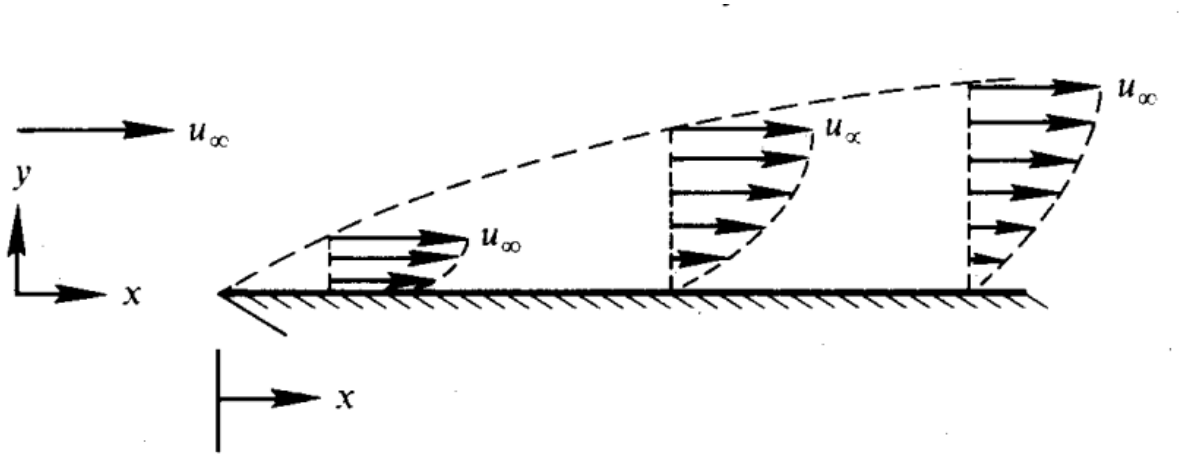
BLASSIUS FLAT PLATE PROBLEM SOLUTION

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

$$\upsilon\frac{\partial^2 u}{\partial y^2}=u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}$$

The boundary conditions are

u=0 v0 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

$u = g(\eta)$

$$\frac{\partial u}{\partial x}=\frac{\partial g(\eta)}{\partial x}=\frac{\partial g(\eta)}{\partial \eta}\frac{\partial \eta}{\partial x}=g'(\eta)yw'(x)$$

$$\frac{\partial u}{\partial y}=\frac{\partial g(\eta)}{\partial y}=\frac{\partial g(\eta)}{\partial \eta}\frac{\partial \eta}{\partial y}=g'(\eta)w(x)$$

$$\frac{\partial^2 u}{\partial y^2}=\frac{\partial^2 g(\eta)}{\partial y^2}=\frac{\partial}{\partial y}\left(\frac{\partial g(\eta)}{\partial y}\right)=\frac{\partial}{\partial \eta}\left(\frac{\partial g(\eta)}{\partial y}\right)\frac{\partial \eta}{\partial y}=f''(\eta)w^2(x)$$

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:

$$v=\frac{\nu g''(\eta)w^2(x)-g(\eta)g'(\eta)yw'(x)}{g'(\eta)w(x)}$$

$$\frac{\partial v}{\partial y}=-g'(\eta)yw'(x)$$

$$\frac{\partial}{\partial y}\left(\frac{\nu g''(\eta)w^2(x)-g(\eta)g'(\eta)yw'(x)}{g'(\eta)w(x)}\right)=-g'(\eta)yw'(x)$$

$$\nu w(x)\frac{\partial}{\partial y}\left(\frac{g''(\eta)}{g'(\eta)}\right)-\frac{w'(x)}{w(x)}\frac{\partial}{\partial y}\big(g(\eta)y\big)=-g'(\eta)yw'(x)$$

$$\nu w(x)\frac{\partial}{\partial y}\left(\frac{g''(\eta)}{g'(\eta)}\right)-\frac{w'(x)}{w(x)}\left(\frac{\partial g(\eta)}{\partial \eta}\frac{\partial \eta}{\partial y}y+g(\eta)\right)=-g'(\eta)yw'(x)$$

$$\nu w(x)\frac{\partial}{\partial y}\left(\frac{g''(\eta)}{g'(\eta)}\right)-\frac{w'(x)}{w(x)}\big(g'(\eta)w(x)y+g(\eta)\big)=-g'(\eta)yw'(x)$$

$$\nu w(x)\frac{\partial}{\partial y}\left(\frac{g''(\eta)}{g'(\eta)}\right)-\frac{w'(x)g'(\eta)w(x)y}{w(x)}-\frac{w'(x)}{w(x)}g(\eta)+g'(\eta)yw'(x)=0$$

$$\nu w(x)\frac{\partial}{\partial y}\left(\frac{g''(\eta)}{g'(\eta)}\right)-\frac{w'(x)}{w(x)}g(\eta)=0$$

$$\frac{\partial}{\partial y}\left(\frac{g''(\eta)}{g'(\eta)}\right)=\frac{\partial}{\partial \eta}\left(\frac{g''(\eta)}{g'(\eta)}\right)\frac{\partial \eta}{\partial y}=\frac{\partial}{\partial \eta}\left(\frac{g''(\eta)}{g'(\eta)}\right)w(x)$$

$$\nu w^2(x)\frac{\partial}{\partial \eta}\left(\frac{g''(\eta)}{g'(\eta)}\right)-\frac{w'(x)}{w(x)}g(\eta)=0$$

$$\frac{1}{g(\eta)}\frac{\partial}{\partial \eta}\left(\frac{g''(\eta)}{g'(\eta)}\right)=\frac{w'(x)}{\nu w^3(x)}$$

In this form, the differential equation is separated so it can be solved as two different set of equation with the common arbitrary constant.

$$\frac{1}{f(\eta)}\frac{\partial}{\partial \eta}\left(\frac{f''(\eta)}{f'(\eta)}\right)=\frac{g'(x)}{\nu g^3(x)}=-k$$

$$\frac{dw(x)}{w^3(x)}=-k\nu dx$$

$$-\frac{1}{2w(x)}=-k\nu x+C$$

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 thus follows that C=0. Then

$w(x)=\frac{1}{\sqrt{2k\nu x}}\qquad \eta=yw(x)=\frac{y}{\sqrt{2k\nu x}}$ So from this result, it is clear that velocity profile

$$u=f\left(\frac{y}{\sqrt{x}}\right)$$

The left hand side of the equation:

$$\frac{1}{g(\eta)}\frac{d}{d\eta}\left(\frac{g''(\eta)}{g'(\eta)}\right)=-k$$

$$d\left(\frac{g''(\eta)}{g'(\eta)}\right)=-kg(\eta)d\eta$$

$$\frac{g''(\eta)}{g'(\eta)}=-k\int g(\eta)d\eta+C$$

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

$$\frac{g''(\eta)}{g'(\eta)} = -k\int g(\eta)d\eta$$

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,

$$\frac{f'''(\eta)}{f''(\eta)} = -k\int u_\infty \frac{df(\eta)}{d\eta} d\eta = -ku_\infty f(\eta)$$

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 $\eta = \frac{y}{\sqrt{vx/u_\infty}}$ and $f'(\eta) = \frac{u}{u_\infty}$

Boundary conditions:

u=0 at y=0

v=0 at y=0

$u \to u_\infty$ at $y \to \infty$

Then $f'(0) = 0 \quad f'(\infty) = 1$

Since $f''(0) = 0 \quad f'''(0) = 0$ (provided that v(0)=0)

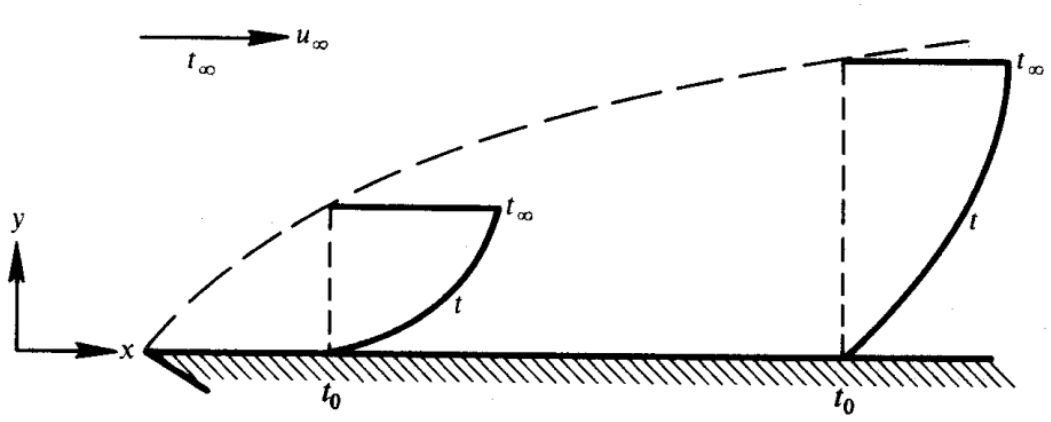
Then from the differential equation . $f(0) f''(0) = 0$ It is not possible that all the derivatives are zero, in fact $f''(0) = 0$ would correspond to zero shear stress at the wall, so we conclude that

$f(0) = 0$

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 velociyies u and v. It is defined as:

$u = \frac{\partial \psi}{dy} \quad v = -\frac{\partial \psi}{dx}$ Due to the definition, conservation of mass is always satisfied. The relation between ψ and f is as follows: $\psi = \sqrt{vxu_\infty} f(\eta)$

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 $\theta = \dfrac{t_0 - t}{t_0 - t_\infty}$ 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 $\eta = \dfrac{y}{\sqrt{vx/u_\infty}}$

Energy equation becomes:

$$\frac{\partial^2 \theta}{\partial \eta^2} + \frac{\Pr}{2} f \frac{\partial \theta}{\partial \eta} = 0$$

$\theta(0) = 0$

$\theta(\infty) = 1$

Equation can be solved directly as a set of differential equations. Equation set change as follows:

$$f'''(\eta)+\frac{1}{2}f(\eta)f''(\eta)=0$$

$$\theta''+\frac{\Pr}{2}f\theta'=0$$

Conversion of the equation:

$$x[0]=\eta$$

$$x[1]=f$$

$$x[2]=f'$$

$$x[3]=f''$$

$$x[4]=\theta$$

$$x[5]=\theta'$$ Differential equation set becomes:

$$\frac{dx[1]}{dx[0]}=x[2]$$

$$\frac{dx[2]}{dx[0]}=x[3]$$

$$\frac{dx[3]}{dx[0]}=-0.5*x[1]*x[3]$$

$$\frac{dx[4]}{dx[0]}=x[5]$$

$$\frac{dx[4]}{dx[0]}=-\frac{\Pr}{2}x[1]*x[5]$$

Boundary conditions:

x[4]=0 at x[0]=0

x[4]=1 at x[0]= ∞

x[2]=0 at x[0]=0

x[2]=1 at x[0]= ∞

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 $\frac{dy}{dx}=f(x,y)$

$y_{n+1} = y_n + h*\phi(x_i,y_i,h)$

$$\phi(x_1, y_1, h) = \sum_{i=1}^{n} b_i k_i$$

In this equation a is constant values depends on polynamial degree of Runge-Kutta equation and coefficients k, can be defined as:

$k_1=f(x_i,y_i)$

$k_2=f(x_i+c_2h,y_i+a_{21}k_1h)$

$k_3=f(x_i+c_3h,y_i+a_{31}k_1h+ q_{32}k_2h)$

….

$k_n=f(x_i+c_nh,y_i+a_{n,1}k_1h+ \ldots+ a_{n,n-1}k_{n-1}h)$

In more generalized form, these equations can be written as:

$$x_i = x_n + c_i h \quad \text{i=1…s}$$

$$y_i = y_n + h\sum_{j=1}^{i-1} a_{ij} k_j$$

$$k_i = f(x_i, y_i)$$

$y_{n+1} = y_n + h\sum_{j=1}^{s} b_i k_i$ 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:

$$\begin{array}{c|cccccc} c_1 & a_{11} & a_{12} & a_{13} & \ldots & a_{1s} \\ c_2 & a_{21} & a_{22} & a_{23} & \ldots & a_{2s} \\ c_3 & a_{31} & a_{32} & a_{33} & \ldots & a_{3s} \\ \vdots & \vdots & \vdots & \vdots & \ldots & \vdots \\ c_s & a_{s1} & a_{s2} & a_{s3} & \ldots & a_{s4} \\ \hline & b_1 & b_2 & b_3 & \ldots & b_s \end{array}$$

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:

$y_{i+1} = y_i + (1/90)*(7k_1 + 32k_3 +12k_4+32k_5+7k_6)h$

$k_1=f(x_i,y_i)$

$k_2=f(x_i+0.25h , y_i+0.25k_1h)$

$k_3=f(x_i+0.25h , y_i+0.125k_1h+0.125k_2h)$

$k_4=f(x_i+0.5h , y_i – 0.5k_2h+k_3h)$

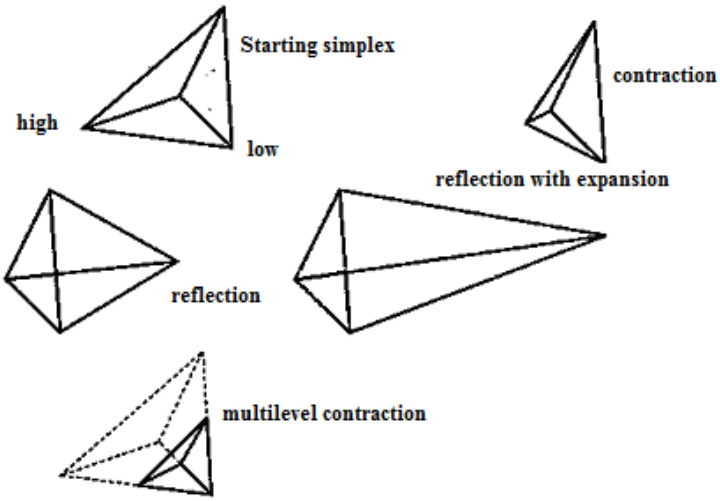
$k_5=f(x_i+0.75h , y_i + (3/16)k_1h+(9/16)k_4h)$

$k_6=f(x_i+h , y_i - (3/7)k_1h+(2/7)k_2h+(12/7)k_3h - (12/7)k_4h+(8/7)k_5h)$

This equation can be given as Buthcher tableu as:

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
	1/90	7/90	32/90	12/90	7/90

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

- **1. Order** according to the values at the vertices:

$$f(\mathbf{x}_1) \leq f(\mathbf{x}_2) \leq \cdots \leq f(\mathbf{x}_{n+1})$$

- **2.** Calculate x_o, the center of gravity of all points except x_{n+1}.

- **3. Reflection**

Compute reflected point $\mathbf{x}_r = \mathbf{x}_o + \alpha(\mathbf{x}_o - \mathbf{x}_{n+1})$
If the reflected point is better than the second worst, but not better than the best, i.e.:
$f(\mathbf{x}_1) \leq f(\mathbf{x}_r) < f(\mathbf{x}_n)$,
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, $f(\mathbf{x}_r) < f(\mathbf{x}_1),$
then compute the expanded point $\mathbf{x}_e = \mathbf{x}_o + \gamma(\mathbf{x}_o - \mathbf{x}_{n+1})$
If the expanded point is better than the reflected point, $f(\mathbf{x}_e) < f(\mathbf{x}_r)$
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 $f(\mathbf{x}_r) \geq f(\mathbf{x}_n)$
Compute contracted point $\mathbf{x}_c = \mathbf{x}_{n+1} + \rho(\mathbf{x}_o - \mathbf{x}_{n+1})$
If the contracted point is better than the worst point, i.e. $f(\mathbf{x}_c) \leq f(\mathbf{x}_{n+1})$
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 $\in \{2, \ldots, n+1\}$. go to step 1.

Note: $\alpha, \gamma, \rho$and σ are respectively the reflection, the expansion, the contraction and the shrink coefficient. Standard values are $\alpha = 1$, $\gamma = 2$, $\rho = 1 / 2$ and $\sigma = 1 / 2$.

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 $dx=\begin{Bmatrix} da \\ db \\ dc \end{Bmatrix}$

$$x_0=\begin{Bmatrix} a \\ b \\ c \end{Bmatrix} \quad x_1=\begin{Bmatrix} a+da \\ b \\ c \end{Bmatrix} \; x_2=\begin{Bmatrix} a \\ b+db \\ c \end{Bmatrix} \; x_3=\begin{Bmatrix} a \\ b \\ c+dc \end{Bmatrix}$$

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;
  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[]={0.001,0.002,0.003,0.004,0.005,0.006,0.007,0.008,0.009,0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.0
9,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.
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[]={200,200,200,200,200,200,200,200,200,100.0,100,100,100,100,100,100,100,100,50,50,50,50,50,50,5
0,50,50,25,25,25,25,25,25,25,25,25,10,10,10,10,10,10,10,10,10,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,5.3,5.3,5.3,5
.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};
 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:

$$h_x = \frac{q''_s}{T_s - T\infty} = -\frac{T_\infty - T_s}{T_s - T_\infty} k \frac{\partial \theta}{\partial y}\bigg|_{y=0}$$

$$h_x = k\left(\frac{u_\infty}{\nu x}\right)^{1/2} \frac{\partial \theta(\eta)}{\partial \eta}\bigg|_{\eta=0} = \frac{k}{x}\left(\frac{u_\infty x}{\nu}\right)^{1/2} \frac{\partial \theta(\eta)}{\partial \eta}\bigg|_{\eta=0}$$

$$Nu_x = \frac{h_x x}{k} = \mathrm{Re}_x^{1/2} \frac{\partial \theta(\eta)}{\partial \eta}\bigg|_{\eta=0}$$

Partial continuous curve fitting values are as follows:

$$Nu_x = 0.44547170\ 1\,\mathrm{Re}^{0.5} * \mathrm{Pr}^{0.46914552} \qquad 10^{-3} \le \mathrm{Pr} \le 10^{-2}$$

$$Nu_x = 0.34666757\ 4\,\mathrm{Re}^{0.5} * \mathrm{Pr}^{0.40159677} \qquad 10^{-2} \le \mathrm{Pr} \le 0.5$$

$$Nu_x = 0.33206065\ 5\,\mathrm{Re}^{0.5} * \mathrm{Pr}^{0.33679327} \qquad 0.5 \le \mathrm{Pr} \le 1000$$

Energy equation can also be directly integrated as follows:

$\frac{\partial^2 \theta}{\partial \eta^2} + \frac{\mathrm{Pr}}{2} f \frac{\partial \theta}{\partial \eta} = 0$ can be written as

$$\frac{\partial \theta'}{\partial \eta} + \frac{\mathrm{Pr}}{2} f\theta' = 0$$

$$\frac{\partial \theta'}{\theta'} + \frac{\mathrm{Pr}}{2} f d\eta = 0$$

$$\ln(\theta') = -\frac{\mathrm{Pr}}{2}\int_0^\eta f(\eta) d\eta + \mathrm{C}$$

$$\theta' = C_1 \exp\left(-\frac{\mathrm{Pr}}{2}\int_0^\eta f(\eta) d\eta\right)$$

$$\theta(\eta) = C_1 \int_0^\eta \left[\exp\left(-\frac{\mathrm{Pr}}{2}\int_0^\eta f(\eta) d\eta\right)\right] d\eta + C_2$$

By applying the boundary condition $\theta = 0$ at $\eta = 0$ $C_2 = 0$

For the boundary condition $\theta = 1$ 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

$$Nu_x = \frac{h_x x}{k} = \mathrm{Re}_x^{1/2}\theta'(\eta)\big|_{\eta=0}$$

and from the above equation

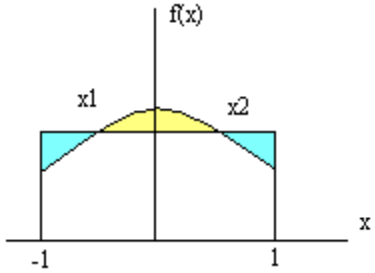
$$\theta'(\eta) = \frac{1}{\int_0^{\infty}\left[\exp\left(-\frac{\Pr}{2}\int_0^{\eta} f(\eta)d\eta\right)\right]d\eta} = C_1$$

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

$$f(\eta) = a_0 + a_1\eta + a_2\eta^2 + a_3\eta^3 + a_4\eta^4 + a_5\eta^5 + a_6\eta^6 + a_7\eta^7 + a_7\eta^8 + a_9\eta^9 + a_{10}\eta^{10}$$

$$\int_0^{\eta} f(\eta)d\eta = a_0\eta + \frac{a_1}{2}\eta^2 + \frac{a_2}{3}\eta^3 + \frac{a_3}{4}\eta^4 + \frac{a_4}{5}\eta^5 + \frac{a_5}{6}\eta^6 + \frac{a_6}{7}\eta^7 + \frac{a_7}{8}\eta^8 + \frac{a_8}{9}\eta^9 + \frac{a_9}{10}\eta^{10} + \frac{a_{10}}{11}\eta^{11}$$

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:

$$I_w(-1,1) \cong \int_{-1}^{1} f(x)w(x)dx \cong \sum_{k=1}^{n} c_k f(x_k)$$

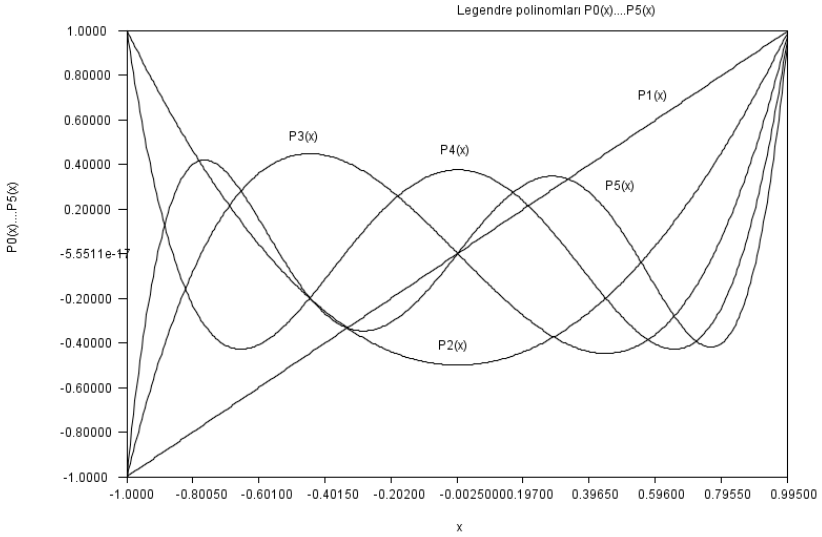
Can be written. In this equation w(x) is the weight factor. In the first of a such formulation, Gauss-Legendre integral formulation, weight factor can be taken as 1.

$$I_w(-1,1) \cong \int_{-1}^{1} f(x)dx \cong \sum_{k=1}^{n} c_k f(x_k)$$

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

$$c_1 x_1^{2j} + c_2 x_2^{2j} + ... + c_n x_n^{2j} = \frac{2}{2j+1}, \quad 0 \leq j \leq n \text{ or}$$

$$c_k = \frac{2(1-x_k^2)}{[nP_{n-1}(x_k)]^2}$$

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]*x11/11.0;
    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 IOExcception
  {  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};
    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 for $f(\eta)$ 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

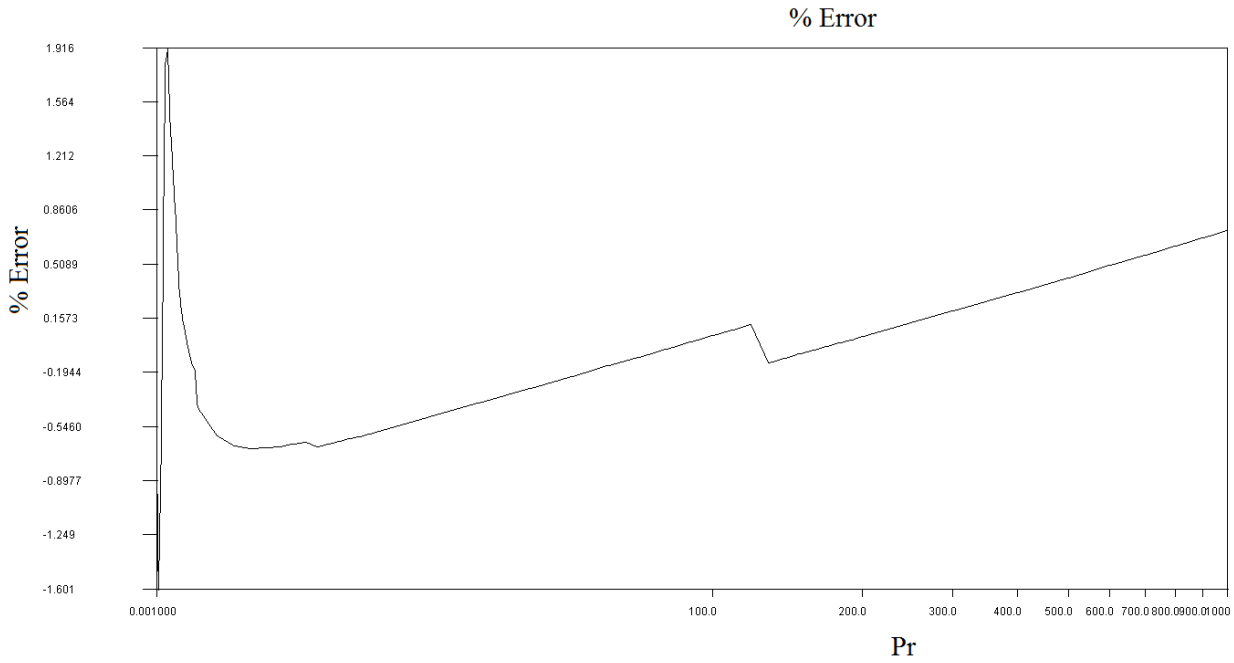
Partial continuous curve fitting values are as follows:

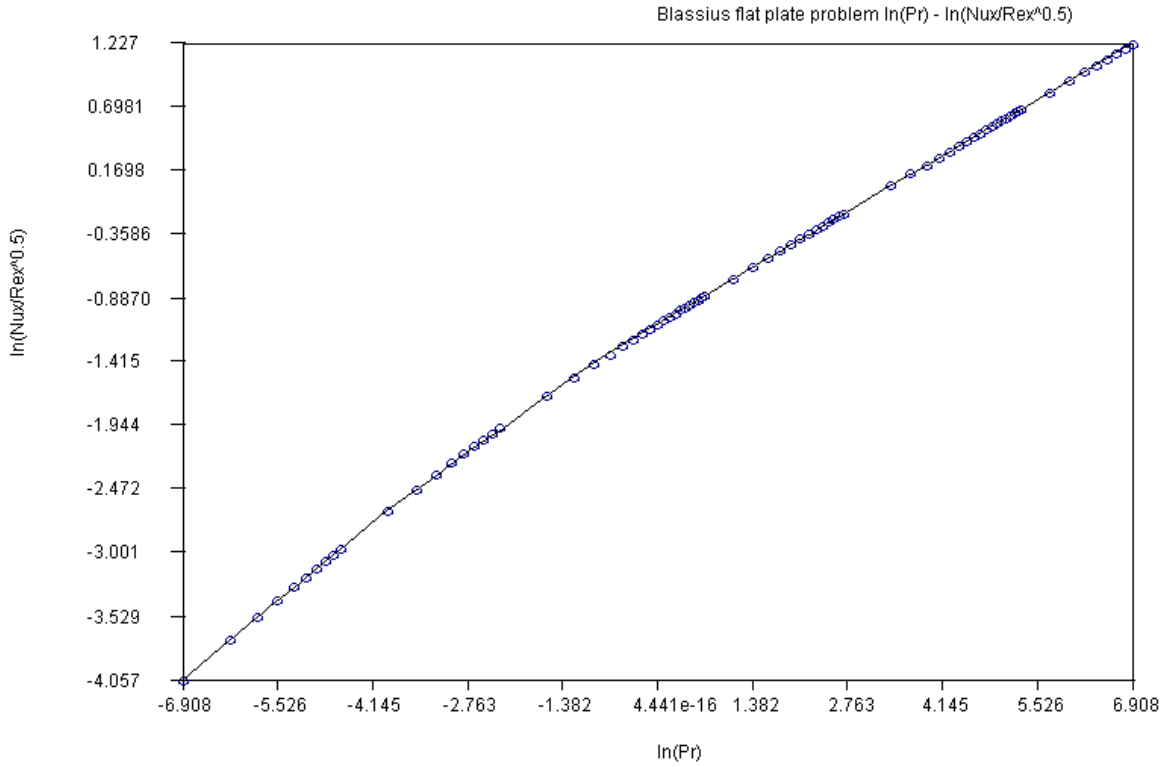
$$Nu_x = 0.45584275\ \ \mathrm{Re}^{0.5} * \mathrm{Pr}^{0.47294717} \qquad 10^{-3} \le \mathrm{Pr} \le 10^{-2}$$

$$Nu_x = 0.35271867\ \ \mathrm{Re}^{0.5} * \mathrm{Pr}^{0.40922589} \qquad 10^{-2} \le \mathrm{Pr} \le 0.5$$

$$Nu_x = 0.33253715\ \ \mathrm{Re}^{0.5} * \mathrm{Pr}^{0.33694685} \qquad 0.5 \le \mathrm{Pr} \le 1000$$

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.



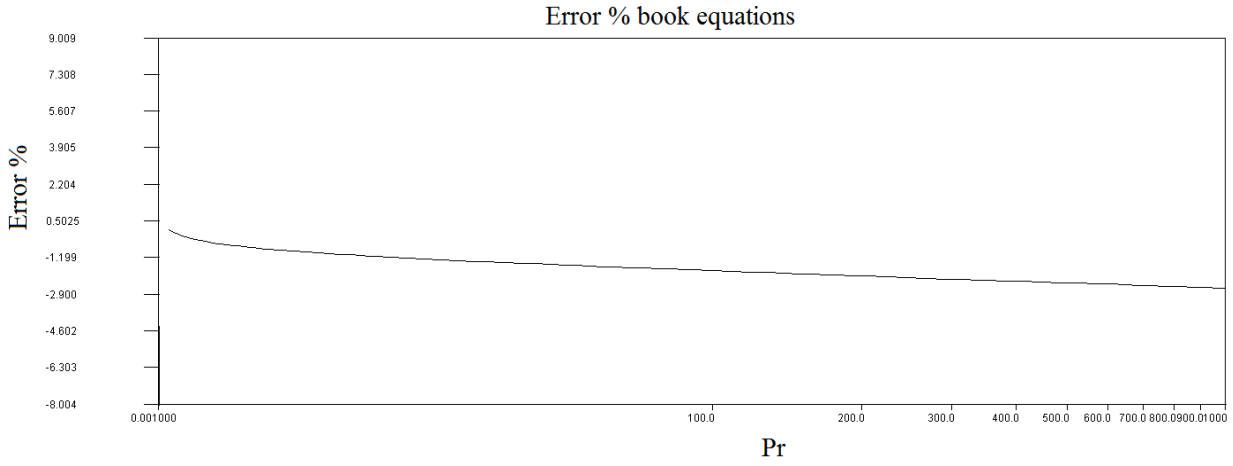


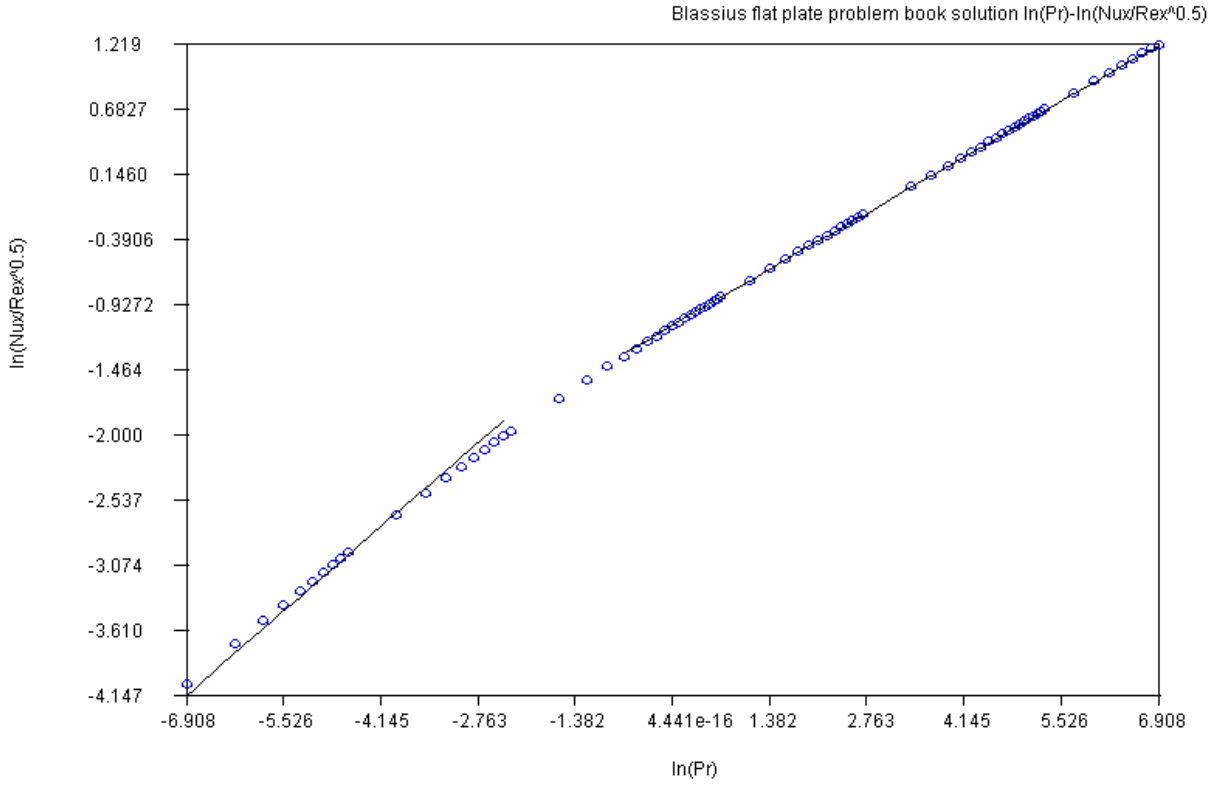
In most of the region plot errors are less than 1 %

As comparison solutions listed in books (a heat and mass transfer by Eckert and Drake) listed the following equations

$$Nu_x = 0.5\,\mathrm{Re}^{0.5} * \mathrm{Pr}^{0.5} \qquad 0.005 \le \mathrm{Pr} \le 0.05$$

$$Nu_x = 0.332\mathrm{Re}^{0.5} * \mathrm{Pr}^{0.33333333} \qquad 0.6 \le \mathrm{Pr}$$





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 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