BASICS OF COMPUTATIONAL FLUID DYNAMICS ASSIGNMENT

Numerical Evaluation of Temperature distribution in a slab subjected to steady state heat conduction

2-D STEADY STATE HEAT CONDUCTION PROBLEM

BY :

B.R.STHAVISHTHA

13ME125

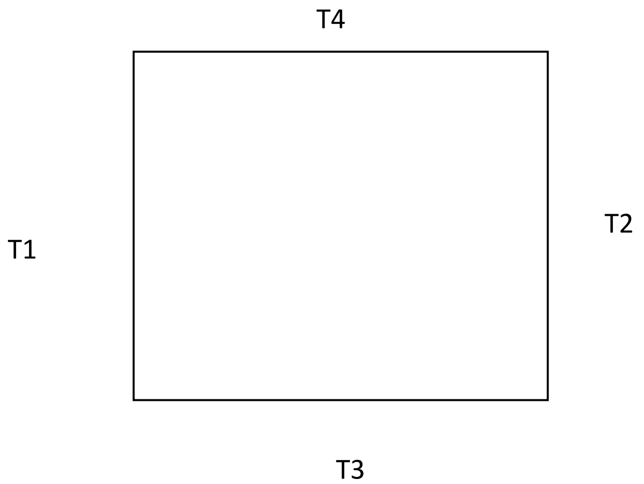
M1

Table of Contents

Problem statement

Part -I

The dimensions of the slab are 1m x 1m.



The problem expects us to find the temperature distribution in a slab subjected to steady state conduction. The discretized equations are solved using Point Gauss Siedel method, successive over relaxation (SOR) and Line Gauss Siedel method (LGS). A comparison regarding the computational time is also made for each method. The temperature contour plots are also generated for each Dirichlet boundary condition in MATLAB. The temperature contour plots generated by my code has been validated with FEM results of FEATool. FEATool[1] is a MATLAB toolbox for modeling and simulation of partial differential equations type physics and mathematical problems with the finite element method (FEM).

Part - II

User defined functions **tdma** and **gausselimination** are created which print the solutions of the linear system of equations when the equations are input to these functions in the form a matrix.

[1] www.precisesimulation.com/featool/

Literature

Direct Methods[1]

For a system of N equations with N unknowns, these methods require a simultaneous storage of all N^2 coefficients of the set of equations in the core memory. Some examples of direct methods include TDMA (Tri-diagonal matrix algorithm), Cramer's rule and Gauss Elimination.

Iterative Methods[2]

These methods are based on the repeated application of a relatively simple algorithm leading to eventual convergence after a large number of repetitions. The iterative methods used in this assignment include point Gauss Siedel method and Line Gauss siedel method. The advantage of iterative methods is that only non-zero coefficients need to be stored in the memory.

Jacobi and Gauss Siedel iterative methods are easy to implement but they can be slow to converge when the system of equations is large. Hence they are not widely used for solving multi-dimensional problems. Though the TDMA method is a direct method, it can be used iteratively along with the Gauss Siedel method in a line-by-line fashion (Line Gauss Siedel method- LGS), to solve multidimensional problems. Hence this is used in this assignment.

Point Gauss Siedel method[3]

The general form of the algebraic equations for the unknown variables is:

$$\sum_{j=1}^{i-1} A_{ij}\emptyset_j + A_{ii}\emptyset_i + \sum_{j=i+1}^{n} A_{ij}\emptyset_j = B_i$$

Solving for φ_i, we have:

$$\emptyset_i^{k+1} = \frac{B_i}{A_{ii}} - \sum_{j=1}^{i-1}\frac{A_{ij}}{A_{ii}}\emptyset_j^k - \sum_{j=i+1}^{n}\frac{A_{ij}}{A_{ii}}\emptyset_j^k$$

In this method, the updated nodal variables $\emptyset_j^{k+1}$ are immediately used on the right-hand side of the above equation as soon as they are available. For a two-

[2] Versteeg and Malalasekara 1995, *An introduction to Computational Fluid Dynamics* - the Finite volume method, Longman Scientific and Technical

[3] Tu, Jiyuan et al, *Computational Fluid Dynamics, A Practical Approach*

dimensional case, since this method is applied at each nodal point, it is called Point Gauss Siedel method.

The convergence of the Point Gauss Siedel method can be enhanced by the technique **Successive Over relaxation**. This is an extrapolation procedure in which the intermediate nodal variables $\emptyset_j^{k+1}$ are further advanced by a weighted average of the current values of $\emptyset_j^{k+1}$ with the previous values of $\emptyset_j^{k}$ .The extrapolated values of $\emptyset_j^{k}$ are obtained as follows:

$$\emptyset_i^{k+1} = (1-\lambda)\emptyset_i^{k} + \lambda\emptyset_i^{k+1}$$

In the above equation, λ is a relaxation factor whose value is found by trial and error. Convergence can be achieved at a faster rate for $1< \lambda<2$ when used along with Gauss Siedel method.

Line Gauss Siedel Method (LGS)[4]

This is a simple, but efficient algorithm that combines direct method (TDMA) and Gauss-Siedel iteration method[5]. The discretized equation is obtained as $a_P T_P = a_E T_E + a_W T_W + a_N T_N + a_S T_S + b$ from numerical solution of 2-dimensional steady-state conduction.

This method can be applied either in x-direction or y-direction. To apply this method in the x-direction, we can choose the *jth* row to apply the TDMA. To do this, it is assumed that the temperature of *(j-1)th* and *(j+1)th* rows are known from the previous iteration. For the first iteration at the current time step, they can be taken as values at the previous time step.

$$A_{i,j}T_{i,j} = B_{i,j}T_{i+1,j} + C_{i,j}T_{i-1,j} + (D_{i,j}T_{i,j+1} + E_{i,j}T_{i,j-1} + S)$$

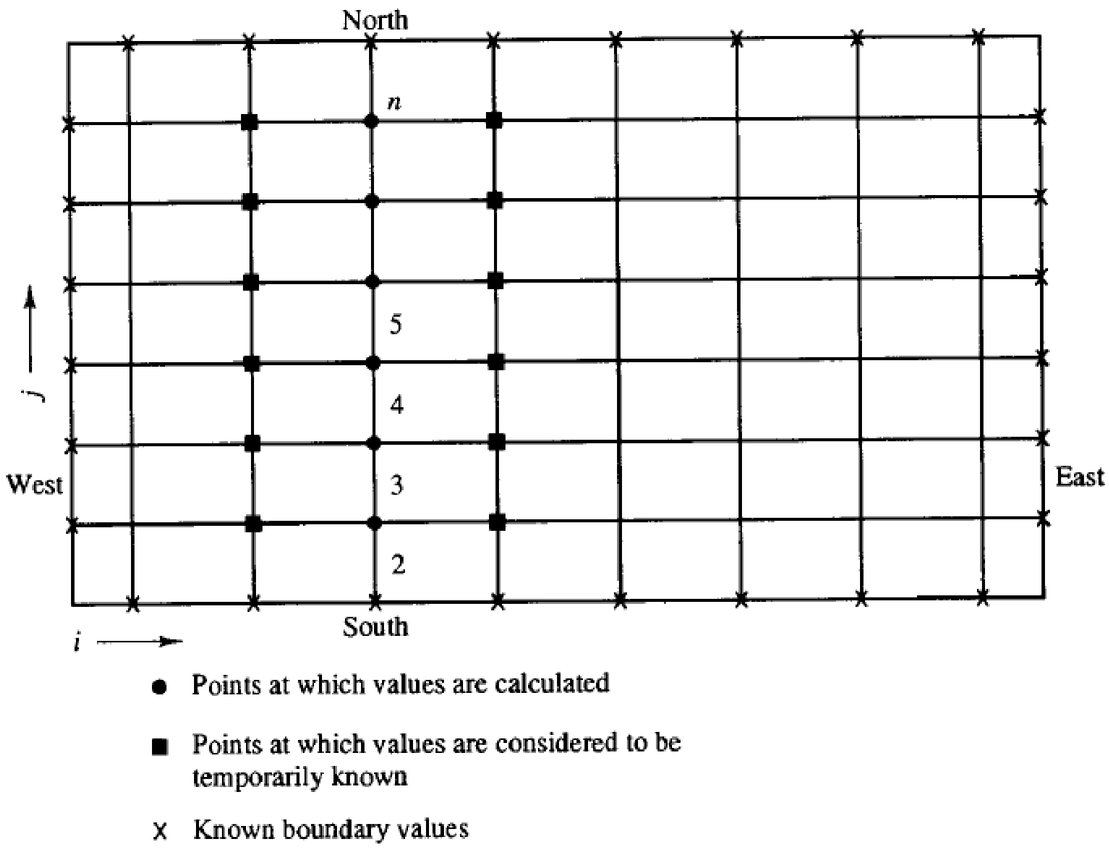
The terms in brackets can be evaluated from the previous iteration. The above equation can be solved using TDMA to get $T_{i,j}$ in the *jth* row. Similarly, the above equation can be modified to proceed along the y-direction.

Once the procedure is completed along the x-direction twice (from j= 1 to N and then from j=N to 1) and along the y-direction twice (from i=1 to M and then from i= M to 1)-referred to as one iteration-the information from boundary conditions on all four sides can be propagated throughout the computational domain.

---

[4] https://www.thermalfluidscentral.org/encyclopedia

[5] Patankar ,S.V., 1980, *Numerical Heat Transfer and Fluid Flow,* Hemisphere, Washington DC



Finite Volume method for 2-Dimensional Steady State Heat Conduction in a slab

The governing equation for 2-Dimensional steady state diffusion without source term is :

$$\frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right) + \mathrm{H} = 0, \text{ where H = volumetric heat addition}$$

Integrating over the control volume, k = thermal conductivity of material

$$\int \frac{\partial}{\partial \mathrm{x}}\left(\mathrm{k}\frac{\partial \mathrm{T}}{\partial \mathrm{x}}\right) \mathrm{dV} + \int \frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right) dV + \mathrm{H}\int dV = 0$$

By Gauss-Divergence theorem,

$$\sum_1^4 \left(k\frac{\partial T}{\partial x}\right).A_i^x + \sum_1^4 \left(k\frac{\partial T}{\partial y}\right).A_i^y + \mathrm{H}\,\Delta\mathrm{x}\,\Delta\mathrm{y} = 0 \text{ ---------------------- (1)}$$

Assuming that the thermal conductivity k is constant, the equations are:

$$\sum_1^4 \left(\frac{\partial T}{\partial x}\right).A_i^x + \sum_1^4 \left(\frac{\partial T}{\partial y}\right).A_i^y + \frac{\mathrm{H}\,\Delta\mathrm{x}\,\Delta\mathrm{y}}{k} = 0$$

$\sum_1^4 \left(\frac{\partial T}{\partial x}\right).A_i^x = (\frac{\partial T}{\partial x})_E.A_E$ - $(\frac{\partial T}{\partial x})_W.A_w$ as projected areas $A_N^x = A_s^x = 0$

$\sum_1^4 \left(\frac{\partial T}{\partial y}\right).A_i^y = (\frac{\partial T}{\partial x})_N.A_N$ - $(\frac{\partial T}{\partial x})_S.A_S$ as projected areas $A_E^x = A_W^x = 0$

Note that $A_E = \Delta y = A_W$ and

$A_N = \Delta x = A_s$

By central differencing scheme,

$(\frac{\partial T}{\partial x})_E = \frac{T_E - T_P}{\Delta x}$ and $(\frac{\partial T}{\partial x})_W = \frac{T_P - T_W}{\Delta x}$

$(\frac{\partial T}{\partial y})_N = \frac{T_N - T_P}{\Delta y}$ and $(\frac{\partial T}{\partial y})_S = \frac{T_P - T_S}{\Delta y}$

Substituting them in Eqn (1),

$$(T_E + T_W - 2T_P)\frac{\Delta y}{\Delta x} + (T_N + T_S - 2T_P)\frac{\Delta x}{\Delta y} + \frac{\mathrm{H}\,\Delta x\,\Delta y}{k} = 0$$

Dividing by Δx Δy, we get

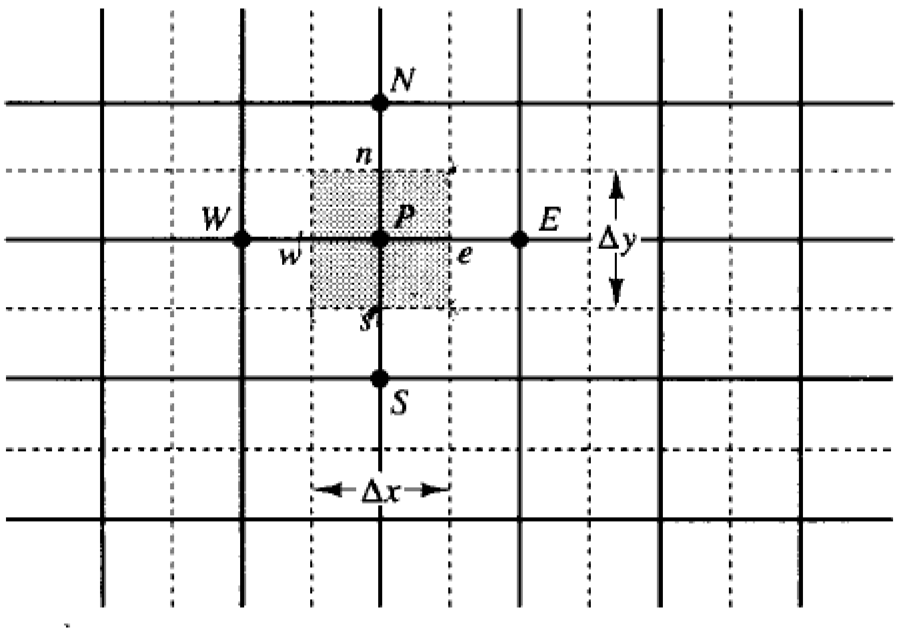
$$2T_P\left(\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2}\right) = \frac{(T_E + T_W)}{(\Delta x)^2} + \frac{(T_N + T_S)}{(\Delta y)^2} + \frac{\mathrm{H}}{k}$$

The discretized equation is:

$$a_P T_P = a_E T_E + a_N T_N + a_S T_S + a_W T_W + S$$

Node	Coefficient a_P	Coefficient a_E	Coefficient a_N	Coefficient a_S	Coefficient a_W	Source Term S
First node (1,1)	$3(\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2})$	$\frac{1}{(\Delta x)^2}$	$\frac{1}{(\Delta y)^2}$	0	0	$2(\frac{T_1}{(\Delta x)^2} + \frac{T_3}{(\Delta y)^2}$ $\frac{\mathrm{H}}{k}$
Last node (m,n)	$3(\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2})$	0	0	$\frac{1}{(\Delta y)^2}$	$\frac{1}{(\Delta x)^2}$	$2(\frac{T_2}{(\Delta x)^2} + \frac{T_4}{(\Delta y)^2}) + \frac{\mathrm{H}}{k}$
Node at (1,n)	$3(\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2})$	0	$\frac{1}{(\Delta y)^2}$	0	$\frac{1}{(\Delta x)^2}$	$2(\frac{T_2}{(\Delta x)^2} + \frac{T_3}{(\Delta y)^2}) + \frac{\mathrm{H}}{k}$

Node at (m,1)	$3(\frac{1}{(\Delta x)^2}+\frac{1}{(\Delta y)^2})$	$\frac{1}{(\Delta x)^2}$	0	$\frac{1}{(\Delta y)^2}$	0	$2(\frac{T_1}{(\Delta x)^2}+\frac{T_4}{(\Delta y)^2})+\frac{H}{k}$
Nodes in first row	$\frac{2}{(\Delta x)^2}+\frac{3}{(\Delta y)^2}$	$\frac{1}{(\Delta x)^2}$	$\frac{1}{(\Delta y)^2}$	0	$\frac{1}{(\Delta x)^2}$	$\frac{2T_3}{(\Delta y)^2}+\frac{H}{k}$
Nodes in first column	$\frac{3}{(\Delta x)^2}+\frac{2}{(\Delta y)^2}$	$\frac{1}{(\Delta x)^2}$	$\frac{1}{(\Delta y)^2}$	$\frac{1}{(\Delta y)^2}$	0	$\frac{2T_1}{(\Delta x)^2}+\frac{H}{k}$
Nodes in last row	$\frac{2}{(\Delta x)^2}+\frac{3}{(\Delta y)^2}$	$\frac{1}{(\Delta x)^2}$	0	$\frac{1}{(\Delta y)^2}$	$\frac{1}{(\Delta x)^2}$	$\frac{2T_4}{(\Delta y)^2}+\frac{H}{k}$
Nodes in last column	$\frac{3}{(\Delta x)^2}+\frac{2}{(\Delta y)^2}$	0	$\frac{1}{(\Delta y)^2}$	$\frac{1}{(\Delta y)^2}$	$\frac{1}{(\Delta x)^2}$	$\frac{2T_2}{(\Delta x)^2}+\frac{H}{k}$
Other inner nodes	$2(\frac{1}{(\Delta x)^2}+\frac{1}{(\Delta y)^2})$	$\frac{1}{(\Delta x)^2}$	$\frac{1}{(\Delta y)^2}$	$\frac{1}{(\Delta y)^2}$	$\frac{1}{(\Delta x)^2}$	$\frac{H}{k}$



Results (Part- I)

Case 1 - left side exposed to a higher temperature with respect to right side and bottom side exposed to high temperature with respect to the top side

20 X 20 nodes

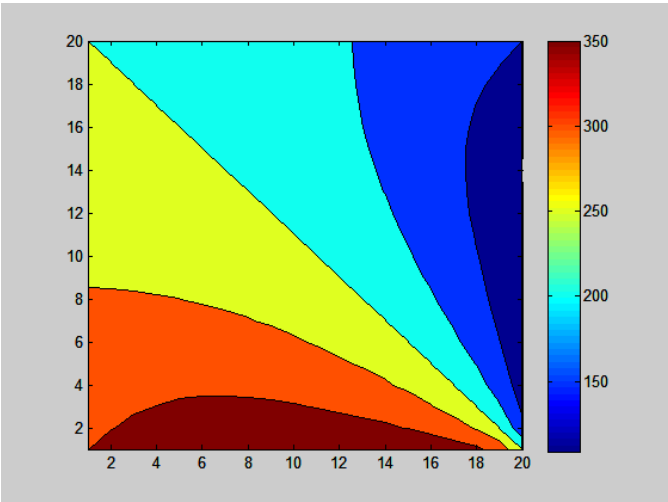
Total time taken: 246.987 seconds

T1=300 °C

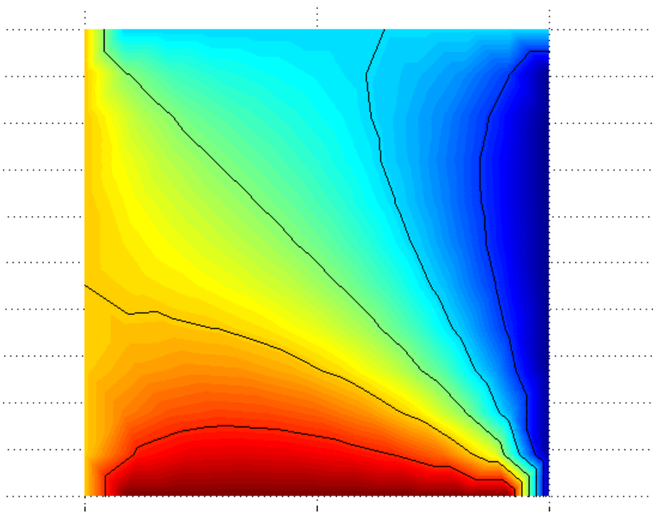
T2=100 °C

T3=400 °C

T4=200 °C



Results of my code



Results of FEATool

Case 2 - left side exposed to a higher temperature with respect to the right side and bottom side exposed to a lower temperature with respect to the top side

20 x 20 nodes

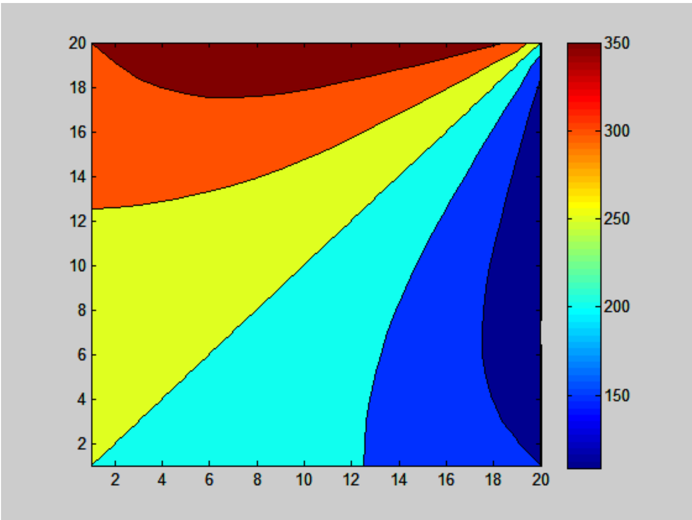
Total Time taken=279.20 sec

T1=300 °C

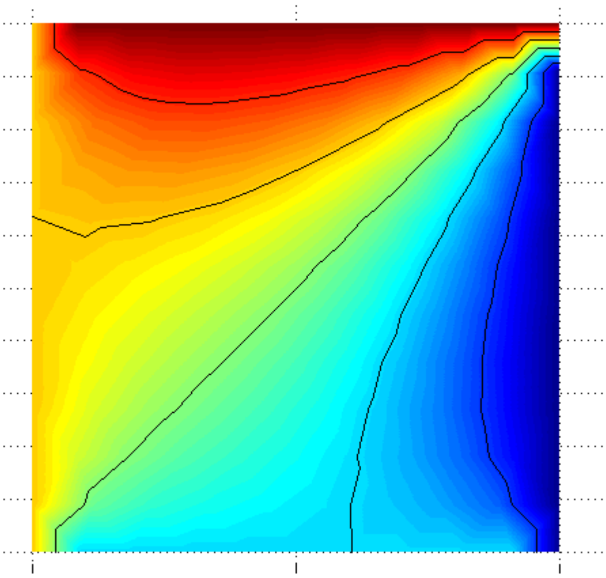
T2=100 °C

T3=200 °C

T4=400 °C



Results of my code



Results of FEATool

Case 3 - left side exposed to a lower temperature with respect to the right side and bottom side exposed to a higher temperature with respect to the top side

20 x 20 nodes

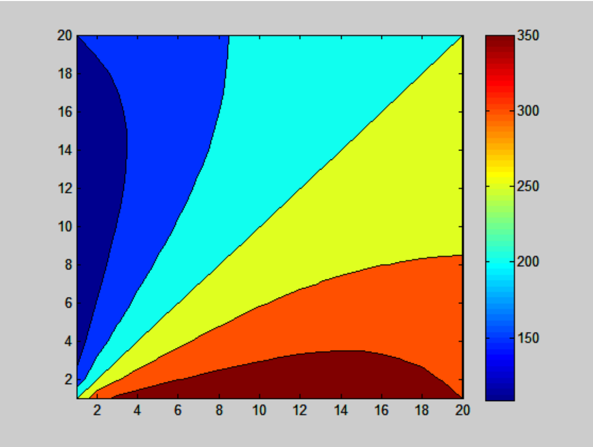
Total Time taken=325.737 sec

T1=100 °C

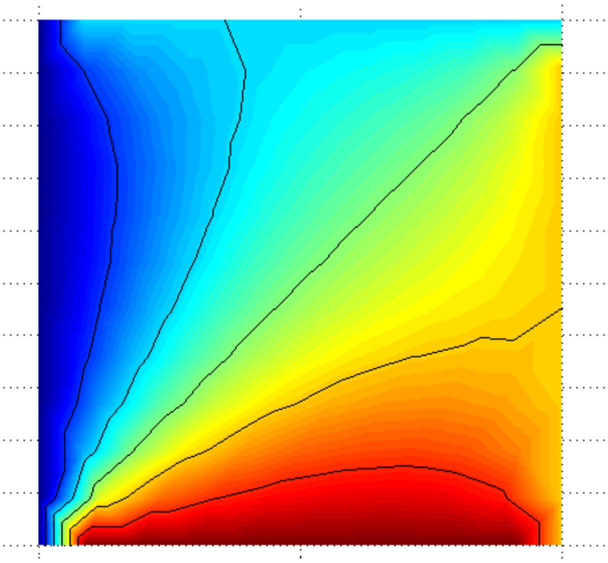
T2=300 °C

T3=400 °C

T4=200 °C



Results of my code



Results of FEATool

Case 4 - left side exposed to a lower temperature with respect to the right side and bottom side exposed to a lower temperature with respect to the top side

20x 20 nodes

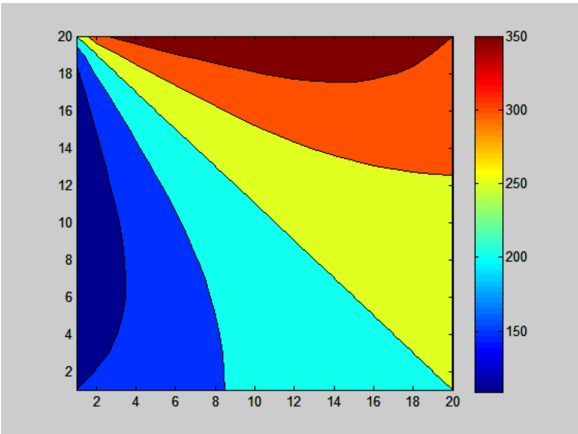
Total Time taken: 317.015 seconds

T1=100 °C

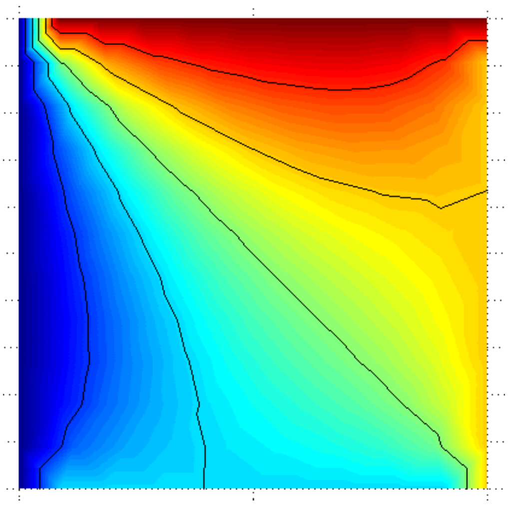
T2=300 °C

T3=200 °C

T4=400 °C



Results of my code



Results of FEATool

Case 5 - constant boundary conditions on left and right sides high temperature at bottom side with respect to the top side

20x 20 nodes

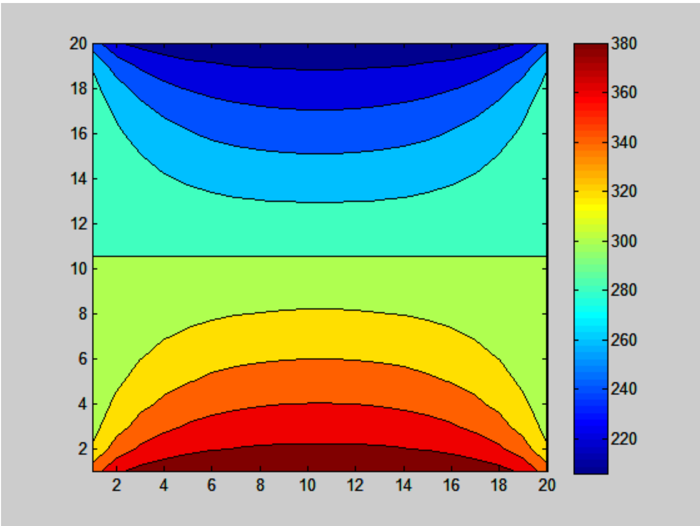
Total Time taken: 159.884 seconds

T1=300 °C

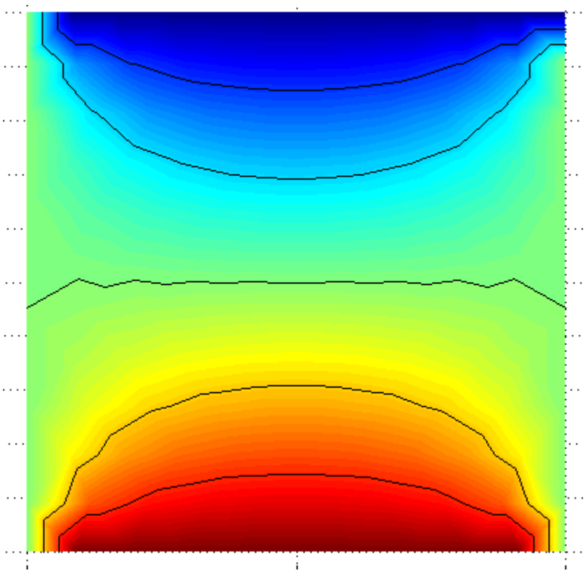
T2=300 °C

T3=400 °C

T4=200 °C



Results of my code



Results of FEATool

Case 6 - constant boundary conditions on left and right sides high temperature at top side with respect to the bottom side

20x 20 nodes

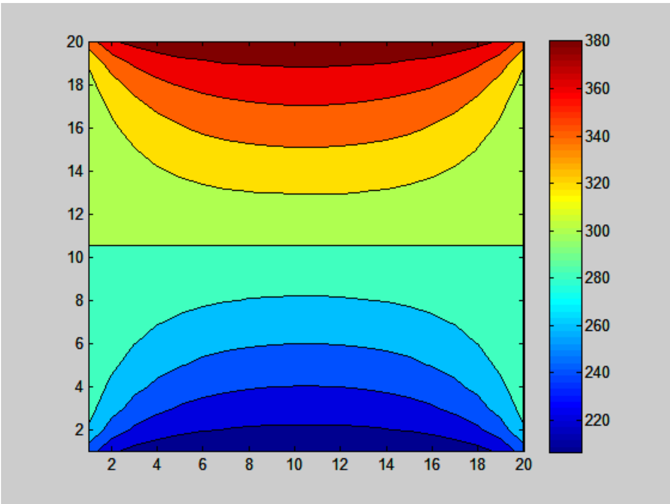
Total Time taken: 307.282 seconds

T1=300 °C

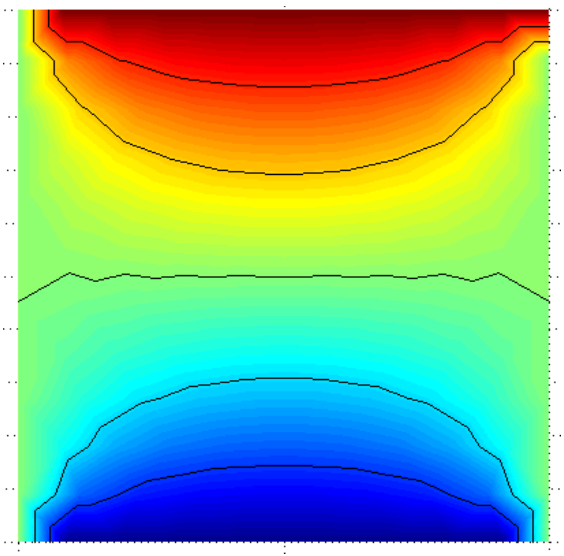
T2=300 °C

T3=200 °C

T4=400 °C



Results of my code



Results of FEATool

Case 7 - constant boundary conditions on top and bottom sides high temperature at left side with respect to the right side

20x 20 nodes

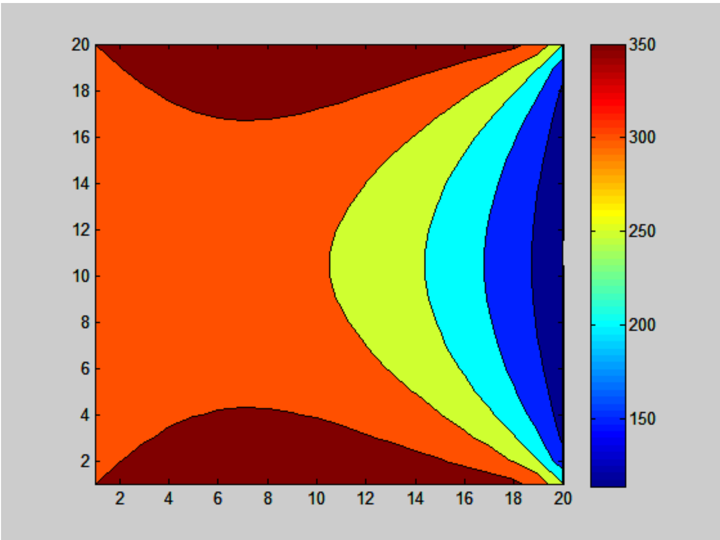
Total Time taken: 168.272seconds

T1=300 °C

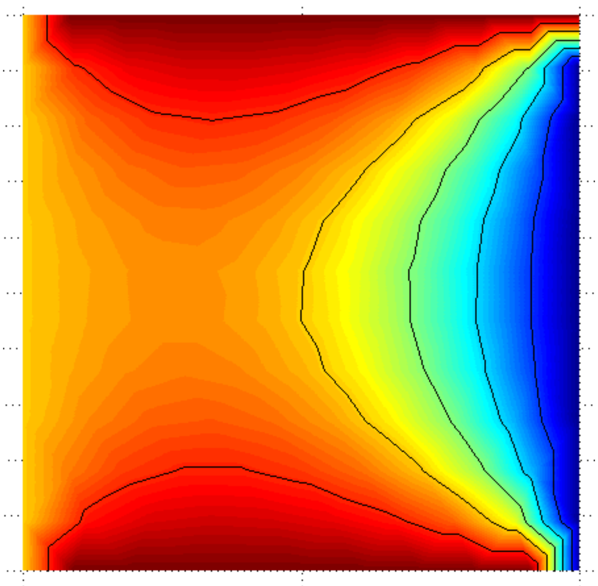
T2=100 °C

T3=400 °C

T4=400 °C



Results of my code



Results of FEATool

Case 8 - constant boundary conditions on top and bottom sides high temperature at right side with respect to the left side

20x 20 nodes

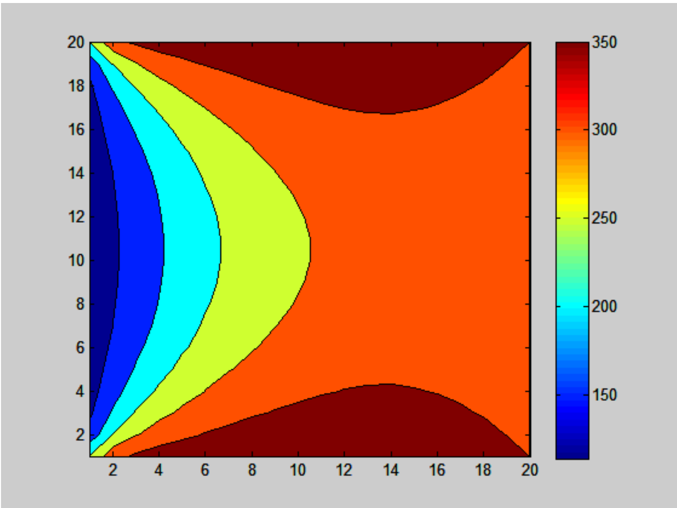
Total Time taken: 321.868 seconds

T1=100 °C

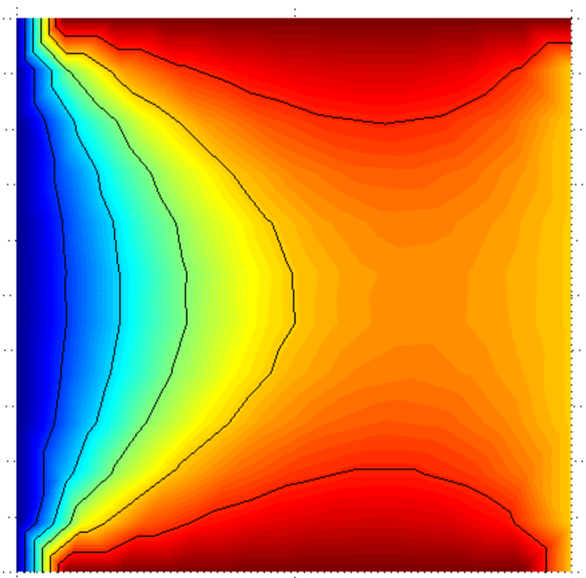
T2=300 °C

T3=400 °C

T4=400 °C



Results of my code



Results of FEATool

Case 9 - left side exposed to a higher temperature with respect to right side and bottom side exposed to high temperature with respect to the top

20 X 20 nodes

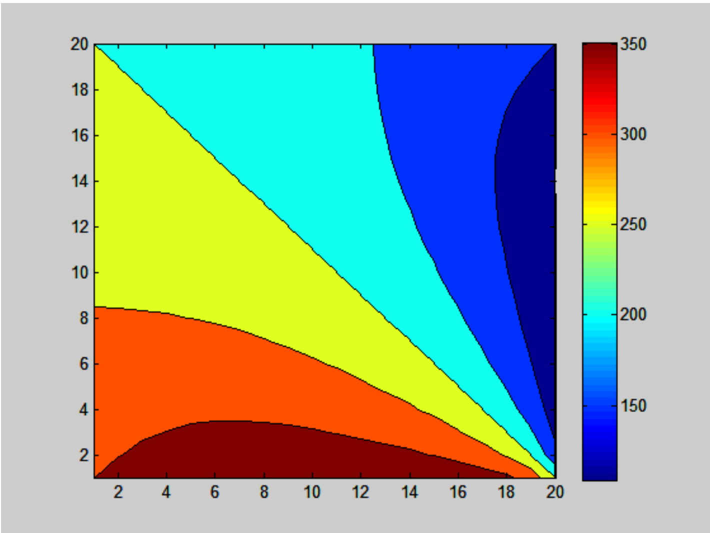
Total time taken: 608.930 seconds

T1=300 °C

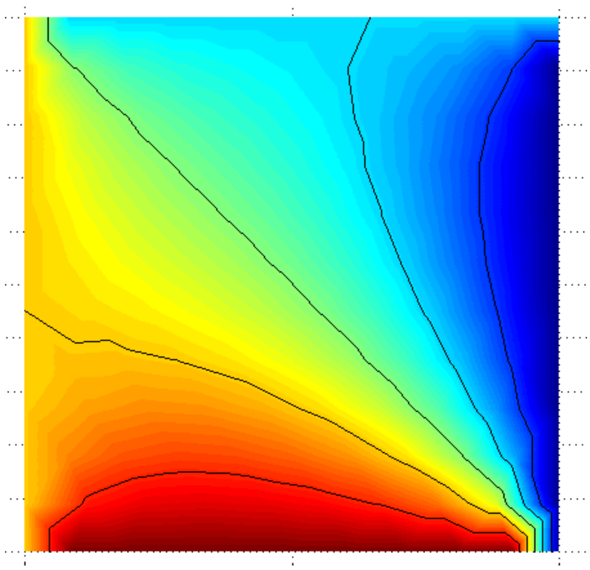
T2=100 °C

T3=400 °C

T4=200 °C



Results of my code



Results of FEATool

Case 10 (Successive Over relaxation) - left side exposed to a higher temperature with respect to right side and bottom side exposed to high temperature with respect to the top side

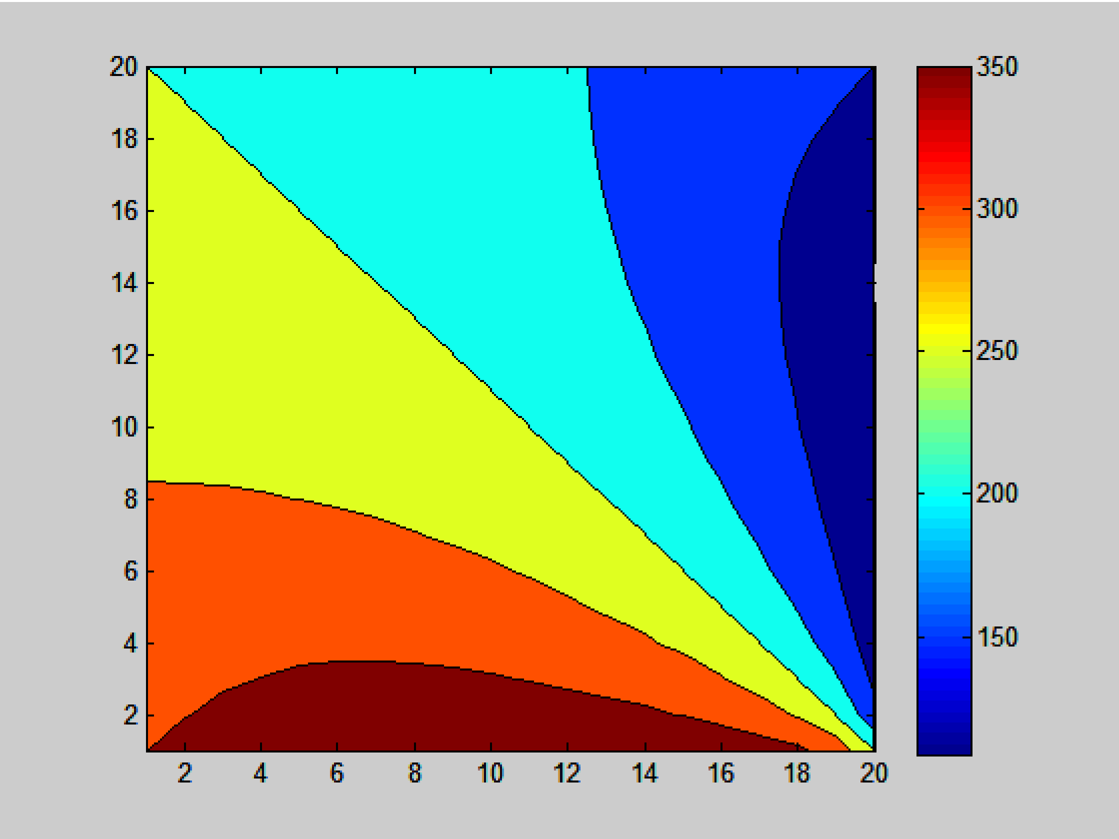
20 X 20 nodes

T1=300 °C

T2=100 °C

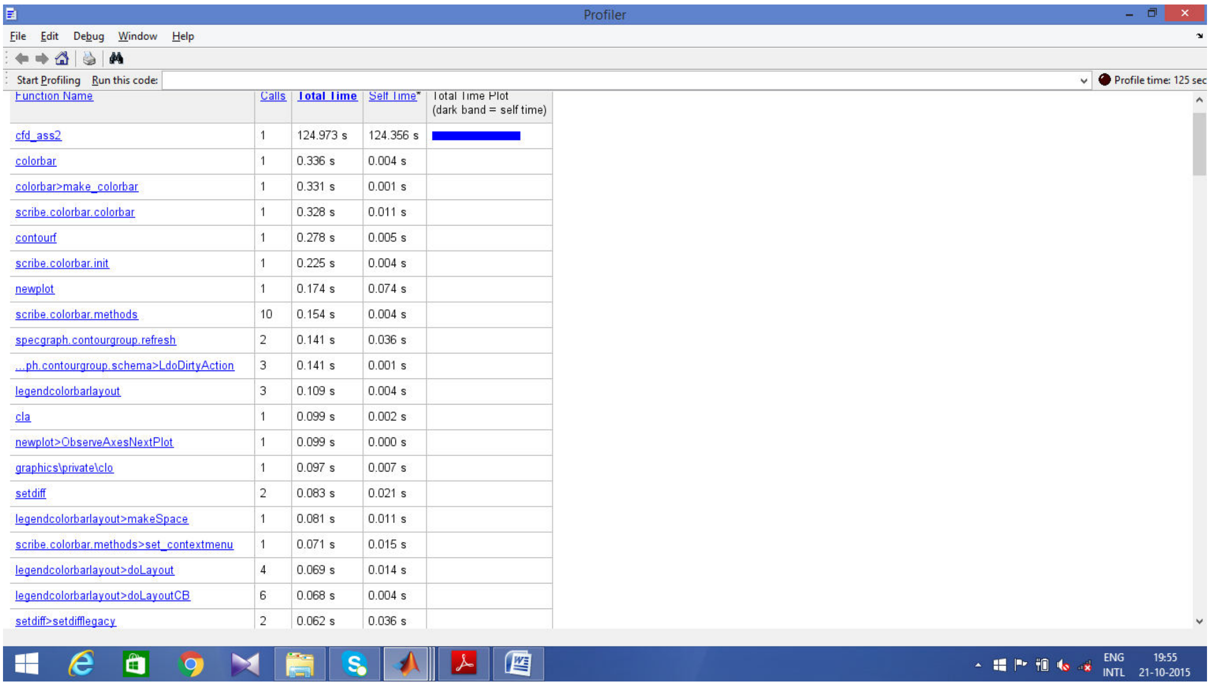
T3=400 °C

T4=200 °C



Over-relaxation parameter = 1.5

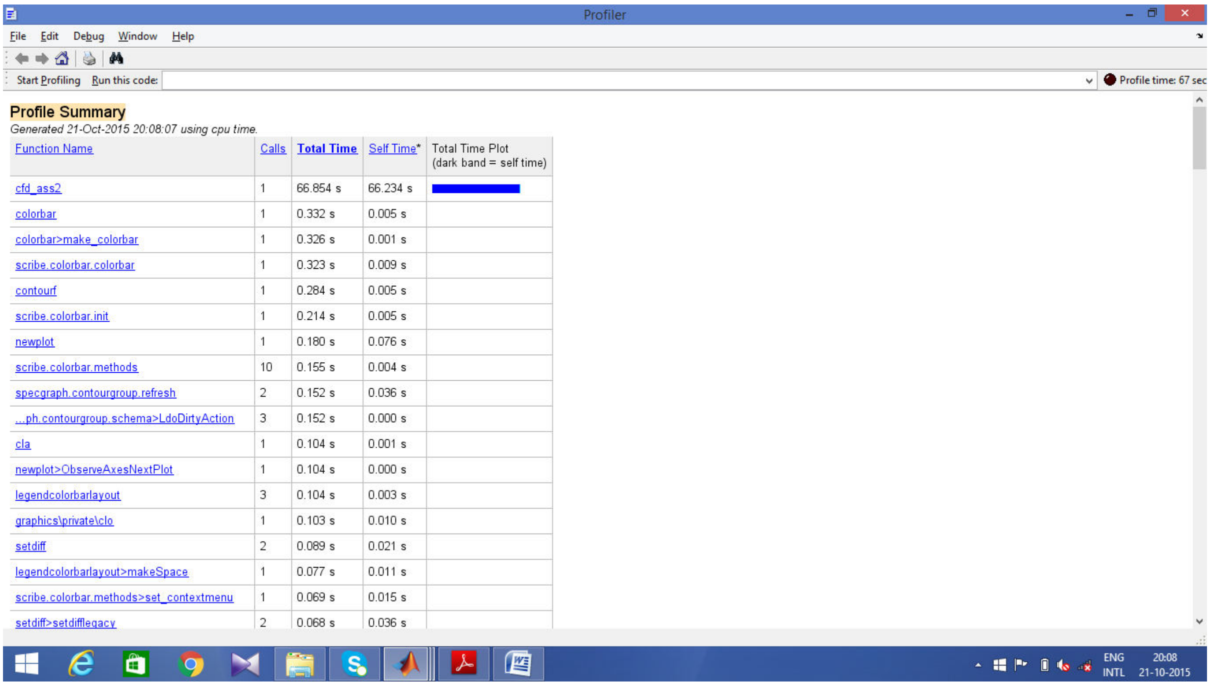
Total time taken for convergence: 124.973 seconds



Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
cfd_ass2	1	124.973 s	124.356 s	
colorbar	1	0.336 s	0.004 s	
colorbar>make_colorbar	1	0.331 s	0.001 s	
scribe.colorbar.colorbar	1	0.328 s	0.011 s	
contourf	1	0.278 s	0.005 s	
scribe.colorbar.init	1	0.225 s	0.004 s	
newplot	1	0.174 s	0.074 s	
scribe.colorbar.methods	10	0.154 s	0.004 s	
specgraph.contourgroup.refresh	2	0.141 s	0.036 s	
...ph.contourgroup.schema>LdoDirtyAction	3	0.141 s	0.001 s	
legendcolorbarlayout	3	0.109 s	0.004 s	
cla	1	0.099 s	0.002 s	
newplot>ObserveAxesNextPlot	1	0.099 s	0.000 s	
graphics\private\clo	1	0.097 s	0.007 s	
setdiff	2	0.083 s	0.021 s	
legendcolorbarlayout>makeSpace	1	0.081 s	0.011 s	
scribe.colorbar.methods>set_contextmenu	1	0.071 s	0.015 s	
legendcolorbarlayout>doLayout	4	0.069 s	0.014 s	
legendcolorbarlayout>doLayoutCB	6	0.068 s	0.004 s	
setdiff>setdifflegacy	2	0.062 s	0.036 s	

Over-relaxation parameter = 1.75

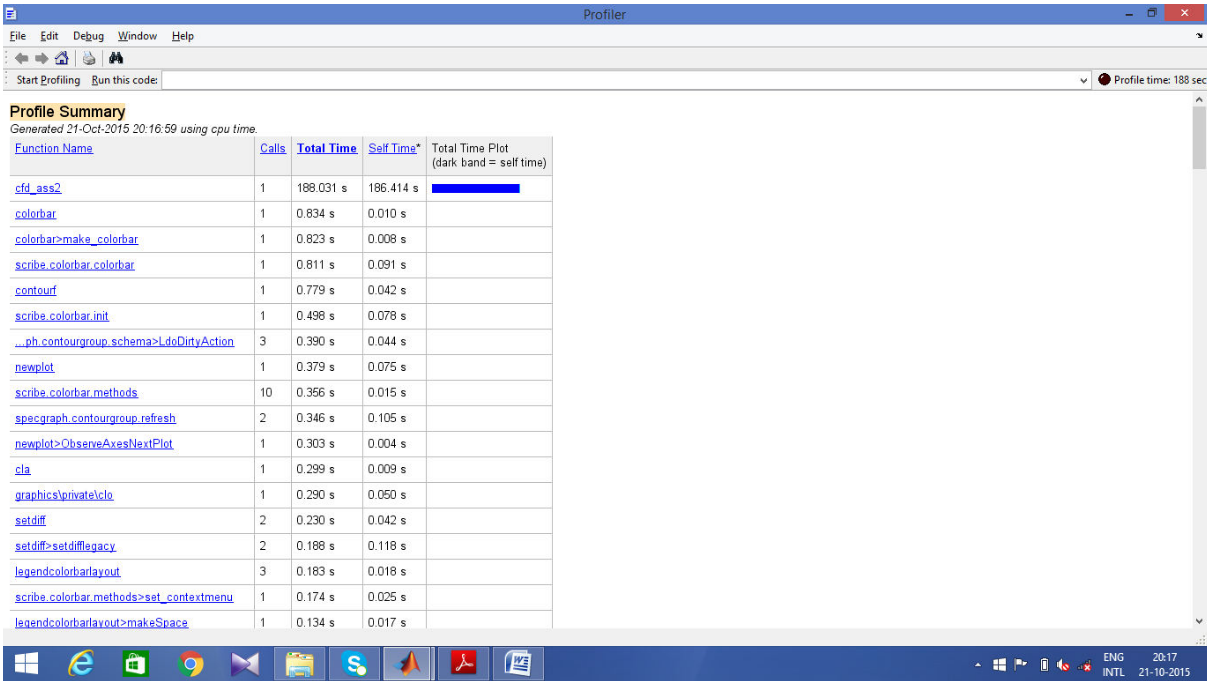
Total time taken for convergence: 66.854 seconds



Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
cfd_ass2	1	66.854 s	66.234 s	
colorbar	1	0.332 s	0.005 s	
colorbar>make_colorbar	1	0.326 s	0.001 s	
scribe.colorbar.colorbar	1	0.323 s	0.009 s	
contourf	1	0.284 s	0.005 s	
scribe.colorbar.init	1	0.214 s	0.005 s	
newplot	1	0.180 s	0.076 s	
scribe.colorbar.methods	10	0.155 s	0.004 s	
specgraph.contourgroup.refresh	2	0.152 s	0.036 s	
...ph.contourgroup.schema>LdoDirtyAction	3	0.152 s	0.000 s	
cla	1	0.104 s	0.001 s	
newplot>ObserveAxesNextPlot	1	0.104 s	0.000 s	
legendcolorbarlayout	3	0.104 s	0.003 s	
graphics\private\clo	1	0.103 s	0.010 s	
setdiff	2	0.089 s	0.021 s	
legendcolorbarlayout>makeSpace	1	0.077 s	0.011 s	
scribe.colorbar.methods>set_contextmenu	1	0.069 s	0.015 s	
setdiff>setdifflegacy	2	0.068 s	0.036 s	

Over-relaxation parameter = 1.25

Total time taken for convergence: 188.031 seconds

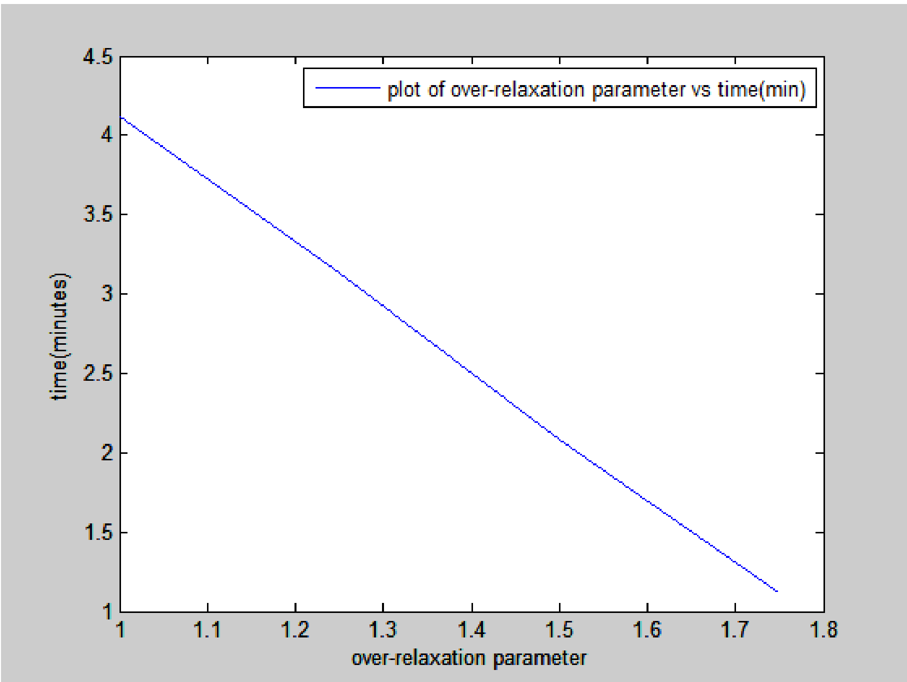


Profile Summary

Generated 21-Oct-2015 20:16:59 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
cfd_ass2	1	188.031 s	186.414 s	
colorbar	1	0.834 s	0.010 s	
colorbar>make_colorbar	1	0.823 s	0.008 s	
scribe.colorbar.colorbar	1	0.811 s	0.091 s	
contourf	1	0.779 s	0.042 s	
scribe.colorbar.init	1	0.498 s	0.078 s	
...ph.contourgroup.schema>LdoDirtyAction	3	0.390 s	0.044 s	
newplot	1	0.379 s	0.075 s	
scribe.colorbar.methods	10	0.356 s	0.015 s	
specgraph.contourgroup.refresh	2	0.346 s	0.105 s	
newplot>ObserveAxesNextPlot	1	0.303 s	0.004 s	
cla	1	0.299 s	0.009 s	
graphics\private\clo	1	0.290 s	0.050 s	
setdiff	2	0.230 s	0.042 s	
setdiff>setdifflegacy	2	0.188 s	0.118 s	
legendcolorbarlayout	3	0.183 s	0.018 s	
scribe.colorbar.methods>set_contextmenu	1	0.174 s	0.025 s	
legendcolorbarlayout>makeSpace	1	0.134 s	0.017 s	

Inferences and Discussions



The above plot shows that increasing the over-relaxation parameter decreases the computational time.

Since the appropriate over-relaxation parameter has to be determined by a trial and error procedure, the above graph of over-relaxation parameter vs. time shows that the convergence time decreases as the parameter is increased. From the above graphical variation, any appropriate value of over-relaxation parameter can be chosen based on the time constraints.

For the same convergence criteria for all cases, computational time taken by each method:

Point Gauss Siedel method - 246.987 sec (Case 1)

Successive Over-relaxation - 66.854sec (for relaxation parameter = 1.75)

Line Gauss Siedel method - 14.603 sec (Case 1)

This shows that the Line Gauss Siedel method achieves a less computational time than the other 2 methods for the same convergence criteria. Hence it is the most widely used method for solving multi-dimensional problems.

Future Scope

Though better numerical iterative methods for solving linear algebraic equations are available such as Multigrid methods and red block variation of Gauss Siedel method, they have not been investigated due to lack of time.

Appendix - Program codes in MATLAB

Part - I

Code for steady state heat conduction using Point Gauss Siedel method

```
mat=zeros(2000,200);
value=zeros(1,2000);
disp('steady state heat conduction problem');
m=input('enter the number of nodes in the x direction');
n=input('enter the number of nodes in y direction');
t1=input('enter the temperature of left wall');
t2=input('enter the temperature of right wall');
t3=input('enter the temperature of bottom wall');
t4=input('enter the temperature of top wall');
kth=input(' enter the thermal conductivity of material');
h=input('enter the value of  volumetric heat addition ');
if m==" "
   m=20;
elseif n==" "
   n=20;
elseif t1==" "
   t1=300;
elseif t2==" "
   t2=100;
elseif t3==" "
   t3=400;
elseif t4==" "
   t4=200;
end
t=zeros(m,n);
t_old=zeros(m*n,1);          %t_old refers to the temperatures in the previous iteration
t_new=zeros(m*n,1);          %t_new refers to the temperatures in the new iteration
deltax=1/m;
deltay=1/n;
```

```
w1=1/(deltax*deltax);
w2=1/(deltay*deltay);
for i=1:m
  for j=1:n      %initialising the coefficients ap,ae,an,as,aw and source term to all nodes obtained by fvm
    if i==1
      if j==1
        ap=-3*(w1+w2);
        ae=w1;
        an=w2;
        s=2*t1*w1+2*t3*w2 + (h/k);
        value(1)=-s;
        for k=1:m
          for l=1:n
            if k==1 && l==1
              mat(1,1)=ap;
            elseif k==2 && l==1
              mat(1,n+1)=an;
            elseif k==1 && l==2
              mat(1,2)=ae;
            end
          end
        end
      elseif j==n
        ap=-3*(w1+w2);
        aw=w1;
        an=w2;
        s=2*t2*w1+2*t3*w2+ (h/k);
        value(j)=-s;
        for k=1:m
          for l=1:n
            if k==1 && l==n
              mat(n,n)=ap;
```

```
            elseif k==1 && l==n-1
                mat(n,n-1)=aw;
            elseif k==2 && l==n
                mat(n,2*n)=an;
            end
        end
    end
  else
    ap=-(2*w1+3*w2);
    s=2*t3*w2+ (h/k);
    ae=w1;
    aw=w1;
    an=w2;
    value(j)=-s;
    for k=1:m
        for l=1:n
                if k==1 && l==j
                    mat(j+(i-1)*n,l+(k-1)*n)=ap;
                elseif k==1 && l==j-1
                    mat(j+(i-1)*n,l+(k-1)*n)=aw;
                elseif k==1 && l==j+1
                    mat(j+(i-1)*n,l+(k-1)*n)=ae;
                elseif k==2 && l==j
                    mat(j+(i-1)*n,l+(k-1)*n)=an;
                end
        end
    end
  end
elseif i==m
  if j==1
    ap=-3*(w2+w1);
    as=w2;
```

```
    ae=w1;
    s=2*(t1*w1+t4*w2) + (h/k);
    value(j+(i-1)*n)=-s;
    for k=1:m
        for l=1:n
            if k==m && l==1
                mat(j+(i-1)*n,l+(k-1)*n)=ap;
            elseif k==m-1 && l==1
                mat(j+(i-1)*n,l+(k-1)*n)=as;
            elseif k==m && l==2
                mat(j+(i-1)*n,l+(k-1)*n)=ae;
            end
        end
    end
elseif j==n
    ap=-3*(w2+w1);
    as=w2;
    aw=w1;
    s=2*(t2*w1+t4*w2+ (h/k));
    value(j+(i-1)*n)=-s;
    for k=1:m
        for l=1:n
            if k==m && l==n
                mat(j+(i-1)*n,l+(k-1)*n)=ap;
            elseif k==m && l==n-1
                mat(j+(i-1)*n,l+(k-1)*n)=aw;
            elseif k==m-1 && l==n
                 mat(j+(i-1)*n,l+(k-1)*n)=as;
            end
        end
    end
else
```

```
        ap=-(2*w1+3*w2);
        s=2*w2*t4+ (h/k);
        ae=w1;
        aw=w2;
        value(j+(i-1)*n)=-s;
        for k=1:m
          for l=1:n
              if k==m && l==j
                mat(j+(i-1)*n,l+(k-1)*n)=ap;
              elseif k==m-1 && l==j
                 mat(j+(i-1)*n,l+(k-1)*n)=as;
              elseif k==m && l==j-1
                 mat(j+(i-1)*n,l+(k-1)*n)=aw;
              elseif k==m && l==j+1
                 mat(j+(i-1)*n,l+(k-1)*n)=ae;
              end
          end
        end
    end
elseif j==1
    if i~=1 && i~=m
        ap=-(3*w1+2*w2);
        s=2*t1*w1+ (h/k);
        ae=w1;
        an=w2;
        as=w2;
        value(j+(i-1)*n)=-s;
        for k=1:m
          for l=1:m
            if k==i && l==j
              mat(j+(i-1)*n,l+(k-1)*n)=ap;
            elseif k==i-1 && l==j
```

```
                    mat(j+(i-1)*n,l+(k-1)*n)=as;
                elseif k==i+1 && l==j
                    mat(j+(i-1)*n,l+(k-1)*n)=an;
                elseif k==i && l==j+1
                   mat(j+(i-1)*n,l+(k-1)*n)=ae;
                end
            end
        end
    end
elseif j==n
     if i~=1 && i~=m
        ap=-(3*w1+2*w2);
        s=2*t2*w1+ (h/k);
        aw=w1;
        an=w2;
        as=w2;
        value(j+(i-1)*n)=-s;
        for k=1:m
            for l=1:n
                if k==i && l==j
                    mat(j+(i-1)*n,l+(k-1)*n)=ap;
                elseif k==i-1 && l==j
                    mat(j+(i-1)*n,l+(k-1)*n)=as;
                 elseif k==i+1 && l==j
                    mat(j+(i-1)*n,l+(k-1)*n)=an;
                elseif k==i && l==j-1
                    mat(j+(i-1)*n,l+(k-1)*n)=aw;
                end
            end
        end
    end
else
```

```
            ap=-2*(w1+w2);
            ae=w1;
            aw=w1;
            an=w2;
            as=w2;
            s=0.0+ (h/k);
            value(j+(i-1)*n)=-s;
            for k=1:m
                for l=1:n
                    if k==i && l==j
                         mat(j+(i-1)*n,l+(k-1)*n)=ap;
                    elseif k==i-1 && l==j
                              mat(j+(i-1)*n,l+(k-1)*n)=as;
                    elseif k==i+1 && l==j
                              mat(j+(i-1)*n,l+(k-1)*n)=an;
                    elseif k==i && l==j-1
                              mat(j+(i-1)*n,l+(k-1)*n)=aw;
                    elseif k==i && l==j+1
                        mat(j+(i-1)*n,l+(k-1)*n)=ae;
                    end
                end
            end
        end
    end
end
iter=0;
for i=1:m
    for j=1:n
     t_old(j+(i-1)*n,1)=min(t1,t3);
    end
end
diff=40;
```

```
n_d=1;
while diff>1e-02              %convergence criteria
    iter=n_d;
for i=1:m*n
 sum=0.0;
    if i==1
        iter=1*n;
         for k=1:m*n
            if k~=i
        sum=sum+mat(i,k)*t_old(k);
            end
        end
        t_new(i)=(value(i)-sum)/mat(1,1);
    diff=abs(t_old(1)-t_new(1));        %calculating the difference between the successive iterations
    t_old(1)=t_new(1);        %the temperature calculated in the current iteration is stored in t_old
    else
         for k=1:m*n
             if k~=i
                 sum=sum+mat(i,k)*t_old(k);
             end
         end
        t_new(i)=(value(i)-1*sum)/mat(i,i);
     end
     diff1=abs(t_old(i)-t_new(i));
     t_old(i)=t_new(i);
     diff=max(diff,diff1);     %finding the maximum difference between the t_old and t_new
end
    n_d=n_d+1;     %increasing the iteration number
end
for s=1:m
    for q=1:n
        t(s,q)=t_new(q+(s-1)*n);          %storing the temperatures of all nodes in a single matrix t
```

```
    end
end
[x,y]=meshgrid(1:m,1:n);
contourf(x,y,t);                  %generating the temperature contours by using the matrix t
colorbar;
```

Code for Successive Over-relaxation (in conjunction with point gauss siedel method)

```
while diff>1e-02  %convergence criteria
    iter=n_d;
for i=1:m*n     %moving through all nodal points and finding temperature
 sum=0.0;
    if i==1
        iter=1*n;
         for k=1:m*n
            if k~=i
        sum=sum+mat(i,k)*t_old(k);
            end
        end
        t_new_d(i)=(value(i)-sum)/mat(1,1);
        ra=mat(1,1)/alpha;
      t_new(i)=((t_new_d(i)*mat(1,1))+(rat*mat(1,1)*t_old(1)))/ra;          %increasing the calculated temperature
by a factor
      diff=abs(t_old(1)-t_new(1));
    t_old(1)=t_new(1);
    else
         for k=1:m*n
             if k~=i
                 sum=sum+mat(i,k)*t_old(k);
             end
         end
        t_new_d(i)=(value(i)-1*sum)/mat(i,i);
        ra=mat(i,i)/alpha;
        t_new(i)=((t_new_d(i)*mat(i,i))+(rat*mat(i,i)*t_old(i)))/ra;
```

```
    end
    diff1=abs(t_old(i)-t_new(i));
    t_old(i)=t_new(i);
    diff=max(diff,diff1);
end
   n_d=n_d+1;
end
```

Code for Line Gauss Siedel method (LGS)

```
for i=1:20        %1 iteration involves 4 moves through x and y axis - briefly explained in literature section
[t_old,t_new,diff]=line_by_linex(t_old,t_new,1,1,m,m,n,value,mat,diff);
[t_old,t_new,diff]=line_by_linex(t_old,t_new,m,-1,1,m,n,value,mat,diff);
[t_old,t_new,diff]=line_by_liney(t_old,t_new,1,1,n,m,n,value,mat,diff);
[t_old,t_new,diff]=line_by_liney(t_old,t_new,n,-1,1,m,n,value,mat,diff);
iter=iter+1;
end
```

Code for line-by-line gauss siedel function along x-direction(user-defined function used along with Line gauss siedel method)

```
function [ t_old,t_new,diff ] = line_by_linex( t_old,t_new,z1,z2,z3,m,n,value,mat,diff )
%this function implements the line by line gauss siedel method along the
%x-direction
mat1=zeros(n,n);
value1=zeros(1,n);
for i=z1:z2:z3
  if i==1
    for j=1:n
        if j==1
        value1(j)= (mat(i,n+j)*t_old(j+(i)*n,1)-value(j+(i-1)*n));
        mat1(j,j+1)=mat(i,j+1);
        mat1(j,j)=mat(i,i);
        elseif j==n
            value1(j)=(mat(j,n+j)*t_old(j+(i)*n,1)-value(j+(i-1)*n));
            mat1(j,j-1)=mat(j+(i-1)*n,j-1);
            mat1(j+(i-1)*n,j)=mat(j+(i-1)*n,j);
```

```
        else
            value1(j)=(mat(j,n+j)*t_old(j+i*n,1)-value(j+(i-1)*n));
            mat1(j,j-1)=mat(j,j-1);
            mat1(j,j+1)=mat(j,j+1);
             mat1(j+(i-1)*n,j)=mat(j+(i-1)*n,j);
        end
        t_new(1:n,1)=tdma(mat1,-value1,n);
        diff=abs(t_new(1,1)-t_old(1,1));
        for k=2:n
            diff1=abs(t_new(k,1)-t_old(k,1));
            diff=max(diff,diff1);
        end
        t_old(1:n,1)=t_new(1:n,1);
      end
elseif i==m
    for j=1:n
            if j==1
                value1(j)=(mat(j+(i-1)*n,j+(i-2)*n)*t_old(j+(i-2)*n,1)-value(j+(i-1)*n));
                mat1(j,j+1)=mat(j+(i-1)*n,j+(i-1)*n+1);
                mat1(j,j)=mat(j+(i-1)*n,j+(i-1)*n);
             elseif j==n
                value1(j)=(mat(j+(i-1)*n,j+(i-2)*n)*t_old(j+(i-2)*n,1)-value(j+(i-1)*n));
                mat1(j,j-1)=mat(j+(i-1)*n,j+(i-1)*n-1);
                mat1(j,j)=mat(j+(i-1)*n,j+(i-1)*n);
            else
                value1(j)=(mat(j+(i-1)*n,j+(i-2)*n)*t_old(j+(i-2)*n,1)-value(j+(i-1)*n));
                mat1(j,j-1)=mat(j+(i-1)*n,j+(i-1)*n-1);
                mat1(j,j+1)=mat(j+(i-1)*n,j+(i-1)*n+1);
                mat1(j,j)=mat(j+(i-1)*n,j+(i-1)*n);
            end
          t_new((m-1)*n+1:m*n,1)=tdma(mat1,-value1,n);
          for k=(m-1)*n+1:m*n
```

```
            diff1=abs(t_new(k,1)-t_old(k,1));
            diff=max(diff,diff1);
          end
          t_old((m-1)*n+1:m*n,1)= t_new((m-1)*n+1:m*n,1);
    end
  else
    for j=1:m
        if j==1
          value1(j)=(mat(j+(i-1)*n,j+(i-2)*n)*t_old(j+(i-2)*n,1)-value(j+(i-1)*n)+mat(j+(i-
1)*n,j+i*n)*t_old(j+i*n,1));
          mat1(j,j+1)=mat(j+(i-1)*n,j+(i-1)*n+1);
          mat1(j,j)=mat(j+(i-1)*n,j+(i-1)*n);
        elseif j==m
          value1(j)=(mat(j+(i-1)*n,j+(i-2)*n)*t_old(j+(i-2)*n,1)-value(j+(i-1)*n)+mat(j+(i-
1)*n,j+i*n)*t_old(j+i*n,1));
          mat1(j,j-1)=mat(j+(i-1)*n,j+(i-1)*n-1);
          mat1(j,j)=mat(j+(i-1)*n,j+(i-1)*n);
        else
          value1(j)=(mat(j+(i-1)*n,j+(i-2)*n)*t_old(j+(i-2)*n,1)-value(j+(i-1)*n)+mat(j+(i-
1)*n,j+i*n)*t_old(j+i*n,1));
          mat1(j,j-1)=mat(j+(i-1)*n,j+(i-1)*n-1);
          mat1(j,j+1)=mat(j+(i-1)*n,j+(i-1)*n+1);
          mat1(j,j)=mat(j+(i-1)*n,j+(i-1)*n);
        end
         t_new((i-1)*n+1:(i)*n,1)=tdma(mat1,-value1,n);
         for k=(i-1)*n+1:(i)*n
            diff1=abs(t_new(k,1)-t_old(k,1));
            diff=max(diff,diff1);
         end
         t_old((i-1)*n+1:(i)*n,1)=t_new((i-1)*n+1:(i)*n,1);
    end
  end
end
end
```

Code for line-by-line gauss siedel function along y-direction(user-defined function used along with Line gauss siedel method)

```
function [ t_old,t_new,diff] = line_by_liney( t_old,t_new,z1,z2,z3,m,n,value,mat,diff )

%this function implements the line by line gauss siedel method along the

%y-direction

mat1=zeros(n,n);

value1=zeros(1,n);

for j=z1:z2:z3

   if j==1

      for i=1:m

         if i==1

            value1(i)= (mat(i,2)*t_old(2,1)-value(1));

            mat1(i,i+1)=mat(j,i+n);

            mat1(j,j)=mat(j,j);

         elseif i==m

            value1(i)=(mat(j+(i-1)*n,j+(i-1)*n+1)*t_old(j+(i-1)*n+1,1)-value(j+(i-1)*n));

            mat1(i,i-1)=mat(j+(i-1)*n,j+(i-2)*n);

            mat1(i,i)=mat(j+(i-1)*n,j+(i-1)*n);

         else

            value1(i)=(mat(j+(i-1)*n,j+(i-1)*n+1)*t_old(j+(i-1)*n+1,1)-value(j+(i-1)*n));

            mat1(i,i+1)=mat(j+(i-1)*n,j+(i)*n);


 mat1(i,i)=mat(j+(i-1)*n,j+(i-1)*n);

            mat1(i,i-1)=mat(j+(i-1)*n,j+(i-2)*n);

  end

      end

         t_new(1:n:(m-1)*n+1,1)=tdma(mat1,-value1,m);

         for k=1:n:(m-1)*n+1

            diff1=abs(t_new(k,1)-t_old(k,1));

            diff=max(diff,diff1);

         end

         t_old(1:n:(m-1)*n+1,1)=t_new(1:n:(m-1)*n+1,1);

   elseif j==m
```

```
        for i=1:m
            if i==1
                value1(i)=(mat(j+(i-1)*n,j+(i-1)*n-1)*t_old(j+(i-1)*n-1,1)-value(j+(i-1)*n));
                mat1(i,i)=mat(j+(i-1)*n,j+(i-1)*n);
                mat1(i,i+1)=mat(j+(i-1)*n,j+i*n);
            elseif i==m
                value1(i)=(mat(j+(i-1)*n,j+(i-1)*n-1)*t_old(j+(i-1)*n-1,1)-value(j+(i-1)*n));
                mat1(i,i-1)=mat(j+(i-1)*n,j+(i-2)*n);
                mat1(i,i)=mat(j+(i-1)*n,j+(i-1)*n);
            else
                value1(i)=(mat(j+(i-1)*n,j+(i-1)*n-1)*t_old(j+(i-1)*n-1,1)-value(j+(i-1)*n));
                mat1(i,i-1)=mat(j+(i-1)*n,j+(i-2)*n);
                mat1(i,i)=mat(j+(i-1)*n,j+(i-1)*n);
                mat1(i,i+1)=mat(j+(i-1)*n,j+i*n);
            end
        end
            t_new(m:m:m*n,1)=tdma(mat1,-value1,m);
            for k=m:m:m*n
                diff1=abs(t_new(k,1)-t_old(k,1));
                diff=max(diff,diff1);
            end
            t_old(m:m:m*n,1)=t_new(m:m:m*n,1);
    else
         for i=1:m
            if i==1
                value1(i)=(mat(j+(i-1)*n,j+(i-1)*n-1)*t_old(j+(i-1)*n-1,1)-value(j+(i-1)*n)+mat(j+(i-1)*n,j+(i-
1)*n+1)*t_old(j+(i-1)*n+1,1));
                mat1(i,i)=mat(j+(i-1)*n,j+(i-1)*n);
                mat1(i,i+1)=mat(j+(i-1)*n,j+i*n);
            elseif i==m
                value1(i)=(mat(j+(i-1)*n,j+(i-1)*n-1)*t_old(j+(i-1)*n-1,1)-value(j+(i-1)*n)+mat(j+(i-1)*n,j+(i-
1)*n+1)*t_old(j+(i-1)*n+1,1));
                mat1(i,i-1)=mat(j+(i-1)*n,j+(i-2)*n);
```

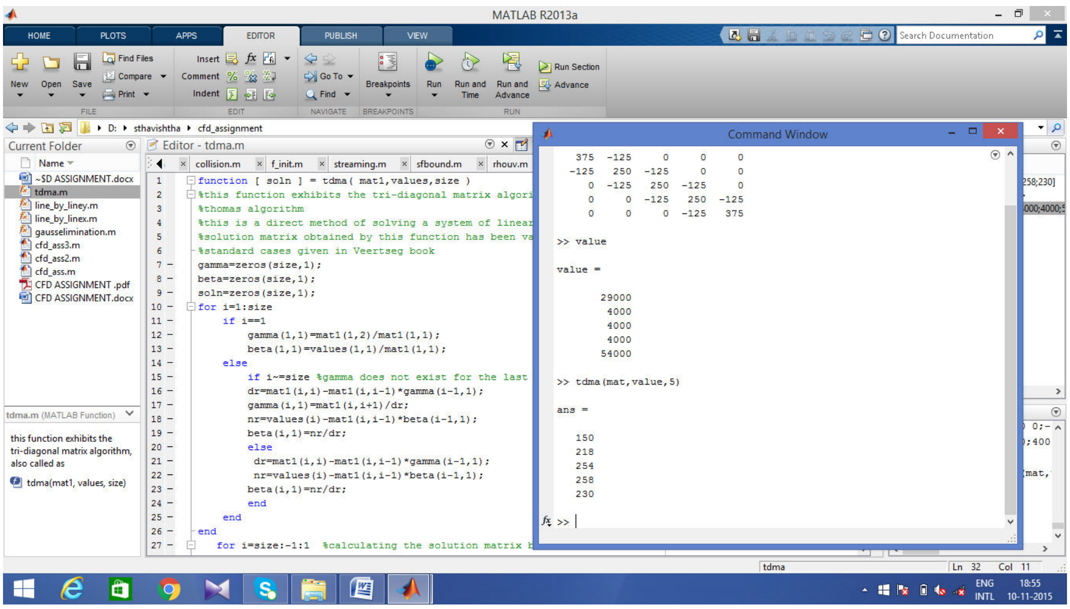
```
            mat1(i,i)=mat(j+(i-1)*n,j+(i-1)*n);
          else
            value1(i)=(mat(j+(i-1)*n,j+(i-1)*n-1)*t_old(j+(i-1)*n-1,1)-value(j+(i-1)*n)+mat(j+(i-1)*n,j+(i-1)*n+1)*t_old(j+(i-1)*n+1,1));
            mat1(i,i-1)=mat(j+(i-1)*n,j+(i-2)*n);
            mat1(i,i)=mat(j+(i-1)*n,j+(i-1)*n);
            mat1(i,i+1)=mat(j+(i-1)*n,j+i*n);
          end
        end
   end
          t_new(j:m:j+(m-1)*m,1)=tdma(mat1,-value1,m);
          for k=j:m:j+(m-1)*m
            diff1=abs(t_new(k,1)-t_old(k,1));
            diff=max(diff,diff1);
          end
          t_old(j:m:j+(m-1)*m,1)=t_new(j:m:j+(m-1)*m,1);
end
end
```

Part - II

Code for TDMA user defined function (This user defined function is used in Line Gauss Siedel method)

```
function [ soln ] = tdma( mat1,values,size )
%this function exhibits the tri-diagonal matrix algorithm, also called as thomas algorithm
%this is a direct method of solving a system of linear equations and the
%solution matrix obtained by this function has been validated with some
%standard cases given in Vertseeg book
gamma=zeros(size,1);
beta=zeros(size,1);
soln=zeros(size,1);
for i=1:size
   if i==1
      gamma(1,1)=mat1(1,2)/mat1(1,1);
```

```
        beta(1,1)=values(1,1)/mat1(1,1);
    else
        if i~=size %gamma does not exist for the last row of elements
        dr=mat1(i,i)-mat1(i,i-1)*gamma(i-1,1);
        gamma(i,1)=mat1(i,i+1)/dr;
        nr=values(i,1)-mat1(i,i-1)*beta(i-1,1);
        beta(i,1)=nr/dr;
        else
         dr=mat1(i,i)-mat1(i,i-1)*gamma(i-1,1);
         nr=values(i,1)-mat1(i,i-1)*beta(i-1,1);
        beta(i,1)=nr/dr;
        end
    end
end
   for i=size:-1:1  %calculating the solution matrix by backward sweeping
       if i==size
           soln(size,1)=beta(size,1);
       else
           soln(i,1)=beta(i,1)-gamma(i,1)*soln(i+1,1);
       end
end
end
```



Code for gauss elimination user-defined function

```
function [soln] = gausselimination( mat1,values,size )

%this function uses the direct method-gaussian elimination to solve a set

%of linear algebraic equations when a matrix is input

soln=zeros(size,1);

beta=values;

for i=1:size

  if i==1            %for the first row of matrix (special case)

    e=mat1(1,1);

  for j=1:size

    mat1(i,j)=mat1(i,j)/e; %dividing the entire row elements by first element

  end

  beta(1)=beta(1)/e;

  for k=i+1:size

    c=mat1(k,1);

    for j=1:size

    mat1(k,j)=mat1(k,j)-c*mat1(1,j); %subtracting the elements of the next rows of the first column from the
first element

    end

    beta(k)=beta(k)-c*beta(1);

  end

   elseif i~=size %for any other row

    f=mat1(i,i);

    for j=1:size

      mat1(i,j)=mat1(i,j)/f;

    end

    beta(i)=beta(i)/f;

    for k=i+1:size

      d=mat1(k,i);

      for j=1:size

        mat1(k,j)=mat1(k,j)-d*mat1(i,j);

      end

      beta(k)=beta(k)-d*beta(i);
```

```
        end
    else
        beta(i)=beta(i)/mat1(i,i);
        mat1(i,i)=1;
    end
end
for m=size:-1:1
    if m==size
        soln(size)=beta(size);
    else
        sum=0.0;
        for g=m+1:size
            sum=sum+mat1(m,g)*soln(g);
        end
        soln(m)=(beta(m,1)-sum)/mat1(m,m);   %calculating the solution by backward sweeping
    end
  end
```