EGM 6342 - CFD

Project - 4

Jithin Gopinadhan
UFID 01247984

PROBLEM STATEMENT

A 2D domain of size $2\pi \times 2\pi$ is given. The flow inside the domain is steady and is given by the following non dimensional velocity components in x & y

$$u = \sin(x)\cos(y)$$
$$v = -\cos(x)\sin(y)$$

Initial condition: The non dimensional temperature is given by
$$T(x, y, t=0) = 0$$

Boundary condition: At time $t=0$, non dimensional temperatures of both left and right walls are increased to 1.0 while the other two walls remain at $T=0$. The boundary conditions stay steady subsequently.

Governing equation: The temperature evolution in the domain is given by the advection-diffusion equation given by

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x} - v\frac{\partial T}{\partial y} + \frac{1}{Pe}\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right)$$

where Pe is the Peclet number. $Pe = 1$.

Objective: Apply finite difference method using a ~~f~~ uniform grid to obtain -

* Temperature field at $t = 2.5, 5, 7.5, 10$ s
* Behavior of temperature profile at $x = \frac{3\pi}{4}$, $y = \frac{\pi}{4}$
* Grid refinement study.

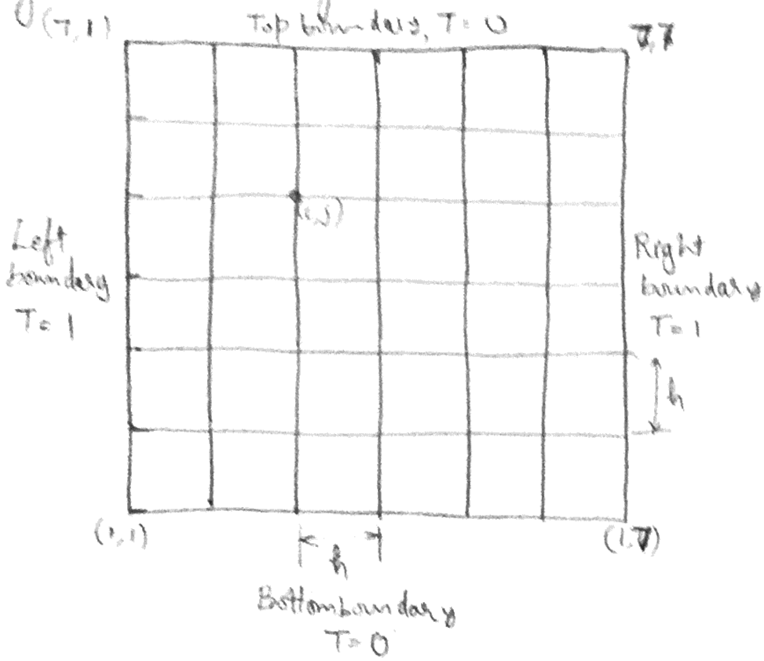
Numerical schemes used :

* Convection terms - Central difference scheme - 2^{nd} order
* Diffusion term - Central difference scheme - 2^{nd} order.
* $\frac{\partial T}{\partial t}$ term - AB2 method - 2^{nd} order
  - RK4 method - 4^{th} order.

Assumptions :

1. The flow inside the domain is viscous, incompressible and in steady state
2. The Peclet number is taken as 1.
3. The functions are continuous and differentiable in the given domain and hence Taylor series approximations are valid.
4. The discretization is carried out on uniform grid spacing with finite difference techniques.
5. Heat transfer modes ~~used~~ present are convection and conduction
6. All the ~~ext~~ nodal values of temperature on the left and right walls including the corner points have an initial value of 1 and stay the same independent of time.
7. All the nodal values of temperature on the upper & lower walls excluding the corner points have an initial value of 0 and stay the same independent of time.
8) Since the exact solution is unavailable, the highest grid resolution (80x80) that is available is assumed to be the exact solution for error calculations.

Grid Schematic for n = 6



Equally spaced grid

$i : 1 \rightarrow 7$

$j : 1 \rightarrow 7$

Discretization

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x} - v\frac{\partial T}{\partial y} + \frac{1}{Pe}\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right)$$

* Convection term

$\rightarrow$ $u = \sin(x).\cos(y)$, $v = -\cos(x)\sin(y)$ } steady

$$\rightarrow \left.\frac{\partial T}{\partial x}\right|_{i,j} = \left[\frac{T_{i+1} - T_{i-1}}{2h}\right]_j$$

$$\rightarrow \left.\frac{\partial T}{\partial y}\right|_{i,j} = \left[\frac{T_{j+1} - T_{j-1}}{2h}\right]_i$$

$T_{i,j} = 1$ if $i = 1$ or $i = N+1$

$= 0$ if $j = 1$ or $j = N+1$ except at corner points

* Diffusion term

$$\rightarrow \left.\frac{\partial^2 T}{\partial x^2}\right|_{i,j} = \left[\frac{T_{i+1} - 2T_i + T_{i-1}}{h^2}\right]_j$$

$$\rightarrow \left.\frac{\partial^2 T}{\partial y^2}\right|_{ij} = \left[\frac{T_{j+1} - 2T_j + T_j - 1}{h^2}\right]$$

Time integration

* Adam Bashforth 2nd order. (Explicit method)

$$T^{n+1} = T^{n} = \frac{1}{2}\Delta t\left[3\frac{\partial T}{\partial t}^{n} - \frac{\partial T}{\partial t}^{n-1}\right]$$

T^{n+1} → Temperature at next time step

T^{n} → Temperature at present time step

$\frac{\partial T}{\partial t}^{n}$ → At current time step

$\frac{\partial T}{\partial t}^{n-1}$ → At previous time step

AB2 is multistep method which requires past value of $\frac{\partial T}{\partial t}$ to predict future value. Hence it does not work at initial value of $t = 0$. So in order to start the loop, we use forward Euler method.

$$T^{1} = T^{0} + \frac{\partial T}{\partial t}\bigg|_{t=0}^{\Delta t} \cdot \Delta t$$

* Runge Kutta IV order. (Explicit method)

$$T^{n+1} = T^{n} + \frac{\Delta t}{6}\left[F_1 + 2F_2 + 2F_3 + F_4\right]$$

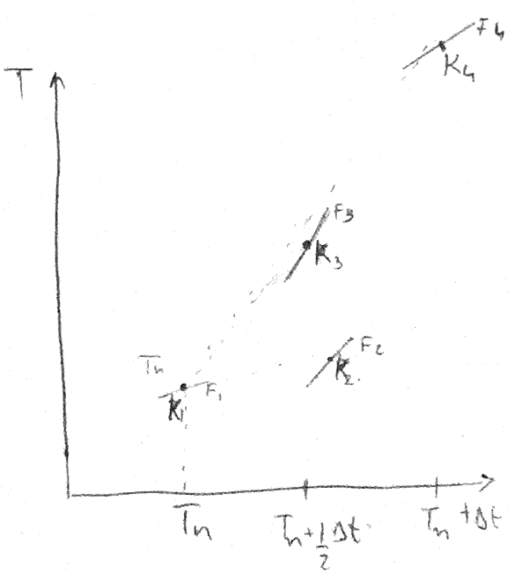
where (F → slope)

$$F_1 = f(K_1, t^n)$$

$$F_2 = f(K_2, t^n + \tfrac{1}{2}\Delta t)$$

$$F_3 = f(K_3, t^n + \tfrac{1}{2}\Delta t)$$

$$F_4 = f(K_4, t^n + \Delta t)$$



and (K → Predicted value.)

$$K_1 = T^n$$

$$K_2 = T^n + \tfrac{1}{2}\Delta t \cdot F_1$$

$$K_3 = T^n + \tfrac{1}{2}\Delta t \cdot F_2$$

$$K_4 = T^n + \Delta t \cdot F_3$$

Methodology - Algorithm used for implementation of numerical scheme.

The code for implementing the numerical scheme was written in MATLAB. The code can be broadly split into three parts as discussed below :

a) Numerical computation of temperature.

Each of the grid sizes are used to initialize the necessary temperature and dT/dt values of the corresponding sized matrices. Further, the spatial loops are used to select each T_{ij} and calculate the corresponding $\frac{\partial T}{\partial x}$, $\frac{\partial T}{\partial y}$, $\frac{\partial^2 T}{\partial y^2}$, $\frac{\partial^2 T}{\partial x^2}$ at each of the interior nodes. The values of T at boundaries stay steady and hence evaluation of these at boundaries are not necessary. The $\frac{\partial T}{\partial t}$ value can be evaluated by substituting above obtained values in the given governing equation. Once $\frac{\partial T}{\partial t}$ value is obtained two schemes are used to evaluate the T value at next time step..

* Adam Bashforth method - 2nd order

The scheme is not self starting. Hence at $t=0$, forward Euler method is used to find temperature at next time step. For further time steps, matrices are used to save value of $\frac{\partial T}{\partial t}$ at present time step and previous time step. These values along with present value of temperature are used to evaluate T at next time step for the entire grid

* RK 4 method

This scheme is self starting and hence does not require other schemes at start up. As explained in the discretization, T^n and $\frac{\partial T}{\partial t}\Big|^n$ are used to evaluate F_1, F_2, F_3, F_4, and K_1, K_2, K_3, K_4. Applying RK4 formula, this gives the values of T^{n+1} on the entire grid for next time step.

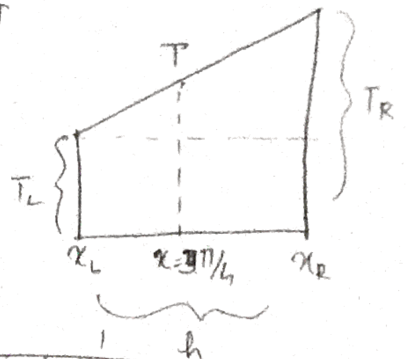
The matrix of temperature values are used to plot the temperature distribution at four time values t = 2.5, 5, 7.5, 10.

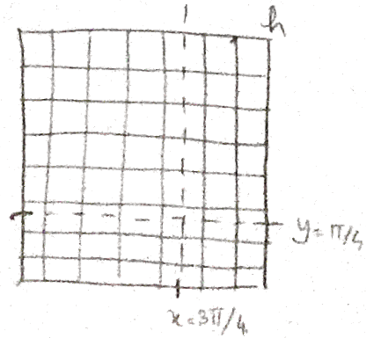
b) Interpolation of temperature along $x = \frac{3\pi}{4}$ and $y = \frac{\pi}{4}$.

The temperature distribution along the line was obtained for the highest grid refinement ~~along these~~ at t = 2.5, 5, 7.5, 10. Since the given x and y value do not necessarily fall on the exact node, it is necessary to do linear interpolation. First the indexes of the upper and lower (or right & left node) nodes are obtained by calculation. Then the temperature at these nodes are used to interpolate the value of T

$$T = T_L + (T_R - T_L)\left[\left(\frac{3\pi}{4} - x_L\right)/h\right]$$

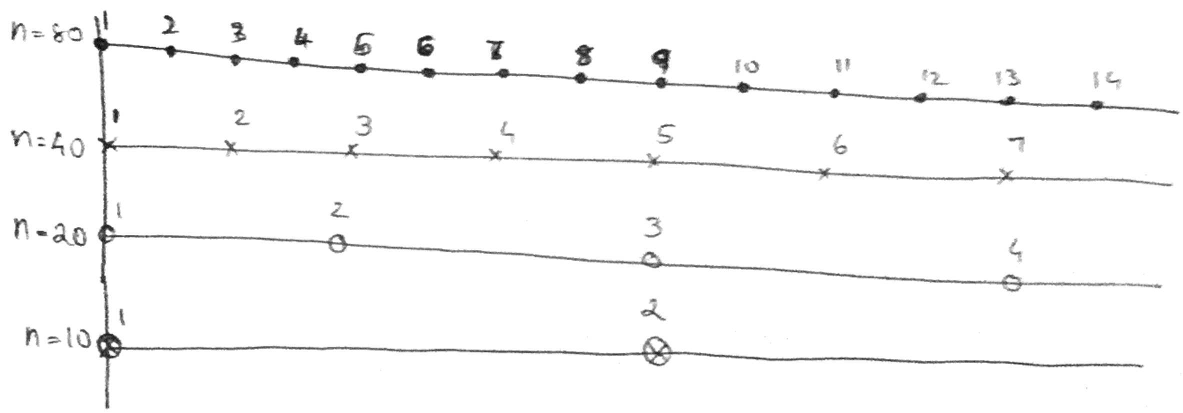
$$T = T_L + (T_R - T_L)\left[\left(\frac{\pi}{4} - y_L\right)/h\right]$$





c) Error Evaluation:

Error evaluation of the schemes is carried out for each of the grid refinements at the final time of $t = 10\,s$. For this, the value of ~~spatial di~~ maximum resolution grid is taken as exact value and the nodes of individual meshes are matched to the nodes on high resolution mesh in order to find the corresponding error matrices.

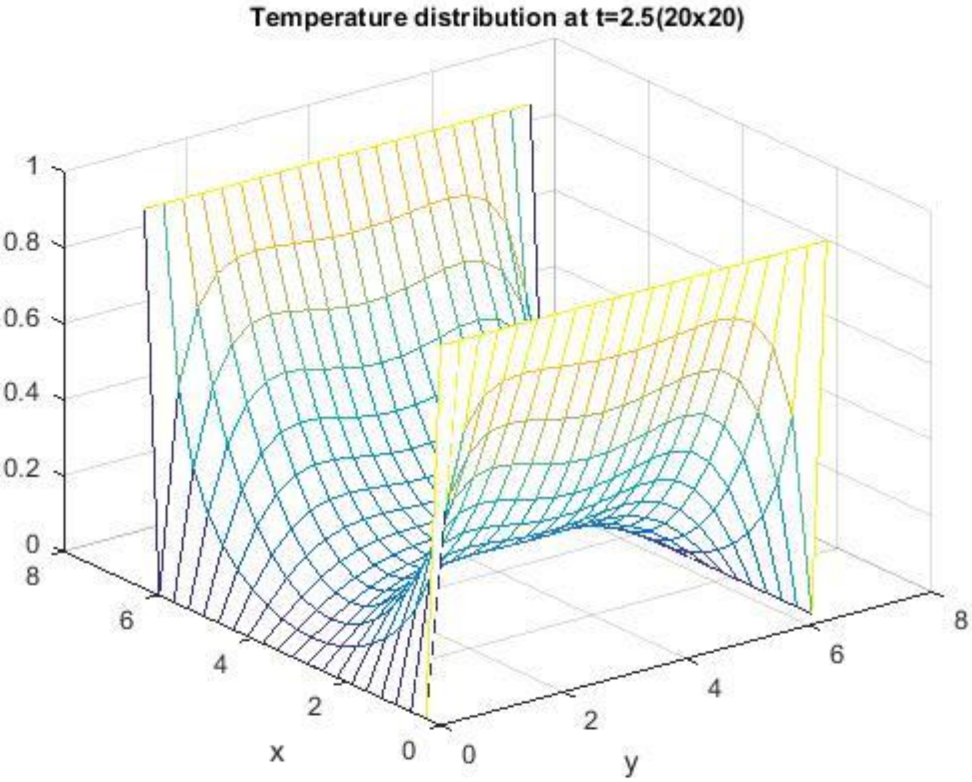


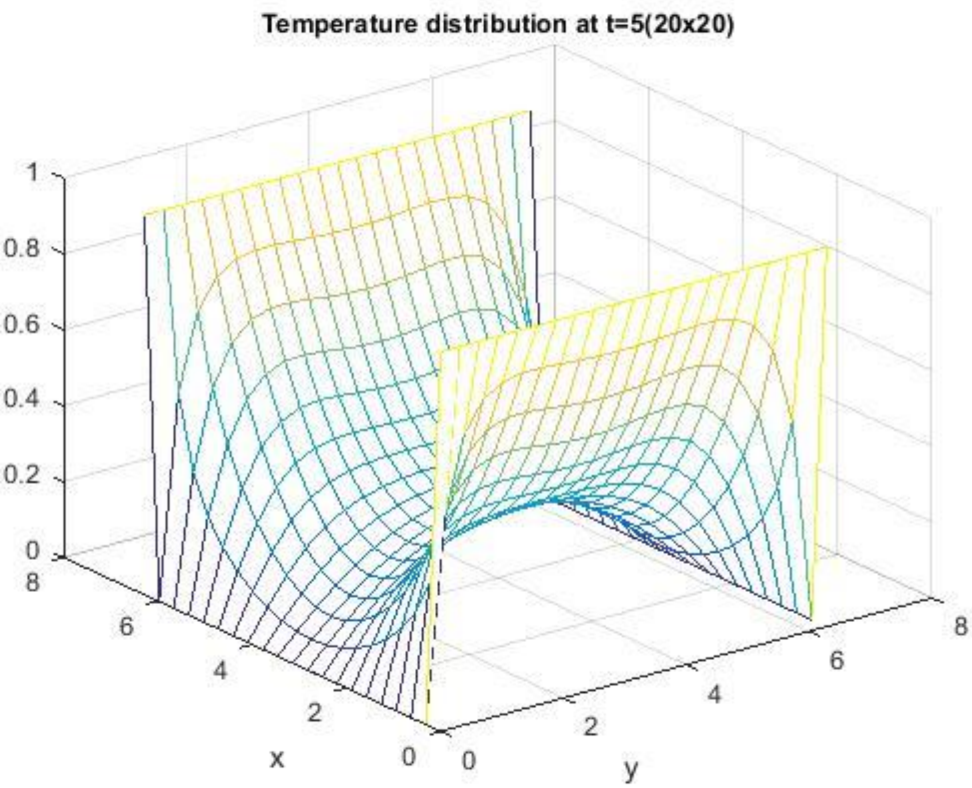
After evaluating the error matrix, L_2 norm of error is given by

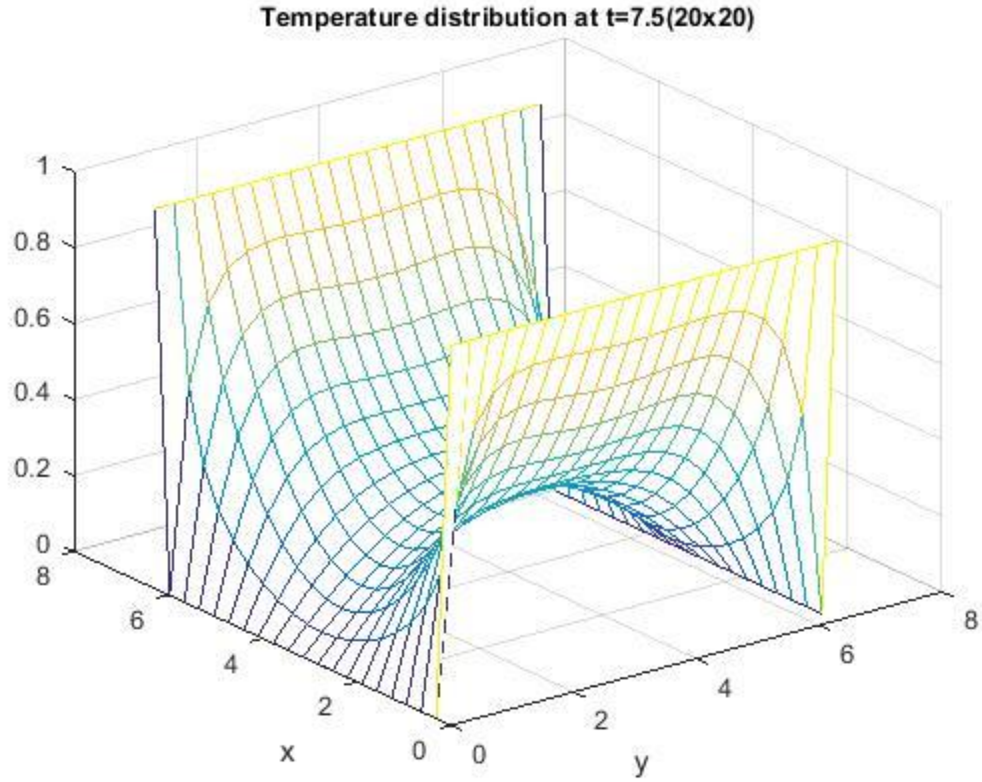
$$E_2 = \sqrt{\frac{\sum_i \sum_j e_{ij}^2}{\text{Total elements}}} = \sqrt{\frac{(\text{norm}(e))^2}{N^2}}$$

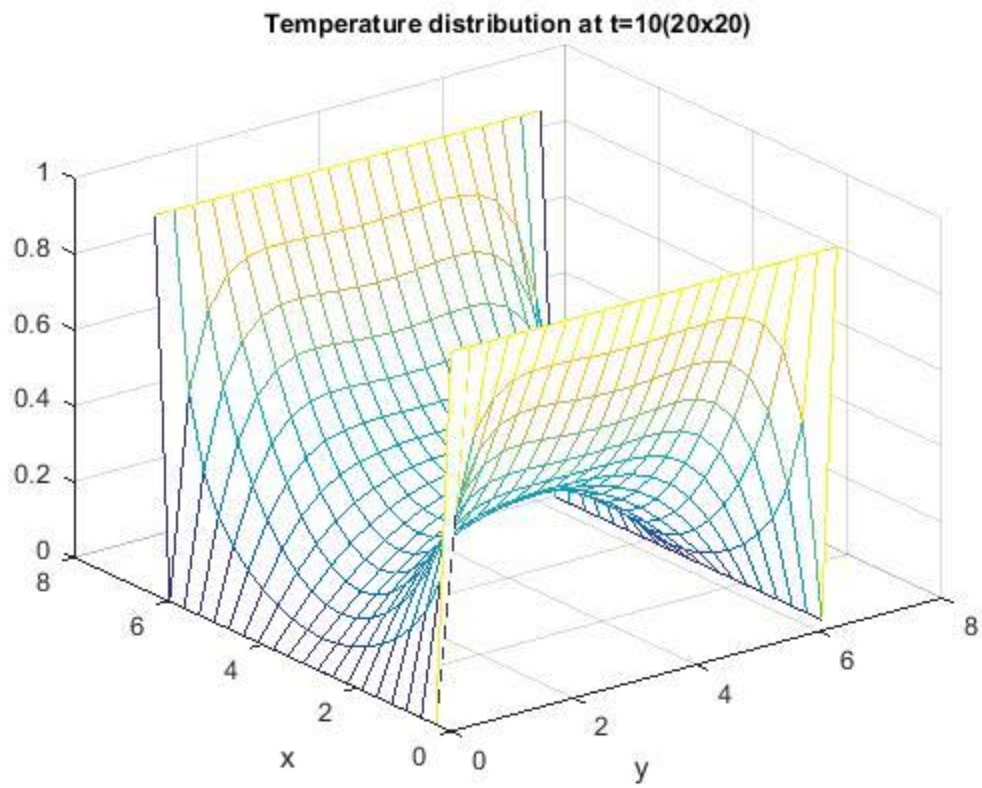
$(\text{norm}(e))^2 \rightarrow$ matlab fn that gives $\sqrt{\sum_i \sum_j e_{ij}^2}$

$$= \frac{\text{norm}(e)}{N}$$

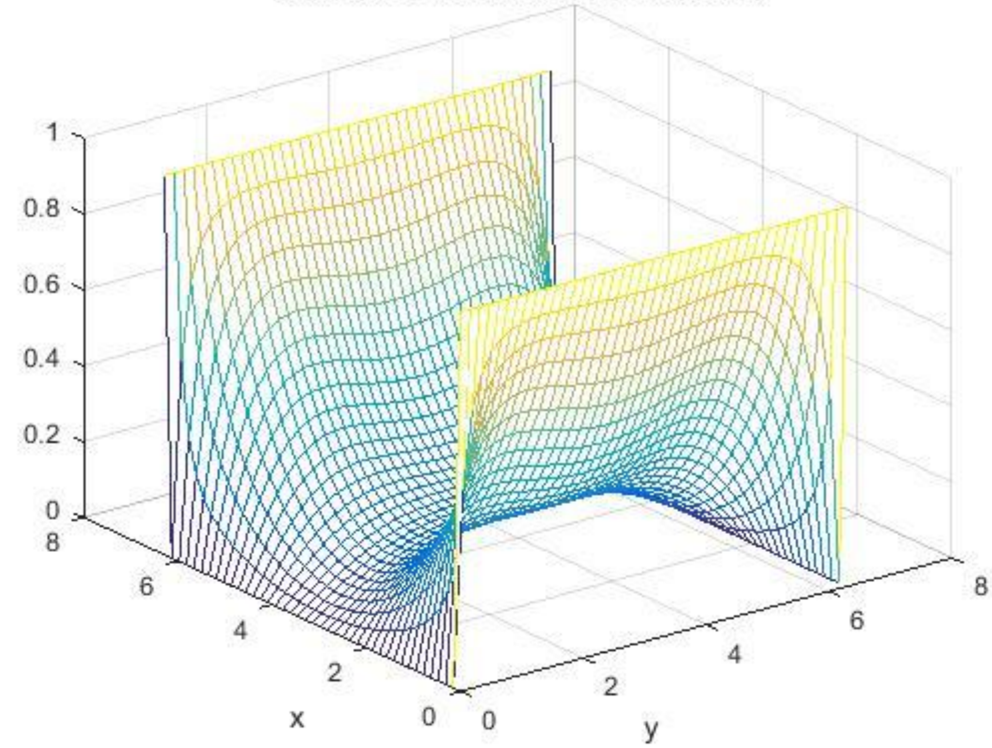

Temperature distribution at t=2.5(20x20)
x
y


Temperature distribution at t=5(20x20)
x
y

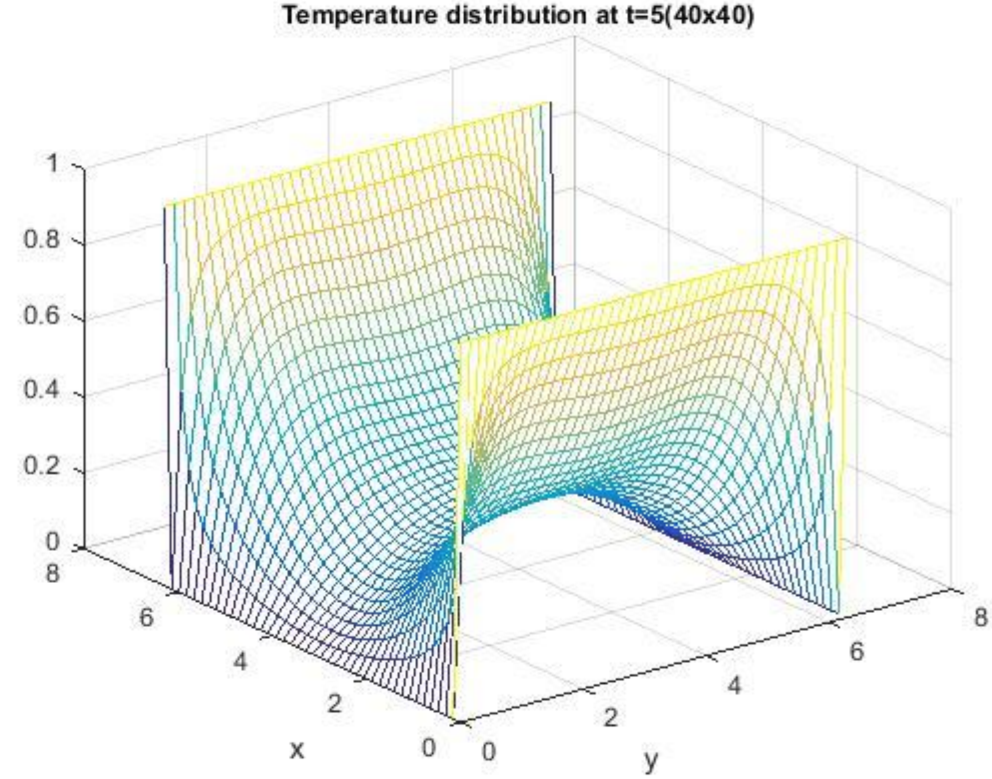

Temperature distribution at t=7.5(20x20)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
0
x
0
2
4
6
8
y

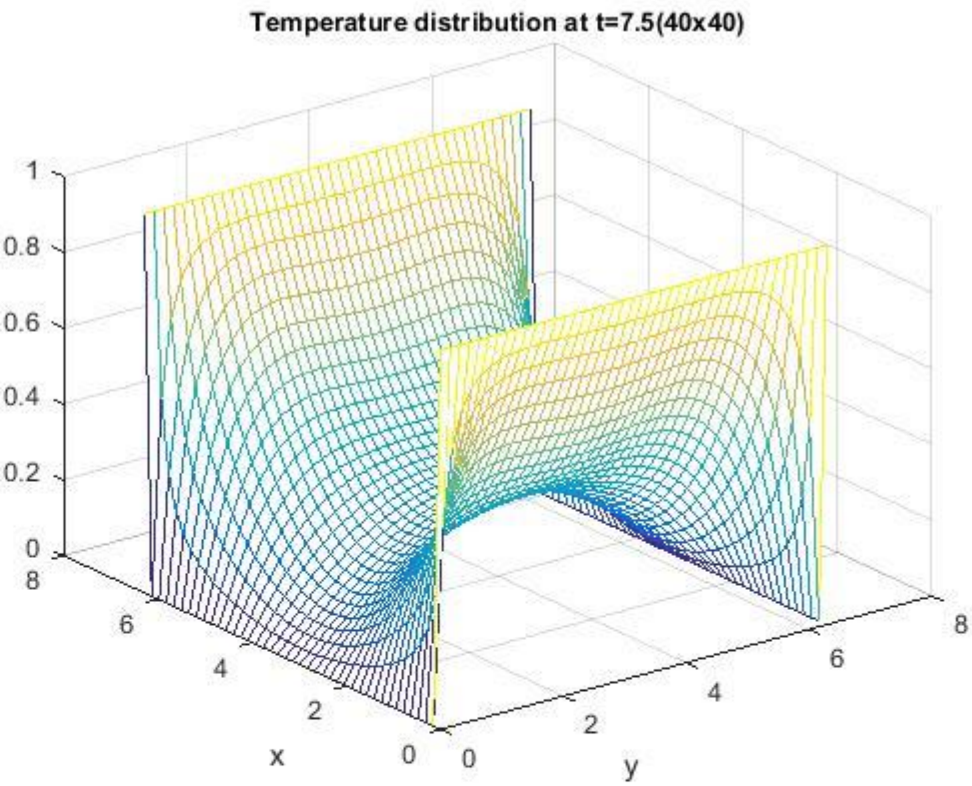

Temperature distribution at t=10(20x20)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
0
x
0
2
4
6
8
y

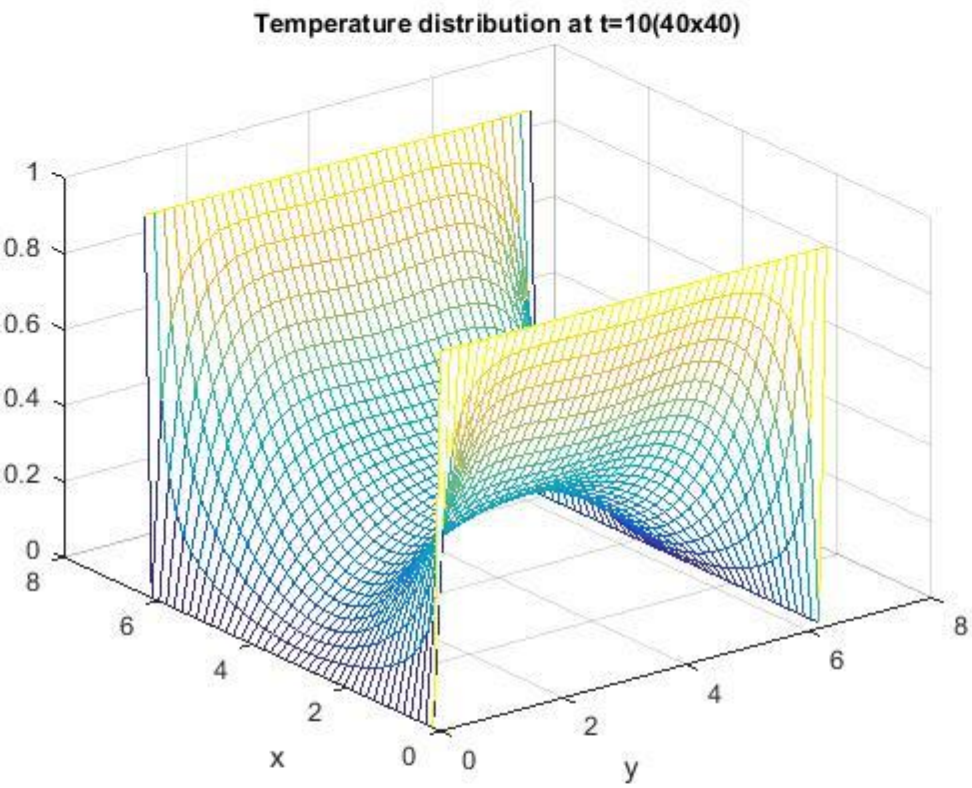
Temperature distribution at t=2.5(40x40)

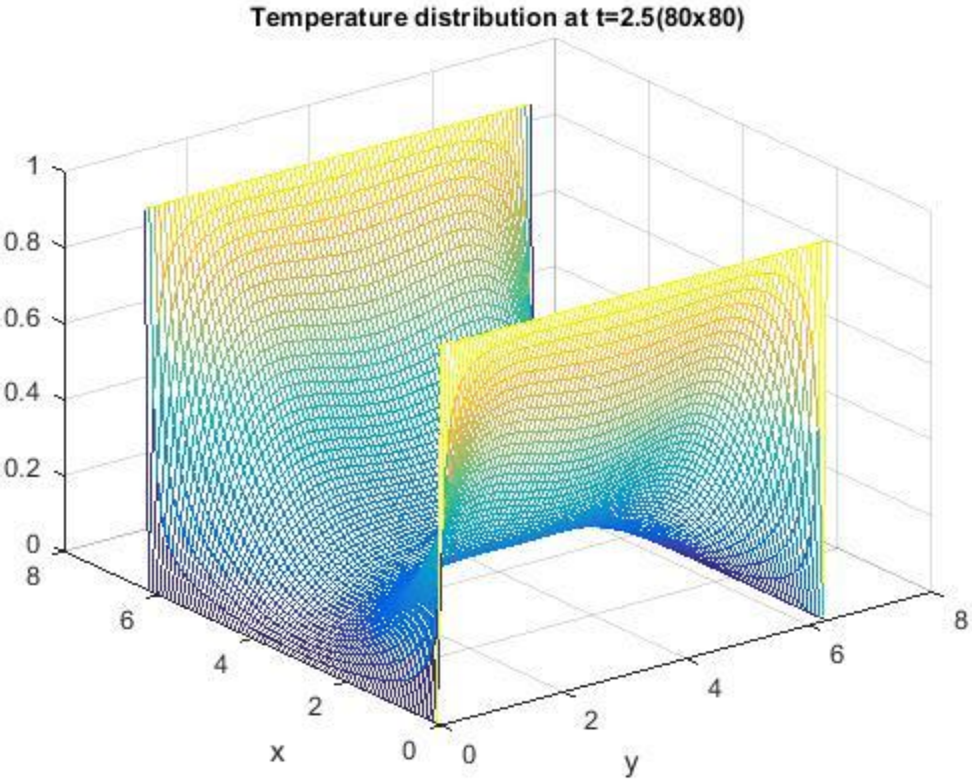


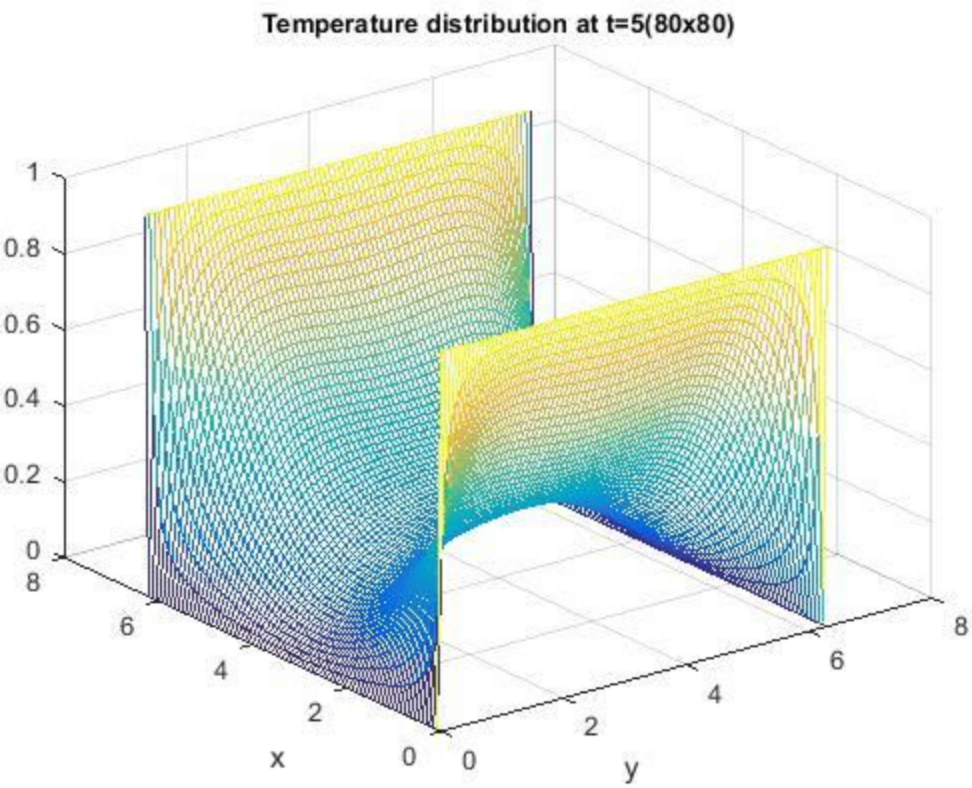
Temperature distribution at t=5(40x40)

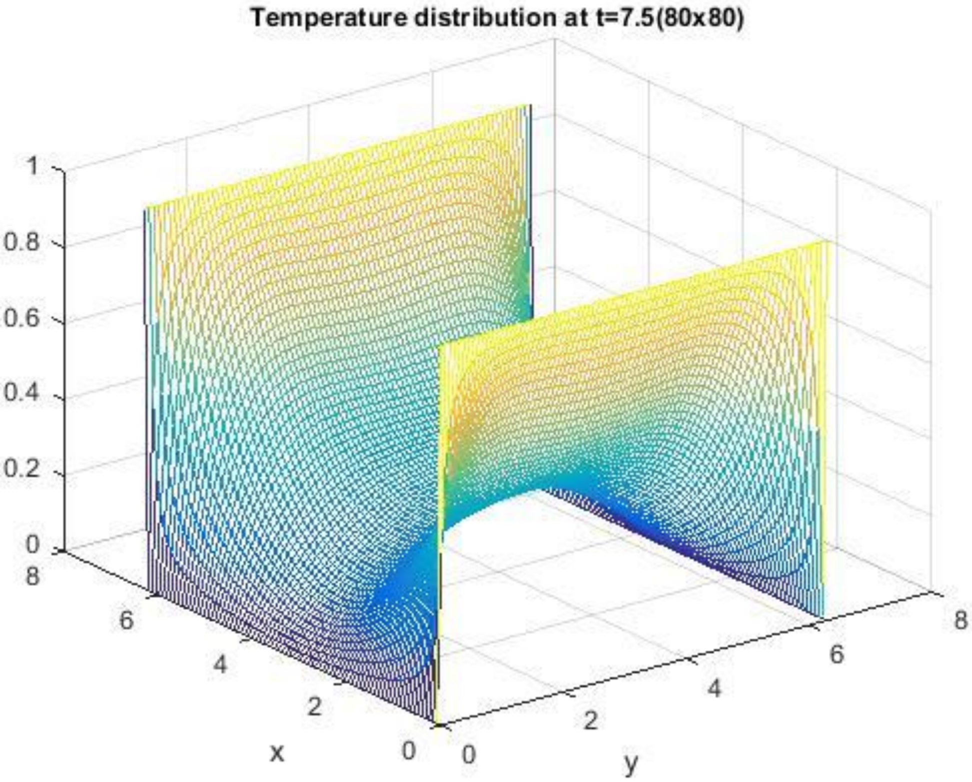



Temperature distribution at t=7.5(40x40)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
0
x
0
2
4
6
8
y

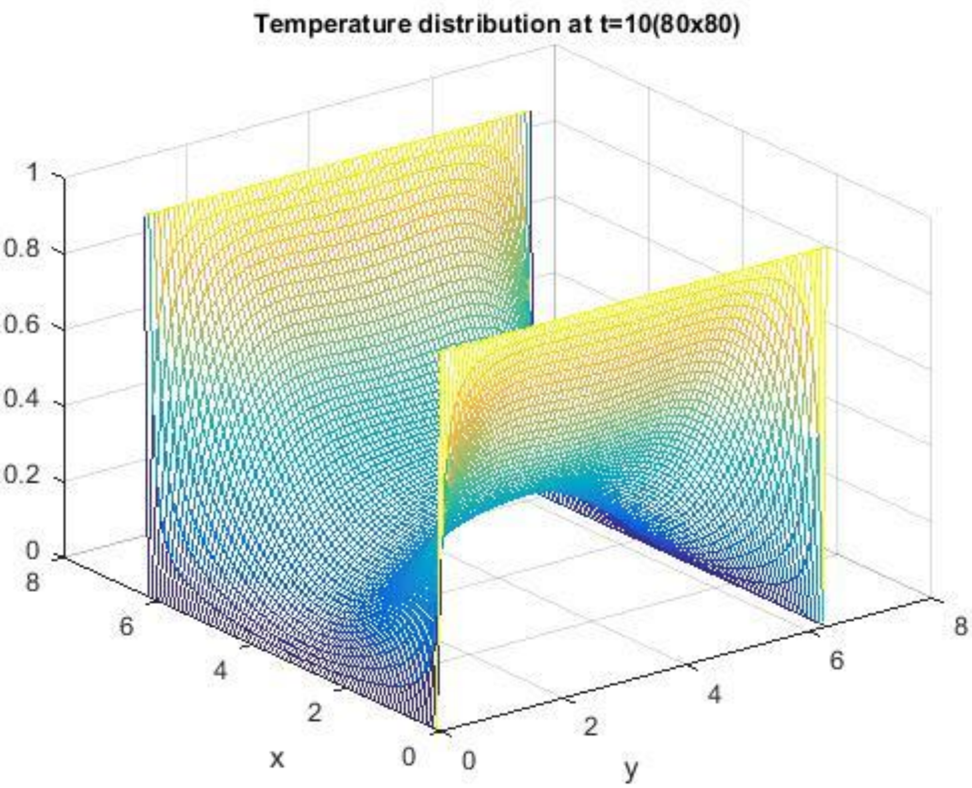

Temperature distribution at t=10(40x40)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
0
x
0
2
4
6
8
y


Temperature distribution at t=2.5(80x80)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
x
0
0
2
4
6
8
y

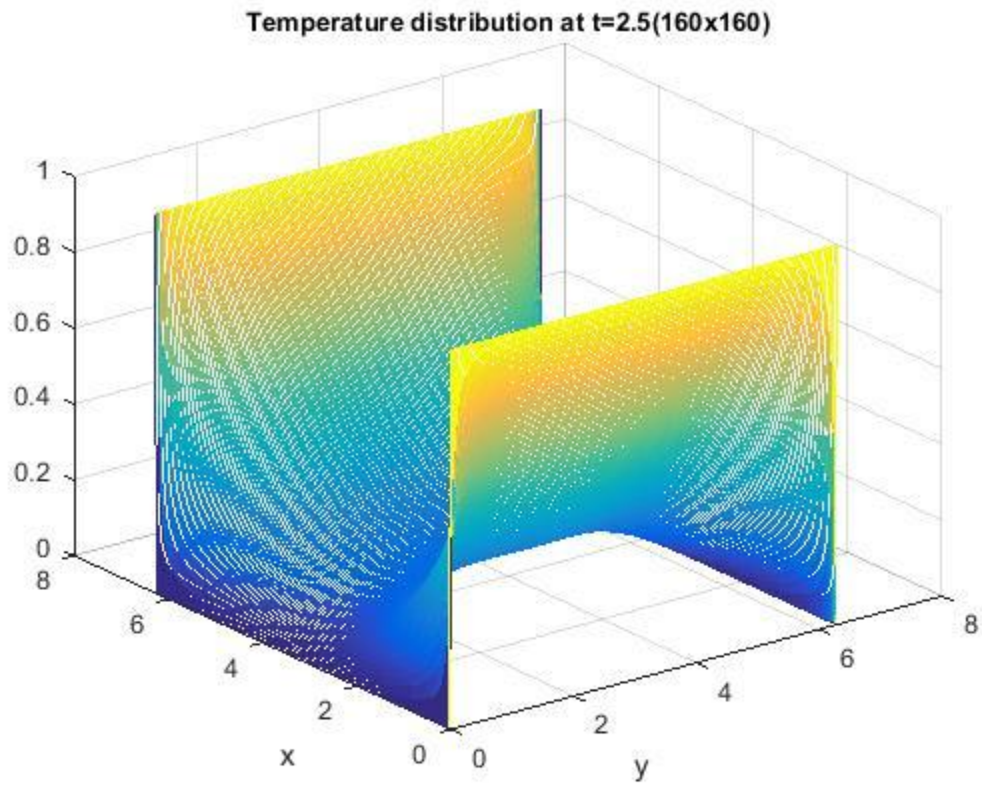

Temperature distribution at t=5(80x80)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
x
0
0
2
4
6
8
y

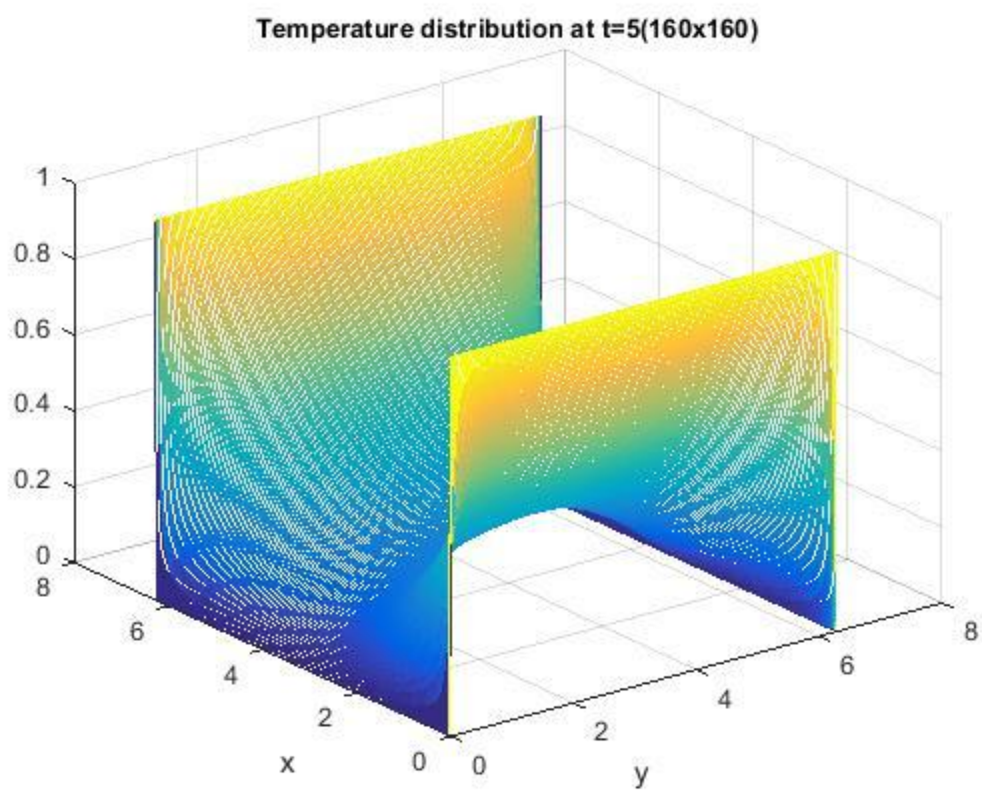


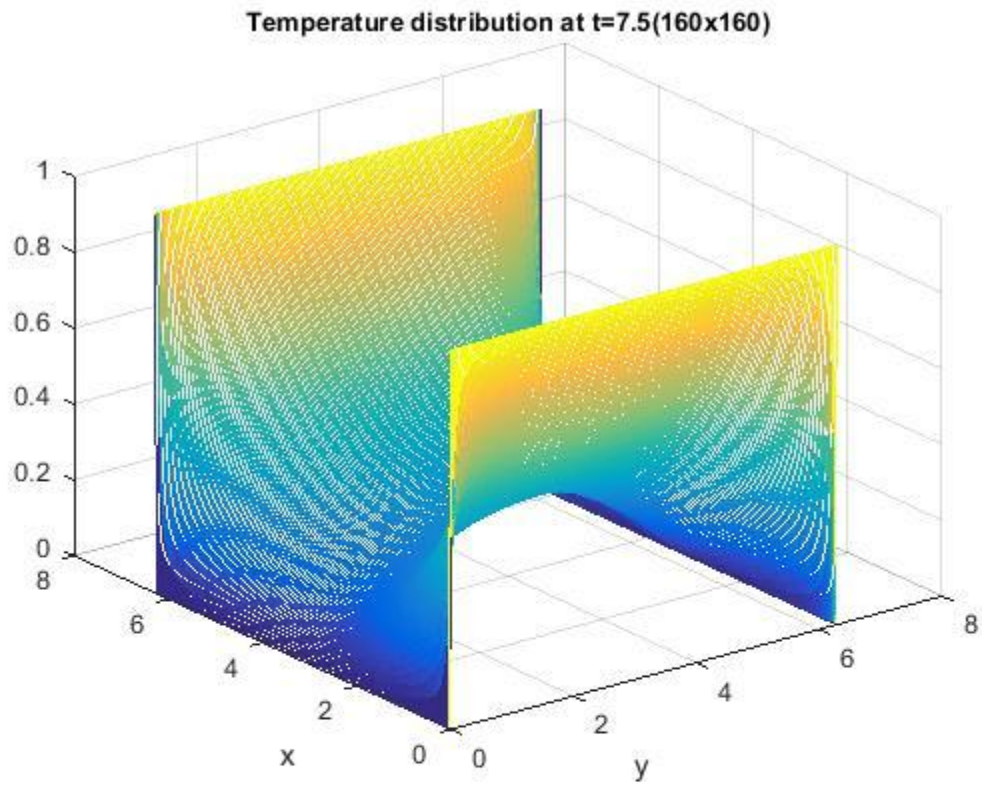
Temperature distribution at t=7.5(80x80)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
0
x
0
2
4
6
8
y



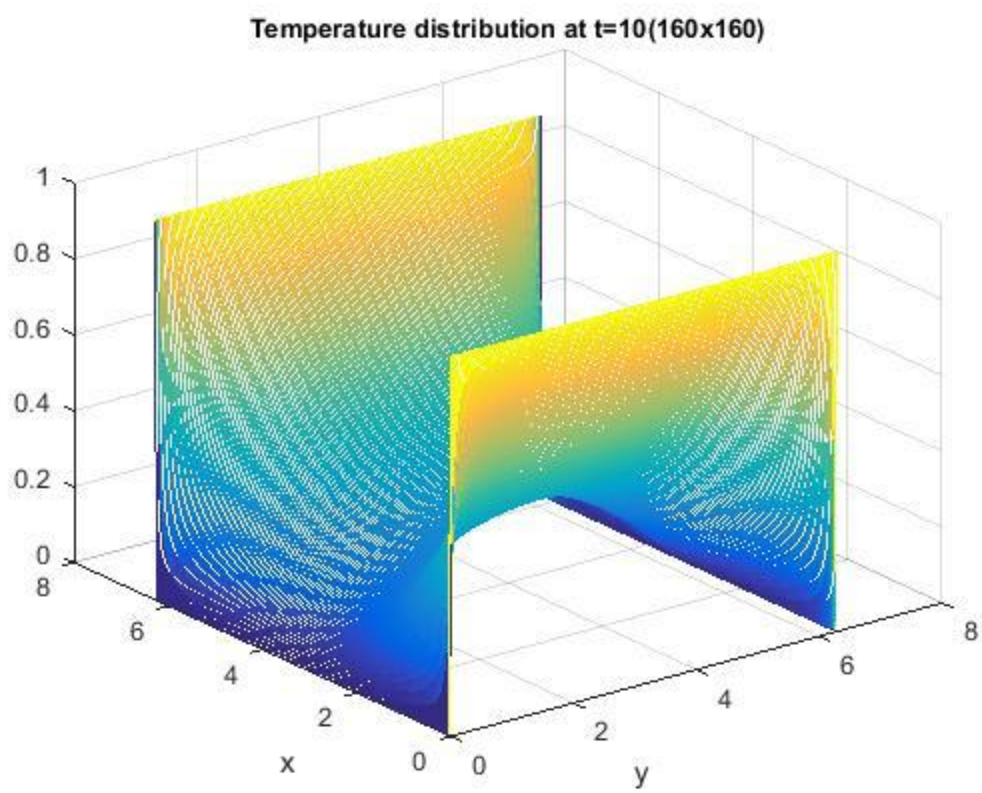
Temperature distribution at t=10(80x80)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
0
x
0
2
4
6
8
y


Temperature distribution at t=2.5(160x160)
x
y

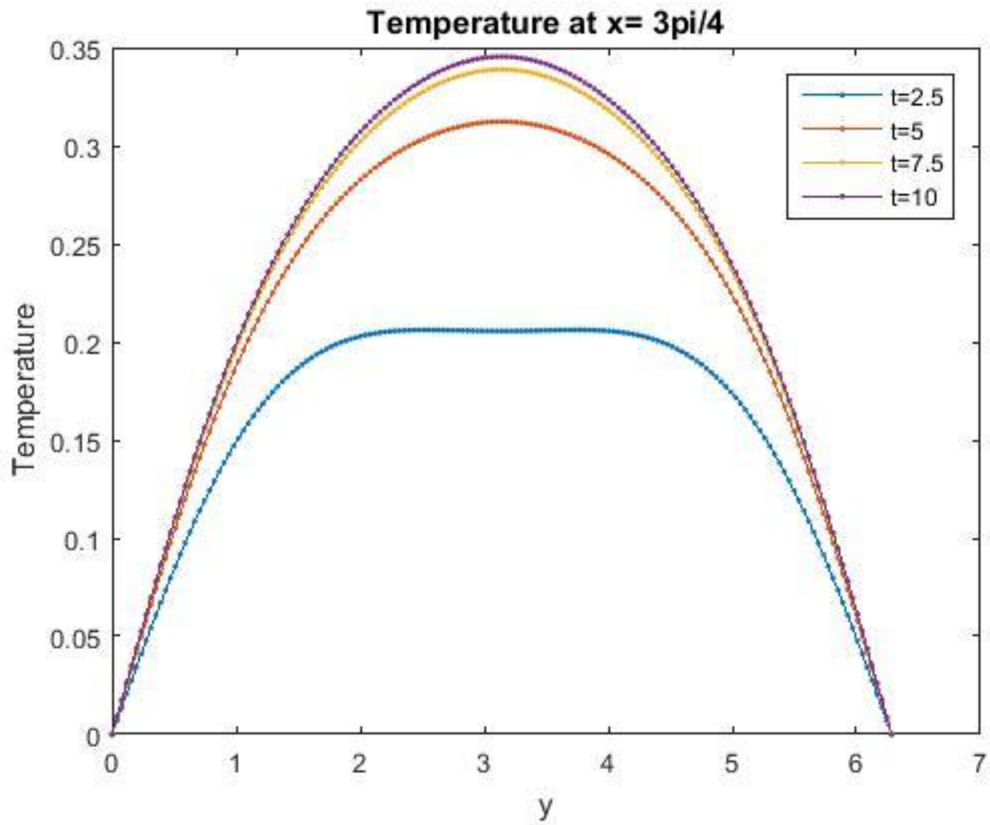

Temperature distribution at t=5(160x160)
x
y

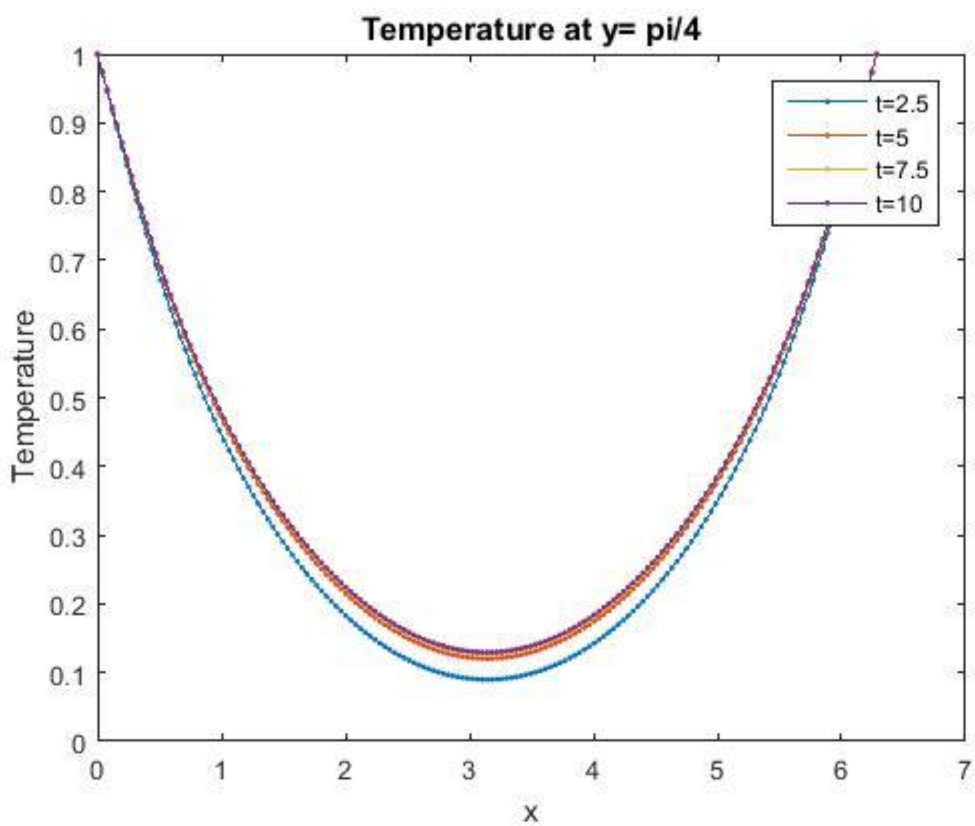


Temperature distribution at t=7.5(160x160)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
0
x
0
2
4
6
8
y

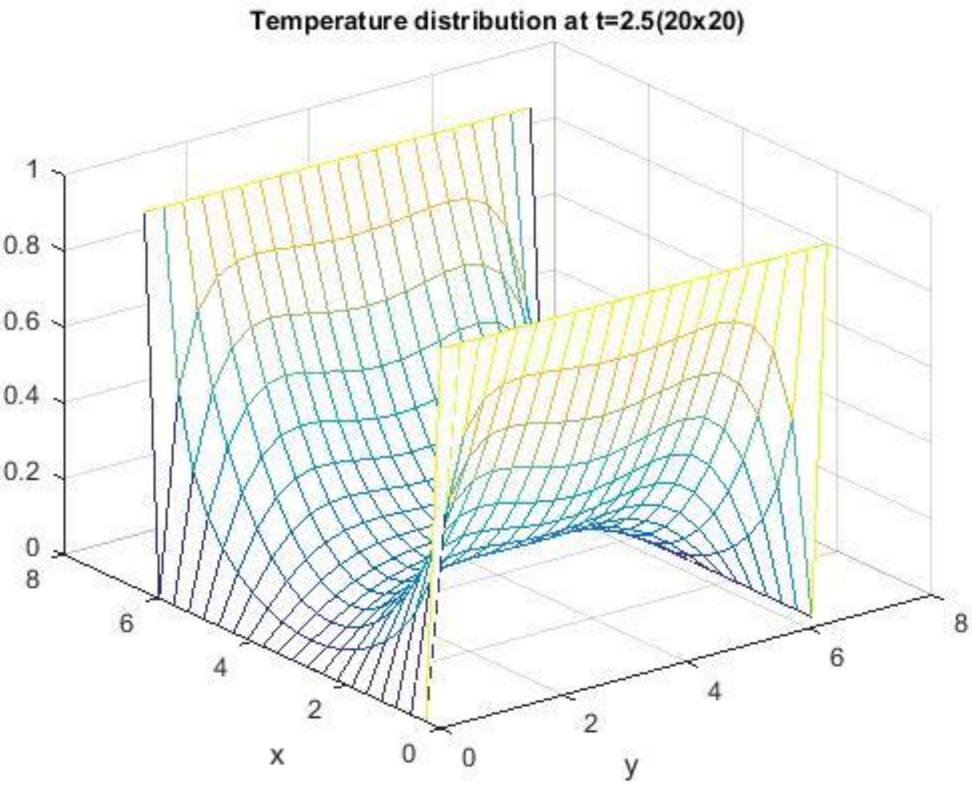


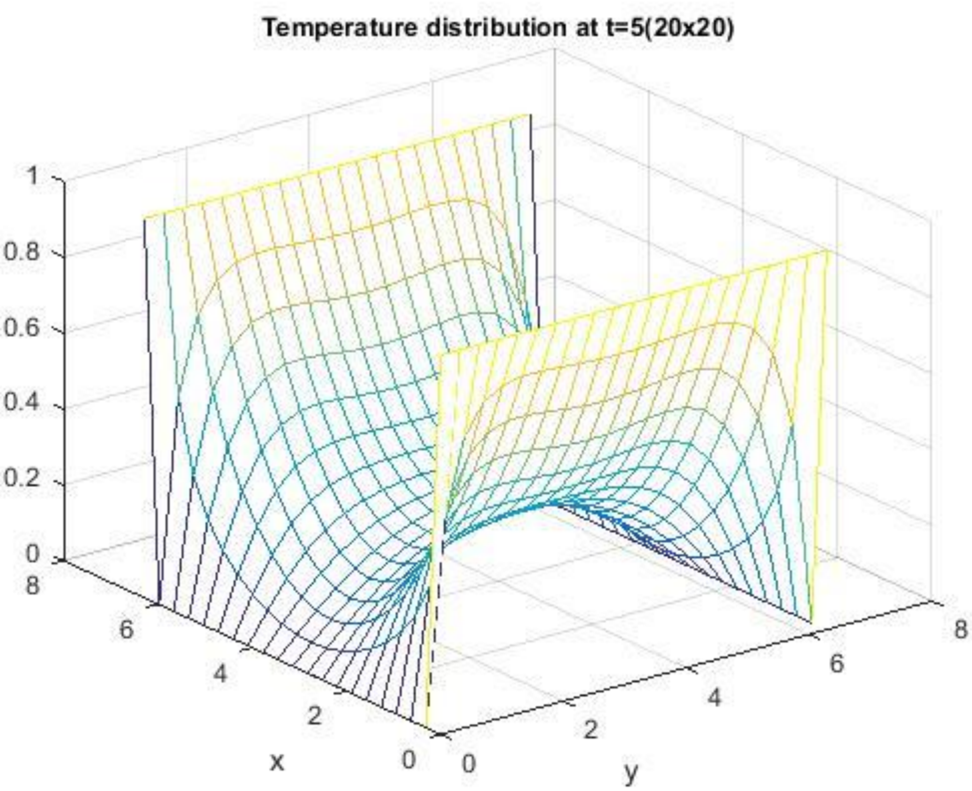
Temperature distribution at t=10(160x160)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
0
x
0
2
4
6
8
y


Temperature at x= 3pi/4
Temperature
y
t=2.5
t=5
t=7.5
t=10

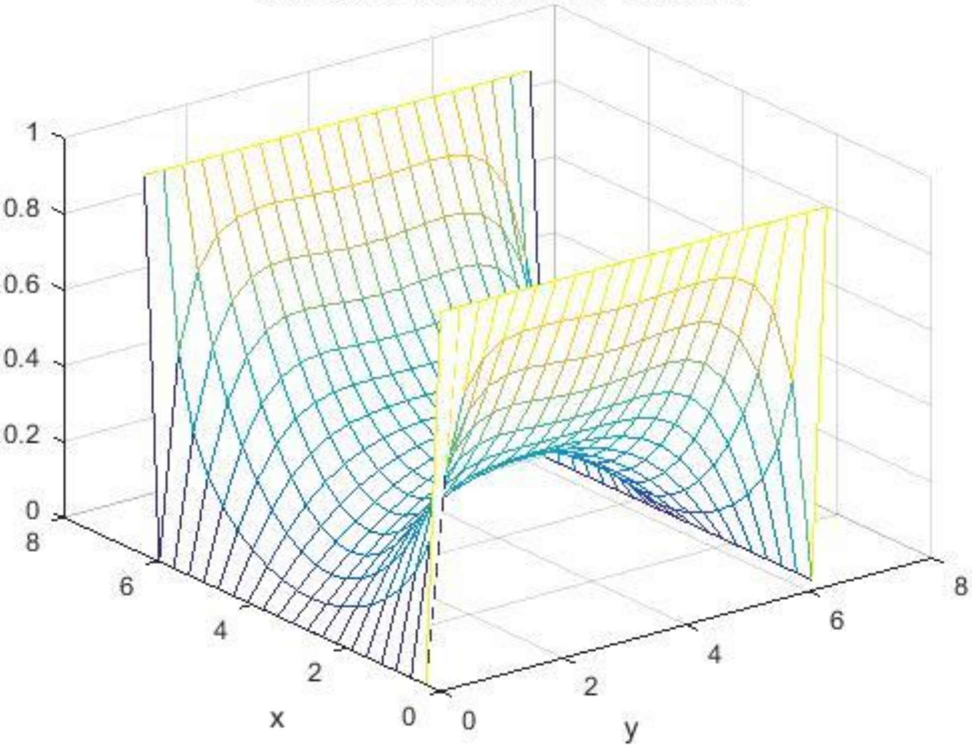

Temperature at y= pi/4
Temperature
x
t=2.5
t=5
t=7.5
t=10

RK4 Plots

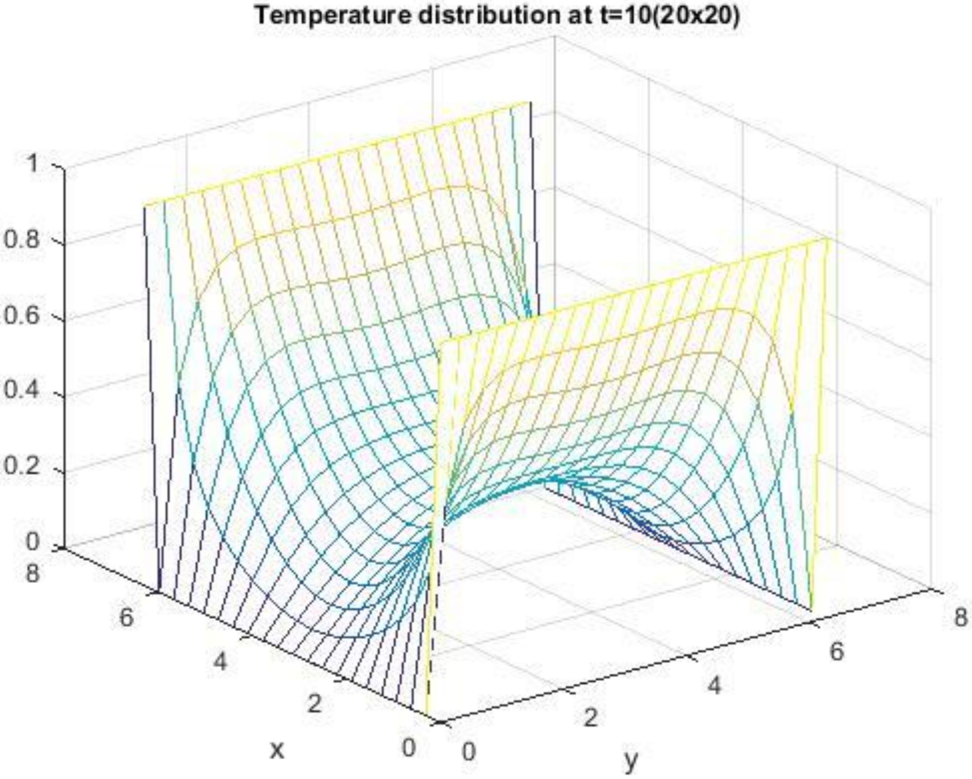

Temperature distribution at t=2.5(20x20)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
0
x
0
2
4
6
8
y


Temperature distribution at t=5(20x20)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
0
x
0
2
4
6
8
y

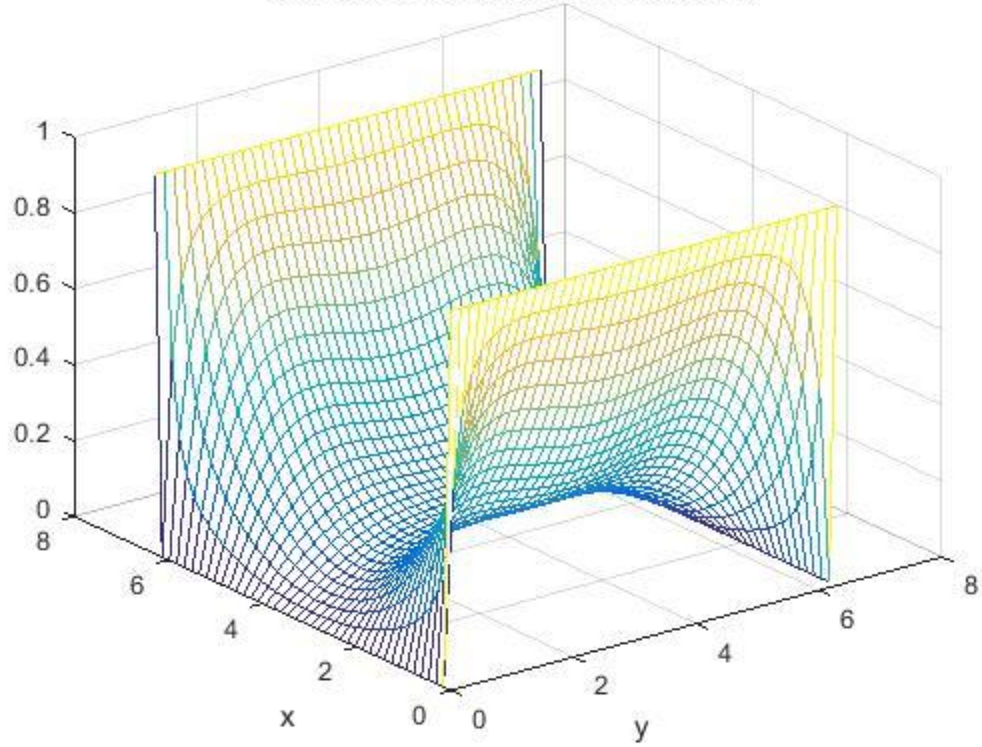
Temperature distribution at t=7.5(20x20)



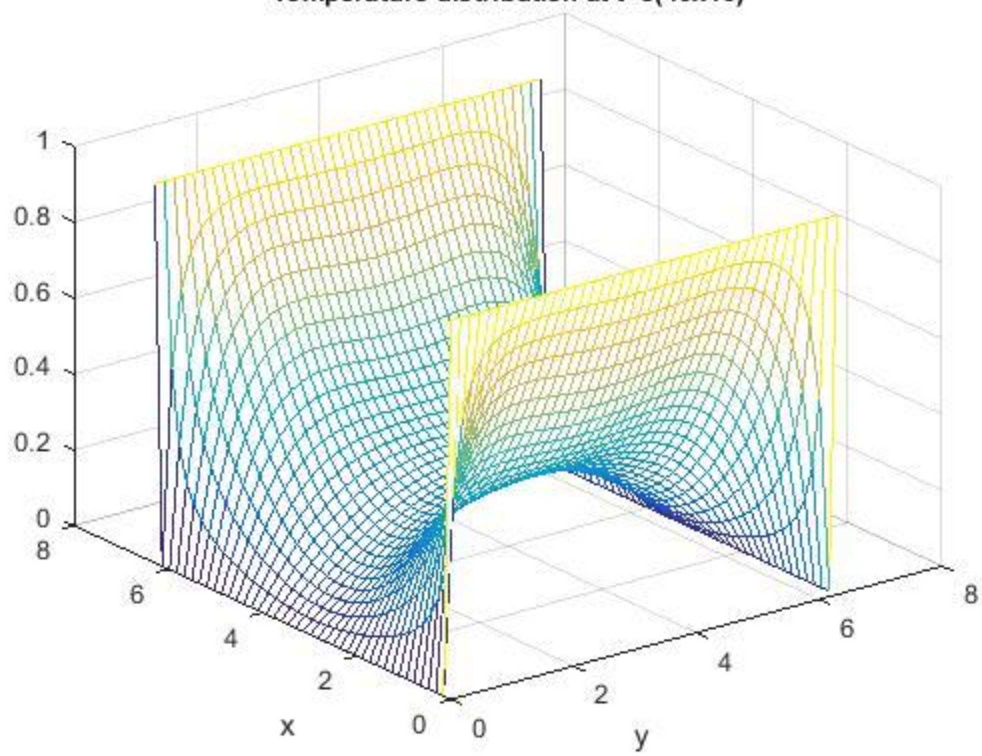
Temperature distribution at t=10(20x20)



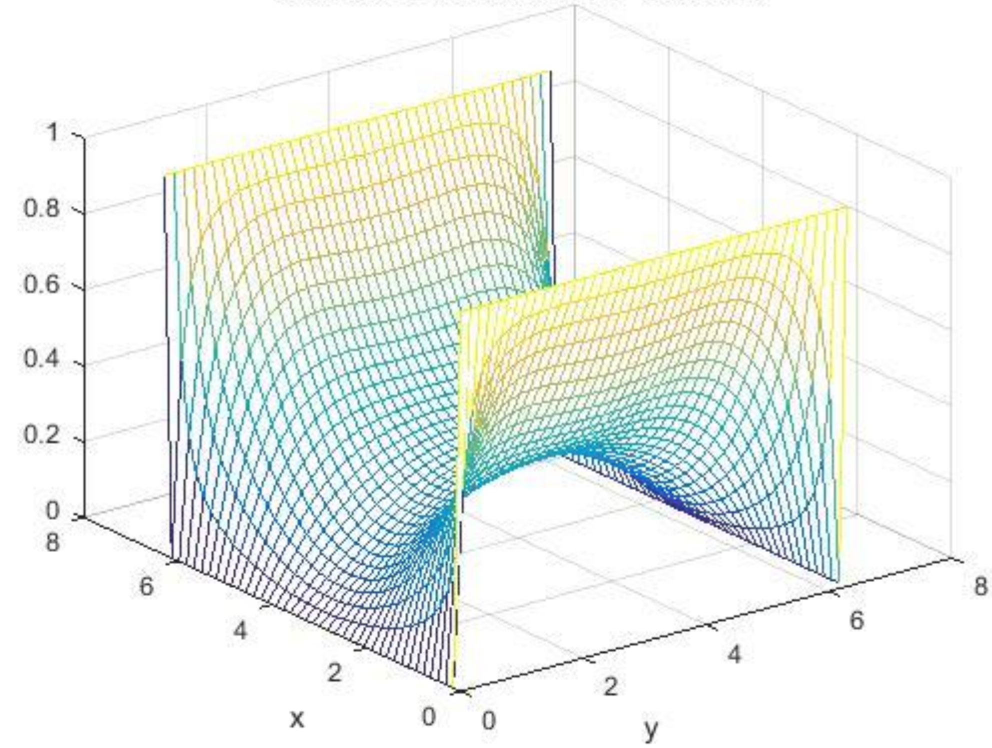
Temperature distribution at t=2.5(40x40)



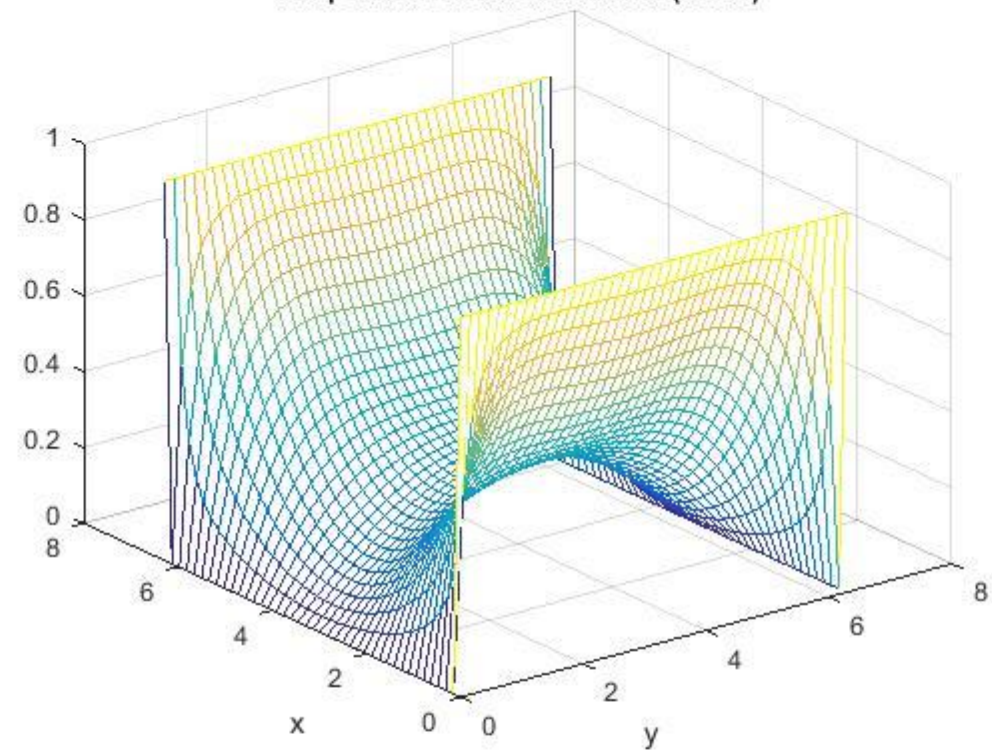
Temperature distribution at t=5(40x40)

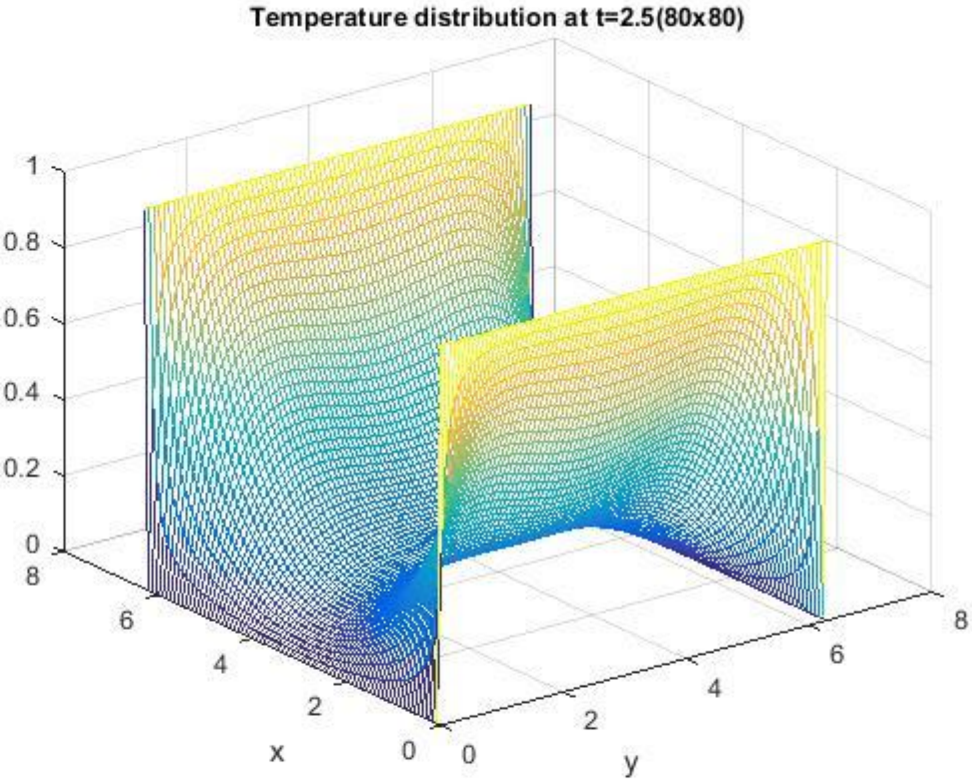


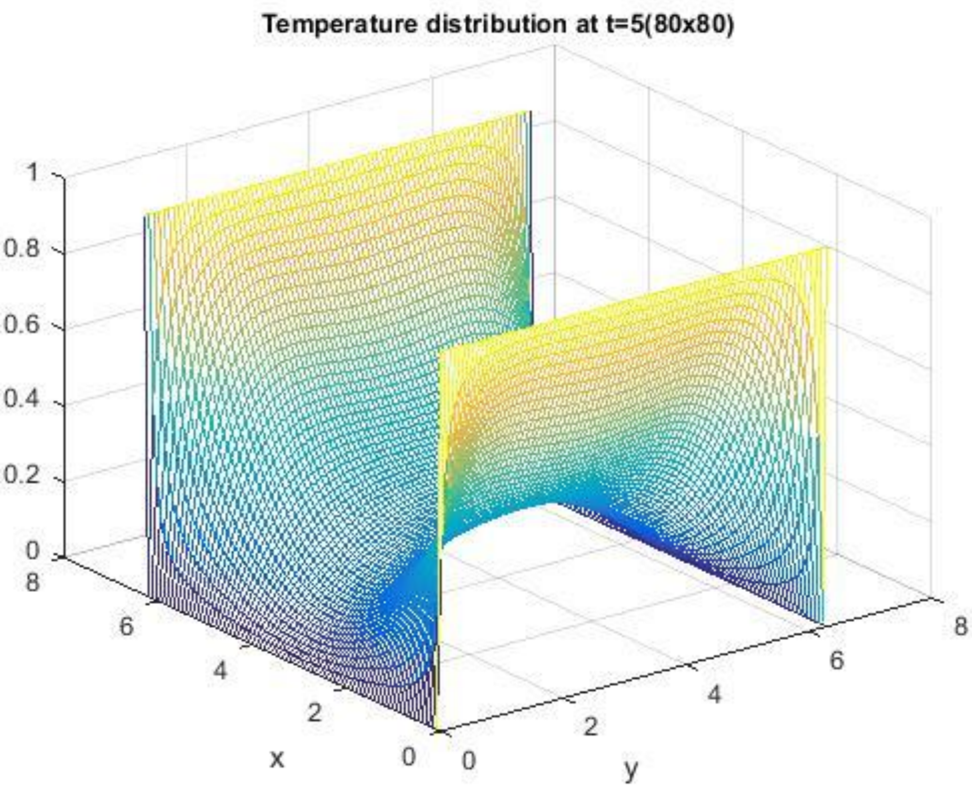
Temperature distribution at t=7.5(40x40)

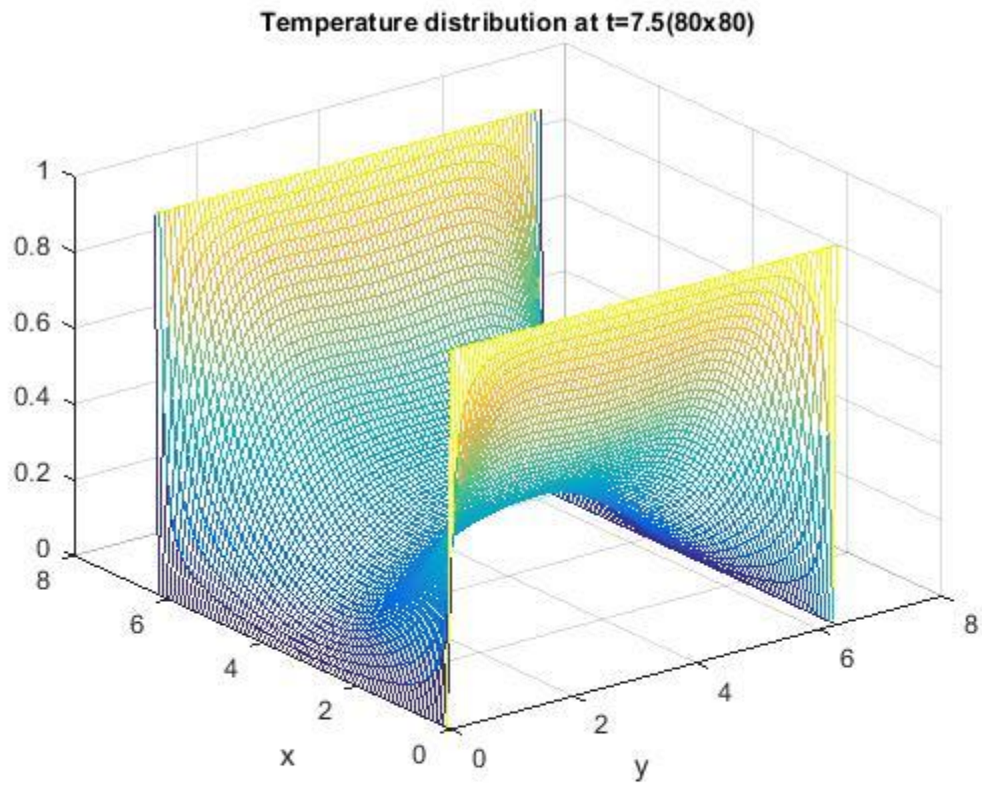


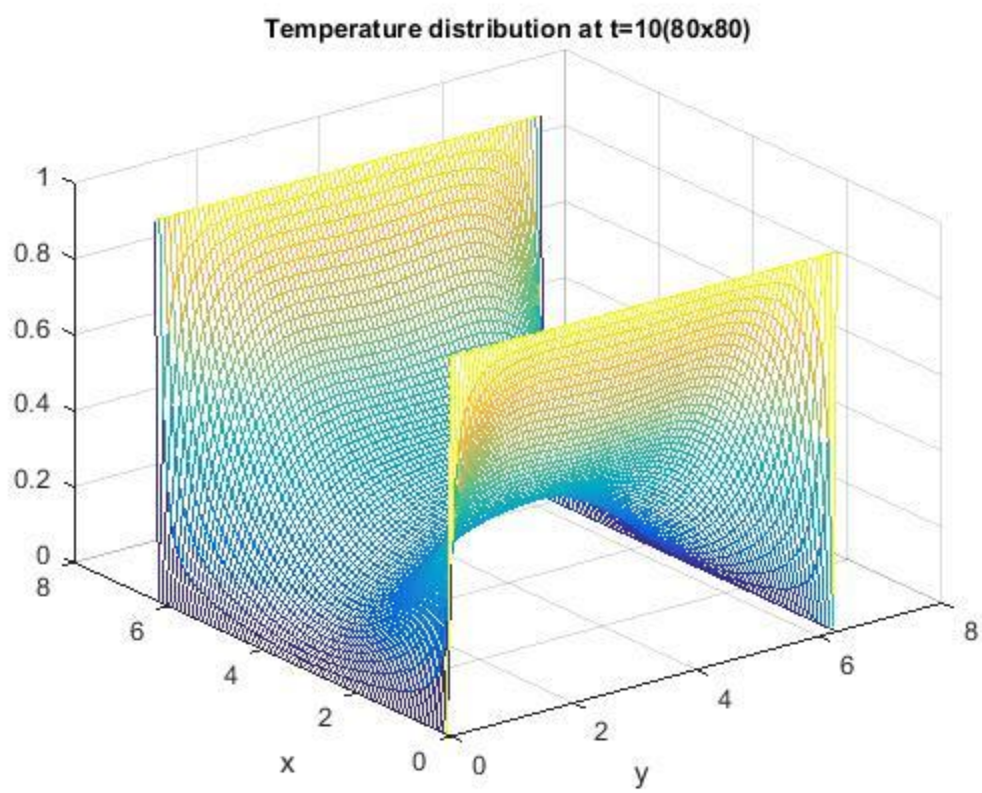
Temperature distribution at t=10(40x40)

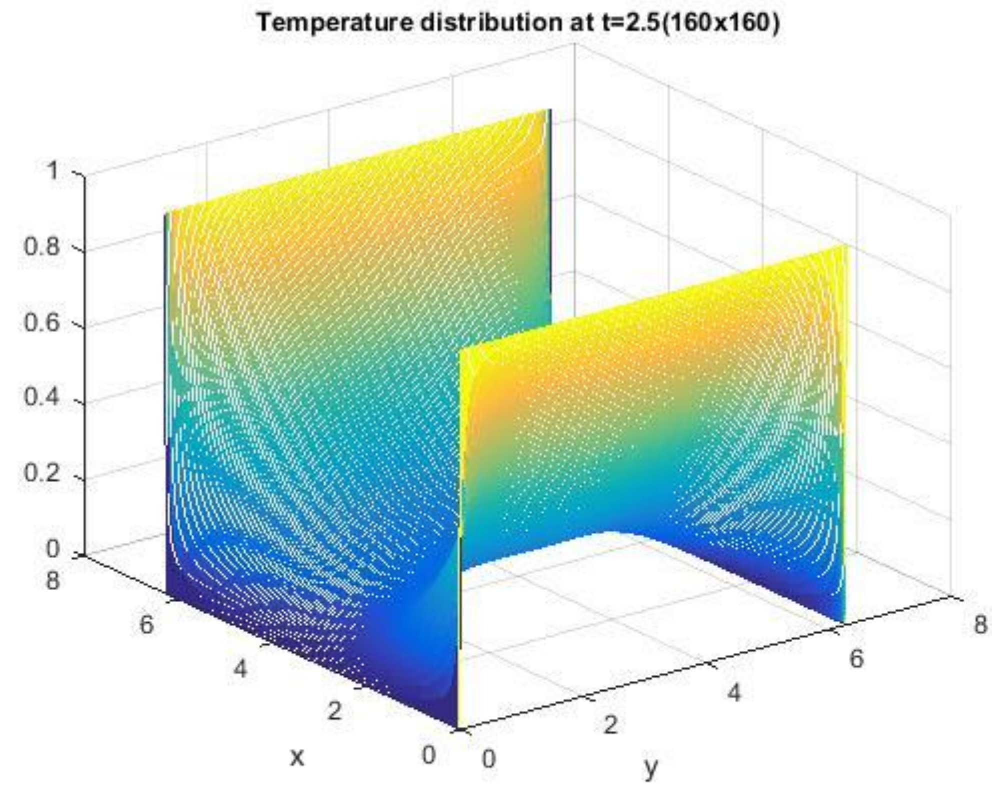


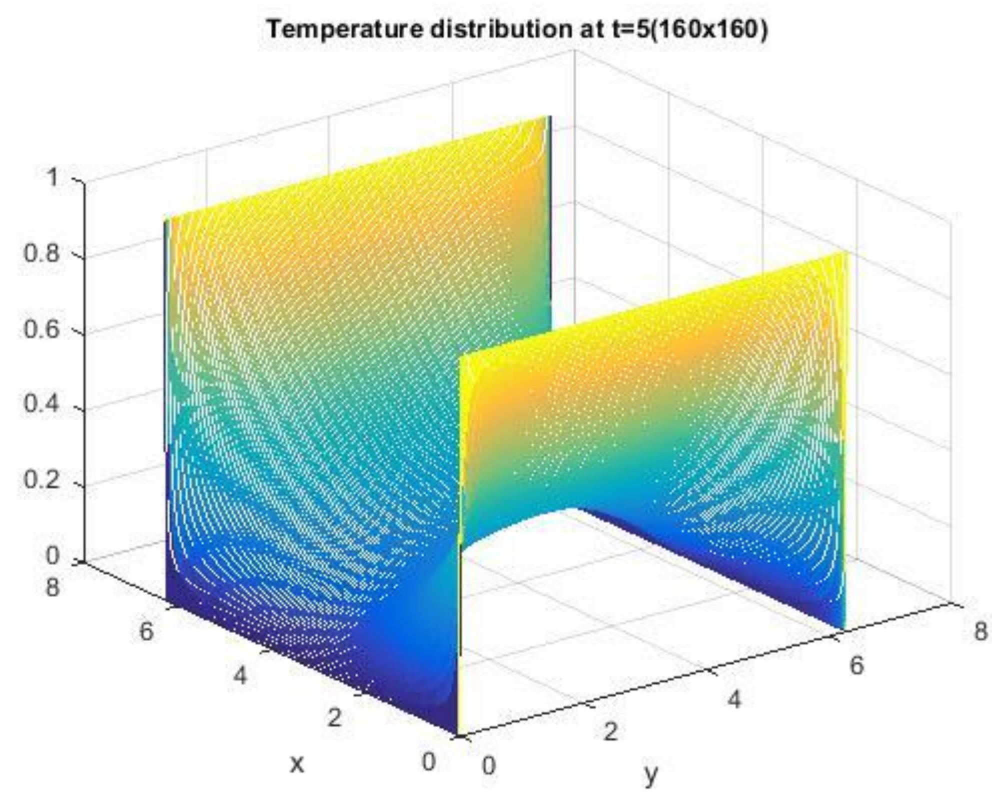

Temperature distribution at t=2.5(80x80)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
0
x
0
2
4
6
8
y


Temperature distribution at t=5(80x80)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
0
x
0
2
4
6
8
y

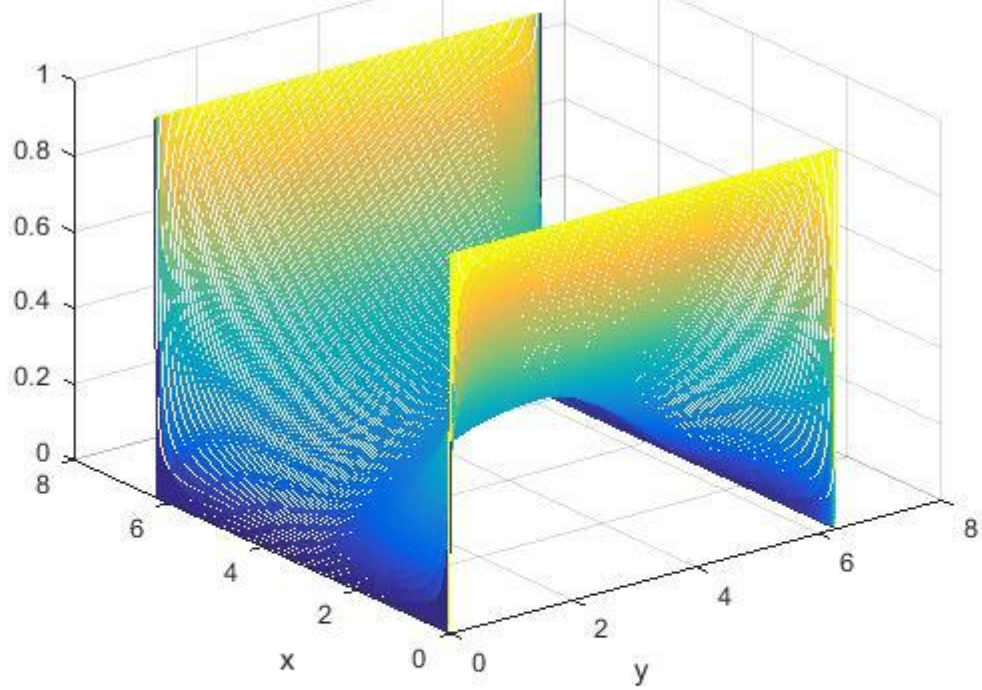

Temperature distribution at t=7.5(80x80)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
x
0
0
2
4
6
8
y


Temperature distribution at t=10(80x80)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
x
0
0
2
4
6
8
y

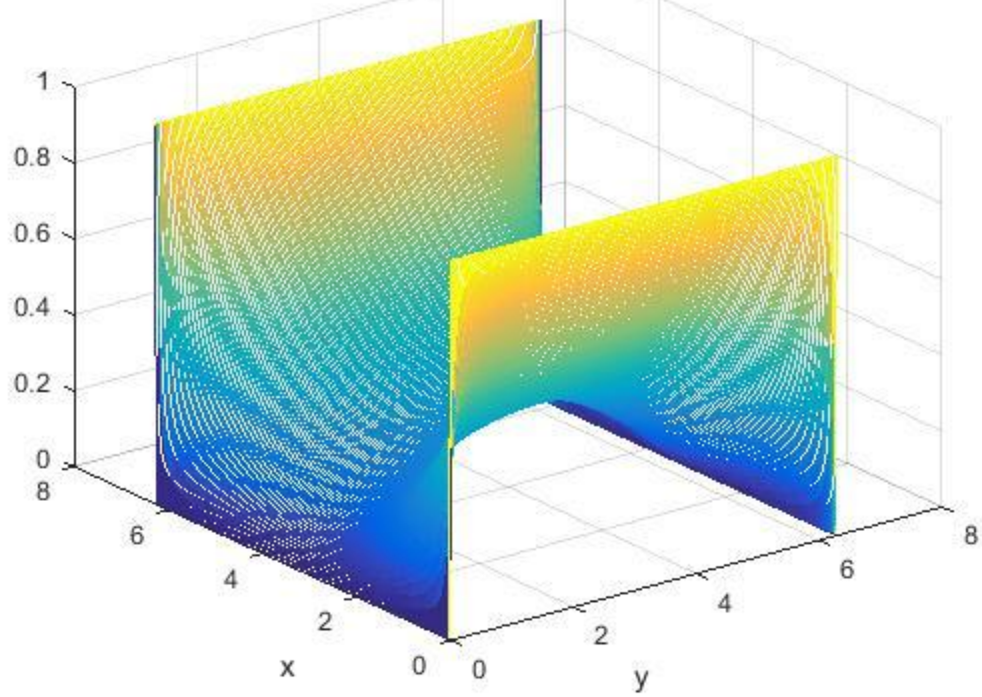

Temperature distribution at t=2.5(160x160)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
0
x
0
2
4
6
8
y

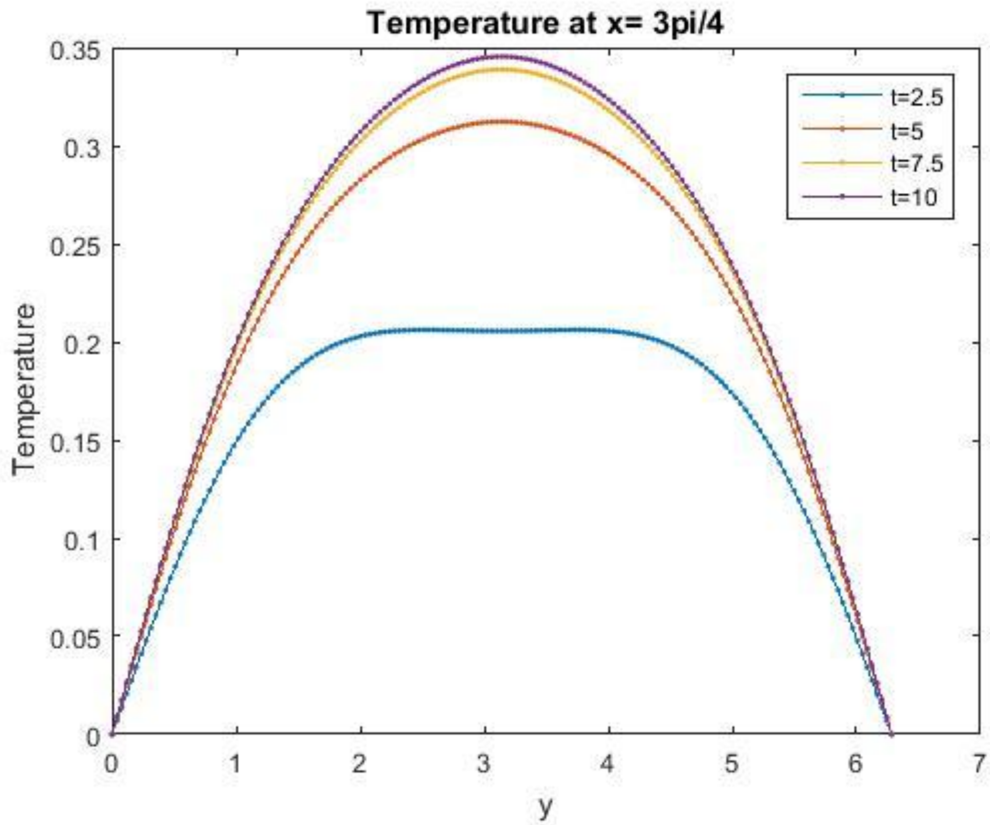

Temperature distribution at t=5(160x160)
1
0.8
0.6
0.4
0.2
0
8
6
4
2
0
x
0
2
4
6
8
y

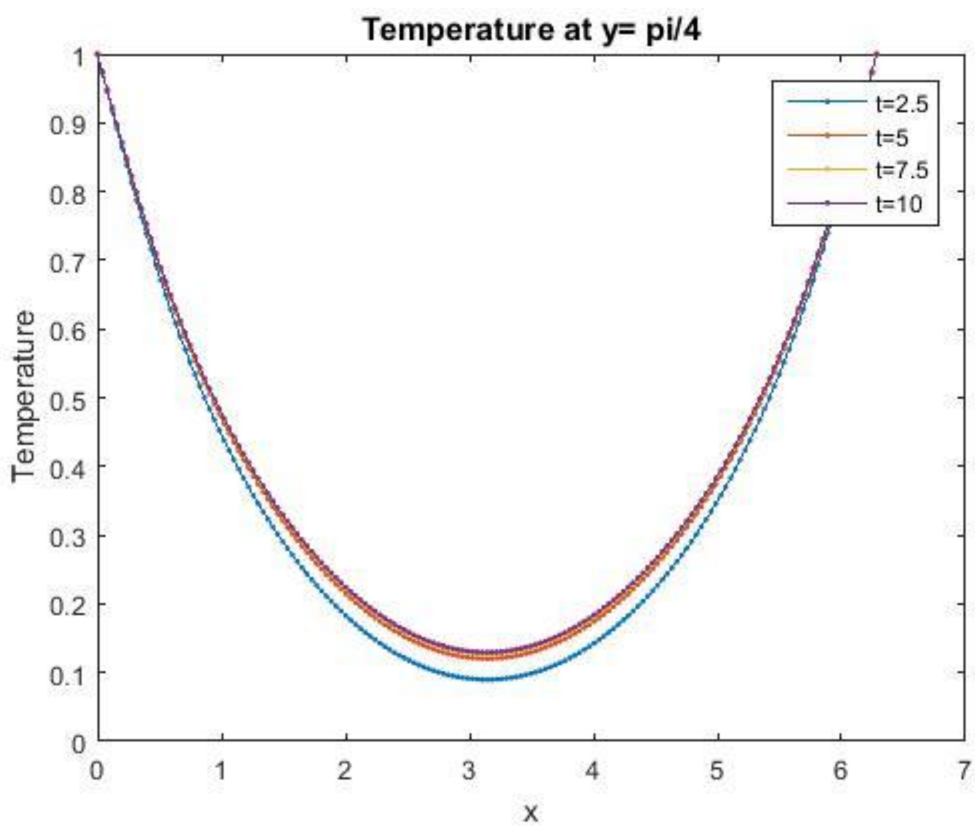
Temperature distribution at t=7.5(160x160)



Temperature distribution at t=10(160x160)




Temperature at x= 3pi/4
Temperature
y
t=2.5
t=5
t=7.5
t=10
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0
1
2
3
4
5
6
7


Temperature at y= pi/4
Temperature
x
t=2.5
t=5
t=7.5
t=10
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
1
2
3
4
5
6
7

Results and Discussion

Analysis of plots
The behavior of the temperature distribution with respect to time can be observed clearly in the plots for each grid resolution. The scheme is consistent and stable in the range of grid resolution taken and hence gives plots with consistent and stable values. It can be observed that as time increases, the diffusion term becomes dominant nearer the centre and temperature field rises up. But nearer the walls, the conduction term is dominant and the temperature stays close to the boundary values. This can be also seen when observing the temperature distribution along x=3pi/4 and y=pi/4. As the time increases, it can also be seen that temperature distribution plots tends to converge to a steady state value.

Error Analysis

<u>AB2 Method</u>

The code was run for grid sizes of 10, 20 40, 80 and dt=0.0005. The scheme was stable and the following L2 norms of errors were obtained assuming 80 to be the exact solution.
E10 = 0.006535246586311
E20 = 0.002102308949674
E40 = 5.240315064309386e-04

The grid resolution was further improved to 20, 40, 80, 160 and the code was run for the same dt. But the solution is unstable for N=160. The dt value was further reduced until stable result was obtained at dt= 0.0001. Following errors obtained assuming 160 to be exact solution
E20 = 0.002218985351091
E40 = 6.686055290228662e-04
E80 = 2.004299242970253e-04

<u>RK4</u>

The code was run for grid sizes of 10, 20 40, 80 and dt=0.0005. The scheme was stable and the following L2 norms of errors were obtained assuming 80 to be the exact solution.
E10 = 0.006535246702314
E20 = 0.002102309003005
E40 = 5.240315191225178e-04

The grid resolution was further improved to 20, 40, 80, 160 and the code was run for the same dt. Solution is found to be stable for N=160. Following errors obtained assuming 160 to be exact solution
E20 = 0.002218986848329
E40 = 6.686057242849385e-04
E80 = 2.004299314599854e-04

Error is becoming of the order of 10^{-4} at higher grid resolution for both the methods. This means solution is converging and it is becoming independent of the grid. It is also seen that RK4 gives stable solution at higher *dt* values due to high stability region. It is also computationally less demanding than AB2 for similar accuracy. AB2 requires smaller time steps for stability and hence is time consuming and resource intensive. The order of both the schemes when the higher grid resolution of 160 is considered comes to about 1.78. This is the true order of accuracy of the schemes when the parameters are chosen as given here.

For both the methods used here, the stability region is small and hence the computational power required to get fast solutions are very high. This is because very low values of *dt* are required to keep the scheme stable. Hence a method with higher stability region would enable the use of larger *dt* values and hence gives a faster solution. Hence implicit schemes like Crank Nicolson can be used for time integration so as to improve the speed for same accuracy requirement.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%CFD Project 4
%Program 1
%Transient advection diffusion equation
%Central difference second order in space
%AB2 in time
%
%Author: Jithin Gopinadhan
%Date : 11/30/2015
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
clear all

L=2*pi();            %Length of domain
N=[20 40 80 160];    %Grid sizes
Time=10;             %Maximum time for simulation
dt=0.0001;           %Delta t
M=Time/dt;           %Number of iterations in time

%Initializing error matrices for different grid sizes
Err20=zeros(N(1)-1);
Err40=zeros(N(2)-1);
Err80=zeros(N(3)-1);
%Initializing matrix to save data at x=3pi/4 and y=pi/4
Pos_data_x=zeros(5,N(4)+1);
Pos_data_y=zeros(5,N(4)+1);

%This loop runs for the different grid
%sizes specified above
for grid=1:4

    h=L/N(grid);     %Symmetric grid: delta x = delta y = h

    %Initializing temperature and its time derivative
    %for two steps in time
    Tn=zeros(N(grid)+1);     %Temperature at nth time
    Tnp1=zeros(N(grid)+1);   %Temperature at (n+1)th time
    dTn=zeros(N(grid)+1);    %dT/dt value at current time step
    dTnm1=zeros(N(grid)+1);  %dT/dt value at previous time step

    %Initializing boundary conditions
    for i=1:N(grid)+1
        Tn(1,i)=1;
        Tn(N(grid)+1,i)=1;
    end

    %Time loop
    for t=0:dt:Time
        %Spatial loops
        for i=1:N(grid)+1
            for j=1:N(grid)+1
                x=h*(i-1);
                y=h*(j-1);
                u=sin(x)*cos(y); %Velocity components
                v=-cos(x)*sin(y);
```

```
                %Interior points
                if(i>1 && i<(N(grid)+1) && j>1 && j<(N(grid)+1))
                    %Evaluation of individual terms
                    dTx= (Tn(i+1,j)-Tn(i-1,j))/(2*h);
                    dTy= (Tn(i,j+1)-Tn(i,j-1))/(2*h);
                    d2Tx= (Tn(i+1,j)- 2*Tn(i,j)+Tn(i-1,j))/(h^2);
                    d2Ty= (Tn(i,j+1)- 2*Tn(i,j)+Tn(i,j-1))/(h^2);
                    %Evaluation of dT/dt
                    dTn(i,j) = -u*dTx - v*dTy + d2Tx + d2Ty;
                end
            end
        end %End of spatial loops

        %Using first order for t=0
        if(t==0)
            Tnp1 = Tn + (dTn *dt);
        end
        %Using AB2 for all other
        if(t>0)
            Tnp1 = Tn + (0.5*dt)*(3* dTn - dTnm1);
        end

        %Assigning T and dT/dt values to correct variable
        %before stepping forward in time
        Tn=Tnp1;
        dTnm1=dTn;

        %Plotting temperature profile for required times
        if(t==2.5 || t==5 || t==7.5 || t==10)
            [X,Y] = meshgrid(0:h:2*pi());
            figure,mesh(X,Y,Tnp1)
            s1=num2str(t);
            s2=num2str(N(grid));
            title(['Temperature distribution at t=',s1,'(',s2,'x',s2,')']
,'FontSize',10)
            xlabel('y');
            ylabel('x');
        end

        %Evaluation of T along x=3*pi/4 and y=pi/4
        %for required times at highest grid resolution
        if(grid==4 && (t==2.5 || t==5 || t==7.5 || t==10))
            T80=Tn;
            i_pos=(3*N(grid) +8)/8; %3pi/4
            j_pos=(N(grid) +8)/8;   %pi/4
            x_pos=3*pi()/4;
            y_pos=pi()/4;
            %Finding index of next highest nodes
            i_ceil=ceil(i_pos);
            j_ceil=ceil((j_pos));
            %Finding position of adjacent nodes
            x1= (i_ceil-2)*h;
            x2= (i_ceil-1)*h;
            y1= (j_ceil-2)*h;
            y2= (j_ceil-1)*h;
```

```
            for q=1:N(grid)+1
                %Linear interpolation of temperature based on temperature
                %values of adjacent nodes
                T_xpos=T80(i_ceil-1,q)+ (T80(i_ceil+1,q)-T80(i_ceil-
1,q))*((x_pos-x1)/h);
                T_ypos=T80(q,j_ceil-1)+ (T80(q,j_ceil+1)-T80(q,j_ceil-
1))*((y_pos-y1)/h);
                len=(q-1)*h;
                %Saving data for plotting later
                Pos_data_x(q,1)=len;
                Pos_data_x(q,(t/2.5)+1)=T_xpos;
                Pos_data_y(q,1)=len;
                Pos_data_y(q,(t/2.5)+1)=T_ypos;

            end
        end

    end %End of time loop

    %Saving values at t=10 for grid refinement study
    if(grid==1)
        T20=Tn;
    end
    if(grid==2)
        T40=Tn;
    end
    if(grid==3)
        T80=Tn;
    end
    if(grid==4)
        T160=Tn;
    end
end

%Evaluation of errors of 3 lower grid refinements against
%highest grid refinemnt assuming it to be exact
for i=1:N(1)-1
    for j=1:N(1)-1
        Err20(i,j)=T160((1+i*8),(1+j*8))-T20(i+1,j+1);
    end
end
for i=1:N(2)-1
    for j=1:N(2)-1
        Err40(i,j)=T160((1+i*4),(1+j*4))-T40(i+1,j+1);
    end
end
for i=1:N(3)-1
    for j=1:N(3)-1
        Err80(i,j)=T160((1+i*2),(1+j*2))-T80(i+1,j+1);
    end
end

%Evaluation of L2 norm for errors
E20=norm(Err20)/(N(1)-1); E40=norm(Err40)/(N(2)-1); E80=norm(Err80)/(N(3)-1);
%Evaluation of order of convergence
```

```
order=log((E80-E40)/(E40-E20))/log(0.5)

%Plotting temperature profiles along x=3pi/4
figure,plot(Pos_data_x(:,1),Pos_data_x(:,2),'.-')
hold on
plot(Pos_data_x(:,1),Pos_data_x(:,3),'.-')
hold on
plot(Pos_data_x(:,1),Pos_data_x(:,4),'.-')
hold on
plot(Pos_data_x(:,1),Pos_data_x(:,5),'.-')
title('Temperature at x= 3pi/4','FontSize',12)
xlabel('y');
ylabel('Temperature');
legend('t=2.5','t=5','t=7.5','t=10')

%Plotting temperature profiles along y=pi/4
figure,plot(Pos_data_y(:,1),Pos_data_y(:,2),'.-')
hold on
plot(Pos_data_y(:,1),Pos_data_y(:,3),'.-')
hold on
plot(Pos_data_y(:,1),Pos_data_y(:,4),'.-')
hold on
plot(Pos_data_y(:,1),Pos_data_y(:,5),'.-')
title('Temperature at y= pi/4','FontSize',12)
xlabel('x');
ylabel('Temperature');
legend('t=2.5','t=5','t=7.5','t=10')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%CFD Project 4
%Program 2
%Transient advection diffusion equation
%Central difference second order in space
%RK4 in time
%
%Author: Jithin Gopinadhan
%Date : 11/30/2015
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
clc

L=2*pi();            %Length of domain
N=[20 40 80 160];     %Grid sizes
Time=10;             %Maximum time for simulation
dt=0.0005;           %Delta t
M=Time/dt;           %Number of iterations in time

%Initializing error matrices for different grid sizes
Err20=zeros(N(1)-1);
Err40=zeros(N(2)-1);
Err80=zeros(N(3)-1);
%Initializing matrix to save data at x=3pi/4 and y=pi/4
Pos_data_x=zeros(5,N(4)+1);
Pos_data_y=zeros(5,N(4)+1);

%This loop runs for the different grid
%sizes specified above
for grid=1:4

    h=L/N(grid);%Symmetric grid: delta x = delta y = h
    %Initializing temperature and its time derivative
    %for two steps in time
    Tn=zeros(N(grid)+1);     %Temperature at nth time
    Tnp1=zeros(N(grid)+1);   %Temperature at (n+1)th time
    dTn=zeros(N(grid)+1);    %dT/dt value at current time step
    %Initializing matrices to store RK4 parameters
    F1=zeros(N(grid)+1);
    F2=zeros(N(grid)+1);
    F3=zeros(N(grid)+1);
    F4=zeros(N(grid)+1);
    K1=zeros(N(grid)+1);
    K2=zeros(N(grid)+1);
    K3=zeros(N(grid)+1);
    K4=zeros(N(grid)+1);
    %Initializing boundary conditions
    for i=1:N(grid)+1
        Tn(1,i)=1;
        Tn(N(grid)+1,i)=1;
    end

    %Time loop
    for t=0:dt:Time
        K1=Tn;%RK4 parameter
```

```
%Spatial loops to evaluate RK4 parameter F1
for i=1:N(grid)+1
    for j=1:N(grid)+1

        x=h*(i-1);
        y=h*(j-1);
        u=sin(x)*cos(y);     %Velocity components
        v=-cos(x)*sin(y);

        if(i>1 && i<(N(grid)+1) && j>1 && j<(N(grid)+1))
            dTx= (Tn(i+1,j)-Tn(i-1,j))/(2*h);
            dTy= (Tn(i,j+1)-Tn(i,j-1))/(2*h);
            d2Tx= (Tn(i+1,j)- 2*Tn(i,j)+Tn(i-1,j))/(h^2);
            d2Ty= (Tn(i,j+1)- 2*Tn(i,j)+Tn(i,j-1))/(h^2);
            %Evaluating predicted slope at t
            F1(i,j) = -u*dTx - v*dTy + d2Tx + d2Ty;
        end
    end
end
%Evaluating first predicted values at t+(dt/2)
K2= Tn + 0.5*dt*F1;

for i=1:N(grid)+1
    for j=1:N(grid)+1

    x=h*(i-1);
    y=h*(j-1);
    u=sin(x)*cos(y);
    v=-cos(x)*sin(y);

    if(i>1 && i<(N(grid)+1) && j>1 && j<(N(grid)+1))
        dTx= (K2(i+1,j)-K2(i-1,j))/(2*h);
        dTy= (K2(i,j+1)-K2(i,j-1))/(2*h);
        d2Tx= (K2(i+1,j)- 2*K2(i,j)+K2(i-1,j))/(h^2);
        d2Ty= (K2(i,j+1)- 2*K2(i,j)+K2(i,j-1))/(h^2);
        %Evaluating first predicted slope at t+(dt/2 at K2
        F2(i,j) = -u*dTx - v*dTy + d2Tx + d2Ty;
    end
    end
end

%Evaluating second predicted values at t+(dt/2)
K3=Tn+ 0.5*dt*F2;


for i=1:N(grid)
    for j=1:N(grid)

    x=h*(i-1);
    y=h*(j-1);
    u=sin(x)*cos(y);
    v=-cos(x)*sin(y);

        if(i>1 && i<(N(grid)+1) && j>1 && j<(N(grid)+1))
            dTx= (K3(i+1,j)-K3(i-1,j))/(2*h);
```

```
                dTy= (K3(i,j+1)-K3(i,j-1))/(2*h);
                d2Tx= (K3(i+1,j)- 2*K3(i,j)+K3(i-1,j))/(h^2);
                d2Ty= (K3(i,j+1)- 2*K3(i,j)+K3(i,j-1))/(h^2);
                %Evaluating second predicted slope at t+(dt/2) at K3
                F3(i,j) = -u*dTx - v*dTy + d2Tx + d2Ty;
            end
        end
    end
    %Evaluating predicted values at t+dt
    K4=Tn+ dt*F3;

        for i=1:N(grid)
            for j=1:N(grid)

            x=h*(i-1);
            y=h*(j-1);
            u=sin(x)*cos(y);
            v=-cos(x)*sin(y);

            if(i>1 && i<(N(grid)+1) && j>1 && j<(N(grid)+1))
                dTx= (K4(i+1,j)-K4(i-1,j))/(2*h);
                dTy= (K4(i,j+1)-K3(i,j-1))/(2*h);
                d2Tx= (K4(i+1,j)- 2*K4(i,j)+K4(i-1,j))/(h^2);
                d2Ty= (K4(i,j+1)- 2*K4(i,j)+K4(i,j-1))/(h^2);
                %Evaluating predicted slope at t+dt using K4
                F4(i,j) = -u*dTx - v*dTy + d2Tx + d2Ty;
            end
            end
        end
    %RK4 formula to evaluate Temperature at next time step
    Tnp1 = Tn + (dt/6) * (F1 + 2*F2 + 2*F3 + F4);

    %Assigning T values to correct variable
    %before stepping forward in time
    Tn=Tnp1;

    %Plotting temperature profile for required times
    if(t==2.5 || t==5 || t==7.5 || t==10)
        [X,Y] = meshgrid(0:h:2*pi());
        figure,mesh(X,Y,Tnp1)
        s1=num2str(t);
        s2=num2str(N(grid));
        title(['Temperature distribution at t=',s1,'(',s2,'x',s2,')']
,'FontSize',10)
        xlabel('y');
        ylabel('x');
    end

    %Evaluation of T along x=3*pi/4 and y=pi/4
    %for required times at highest grid resolution
    if(grid==4 && (t==2.5 || t==5 || t==7.5 || t==10))
        T80=Tn;
        i_pos=(3*N(grid) +8)/8; %3pi/4
        j_pos=(N(grid) +8)/8;   %pi/4
        x_pos=3*pi()/4;
        y_pos=pi()/4;
```

```
            %Finding index of next highest nodes
            i_ceil=ceil(i_pos);
            j_ceil=ceil((j_pos));
            %Finding position of adjacent nodes
            x1= (i_ceil-2)*h;
            x2= (i_ceil-1)*h;
            y1= (j_ceil-2)*h;
            y2= (j_ceil-1)*h;

            for q=1:N(grid)+1
                %Linear interpolation of temperature based on temperature
                %values of adjacent nodes
                T_xpos=T80(i_ceil-1,q)+ (T80(i_ceil+1,q)-T80(i_ceil-
1,q))*((x_pos-x1)/h);
                T_ypos=T80(q,j_ceil-1)+ (T80(q,j_ceil+1)-T80(q,j_ceil-
1))*((y_pos-y1)/h);
                len=(q-1)*h;
                %Saving data for plotting later
                Pos_data_x(q,1)=len;
                Pos_data_x(q,(t/2.5)+1)=T_xpos;
                Pos_data_y(q,1)=len;
                Pos_data_y(q,(t/2.5)+1)=T_ypos;
            end
        end

    end %End of time loop

    %Saving values at t=10 for grid refinement study
    if(grid==1)
        T20=Tn;
    end
    if(grid==2)
        T40=Tn;
    end
    if(grid==3)
        T80=Tn;
    end
    if(grid==4)
        T160=Tn;
    end
end

%Evaluation of errors of 3 lower grid refinements against
%highest grid refinemnt assuming it to be exact
for i=1:N(1)-1
    for j=1:N(1)-1
        Err20(i,j)=T160((1+i*8),(1+j*8))-T20(i+1,j+1);
    end
end
for i=1:N(2)-1
    for j=1:N(2)-1
        Err40(i,j)=T160((1+i*4),(1+j*4))-T40(i+1,j+1);
    end
end
for i=1:N(3)-1
    for j=1:N(3)-1
```

```
        Err80(i,j)=T160((1+i*2),(1+j*2))-T80(i+1,j+1);
    end
end

%Evaluation of L2 norm for errors
E20=norm(Err20)/(N(1)-1); E40=norm(Err40)/(N(2)-1); E80=norm(Err80)/(N(3)-1);
%Evaluation of order of convergence
order=log((E80-E40)/(E40-E40))/log(0.5)

%Plotting temperature profiles along x=3pi/4
figure,plot(Pos_data_x(:,1),Pos_data_x(:,2),'.-')
hold on
plot(Pos_data_x(:,1),Pos_data_x(:,3),'.-')
hold on
plot(Pos_data_x(:,1),Pos_data_x(:,4),'.-')
hold on
plot(Pos_data_x(:,1),Pos_data_x(:,5),'.-')
title('Temperature at x= 3pi/4','FontSize',12)
xlabel('y');
ylabel('Temperature');
legend('t=2.5','t=5','t=7.5','t=10')

%Plotting temperature profiles along y=pi/4
figure,plot(Pos_data_y(:,1),Pos_data_y(:,2),'.-')
hold on
plot(Pos_data_y(:,1),Pos_data_y(:,3),'.-')
hold on
plot(Pos_data_y(:,1),Pos_data_y(:,4),'.-')
hold on
plot(Pos_data_y(:,1),Pos_data_y(:,5),'.-')
title('Temperature at y= pi/4','FontSize',12)
xlabel('x');
ylabel('Temperature');
legend('t=2.5','t=5','t=7.5','t=10')
```