal tubes: Recent Advances in modelling of heat transfer
to pure fluids and mixtures, J. of the Braz. Soc. Of Mech. Sci. & Eng., January-March 2005, Vol. XXVII, No 1 pp 23-30

113. **Jiang, Puing, Qian, Xin, Li, Xiaobo, Yang, Ronggui,** Three-dimensional anisotropic thermal conductivity tensor of single crystalline β-Ga_2O_3, Appl. Phys. Lett 113, 232105(2018), DOI 10.163/1.5054573

114. Quelin, Xavier, Perrin, Bernard, Louis Gerard, Peretti Pierre, Three-dimensional thermal conductivity-tensor measurement of a polymer crystal by photothermal probe-beam deflection, Physical Review B, Volume 48, Number 6, 1 august 1993,II

115. Engineering toolbox density of solids, https://www.engineeringtoolbox.com/density-solids-d_1265.html

116. Engineering toolboc Specific heat of solids, https://www.engineeringtoolbox.com/specific-heat-solids-d_154.html

117. Zuo, Y.-X., and E. H. Stenby: *Can. J. Chem. Eng.*, **75:** 1130 (1997).

118. Brock, J. R., and R. B. Bird: *AIChE J.*, **1:** 174 (1955).

119. Curl, R. F., Jr., and K. S. Pitzer: *Ind. Eng. Chem.*, **50:** 265 (1958).

120. Stiel, L. I., and G. Thodos: *AIChE J.*, **10:** 26 (1964)

121. Chung, T.-H., M. Ajlan, L. L. Lee, and K. E. Starling: *Ind. Eng. Chem. Res.* **27:** 671 (1988).

122. Chung, T.-H., L. L. Lee, and K. E. Starling: *Ind. Eng. Chem. Fundam.*, **23:** 8 (1984).

123. Reichenberg, D.: ''The Viscosities of Pure Gases at High Pressures,'' *Natl. Eng. Lab., Rept. Chem. 38,* East Kilbride, Glasgow, Scotland, August 1975.

124. Lucas, K.: *Phase Equilibria and Fluid Properties in the Chemical Industry,* Dechema, Frankfurt, 1980, p. 573.

125. Lucas, K.: *Chem. Ing. Tech.*, **53:** 959 (1981).

126. Lucas, K.: personal communications, August 1983, September 1984.

127. Chung, T.-H., M. Ajlan, L. L. Lee, and K. E. Starling: *Ind. Eng. Chem. Res.*, **27:** 671 (1988).

128. Chung, T.-H., L. L. Lee, and K. E. Starling: *Ind. Eng. Chem. Fundam.*, **23:** 8 (1984).

129. Chapman, S., and T. G. Cowling: *The Mathematical Theory of Nonuniform Gases,* Cambridge, New York, 1939.

130. The Properties of Gases & Liquids, Forth edition, Robert C. Reid, John M. Prausnitz, Bruce E. Poling, Mc Graw Hill, ISBN 0-07-051799-1

131. The Properties of Gases and Liquids, Fifth edition, Bruce E. Poling, John M. Prausnitz, John P. O'Connel, Mc Graw Hill

132. Przezdziecki, J. W., and T. Sridhar: *AIChE J.*, **31:** 333 (1985).

133. Wilke, C. R.: *J. Chem. Phys.*, **18:** 517 (1950)

134. Eric W. Lemmon, "Thermodynamic Properties of Air and Mixtures of Nitrogen, Argon and Oxygen From 60 to 2000 K and Pressures to 2000 MPa", J. Phys. Chem. Ref. Data, Vol. 29, No. 3, 2000

135. K. Kadoya, N. Matsunaga, and A. Nagashima, Viscosity and Thermal Conductivity of Dry Air in the Gaseous Phase, Journal of Physical and Chemical Reference Data 14, 947 (1985)

136. E. W. Lemmon & R. T. Javobsen, Viscosity and Thermal Conductivity Equations for Nitrogen, Oxygen, Argon, and Air, International Journal of Thermophysics, Vol. 25, No. 1, January 2004

137. The International Association for the Properties of Water and Steam, Revised Relase on the IAPWS Formulation 1995 for the Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use, Prague, Check Republic 2018, IAPWS R6-95(2018)

138. The International Association for the Properties of Water and Steam, Revised Relase on the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, Lucerne, Switzerland, August 2007, IAPWS R7(2012)

139. J.R. Cooper, R. B. Dooley, The International Association for the Properties of Water and Steam, Guideline on an Equation of State for Humid Air in Contact with SeaWater and Ice, Consistent with the IAPWS Formulation 2008 for the Thermodynamic Properties of SeaWater

140. Numerical Thermodynamics, M. Turhan Coban, www.turhancoban.com

141. Ian H. Bell, Jorrit Wronski, Sylvain Quailin, and Vincent Lemort, Pure and Pseudo-pure Fluid Thermophysical Property Evaluation and the Open-Source Thermophysical Property Library CoolProp, Ind Eng Chem Res 2014 Feb 12;53(6) 2498-2508 DOI 10.1021/ie4033999

142. S. G. S. Beirao, A.P.C. Ribeiro, M.J.V. Lourenço, F.J.V. Santos, C.A. Nieto de Castro, Thermal Conductivity of Humid Air, Int J Thermopys (2012) 33:1686-1703, DOI 10.1007/s10765-012-1245-5
143. Xiaoyan Ji, Thermodynamic properties of Humid air and their application in advanced power generation cycles, Doctaral Thesis in Chemical Engineering, KTH Royal Institute of Technology, Energy Processes, SE-100 44 Stockholm, Sweeden, ISBN 91-7178-437-3
144. R. Feistel, D.G. Wright, D.R. Jackett, K.Miyagawa, J.H. Reissmann, W. Wagner, U. Overhoff, C. Guder, A. Feistal, G.M. Marion, Numerical implementation and ocenographic application oft he thermodynamic potentials of liquid water, water vapor, ice, sewater and humid air- Part 1: Background and equations, Ocean Sci., 6, 663-677,2010 DOI 10.5194/os-6-633-2010
145. R. Feistel, D.G. Wright, D.R. Jackett, K.Miyagawa, J.H. Reissmann, W. Wagner, U. Overhoff, C. Guder, A. Feistal, G.M. Marion, Numerical implementation and ocenographic application oft he thermodynamic potentials of liquid water, water vapor, ice, sewater and humid air- Part 2: The library routines, Ocean Sci., 6, 695-718,2010 DOI 10.5194/os-6-695-2010
146. Simeen Sattar, Thermodynamics of Mixing Real Gases, Jornal of Chemical Education Vol. 77 No 10 October 2000
147. Wolgang Wagner, Hans-Joachim Kretzschmar, International Steam Tables, Properties of water and steam based on industrial formulation IAPWS-IF97, Springer, ISBN 978-3-540-21419-9, DOI 10.1007/978-3-540-74234-0
148. Joseph H. Keenan, Frederick G. Keyes, Philip G. Hill, Joan G. Moore, Steam Tables Thermodynamic properties of Water incuding vapor, liquid and solid phases, Wiley-Interscience, ISBN 0-471-04210-2
149. E. W. Lemmon1, and R. T Jacobsen , "Viscosity and Thermal Conductivity Equations for Nitrogen, Oxygen, Argon, and Air", International Journal of Thermophysics, Vol. 25, No. 1, January 2004 (© 2004)
150. M. Salimi, A. Bahramian, “The Prediction of the Speed of Sound in Hydrocarbon Liquids and Gases: The Peng-Robinson Equation of State Versus SAFT-BACK”, Petroleum Science and Technology, 32:409-417,2014
151. Forster, H. K. and N. Zuber: "Dynamics of Vapour Bubbles and Boiling Hear Transfer”. AIChE J. 1:531-535 (1955)

APPENDIX 1 EXAM QUESTIONS

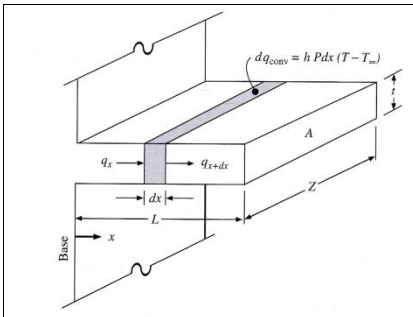
HEAT TRANSFER I group A Fall 2015 Midterm Exam, M. Turhan ÇOBAN

27 November 2015 time : 13.15

QUESTION 1

t=0.5x10^{-3} m thin, L=0.1 m long fins attached to a wall with 1.5x10^{-3} m distance in between (N=500 fins/m). The base temperature is T_b= 127 C. The air temperature is T_∞ =27 C. The width of the wall is z=0.2 m. Find the total heat transfer for a meter of finned Wall (W) . The convective heat transfer coefficient is h=40 W/(m^{2}K). Fin thermal conductivity k=120 W/(mK).

Note: Total heat transfer=heat transfer from the fins + heat transfer from the base wall



$A_f = PL_c \quad L_c = L + (t/2) \quad A_p = tL \quad A_c = tz \quad P = 2z + 2t$

$$\frac{d^2\theta}{dx^2} - \frac{hP}{kA_c}\theta = 0 \qquad \theta = T - T_\infty \qquad \theta_b = T_b - T_\infty \qquad m^2 = \frac{hP}{kA_c}$$

$$\frac{\theta}{\theta_b} = \frac{\cosh[m(L_c - x)]}{\cosh[mL_c]}$$

$$q_f = \sqrt{hPkA_c}\;\theta_b \tanh(\mathrm{mL})$$

$$\eta_f = \frac{\tanh[mL_c]}{[mL_c]}$$

N fin system:

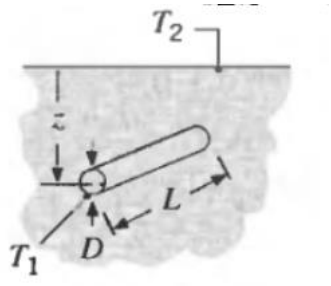
$$q = q_f + q_{base} = N\eta_f hA_f\theta_b + hA_{base}\theta_b$$

QUESTION 2

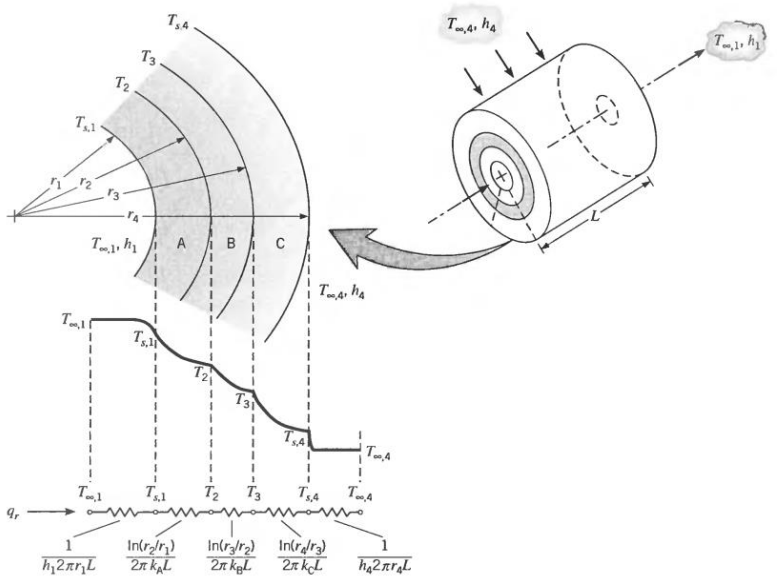
A pipeline, used for the transport of crude oil, is burried in the earth such that its centerline is a distance of 1.5 m below the surface. The pipe has an outer diameter of 0.5 m and is insulated with a layer of cellular glass[k=0.05 W/(mK)] 0.1 m thick. What is the heat loss per unit length of pipe under conditions for which heated oil at 120 C flows through the pipe and the surface of the earth is at a temperature of 0 C.

Horizontal isothermal cylinder of Length L buried in a semi-infinite medium.

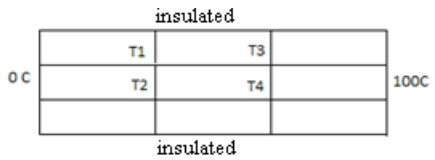
$q = Sk\Delta T_{1-2} \qquad \Delta T_{1-2} = T_1 - T_2 \qquad L >> D \qquad S = \dfrac{2\pi L}{\cosh^{-1}(2z/D)} \qquad L >> D \quad z > 3D/2 \quad S = \dfrac{2\pi L}{\ln(4z/D)}$



$$q_r = \frac{T_{\infty 1} - T_{\infty 2}}{\frac{1}{2\pi r_1 L h_1} + \frac{\ln(r_2/r_1)}{2\pi k_A L} + \frac{\ln(r_3/r_2)}{2\pi k_B L} + \frac{1}{2\pi r_3 L h_3}}$$



QUESTION 3 A 0.15 m x 0.15 m square shape is given with outside temperatures are 0 C and 100 C as shown in the figure. Upper and lower parts are insulated. If Δx=0.05 m and Δy=0.05 m, find temperatures T1,T2,T3 and T4 by using finite difference formulation.

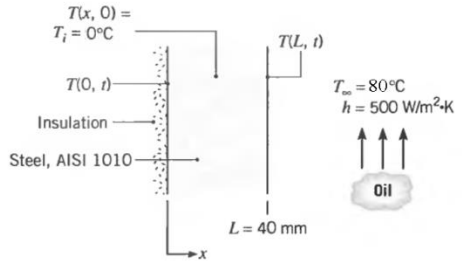


$$T_{m,n+1} + T_{m,n-1} + T_{m+1,n} + T_{m-1,n} + \frac{\dot{q}(\Delta x)^2}{k} - 4T_{m,n} = 0$$

QUESTION 4 : Consider a steel wall that has a thickness of 0.04 m. The wall is heavily insulated on the outside, and the before the initiation of flow on the surface of the wall, the wall are at uniform temperature of 0 C. With initiation of the flow, hot oil at 80 C is flowing from from the surface, creating convective condition corresponding to h=500 W/m^{2}K at the surface. What is the temperature of the steel wall under the insulation (x=0.04 m) 480 seconds after the initiation of flow.Properties of steel used in the wall is as follows: ρ=7832 kg/m^3, C_p=434 J/kgK, k=63.9 W/mK, α=18.8e-6 m^2/s

$$Bi = \frac{hL}{k} \quad Fo = \frac{\alpha t}{L^2} \quad x^* = \frac{x}{L} \qquad \theta^* = \frac{T - T_\infty}{T_i - T_\infty} = \sum_{n=1}^{\infty} C_n \exp(-\xi_n Fo)\cos(\xi_n x^*) \quad C_n = \frac{4\sin(\xi_n)}{2\xi_n + \sin(2\xi_n)}$$

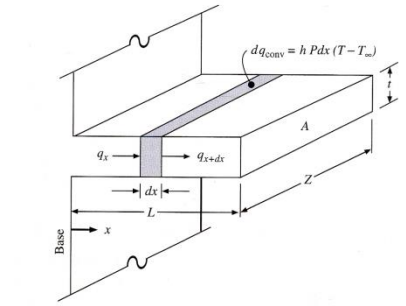
$$\xi_n \tan(\xi_n) - Bi = 0$$



HEAT TRANSFER I Fall 2014 Midterm Exam, M. Turhan ÇOBAN

QUESTION 1

An aliminium fin[k=200 W/mK] 3.0 mm thick and 75 mm long protrudes form a Wall as in the figure. The base is maintained at 300 C, and the ambient temperature is 50 C with h=10 W/(m^2K). Calculate the heat loss from the fin per unit depth of material.

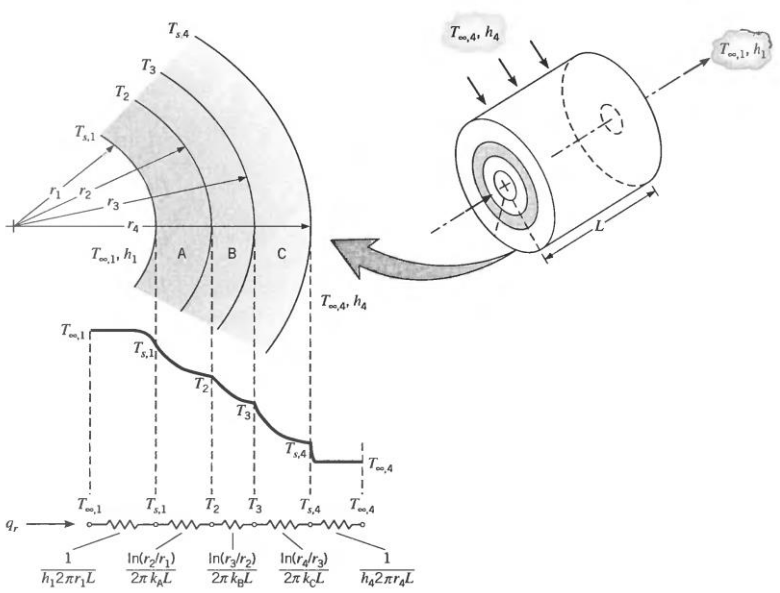


$$\frac{d^2\theta}{dx^2} - \frac{hP}{kA_c}\theta = 0 \qquad \theta = T - T_\infty \qquad m^2 = \frac{hP}{kA_c} \qquad \frac{\theta}{\theta_b} = \frac{\cosh[m(L-x)]}{\cosh[mL]}$$

$$q_f = \sqrt{hPkA_c}\;\theta_b \tanh(mL)$$

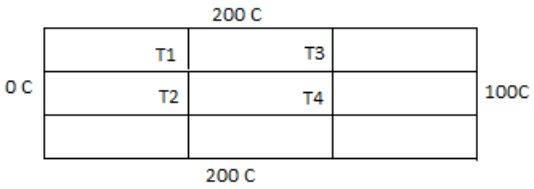
QUESTION 2

An aliminium pipe has an internal radius of r_1=10x10^{-3} m and outside radius of r_1=12x10^{-3} m. Pipe has a thermal conductivity k=200 W/(mK). Pipe is insulated with 5e-2m m insulation layer with a thermal conductivity coefficient of k=0.05 W/(mK). Inside the pipe water at T_i=100 C is flowing with a heat transfer coeefficient h_i=1000 W/(m^2K) and in the outside of the pipe air at T_o=20 C existed with a heat transfer coeefficient h_0=10 W/(m^2K). Total pipe length is L=100 m. Calculate the heat loss from the pipe.



$$q_r = \frac{T_{\infty 1} - T_{\infty 2}}{\frac{1}{2\pi r_1 L h_1} + \frac{\ln(r_2/r_1)}{2\pi k_A L} + \frac{\ln(r_3/r_2)}{2\pi k_B L} + \frac{1}{2\pi r_3 L h_3}}$$

QUESTION 3 A 0.15 m x 0.15 m square shape is given with outside temperatures are 0 C, 100 C, 100 C and 50 C as shown in the figure. If Δx=0.05 m and Δy=0.05 m, find temperatures T1,T2,T3 and T4 by using finite difference formulation.



$$T_{m,n+1} + T_{m,n-1} + T_{m+1,n} + T_{m-1,n} + \frac{\dot{q}(\Delta x)^2}{k} - 4T_{m,n} = 0$$

QUESTION 4 : The temperature of a gas stream is measured with a thermocouple.The junction may be approximated as a sphere of diameter D=1 mm, k=25 W/(mK), ρ=8400 kg/m^3, and C_p=400 J/(kgK). The heat transfer coefficient between the junction and the gas stream is h=100 W/(m^{2}K). How long will it take for the thermocouple to record 99% $\left(\frac{T - T_\infty}{T_b - T_\infty} = \frac{1}{100}\right)$ of applied temperature difference?

$$\frac{T - T_\infty}{T_b - T_\infty} = e^{-mt} \qquad m = \frac{hA}{\rho C_p V}$$

Plane Wall		
Bi^a	ζ_1 (rad)	C_1
0.01	0.0998	1.0017
0.02	0.1410	1.0033
0.03	0.1723	1.0049
0.04	0.1987	1.0066
0.05	0.2218	1.0082
0.06	0.2425	1.0098
0.07	0.2615	1.0114
0.08	0.2791	1.0130
0.09	0.2956	1.0145
0.10	0.3111	1.0161
0.15	0.3779	1.0237
0.20	0.4328	1.0311
0.25	0.4801	1.0382
0.30	0.5218	1.0450
0.4	0.5932	1.0580
0.5	0.6533	1.0701
0.6	0.7051	1.0814
0.7	0.7506	1.0919
0.8	0.7910	1.1016
0.9	0.8274	1.1107
1.0	0.8603	1.1191
2.0	1.0769	1.1785
3.0	1.1925	1.2102
4.0	1.2646	1.2287
5.0	1.3138	1.2402
6.0	1.3496	1.2479
7.0	1.3766	1.2532
8.0	1.3978	1.2570
9.0	1.4149	1.2598
10.0	1.4289	1.2620
20.0	1.4961	1.2699
30.0	1.5202	1.2717
40.0	1.5325	1.2723
50.0	1.5400	1.2727
100.0	1.5552	1.2731
∞	1.5708	1.2733

$Bi = \frac{hL}{k}$	ξ_1	ξ_2	ξ_3	ξ_4
0	0	3.1416	6.2832	9.4248
0.001	0.0316	3.1419	6.2833	9.4249
0.002	0.0447	3.1422	6.2835	9.4250
0.004	0.0632	3.1429	6.2838	9.4252
0.006	0.0774	3.1435	6.2841	9.4254
0.008	0.0893	3.1441	6.2845	9.4256
0.01	0.0998	3.1448	6.2848	9.4258
0.02	0.1410	3.1479	6.2864	9.4269
0.04	0.1987	3.1543	6.2895	9.4290
0.06	0.2425	3.1606	6.2927	9.4311
0.08	0.2791	3.1668	6.2959	9.4333
0.1	0.3111	3.1731	6.2991	9.4354
0.2	0.4328	3.2039	6.3148	9.4459
0.3	0.5218	3.2341	6.3305	9.4565
0.4	0.5932	3.2636	6.3461	9.4670
0.5	0.6533	3.2923	6.3616	9.4775
0.6	0.7051	3.3204	6.3770	9.4879
0.7	0.7506	3.3477	6.3923	9.4983
0.8	0.7910	3.3744	6.4074	9.5087
0.9	0.8274	3.4003	6.4224	9.5190
1.0	0.8603	3.4256	6.4373	9.5293
1.5	0.9882	3.5422	6.5097	9.5801
2.0	1.0769	3.6436	6.5783	9.6296
3.0	1.1925	3.8088	6.7040	9.7240
4.0	1.2646	3.9352	6.8140	9.8119
5.0	1.3138	4.0336	6.9096	9.8928
6.0	1.3496	4.1116	6.9924	9.9667
7.0	1.3766	4.1746	7.0640	10.0339
8.0	1.3978	4.2264	7.1263	10.0949
9.0	1.4149	4.2694	7.1806	10.1502
10.0	1.4289	4.3058	7.2281	10.2003
15.0	1.4729	4.4255	7.3959	10.3898
20.0	1.4961	4.4915	7.4954	10.5117
30.0	1.5202	4.5615	7.6057	10.6543
40.0	1.5325	4.5979	7.6647	10.7334
50.0	1.5400	4.6202	7.7012	10.7832
60.0	1.5451	4.6353	7.7259	10.8172
80.0	1.5514	4.6543	7.7573	10.8606
100.0	1.5552	4.6658	7.7764	10.8871
∞	1.5708	4.7124	7.8540	10.9956

HEAT TRANSFER I Fall 2013 Midterm Exam

QUESTION 1

A plane wall of thickness L=0.2 m, thermal conductivity k=2 W/(mK) has its surface at x=0 insulated, and the other surface at x=L is at T= 0 C . **Heat is generated** within the wall at a rate of

$q = 100\cos\left(\frac{\pi x}{2L}\right)$ W/m^3.

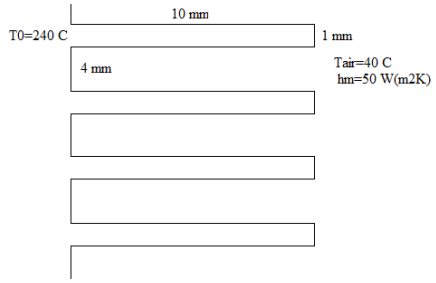
a) Develop an expression for one dimensional steady state temperature distribution in the Wall

b) Develop an expression and solve the temperature of the insulated surface.

QUESTION 2

Aliminium fins of recangular profile are attached on a plane wall with 5 mm spacing The fins have thickness t=1 mm, length L=10 mm, and thermal conductivity of the fin material is k=200 W/(mK). He plane wall is maintained at T_0=240 C, and the fins dissipate heat by convection into the ambient air at T_{air}=40 C with heat transfer coefficient h_m=50 W/(mK)

5. Determine the fin efficiency
6. Determine the area weighted fin efficiency
7. Determine the heat loss from the wall per square meter of wall surface

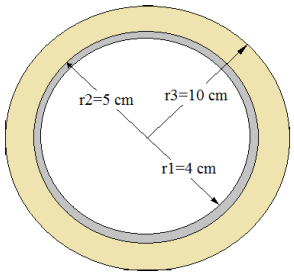


QUESTION 3

A steel pipe has an internal radius of r_1=40x10^{-3} m and outside radius of r_1=50x10^{-3} m. Pipe has a thermal conductivity k=55 W/(mK). Pipe is insulated by using 50x10^{-3} m thick glass-fiber insulation with k=0.05 W/(mK). Inside the pipe water at T_i=100 C is flowing with a heat transfer coeefficient h_i=1200 W/(m^{2}K) and in the outside of the pipe air at T_o=20 C existed with a heat transfer coeefficient h_0=10 W/(m^{2}K).

a) Calculate the heat loss for 100 meters of pipe length (W/m).

b) Calculate Temperature outside of the insulation layer.



HEAT TRANSFER I Fall 2013 Final Exam

QUESTION 1

An iron road of length L=0.3 m, diameter D=$1x10^{-2}$m and thermal conductivity k=60 W/(mK) is thermally attached horizontally to a large tank at temperature T_0=200 C. The road is dissipating heat by convection into the ambient air at T_{inf}=20 C with a covective heat transfer coefficient h=15 W/(m^2K)

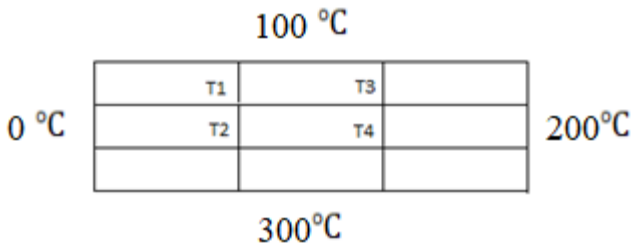
a) Find the heat transfer

b) Find the temperature at the and of the road (L=0.3 m)

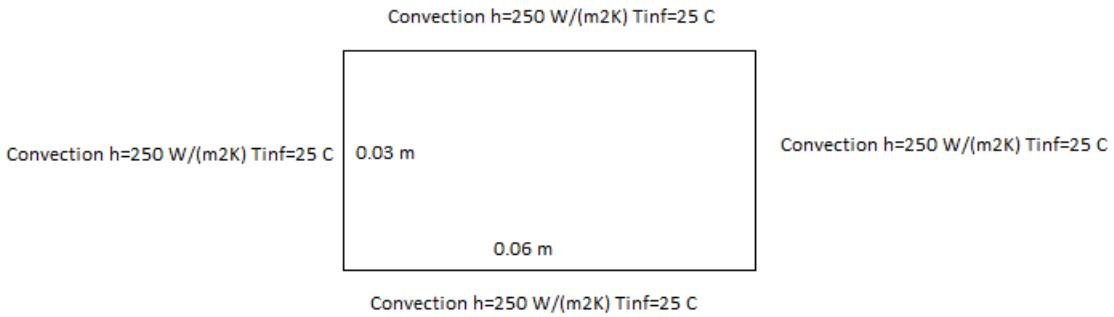
QUESTION 2

A thick wood Wall[α=$0.82x10^{-7}$$m^2$/s, k=0.15 W/(mK)] is initially at a uniform temperature of T_i=20 C. The wood may ignite at T=400C. If the surface is exposed to hot gases at T_{inf}=500 C, and the heat transfer coefficient between the gas and the surface is h=45 W/(m^2K). How long will it take for the surface of the wood to ignition temperature[T=400C]

QUESTION 3 A 0.3 m x 0.3 m square shape is given with outside temperatures are 0 C, 100 C, 200 C and 100 C as shown in the figure. If Δx=0.1 m and Δy=0.1 m, find temperatures T1,T2,T3 and T4 by using finite difference formulation.



QUESTION 4 A rectangular aliminium bar 0.06 m by 0.03 m [k=200 W/(mK), C_p=890 J/(kgK), ρ=2700 kg/m^3,α=$8.4x10^{-5}$$m^2$/s] is initially at a uniform temperature T_i= 175 C, suddenly all four surfaces of the road are subjected to convective cooling with a heat transfer coefficient h=250 W/(m^2K) into an anbient at T_{inf}=25 C . Determine the center temperature T_0 of the bar at t=60 s after the start of the cooling

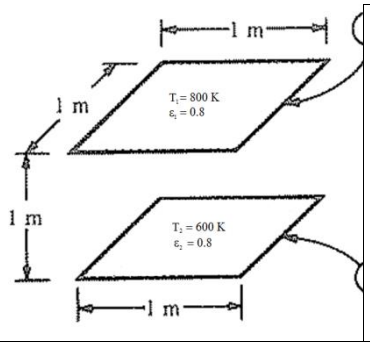


HEAT TRANSFER II Spring 2015 Final Exam

M. Turhan ÇOBAN

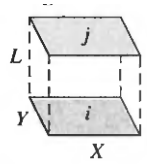
QUESTION 1

Two square plates, each 1m by 1 m are parallel to and directly opposite each other at distance 1 m. The hot plate is at T_1=800 K and has an emmisssivity ε_1=0.8. The colder plate is T_2=600 K and also has an emmisssivity ε_2=0.8. rate by radiation at each plate. The configuration is given in the figure. Calculate the heat transfer by radiation between the plates.



$$q = \frac{E_{b1} - E_{b2}}{R_1 + R_{12} + R_2} \qquad R_1 = \frac{1-\varepsilon_1}{\varepsilon_1 A_1} \qquad R_2 = \frac{1-\varepsilon_2}{\varepsilon_2 A_2}$$

$$R_{12} = \frac{1}{A_1 F_{12}} \qquad E_b = \sigma T^4 \qquad \sigma = 5.67e-8\ W/(m^2K^4)$$



$$\overline{X} = X/L,\ \overline{Y} = Y/L$$

$$F_{ij} = \frac{2}{\pi \overline{X}\overline{Y}} \left(\ln\left[\frac{(1+\overline{X}^2)(1+\overline{Y}^2)}{1+\overline{X}^2+\overline{Y}^2} \right]^{1/2} + \overline{X}(1+\overline{Y}^2)^{1/2} \tan^{-1}\left(\frac{\overline{X}}{(1+\overline{Y}^2)^{1/2}} \right) + \right.$$
$$\left. \overline{Y}(1+\overline{X}^2)^{1/2} \tan^{-1}\left(\frac{\overline{Y}}{(1+\overline{X}^2)^{1/2}} \right) - \overline{X}\tan^{-1}(\overline{X}) - \overline{Y}\tan^{-1}(\overline{Y}) \right)$$

QUESTION 2

A counterflow heat exchanger is used to cool water (Cp_h=4187 J/(kgK) from T_{hi}=90 C to T_{ho}=60 C at a rate of m_h=0.5 kg/s with water (Cp_c=4187 J/(kgK) entering T_{ci}=20 C at a rate of m_c=1 kg/s. The overall heat transfer coefficient is U_m=200 W/(m^2K). Calculate the heat transfer surface required and exit temperature of cold water.

$$C_c = m_c Cp_c \quad C_h = m_h Cp_h \quad \varepsilon = \frac{q}{q_{\max}} \quad \varepsilon = \frac{C_h(T_{hi} - T_{ho})}{C_{\min}(T_{hi} - T_{ci})} \text{ or } \varepsilon = \frac{C_c(T_{c0} - T_{ci})}{C_{\min}(T_{hi} - T_{ci})} \quad NTU = \frac{UA}{C_{\min}}$$

$$C_r = \frac{C_{\min}}{C_{\max}} \quad \text{for counter flow: } \varepsilon = \frac{1 - \exp[-NTU(1 - C_r)]}{1 - C_r \exp[-NTU(1 - C_r)]} \quad (C_r < 1) \quad \varepsilon = \frac{NTU}{1 + NTU} \quad (C_r = 1)$$

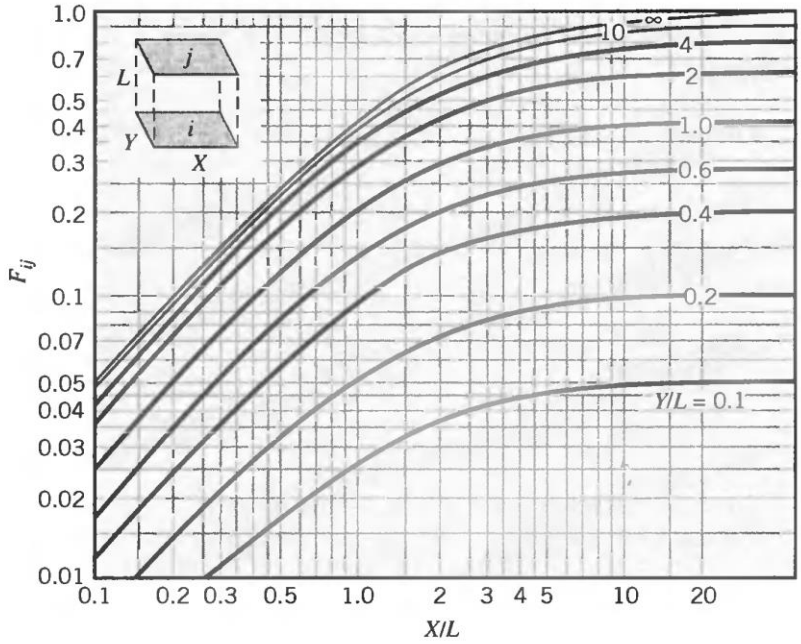
$$\text{for counter flow: } NTU = \frac{1}{C_r - 1} \ln\left(\frac{\varepsilon - 1}{\varepsilon C_r - 1}\right) \quad (C_r < 1) \quad NTU = \frac{\varepsilon}{1 - \varepsilon} \quad (C_r = 1)$$

QUESTION 3 A horizontal electrical heater of outside diameter D=0.025 m and length L=2 m dissipates heat by free convection into atmospheric air at T_∞ =27 C. If the surface temperature of the heater is 227 C, calculate the rate of heat transfer from heater to the air.

$$Nu_D = \left(0.60 + \frac{0.387 Ra_D^{1/6}}{\left(1 + (0.559/\mathrm{Pr})^{9/16}\right)^{8/27}}\right)^2 \quad Ra_D \le 10^{12} \quad Nu_D = \frac{hD}{k} \quad Ra_D = \frac{g\beta(T_s - T_\infty)D^3}{\nu\alpha}$$

$$\beta = -\frac{1}{\rho}\left(\frac{\partial \rho}{\partial T}\right)_P \quad \rho = \frac{P}{RT} \text{ (ideal gaz)}$$

T (K)	ρ (kg/m^3)	c_p (kJ/kg · K)	$\mu \cdot 10^7$ (N · s/m^2)	$\nu \cdot 10^6$ (m^2/s)	$k \cdot 10^3$ (W/m · K)	$\alpha \cdot 10^6$ (m^2/s)	Pr
Air							
100	3.5562	1.032	71.1	2.00	9.34	2.54	0.786
150	2.3364	1.012	103.4	4.426	13.8	5.84	0.758
200	1.7458	1.007	132.5	7.590	18.1	10.3	0.737
250	1.3947	1.006	159.6	11.44	22.3	15.9	0.720
300	1.1614	1.007	184.6	15.89	26.3	22.5	0.707
350	0.9950	1.009	208.2	20.92	30.0	29.9	0.700
400	0.8711	1.014	230.1	26.41	33.8	38.3	0.690
450	0.7740	1.021	250.7	32.39	37.3	47.2	0.686
500	0.6964	1.030	270.1	38.79	40.7	56.7	0.684
550	0.6329	1.040	288.4	45.57	43.9	66.7	0.683
600	0.5804	1.051	305.8	52.69	46.9	76.9	0.685
650	0.5356	1.063	322.5	60.21	49.7	87.3	0.690
700	0.4975	1.075	338.8	68.10	52.4	98.0	0.695
750	0.4643	1.087	354.6	76.37	54.9	109	0.702
800	0.4354	1.099	369.8	84.93	57.3	120	0.709



HEAT TRANSFER II MIDTERM EXAM 16.04.2015 EXAM TIME:2 HOUR

PROBLEM 1

Water at an inlet temperature of 37 C enters into a pipe with $3x10^{-2}$m in diameter end exit at a temperature of 87 C. The water velocity at the pipe inlet is u=3 m/s. If the surface temperature of the pipe is T_s=97 C, calculate

a) the heat transfer coefficient through the pipe

b) Pipe length

c) Total heat transfer rate through the pipe

PROBLEM 2

Saturated water at T_{sat} =100 C flows with a mass flow rate of m=0.1 kg/s through a $2.5x10^{-2}$ m ID vertical tube of L=5 m, subjected to a uniform wall temperature of T_w=110 C. Calculate the two phase heat transfer coefficient where x=0.1

PROBLEM 3 A horizontal cylinder of 0.05 m in diameter and 5 m in length maintained at a uniform temperature of T=100 C is exposed to atmospheric air at 10 C. Calculate

a) the free convection heat transfer coefficient
b) The total heat transfer from the pipe