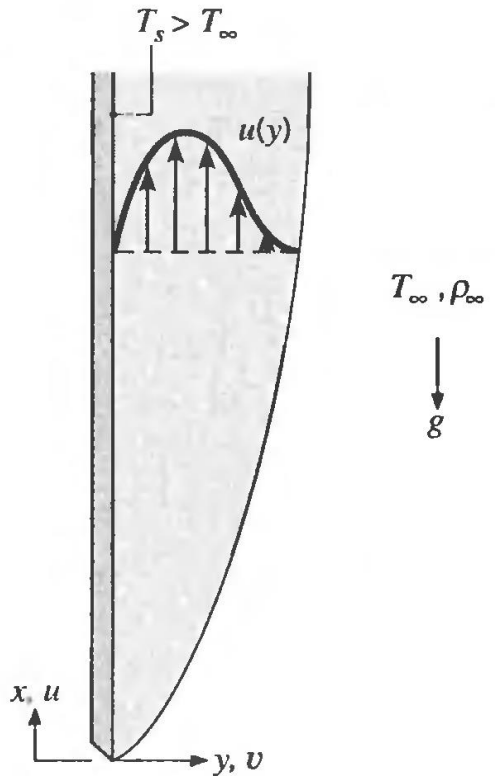
FREE CONVECTION SIMILARITY SOLUTION

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

$$\rho u \frac{\partial u}{\partial x} + \rho v \frac{\partial u}{\partial y} = \frac{\partial}{\partial y}\left(\mu \frac{\partial u}{\partial y}\right) - \frac{dP}{dx} - \rho g$$

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:

$\frac{dP}{dx} = -\rho_\infty g$ if this substituted the previous momentum equation it becomes:

$$\rho u \frac{\partial u}{\partial x} + \rho v \frac{\partial u}{\partial y} = \frac{\partial}{\partial y}\left(\mu \frac{\partial u}{\partial y} \right) + (\rho_\infty - \rho) g$$

Furthermore, if Definition of the thermodynamic property of thermal expansion coefficient is taken into consideration:

$$\beta = -\frac{1}{\rho}\left(\frac{\partial \rho}{\partial T} \right)_P \approx \frac{1}{\rho}\left(\frac{\rho_\infty - \rho}{T_\infty - T} \right)$$

Equation becomes:

$$\rho u \frac{\partial u}{\partial x} + \rho v \frac{\partial u}{\partial y} = \frac{\partial}{\partial y}\left(\mu \frac{\partial u}{\partial y} \right) + g\beta (T - T_\infty)$$

Boundary layer equations can be converted by using stream function concept. Let

$$u = \frac{\partial \varphi}{\partial y} \quad v = -\frac{\partial \varphi}{\partial x} \quad \theta = \frac{T - T_\infty}{T_s - T_\infty}$$

Then equations become:

$$\frac{\partial \varphi}{\partial y}\frac{\partial^2 \varphi}{\partial x \partial y} - \frac{\partial \varphi}{\partial x}\frac{\partial^2 \varphi}{\partial y^2} = \nu \frac{\partial^3 \varphi}{\partial y^3} + g\beta (T_s - T_\infty)\theta$$

$$\frac{\partial \varphi}{\partial y}\frac{\partial \theta}{\partial x} - \frac{\partial \varphi}{\partial x}\frac{\partial \theta}{\partial y} = \alpha \frac{\partial^2 \theta}{\partial y^2} + \theta \frac{\partial \varphi}{\partial y}\frac{dT_s}{dx}$$

A similarity transformation will be applied to this qauations in the form of

$\eta = yH(x)$ where $H(x) = \frac{1}{x}\left(\frac{1}{4}Gr_x\right)^{1/4}$ so $\eta = \frac{y}{x}\left(\frac{1}{4}Gr_x\right)^{1/4}$

And a separation of variable can be applied to the equations in the form of

$\varphi(x,\eta) = \upsilon F(\eta)G(x)$ where

$$G(x) = 4\left(\frac{1}{4}Gr_x\right)^{1/4}$$

Where $Gr_x = \frac{g\beta(T_s(x) - T_\infty)}{\nu^2}$ is the Garshoff number. For the case of $T_s(x)$=constant, the equation becomes:

$$F'''(\eta) + 3F(\eta)F'(\eta) - 2\left(F'(\eta)\right)^2 + \theta(\eta) = 0$$

$$\theta'''(\eta) + 3\,\mathrm{Pr}\,F(\eta)\theta'(\eta) = 0$$

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.

$$h_x = \frac{q_s''}{T_s - T_\infty} \qquad q_s'' = -k\left.\frac{dT}{dy}\right|_{y=0} = -k(T_s - T_\infty)\left(\frac{\partial\theta}{\partial\eta}\frac{\partial\eta}{\partial y}\right)_0$$

$q_s'' = -k(T_s - T_\infty)\theta'(0)H(x)$ Then the local Nusselt number will be defined as:

$$Nu_x = \frac{xh_x}{k} = -\theta'(0)xH(x) = -\frac{\theta'(0)}{\sqrt{2}}Gr_x^{1/4}$$ 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.

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:

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

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:

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

$k_1=f(x_i,y_i)$

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

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

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

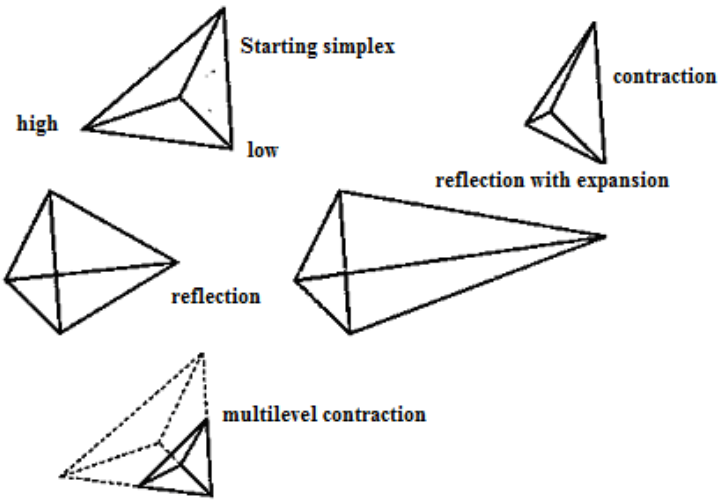
$k_5=f(x_i+0.75h , y_i + (3/16)k_1h+(9/16)k_4h)$

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

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 λ=1 else if $i \neq j$ λ=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 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};
 //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:

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

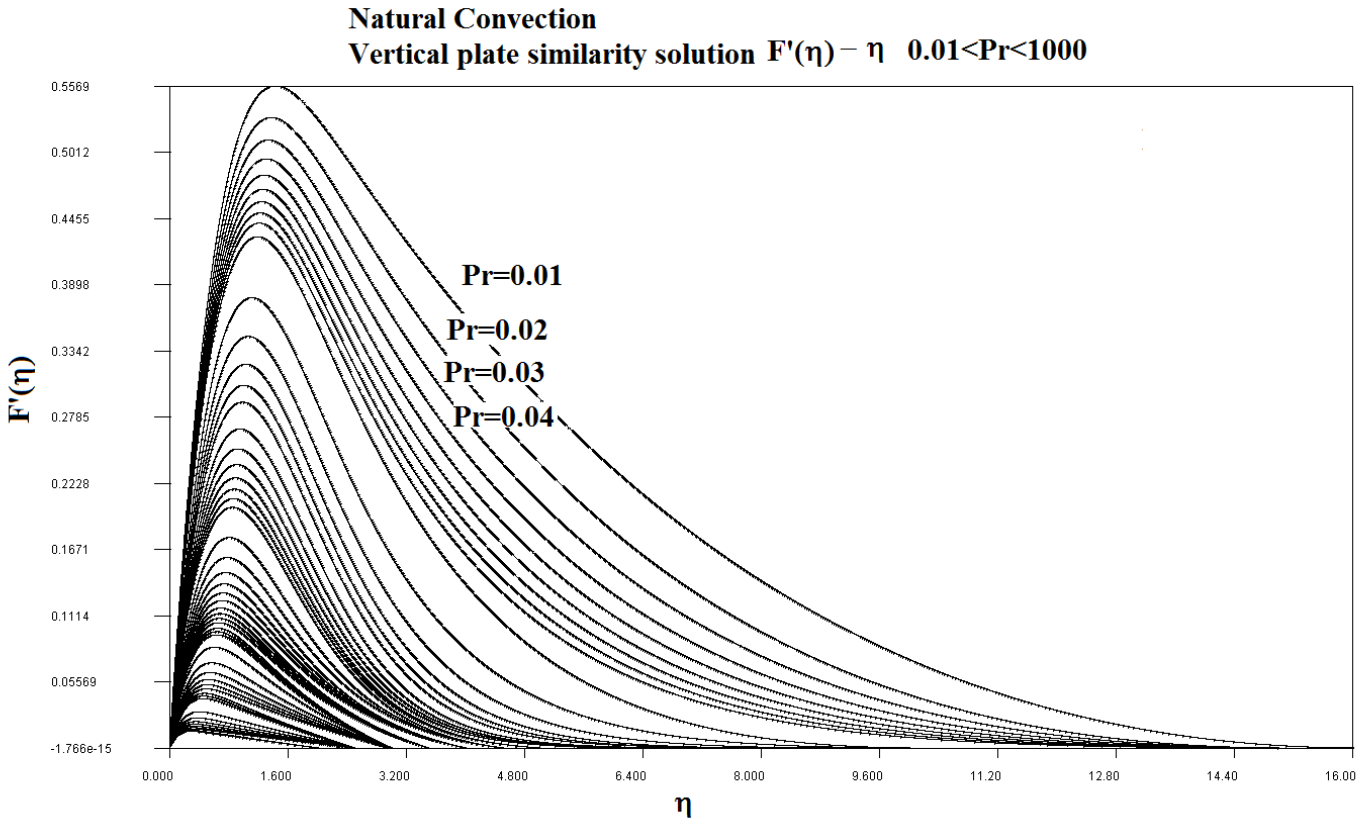
$$h_x = k\left(\frac{u_\infty}{\nu x}\right)^{1/2} \frac{\partial \theta(\eta)}{\partial \eta}\bigg|_{\eta=0} = \frac{k}{x}\left(\frac{u_\infty x}{\nu}\right)^{1/2} \frac{\partial \theta(\eta)}{\partial \eta}\bigg|_{\eta=0}$$

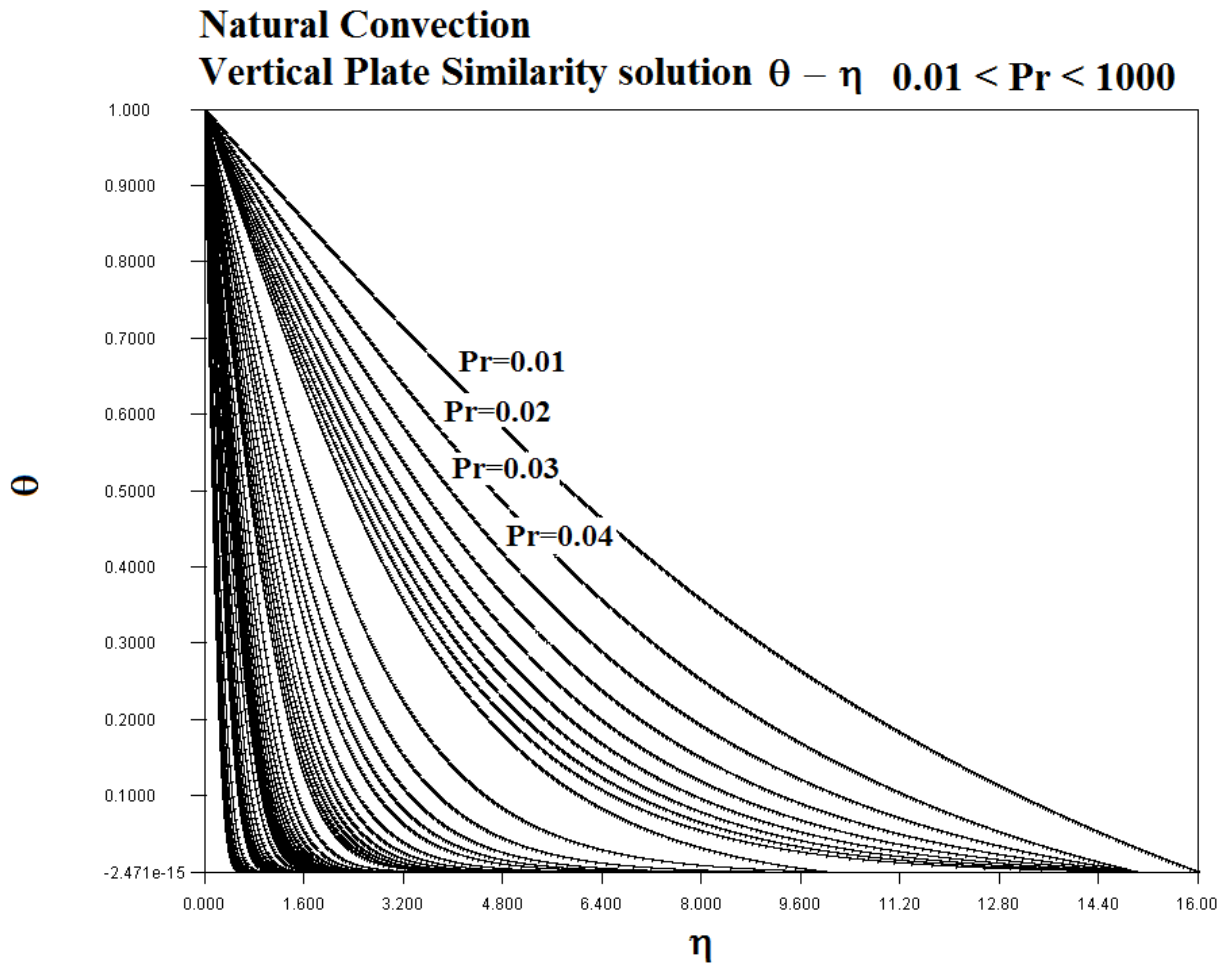
$$Nu_x = \frac{h_x x}{k}$$

Results for $\frac{\partial \theta(\eta)}{\partial \eta}\Big|_{\eta=0}$

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

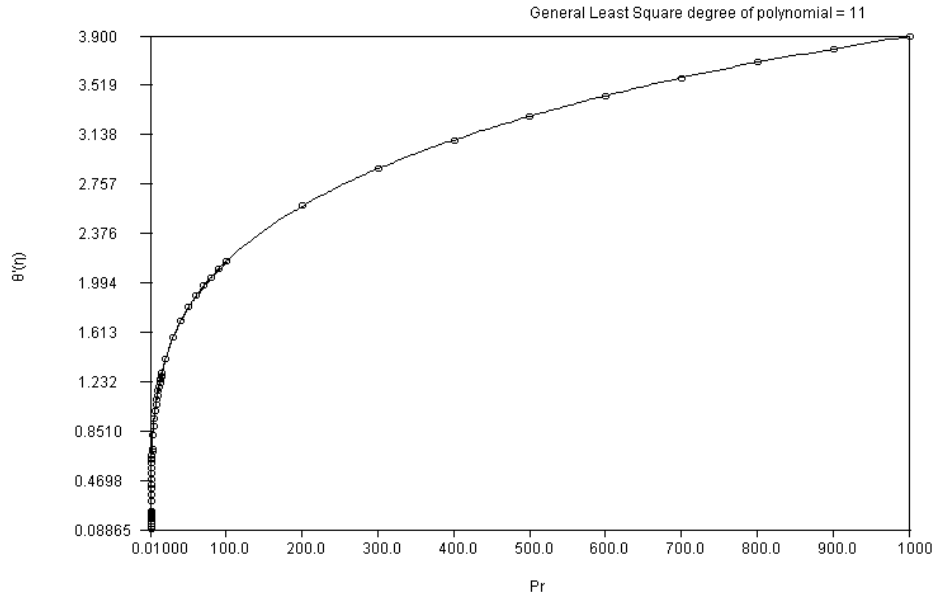
$$\left.\frac{\partial\theta}{\partial y}\right|_{y=0} = \theta' = a_0\,\mathrm{Pr}^{1/4} + a_1\,\mathrm{Pr}^{1/2} + a_2\,\mathrm{Pr}^{3/4} + a_3\,\mathrm{Pr} + a_4\,\mathrm{Pr}^{5/4} + a_5\,\mathrm{Pr}^{3/2} +$$

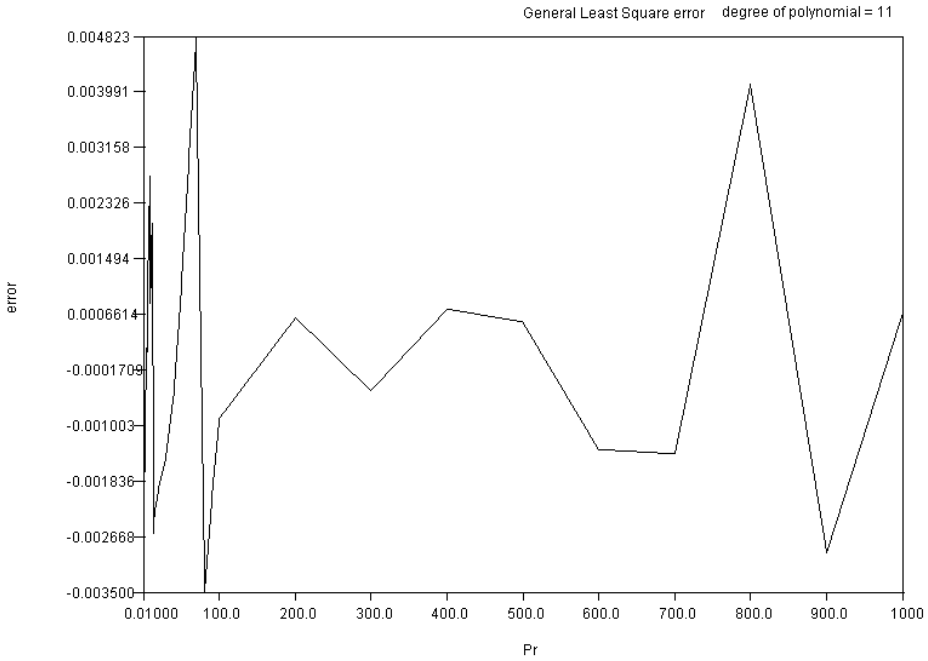
$$a_6\,\mathrm{Pr}^{2/4} + a_7\,\mathrm{Pr}^{2} + a_8\,\mathrm{Pr}^{9/4} + a_9\,\mathrm{Pr}^{5/2} + a_{10}\,\mathrm{Pr}^{11/4} + a_{11}\,\mathrm{Pr}^{3}$$

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

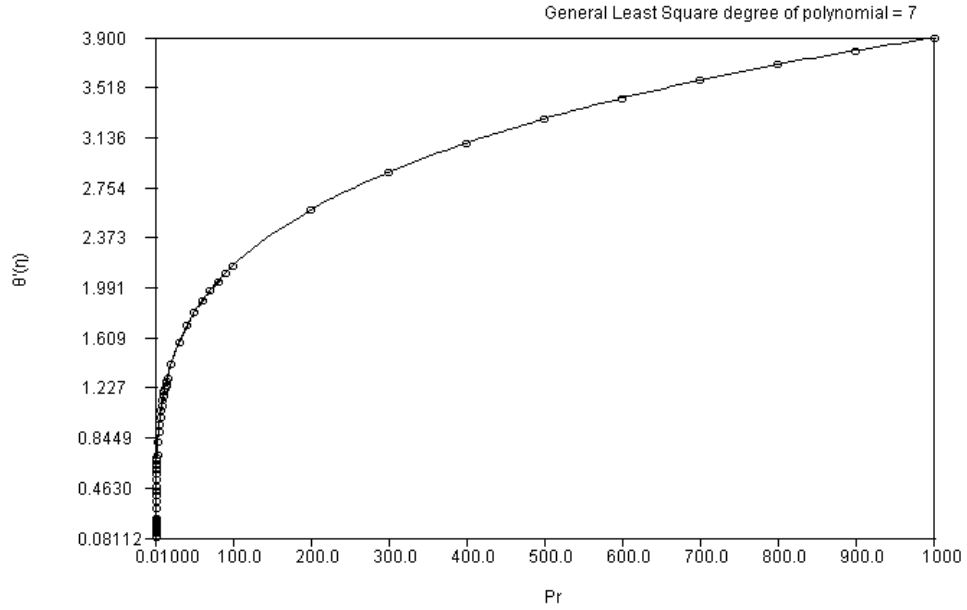
For this polynomial data and fitted line and error shown as follows

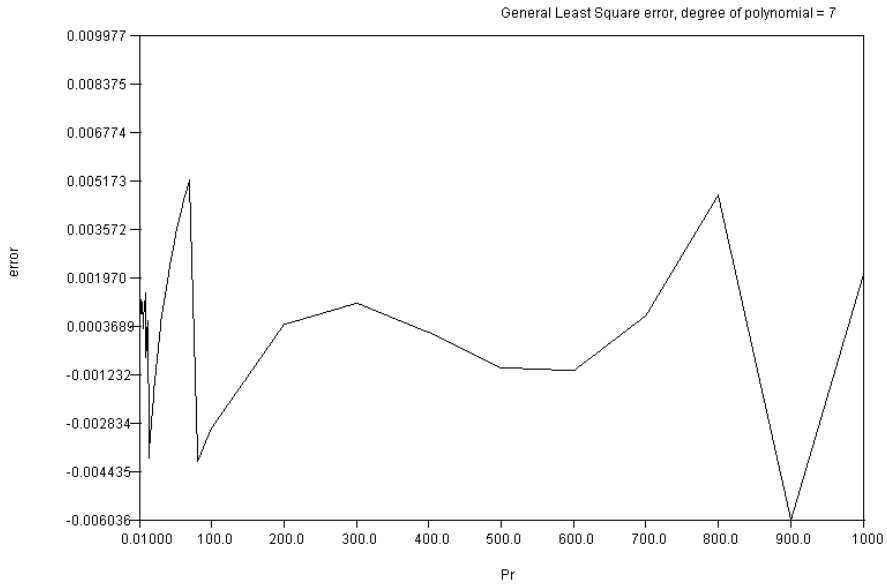




$$\left.\frac{\partial\theta}{\partial y}\right|_{y=0} = \theta' = a_0\,\mathrm{Pr}^{1/4} + a_1\,\mathrm{Pr}^{1/2} + a_2\,\mathrm{Pr}^{3/4} + a_3\,\mathrm{Pr} + a_4\,\mathrm{Pr}^{5/4} + a_5\,\mathrm{Pr}^{3/2} + a_6\,\mathrm{Pr}^{2/4} + a_7\,\mathrm{Pr}^{2}$$

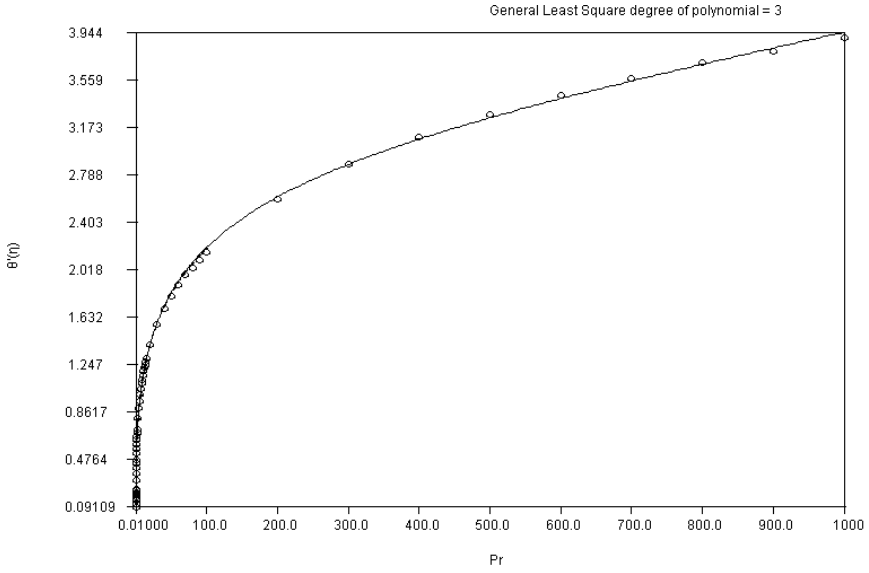
a0	-7.1616625047E-02
a1	1.3294314755E+00
a2	-1.0629736882E+00
a3	4.8311043084E-01
a4	-1.3183854844E-01
a5	2.1366843379E-02
a6	-1.8917609206E-03
a7	7.0288177613E-05

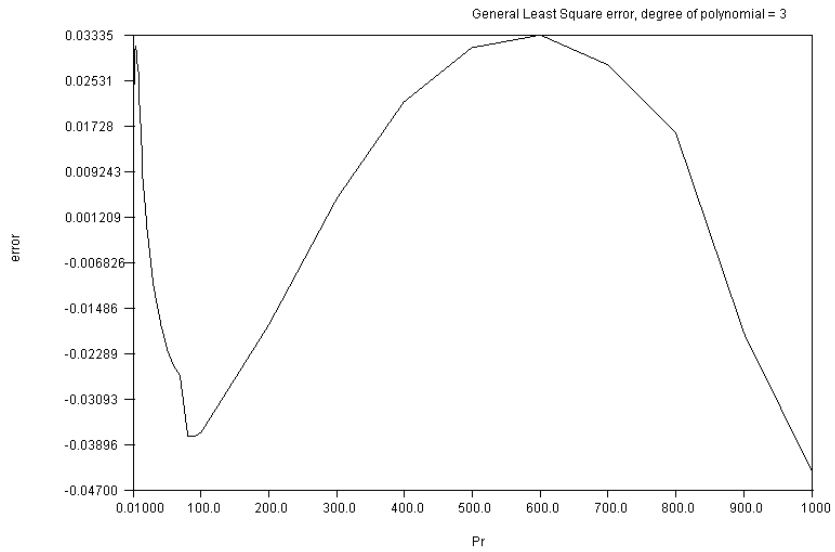




$$\left.\frac{\partial\theta}{\partial y}\right|_{y=0} = \theta' = a_0\,\mathrm{Pr}^{1/4} + a_1\,\mathrm{Pr}^{1/2} + a_2\,\mathrm{Pr}^{3/4} + a_3\,\mathrm{Pr}$$

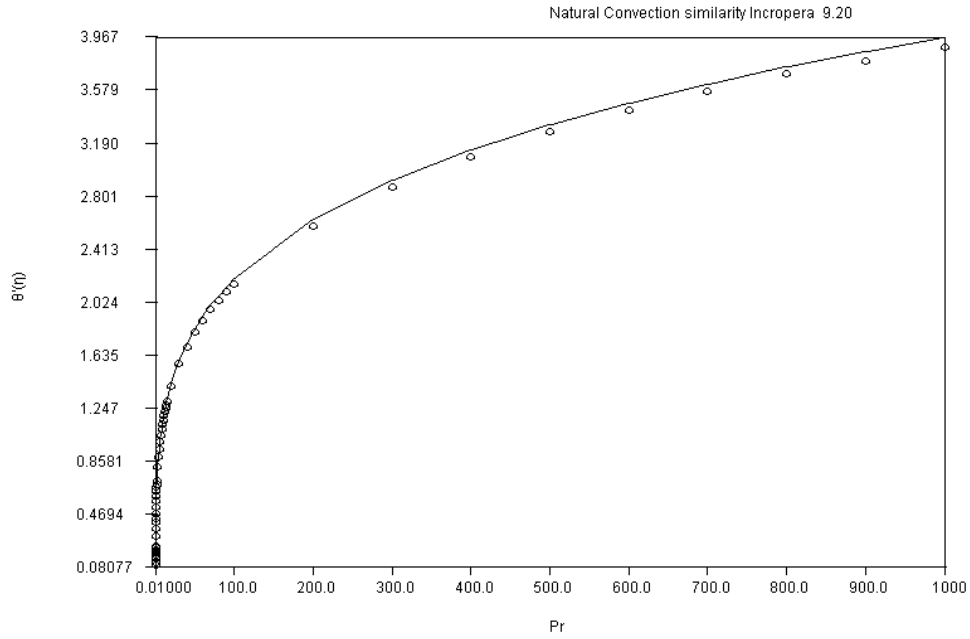
a0	0.337767076
a1	0.279380627
a2	-0.071379462
a3	0.005903166



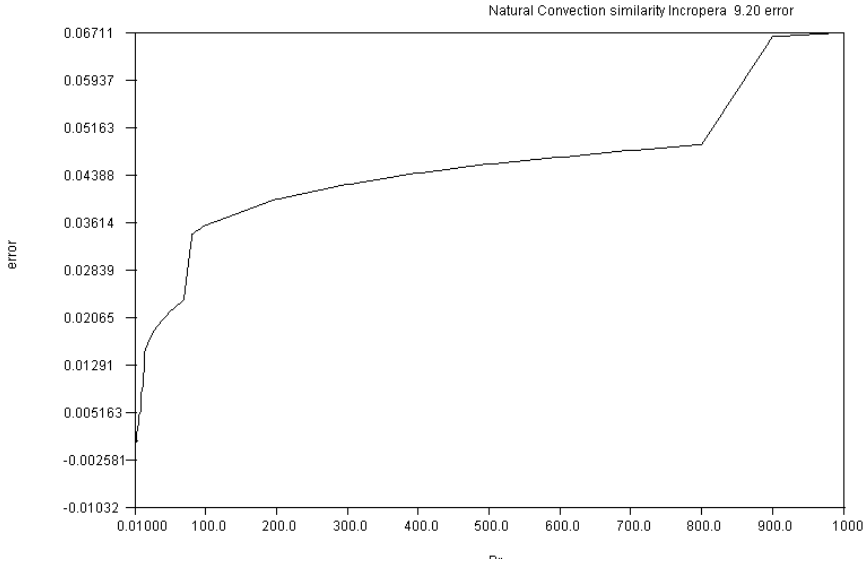


As a solution of this problem following equation is given in Fundamentals of Heat and Mass Transfer, Incropera/DeWitt/Bergman/Lavine, sixth edition

$$\left.\frac{\partial\theta}{\partial y}\right|_{y=0} = \theta' = \frac{0.75\,\mathrm{Pr}^{1/2}}{(0.609+1.221\mathrm{Pr}^{1/2}+1.238\,\mathrm{Pr})^{1/4}}$$



Error level of the equation is as follow:



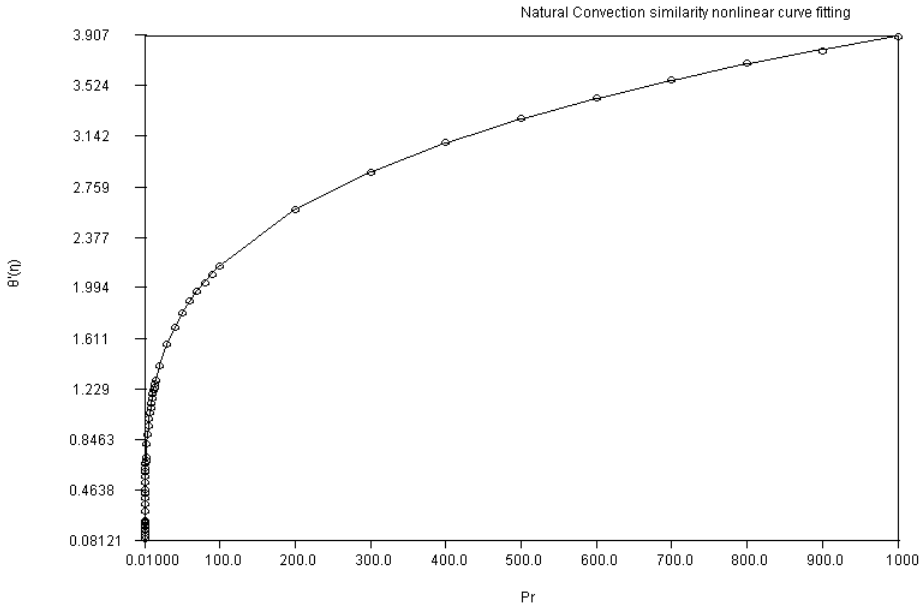
Pr	θ'(η) calculated	θ'(η) Incropera 9.20	Error
0.01	0.091093502	0.081214605	-0.009878897
0.02	0.116834579	0.112577416	-0.004257163
0.03	0.136995175	0.135789496	-0.001205679
0.04	0.154605985	0.154806134	2.00E-04
0.05	0.170229134	0.171157064	9.28E-04
0.06	0.184286962	0.185625468	0.001338506
0.07	0.19708927	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.

$$\left.\frac{\partial\theta}{\partial y}\right|_{y=0} = \theta' = \frac{a_0\,\mathrm{Pr}^{1/2}}{(a_1 + a_2\,\mathrm{Pr}^{1/2} + a_3\,\mathrm{Pr})^{1/4}}$$

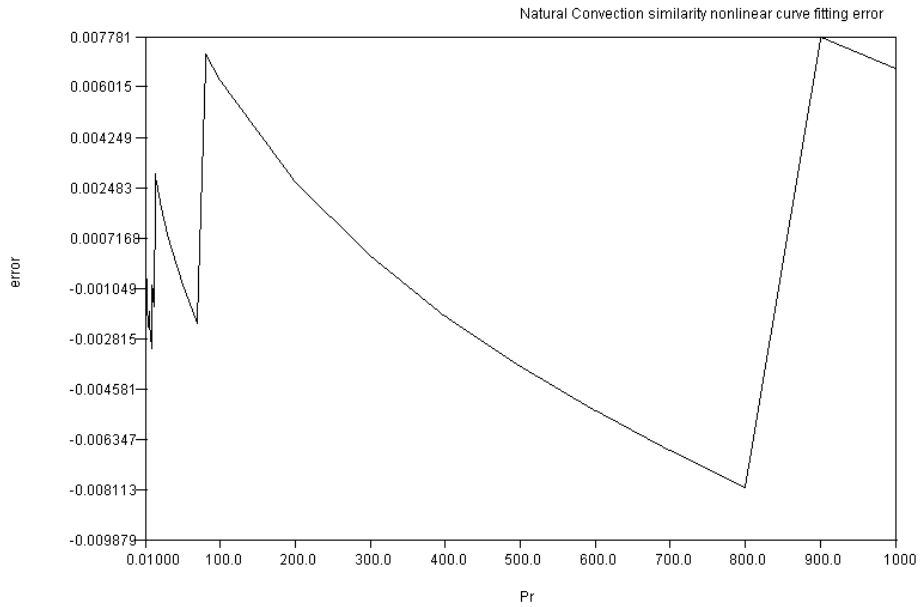
a0	0.756705949
a1	0.621261022
a2	1.187008971
a3	1.369213226



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

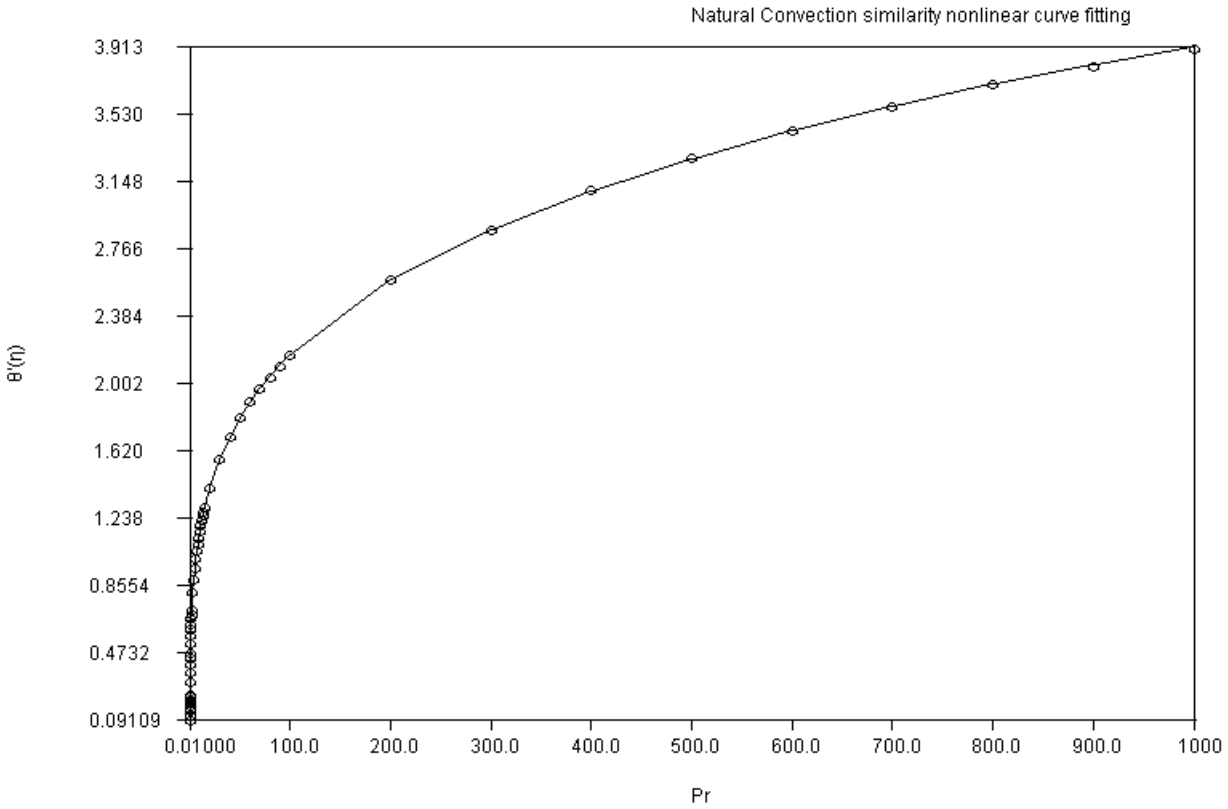
0.02	0.11683458	0.112577416	-0.004257163
0.03	0.13699518	0.135789496	-0.001205679
0.04	0.15460599	0.154806134	2.00E-04
0.05	0.17022913	0.171157064	9.28E-04
0.06	0.18428696	0.185625468	0.001338506
0.07	0.19708927	0.198675703	0.001586433
0.08	0.20886509	0.210609701	0.001744613
0.09	0.21978666	0.221636726	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 eqautions by increasing number of terms as well. Let us assume the equation in the form of

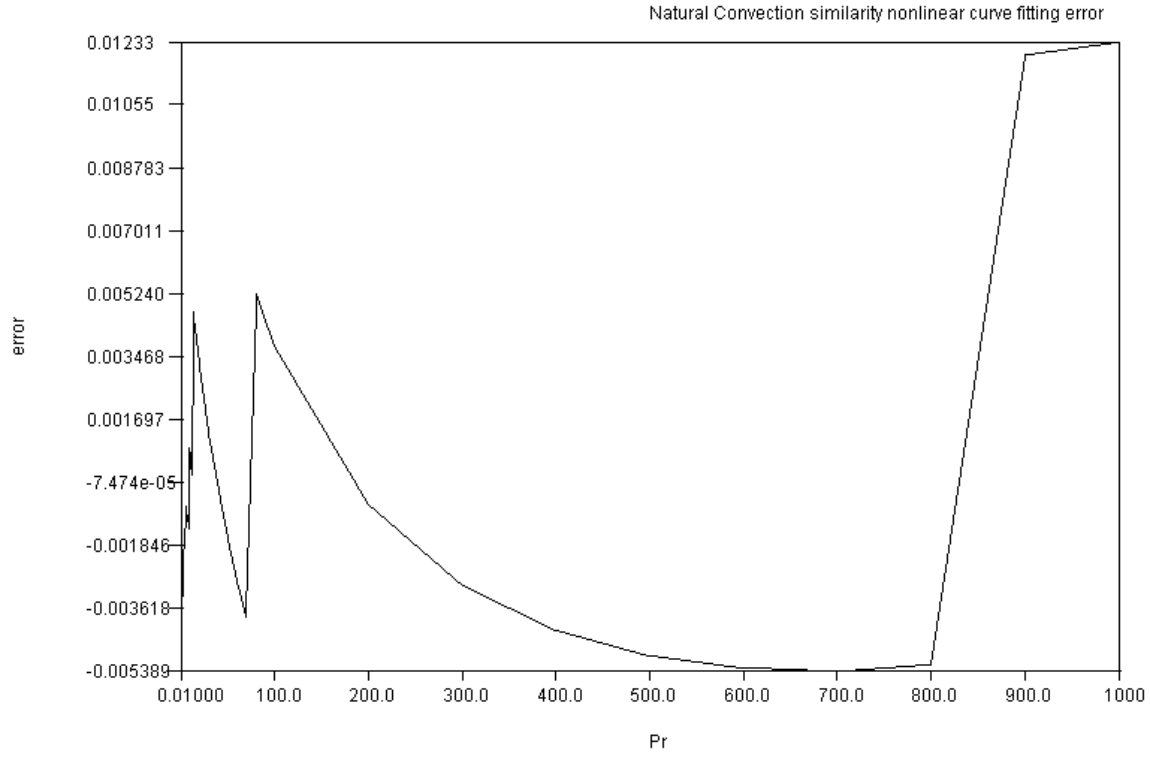
$$\left.\frac{\partial\theta}{\partial y}\right|_{y=0} = \theta' = \frac{a_0 + a_1 \,\mathrm{Pr}^{1/2} + a_2 \,\mathrm{Pr}}{(a_3 + a_4 \,\mathrm{Pr}^{1/2} + a_5 \,\mathrm{Pr})^{1/4}}$$

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.

$$\bar{h} = \frac{1}{L}\int_0^L h dx = \frac{k}{L}\left[\frac{g\beta(T_s - T_\infty)}{4\nu^2}\right]^{1/4}\left(\frac{\partial\theta}{\partial\eta}\right)_{\eta=0}\int_0^L \frac{dx}{x^{1/4}}$$

And

$$\bar{Nu} = \frac{\bar{h}L}{k} = \frac{4}{3}Nu_x$$

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.