Project 2: MECH 6371
Computational Thermal and Fluid Science

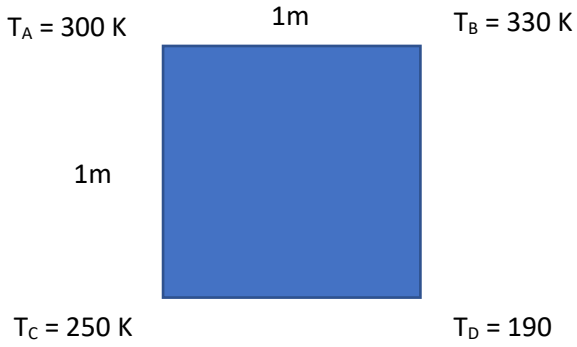
Yajat Pandya
UTD ID: 2021440761

1. Introduction

The problem given is a 2-D transient conduction problem of an Aluminum plate with dimensions of 1m x 1m. The corners of the plate are externally forced to be at a certain temperature with the edges having a linear gradient between the corners.

Initial condition: T (x, y, t=0) = 200 K;

At steady – state:



Linear temperature increase along the edges of the plate.

Required: (All with and without factorization)

1. Steady-state temperature distribution
2. Temperate distribution at t = 100, 500 and 3000 s
3. Comparison with simple explicit scheme with same grid size (developed in project #1)

2. Methodology

Governing equation for the given 2-D plate:

$$\frac{1}{\alpha}\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}$$

Let T (x, y, t) – temperature at any (x, y, t) spatio-temporal coordinate on the plate. For the material Aluminum, the thermal diffusivity is $\alpha = 9.7 \times 10^{-5}\ m^2/s$. It is required to formulate a **Crank – Nicholson** scheme as a finite difference scheme for solving the second order PDE. That is,

$$\frac{{T_{i,j}}^{n+1} - {T_{i,j}}^{n}}{\Delta t} = \frac{\alpha}{2}\left(\frac{{T_{i+1,j}}^{n+1} - 2{T_{i,j}}^{n+1} + {T_{i-1,j}}^{n+1}}{(\Delta x)^2} + \frac{{T_{i+1,j}}^{n} - 2{T_{i,j}}^{n} + {T_{i-1,j}}^{n}}{(\Delta x)^2}\right) + \frac{\alpha}{2}\left(\frac{{T_{i,j+1}}^{n+1} - 2{T_{i,j}}^{n+1} + {T_{i,j-1}}^{n+1}}{(\Delta y)^2} + \frac{{T_{i,j+1}}^{n} - 2{T_{i,j}}^{n} + {T_{i,j-1}}^{n}}{(\Delta y)^2}\right)$$

2.1 Solving without factorization

This requires us to not have a spatial loop of iterating from node-to-node, but instead just have a time-loop and solve a (*n x n*) matrix to compute the temperature at all nodes simultaneously. We solve the implicit formulation with the help of system of equations:

$$\boldsymbol{Ax = B}$$

$$\boldsymbol{x = A^{-1}B}$$

This way, we create a time – loop inside which the temperature *inside* the whole plate will be calculating in a single operation for that timestep. As this method is unconditionally stable, we do not need to care about the dependence of Δt and Δx. Here each step can be computed by inverting a five-diagonal matrix in MATLAB for separate directions.

Here, A is the coefficient matrix with coefficients of all nodes (*n x n* nodes) for the implicit equation of all nodes, and B is the part of the equation which contains the values derived from the previous (n*th*) timestep i.e. the known part of the equation. These set of equations are solved by inverting the matrix A. In our case, we have the prescribed **Drichlet** boundary conditions at all edges of the plate, hence we need to solve the matrix of the dimension of (*n–2 x n–2*).

Dimensionally,

$$\dim(A) = (n-2)^2 \text{ x } (n-2)^2$$

$$\dim(B) = (n-2)^2 \text{ x } 1$$

$$\dim(x) = (n-2)^2 \text{ x } 1$$

It is important to note that the 2nd diagonals of A will have elements equal to zero at every (nx – 2) instances. This is because these two diagonals equate to the contribution from the (i – 1)th and (j – 1)th nodes which are the edge nodes at every (nx – 2) instances. Also note that for the nodes in contact with the edges, the RHS part – B(:) needs to have added a few terms from the LHS which are the temperature at the neighboring

edges. This contribution is already modeled in the creation of B in the if, elseif, else loop format which essentially categorize the points as edge points or corners and help compute B(:) for that node accordingly in lines #47 to #81 in the code#1 attached in appendix.

2.1 Solving with factorization

The Crank – Nicholson scheme can be reduced to the following approximation using simple mathematical operands:

$$\left(1-\frac{\alpha}{2}\Delta t\delta_x^2\right)\left(1-\frac{\alpha}{2}\Delta t\delta_y^2\right)\Delta u_{i,j} = \frac{1}{2}\alpha\Delta t\left[2\frac{u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n}{\Delta x^2} + 2\frac{u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n}{\Delta y^2}\right]$$

Here, the implicit equation is solved using two step approach with Alternating Direction Implicit (ADI) method. The first step solves the system of equations with four unknowns for the pseudo matrix $\Delta T^*(i,j)$ which is in turn used as the RHS part in the second step to get $\Delta T(i,j)$ using the same procedure. Here, $\Delta T(i,j)$ is the matrix of difference of temperature compared to the previous timestep.

$$\Delta T\,(i,j) = T\,(i,j,n+1) - T(i,j,n)$$

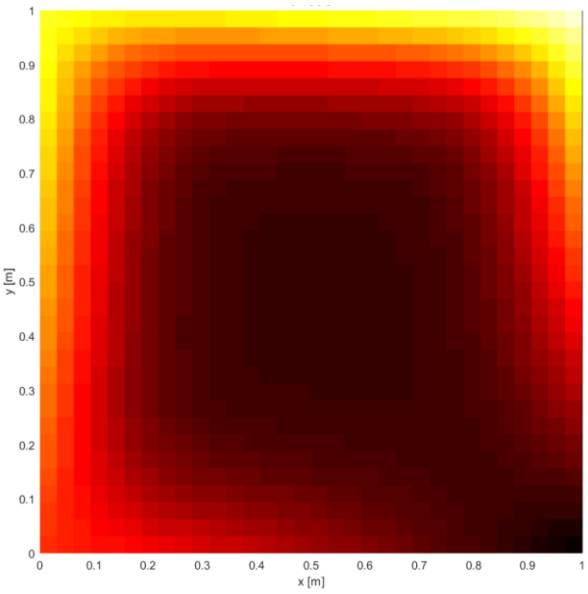
We hence calculate the temperature matrix for next timestep using two-step ADI approach which involves solving the system of equation $\boldsymbol{Ax = B}$ but the difference is A has three diagonals compared to five in the previous method. This is implemented in the code#2 in appendix of this report. The time loop starting from line #34 to #51 has two spatial subloops which comprehend to the above two step scheme each.

From the linear gradient boundary condition at the edges, it is important to note that for all $t > 0$,

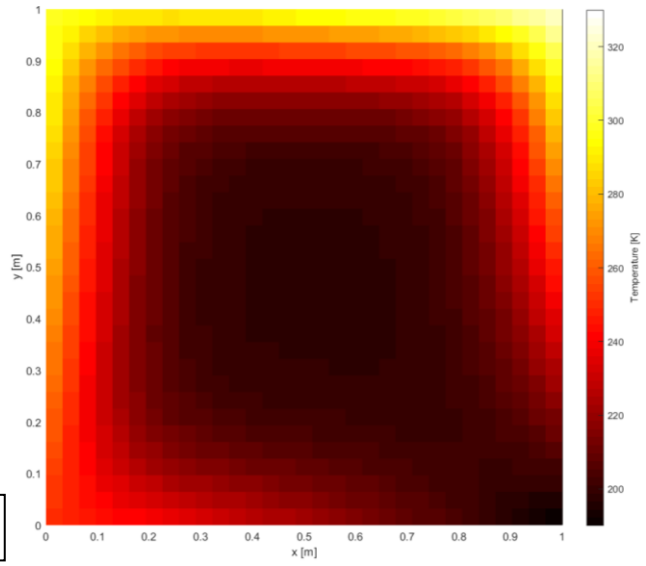
$$T(x,0,t) = T_C + (T_D - T_C)x$$
$$T(x,1,t) = T_A + (T_B - T_A)x$$
$$T(0,y,t) = T_C + (T_A - T_C)y$$
$$T(1,y,t) = T_D + (T_B - T_D)y$$

3. Results

The temperature plots for $t = 100$ s are shown in figure 1 for all three cases – simple explicit, CN without and with factorization. Number of points in each direction is **32** for all the results mentioned due to computational constraints.


y [m]
x [m]
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

(a)


y [m]
x [m]
Temperature [K]
200
220
240
260
280
300
320

(b)

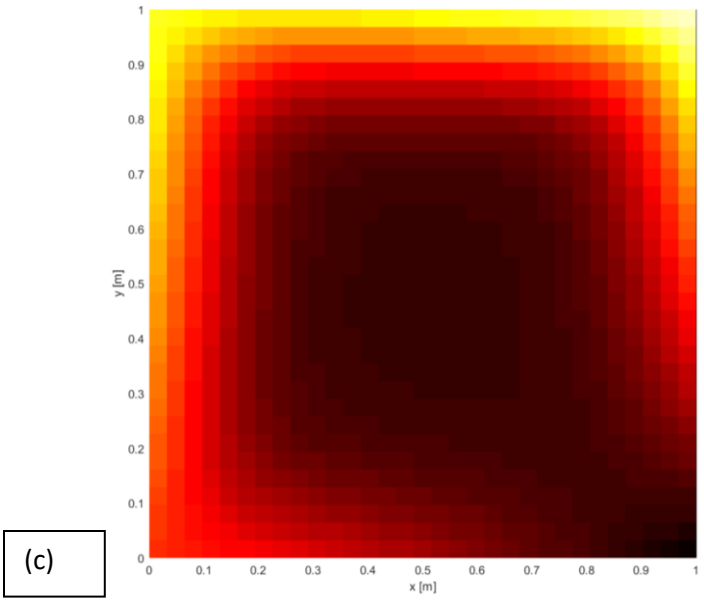
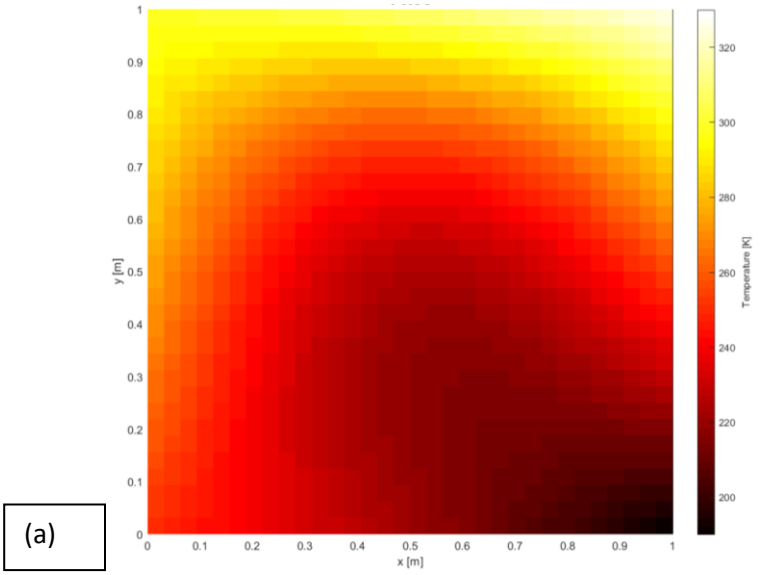


Fig. 1: Temperature plots at t = 100 s for (a) Simple explicit (b) CN without factorization (c) CN with factorization



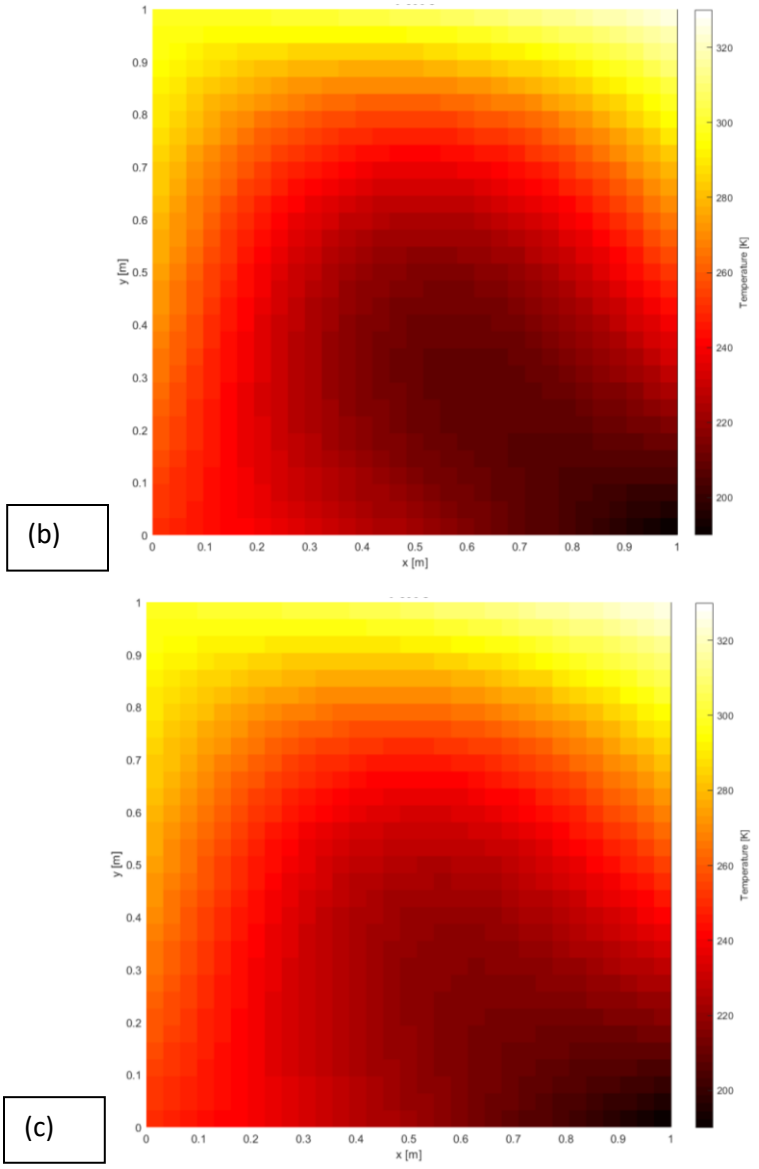
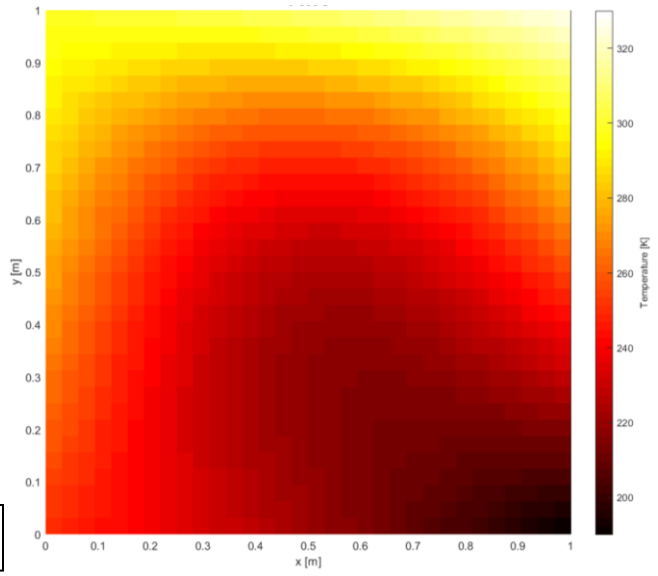
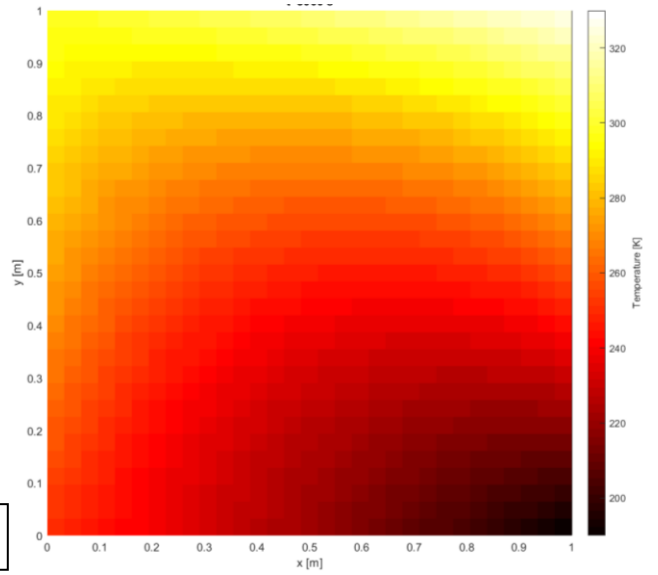


Fig. 2: Temperature plots at t = 500 s for (a) Simple explicit (b) CN without factorization (c) CN with factorization

(a)


y [m]
x [m]
Temperature [K]
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
200
220
240
260
280
300
320

(b)


y [m]
x [m]
Temperature [K]
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
200
220
240
260
280
300
320

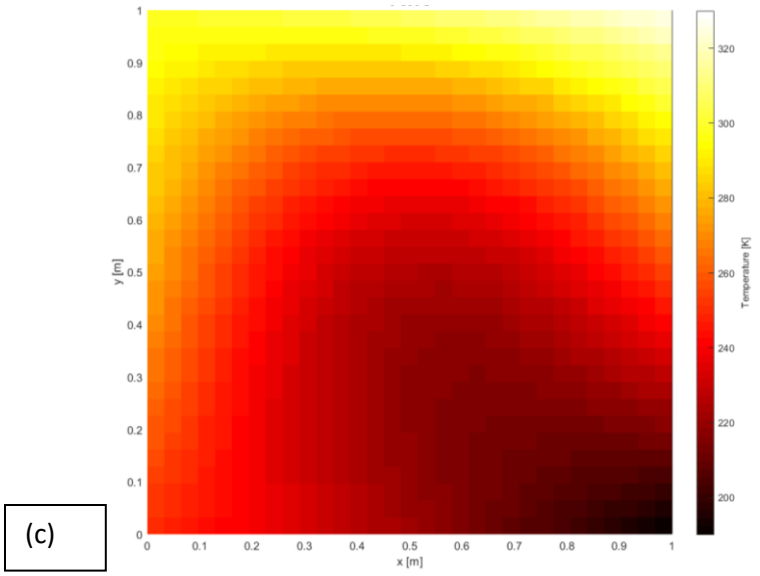


Fig. 3: Temperature plots at t = 3000 s for (a) Simple explicit (b) CN without factorization (c) CN with factorization

Similar plots are shown in figure 2 and 3 for t = 500 and 3000 s respectively.

The overall profiles have achieved convergence in accordance with the forced boundary conditions for the results shown in figure 2. This makes the profiles look extremely polarized from point D to B. Due to this, the rightmost edge, DB, of the plate has maximum notable temperature gradient. It is important to note that the maximum difference in the results is noted for t = 100 s and the minimum difference is noted near the steady – state time i.e. at t = 3000 s. This is due to the fact that the unconditionally stable scheme is perhaps unconditionally stable, but this doesn't mean it is more efficient in terms of the error order compared to the simple explicit scheme.

4. Conclusions

This project consisted of formulating, designing and implementing a numerical solver for a 2D conductive plate with prescribed initial and boundary conditions. The problem algorithm was developed using MATLAB with an implicit Crank – Nicholson finite difference scheme for the conductive heat equation. The CN scheme was formulated using with and without factorization scheme. In both the cases, the results were compared with those obtained from simple explicit scheme in project #1. The stability criteria was independent of the grid size and the time step size as the scheme is unconditionally stable.

It was noted that the two step ADI scheme for Crank – Nicholson took significantly simulation time than the other two schemes. This comes from the fact that it apparently takes lesser time for MATLAB to compute two 3-diagonal matrices than to compute one 5-diagonal matrix in the CN scheme without factorization. However, the results in all three cases show satisfactory correlation to each other. Moreover, it is important to note that for a 1 m plate, it takes approximately 3000 s to reach steady state. This is because Aluminum has a very low thermal diffusivity – meaning the heat effects would take significant time to affect the whole body compared to a highly conductive material like *Silver* or *Copper*.

It was thus possible to visualize the evolution of heat transfer in the 2D plate at every instant, the duration of which would be from the user input with all three types of finite difference schemes – simple explicit, CN without factorization and CN without factorization. The results show significant acceptance with each other and the CN scheme with factorization was the computationally fastest in terms of the simulation time required on the same machine with same flow parameters.

The following MATLAB codes were developed for this project:

CODE #1: CN scheme without factorization -

```
clear
close;

%% Initialization
dt=0.2; %time step in sec
tmax=3000; %simulation time limit
%NX=6; NY=6; %number of grid points including boundary points
%nx=NX-1; ny=NY-1;
nx=32;ny=32;
dx=1/(nx-1);dy=1/(ny-1);
al=9.7*10^(-5); %aluminum thermal diffusivity
r=al*dt/(dx*dx);
T(1:1:nx,1:1:ny,1:1:tmax/dt+1)=200; %Temperature spatio-temporal array
initialization

%% CFD - WITHOUT FACTORIZATION
%%% THIS CODE IS DESIGNED ONLY FOR EQUAL NUMBER OF X & Y NODES - BECAUSE OF
%%% THE SYMMETRY OF THE PROBLEM. THIS IS TO CLARIFY THAT THE STUDENT IS ABLE
TO SOLVE IT FOR UNEQUAL NUMBER AS WELL;
%%% THIS PROBLEM DID NOT REQUIRE SO IT IS NOT ATTEMPTED.

A1=diag((1+2*r).*ones((nx-2)^2,1));
b=(-r/2).*ones(-1+(nx-2)^2,1);
for i=1:size(b)
    if mod(i,nx-2)==0
        b(i)=0;
    end
end
A2=diag(b,1);
A3=diag(b,-1);
A4=diag((-r/2).*ones(-nx+2+(nx-2)^2,1),nx-2);
A5=diag((-r/2).*ones(-nx+2+(nx-2)^2,1),-nx+2);
A=A1+A2+A3+A4+A5;

for nt=2:1:tmax/dt+1
    % boundary conditions
    T(:,1,nt)=250+(190-250).*(0:1:nx-1).*dx;
    T(:,ny,nt)=300+(330-300).*(0:1:nx-1).*dx;
    T(1,:,nt)=250+(300-250).*(0:1:ny-1).*dy;
    T(nx,:,nt)=190+(330-190).*(0:1:ny-1).*dy;
end
%
for nt=2:1:tmax/dt+1
    % crank-nicholson - To compute T(x,y,nt)
    nt
    n=0;
```

```
    for x=2:1:nx-1
        for y=2:1:nx-1
            n=n+1;
            if x==2 && y~=2 && y~=(ny-1)
                B(n)=T(x,y,nt-1)+...
                    (r/2)*((T(x+1,y,nt-1)+T(x-1,y,nt-1)-2*T(x,y,nt-
1))+(T(x,y+1,nt-1)+T(x,y-1,nt-1)-2*T(x,y,nt-1)))...
                    +(r/2)*T(x-1,y,nt);
            elseif y==2 && x~=2 && x~=(nx-1)
                B(n)=T(x,y,nt-1)+...
                    (r/2)*((T(x+1,y,nt-1)+T(x-1,y,nt-1)-2*T(x,y,nt-
1))+(T(x,y+1,nt-1)+T(x,y-1,nt-1)-2*T(x,y,nt-1)))...
                    +(r/2)*T(x,y-1,nt);
            elseif x==(nx-1) && y~=2 && y~=(ny-1)
                B(n)=T(x,y,nt-1)+...
                    (r/2)*((T(x+1,y,nt-1)+T(x-1,y,nt-1)-2*T(x,y,nt-
1))+(T(x,y+1,nt-1)+T(x,y-1,nt-1)-2*T(x,y,nt-1)))...
                    +(r/2)*T(x+1,y,nt);
            elseif y==(ny-1) && x~=2 && x~=(nx-1)
                B(n)=T(x,y,nt-1)+...
                    (r/2)*((T(x+1,y,nt-1)+T(x-1,y,nt-1)-2*T(x,y,nt-
1))+(T(x,y+1,nt-1)+T(x,y-1,nt-1)-2*T(x,y,nt-1)))...
                    +(r/2)*T(x,y+1,nt);
            elseif x==2 && y==2
                B(n)=T(x,y,nt-1)+...
                    (r/2)*((T(x+1,y,nt-1)+T(x-1,y,nt-1)-2*T(x,y,nt-
1))+(T(x,y+1,nt-1)+T(x,y-1,nt-1)-2*T(x,y,nt-1)))...
                    +(r/2)*(T(x-1,y,nt)+T(x,y-1,nt));
            elseif x==2 && y==(ny-1)
                B(n)=T(x,y,nt-1)+...
                    (r/2)*((T(x+1,y,nt-1)+T(x-1,y,nt-1)-2*T(x,y,nt-
1))+(T(x,y+1,nt-1)+T(x,y-1,nt-1)-2*T(x,y,nt-1)))...
                    +(r/2)*(T(x-1,y,nt)+T(x,y+1,nt));
            elseif x==(nx-1) && y==2
                B(n)=T(x,y,nt-1)+...
                    (r/2)*((T(x+1,y,nt-1)+T(x-1,y,nt-1)-2*T(x,y,nt-
1))+(T(x,y+1,nt-1)+T(x,y-1,nt-1)-2*T(x,y,nt-1)))...
                    +(r/2)*(T(x+1,y,nt)+T(x,y-1,nt));
            elseif x==(nx-1) && y==(ny-1)
                B(n)=T(x,y,nt-1)+...
                    (r/2)*((T(x+1,y,nt-1)+T(x-1,y,nt-1)-2*T(x,y,nt-
1))+(T(x,y+1,nt-1)+T(x,y-1,nt-1)-2*T(x,y,nt-1)))...
                    +(r/2)*(T(x+1,y,nt)+T(x,y+1,nt));
            else
                B(n)=T(x,y,nt-1)+...
                    (r/2)*((T(x+1,y,nt-1)+T(x-1,y,nt-1)-2*T(x,y,nt-
1))+(T(x,y+1,nt-1)+T(x,y-1,nt-1)-2*T(x,y,nt-1)));
            end
        end
    end

    % Calculating T at nt timestep
    %TT(:)=A\B';
    ain=inv(A);
    for i=1:size(ain,1)
        for j=1:size(ain,2)
            if ain(i,j)<10^-5
```

```
                ain(i,j)=0;
            end
        end
    end

    TT(:)=ain*B';
    n=0;
    for x=2:1:nx-1
        for y=2:1:ny-1
            n=n+1;
            T(x,y,nt)=TT(n);
        end
    end
end

%% POST PROCESSING
nt=1+100/dt;
figure;set(gcf, 'Position', get(0, 'Screensize'));
pcolor((0:1:nx).*dx,(0:1:ny).*dy,T(:,:,nt)');shading flat;axis
equal;caxis([190, 330]);
colormap(gca,'hot');c = colorbar;c.Label.String = 'Temperature [K]';
xlim([0,1]);ylim([0,1]);
ylabel('y [m]');xlabel('x [m]');title(['t=',num2str((nt-1)*dt),' s']);

nt=1+500/dt;
figure;set(gcf, 'Position', get(0, 'Screensize'));
pcolor((0:1:nx).*dx,(0:1:ny).*dy,T(:,:,nt)');shading flat;axis
equal;caxis([190, 330]);
colormap(gca,'hot');c = colorbar;c.Label.String = 'Temperature [K]';
xlim([0,1]);ylim([0,1]);
ylabel('y [m]');xlabel('x [m]');title(['t=',num2str((nt-1)*dt),' s']);

nt=1+3000/dt;
figure;set(gcf, 'Position', get(0, 'Screensize'));
pcolor((0:1:nx).*dx,(0:1:ny).*dy,T(:,:,nt)');shading flat;axis
equal;caxis([190, 330]);
colormap(gca,'hot');c = colorbar;c.Label.String = 'Temperature [K]';
xlim([0,1]);ylim([0,1]);
ylabel('y [m]');xlabel('x [m]');title(['t=',num2str((nt-1)*dt),' s']);
```

CODE #2: CN scheme with factorization –

```
clear
close;

%% Initialization
dt=0.2; %time step in sec
tmax=3000; %simulation time limit
%NX=6; NY=6; %number of grid points including boundary points
%nx=NX-1; ny=NY-1;
nx=32;ny=32;
dx=1/(nx-1);dy=1/(ny-1);
al=9.7*10^(-5); %aluminum thermal diffusivity
r=al*dt/(dx*dx);
T(1:1:nx,1:1:ny,1:1:tmax/dt+1)=200; %Temperature spatio-temporal array
initialization
```

```
%% CFD - WITH FACTORIZATION
%%% THIS CODE IS DESIGNED ONLY FOR EQUAL NUMBER OF X & Y NODES - BECAUSE OF
%%% THE SYMMETRY OF THE PROBLEM. THIS IS TO CLARIFY THAT THE STUDENT IS ABLE
TO SOLVE IT FOR UNEQUAL NUMBER AS WELL;
%%% THIS PROBLEM DID NOT REQUIRE SO IT IS NOT ATTEMPTED.

 A1=diag((1+r).*ones((nx-2),1));
 b=(-r/2).*ones(-1+(nx-2),1);
 A2=diag(b,1);
 A3=diag(b,-1);
 A=A1+A2+A3;

for nt=2:1:tmax/dt+1
    % boundary conditions
    T(:,1,nt)=250+(190-250).*(0:1:nx-1).*dx;
    T(:,ny,nt)=300+(330-300).*(0:1:nx-1).*dx;
    T(1,:,nt)=250+(300-250).*(0:1:ny-1).*dy;
    T(nx,:,nt)=190+(330-190).*(0:1:ny-1).*dy;
end

for nt=2:1:tmax/dt+1
    % crank-nicholson - To compute T(x,y,nt)
    nt
    clear del1 del2
    for y=2:1:ny-1
        for x=2:1:nx-1
            B(x-1)=r*((T(x+1,y,nt-1)+T(x-1,y,nt-1)-2*T(x,y,nt-
1))+(T(x,y+1,nt-1)+T(x,y-1,nt-1)-2*T(x,y,nt-1)));
        end
        del1(:,y-1)=A\B';
    end

    for x=2:1:nx-1
        del2(x-1,:)=A\del1(x-1,:)';
    end

    % Calculating T at nt timestep
    T(2:nx-1,2:ny-1,nt)=T(2:nx-1,2:ny-1,nt-1)+del2(:,:);
end

%% POST PROCESSING
nt=1+100/dt;
figure;set(gcf, 'Position', get(0, 'Screensize'));
pcolor((0:1:nx-1).*dx,(0:1:ny-1).*dy,T(:,:,nt)');shading flat;axis
equal;caxis([190, 330]);
colormap(gca,'hot');c = colorbar;c.Label.String = 'Temperature [K]';
xlim([0,1]);ylim([0,1]);
ylabel('y [m]');xlabel('x [m]');title(['t=',num2str((nt-1)*dt),' s']);

nt=1+500/dt;
figure;set(gcf, 'Position', get(0, 'Screensize'));
pcolor((0:1:nx-1).*dx,(0:1:ny-1).*dy,T(:,:,nt)');shading flat;axis
equal;caxis([190, 330]);
colormap(gca,'hot');c = colorbar;c.Label.String = 'Temperature [K]';
xlim([0,1]);ylim([0,1]);
ylabel('y [m]');xlabel('x [m]');title(['t=',num2str((nt-1)*dt),' s']);
```

```
nt=1+3000/dt;
figure;set(gcf, 'Position', get(0, 'Screensize'));
pcolor((0:1:nx).*dx,(0:1:ny).*dy,T(:,:,nt)');shading flat;axis
equal;caxis([190, 330]);
colormap(gca,'hot');c = colorbar;c.Label.String = 'Temperature [K]';
xlim([0,1]);ylim([0,1]);
ylabel('y [m]');xlabel('x [m]');title(['t=',num2str((nt-1)*dt),' s']);
```