Numerical Methods of Thermo-Fluid Dynamics I

Winter Semester 2017-2018

DELIVERABLE TASK I:

Numerical Solution of 2D Heat Equation

Report by:

Jishnu Jayaraj (22448952)

Moradiya Tushar (22280590)

Chair of Fluid Mechanics
Department of Biochemical Engineering, Technical Faculty
Friedrich-Alexander University Erlangen-Nuremberg

CONTENTS

1.Introduction

The given problem is

$\frac{\partial w}{\partial t} = \frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2}$

[x,y] ϵ $[0,1]X[0,1]$, $t \epsilon$ $[0,0.016]$

The initial and boundary conditions are:

w(x,y,0)=0,

w(0,y,t)=1-y, w(1,y,t)=1-y2, w(x,0,t)=1, w(x,1,t)=0

We use Finite difference method in order to solve this conservation equation in differential form. The solution domain is covered by a grid. At each grid point, the differential equation is approximated by replacing the partial derivatives by approximations in terms of nodal values of function. The system will be approximated by a system of linear algebraic equations.

It can be solved by explicit or implicit method. In implicit method the solution is found by solving an equation involving both current state and later state of the system. While in explicit method the value at each node is found by using value from previous node.

1.1)Discretizing equation by CDS scheme in space and Crank-Nicolson method in time.

Discretizing in space

$$\frac{\partial^2 w}{\partial x^2} = \frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{\Delta x^2}$$

$$\frac{\partial^2 w}{\partial y^2} = \frac{w_{i,j+1} - 2w_{i,j} + w_{i,j-1}}{\Delta y^2}$$

Discretisation in time

$$\frac{\partial w}{\partial t} = \frac{w_{i,j}^{n+1} - w_{i,j}^{n}}{\Delta t}$$

Since it is a square grid $\Delta x = \Delta y$

Substituting these values in heat eqution we get

$$\frac{w_{i,j}^{n+1} - w_{i,j}^{n}}{\Delta t} = \frac{1}{2(\Delta x)^2} [w_{i+1,j}^{n+1} + w_{i-1,j}^{n+1} + w_{i,j+1}^{n+1} + {}_{i,j-1}^{n+1}w - 4w_{i,j}^{n+1} + w_{i+1,j}^{n} + w_{i-1,j}^{n} + w_{i,j+1}^{n} + w_{i,j-1}^{n} - 4w_{i,j}^{n}]$$

$$w_{i,j}^{n+1} - \frac{\Delta t}{2\Delta x^2}\left[w_{i+1,j}^{n+1} + w_{i-1,j}^{n+1} + w_{i,j+1}^{n+1} + {}_{i,j-1}^{n+1}w - 4w_{i,j}^{n+1}\right] =$$
$$w_{i,j}^{n} + \frac{\Delta t}{2\Delta x^2}\left[w_{i+1,j}^{n+1} + w_{i-1,j}^{n+1} + w_{i,j+1}^{n+1} + {}_{i,j-1}^{n+1}w - 4w_{i,j}^{n+1}\right]$$

1.2)Condition under which Crank-Nicolson method is stable

Von Neumann(Fourier approach)

A very versatile tool for analysing stability is the Fourier method developed by von Neumann. Here initial values at mesh points are expressed in terms of a finite Fourier series, and we consider the growth of individual Fourier components. Using this technique we can compute the stability of finite difference schemes as applied to partial differential equation.

On solving a partial differential equation we are actually getting an approximate solution and there is always an error associated with result (which is equal to the truncated term in taylor series expansion).

$w_i^n = \overline{w_i^n} + \varepsilon_i^n$

If we decompose numerical solution to Fourier harmonics on spatial grid

$$u_m = \xi^j(k)e^{ikn\Delta x}$$

Where u_m = amplitude
k = wave no: of mode
$\xi^j(k)$ = amplification factor

For the given heat equation at $t = t_n$, let $w_{i,j}^n = e^{ik(x+y)}$
And for $t = t_{n+1}$, $w_{i,j}^{n+1} = \lambda e^{ik(x+y)}$
Where $\lambda =$ amplification factor of the error

Substituting this in the given heat equation

$$\lambda.e^{ik(x+y)} - \frac{\lambda\Delta t}{2\Delta x^2}\left[e^{ik(x+\Delta x+y} + e^{ik(x-\Delta x+y} + e^{ik(x+y+\Delta y} + e^{ik(x+y-\Delta y} - 4.e^{ik(x+y)}\right] =$$

$$e^{ik(x+y)} + \frac{\Delta t}{2\Delta x^2}\left[e^{ik(x+\Delta x+y} + e^{ik(x-\Delta x+y} + e^{ik(x+y+\Delta y} + e^{ik(x+y-\Delta y} - 4e^{ik(x+y)}\right]$$

$$\lambda\left[1 - \frac{\Delta t}{2\Delta x^2}\left(e^{ik\Delta x} + e^{-ik\Delta x} + e^{ik\Delta y} + e^{-ik\Delta y} - 4\right)\right]$$
$$= \left[\frac{\Delta t}{\Delta x^2}\left(e^{ik\Delta x} + e^{-ik\Delta x} + e^{ik\Delta y} + e^{-ik\Delta y} - 4\right)\right]$$

We know $\Delta x = \Delta y$ *and* $e^{ik\Delta x} = \cos(k\Delta x) + i\sin(k\Delta x)$

$$\lambda\left[1-\frac{\Delta t}{\Delta x^2}(4\cos k\Delta x-4)\right]=\left[1+\frac{\Delta t}{\Delta x^2}(4\cos k\Delta x-4)\right]$$

$$\lambda\left[1+\tfrac{4\Delta t}{\Delta x^2}sin^2(k\Delta x/2)\right]=\left[1-\tfrac{4\Delta t}{\Delta x^2}sin^2(k\Delta x/2)\right]$$

$$\lambda=\frac{1-2.\Delta t/\Delta x^2\,.\,sin^2(k\Delta x/2)}{1+2.\Delta t/\Delta x^2\,.\,sin^2(k\Delta x/2)}$$

λ is the ratio of amplitude of the errors between two successive time steps and is a measure of change in error after each iteration. From equation we can observe that numerator is always smaller than denominator, thus $\lambda < 1$,always

Therefore we can conclude Crank Nicolson method is unconditionally stable.

1.3) Convergence of solution when grid is refined

Lax equivalence theorem states that in a given property posed linear initial value problem and a consistent finite difference scheme, stability is the requirement of convergence.

Consistency + stability = convergence

A method is said to be consistent if the truncation error tends to zero when the mesh spacing $\Delta t \to 0\ or\ \Delta x \to 0$. Truncation error is usually proportional to a power of the grid spacing $\Delta x\ or$ time step Δt. When we increase the grid spacing (h/2) and substitute it in the Taylor series expansion, we can see that the order of the error term is increased which means the algebraic equation is more stable than that of previous grid spacing.

2. MATLAB codes for Crank-Nicolson and Explicit Euler scheme

The codes were programmed assuming $\Delta t = 0.0001$ and maximum t value as 0.16 in both methods. Because in Explicit Euler the equation will be stable for only particular values of Δt, given by the equation $\Delta t < \Delta x^2/4$. Which is only satisfied by t= 0.0001.
Though Crank Nicolson is unconditionally stable, for the ease of comparison of two methods we have taken the same time step in both cases.

The MATLAB codes are given in appendix and also the .m files are attached within the file

3. Results and Observations

3.1) Temperature evolution

We have considered a point x=y=0.4 and have plotted the variation of temperature with time at this particular point. From the plot we can infer that temperature variation is a smooth curve starting from 0 and finally reaches a steady state value. From the slope of the curve we can conclude that Crank Nicolson converges faster because of its higher slope.

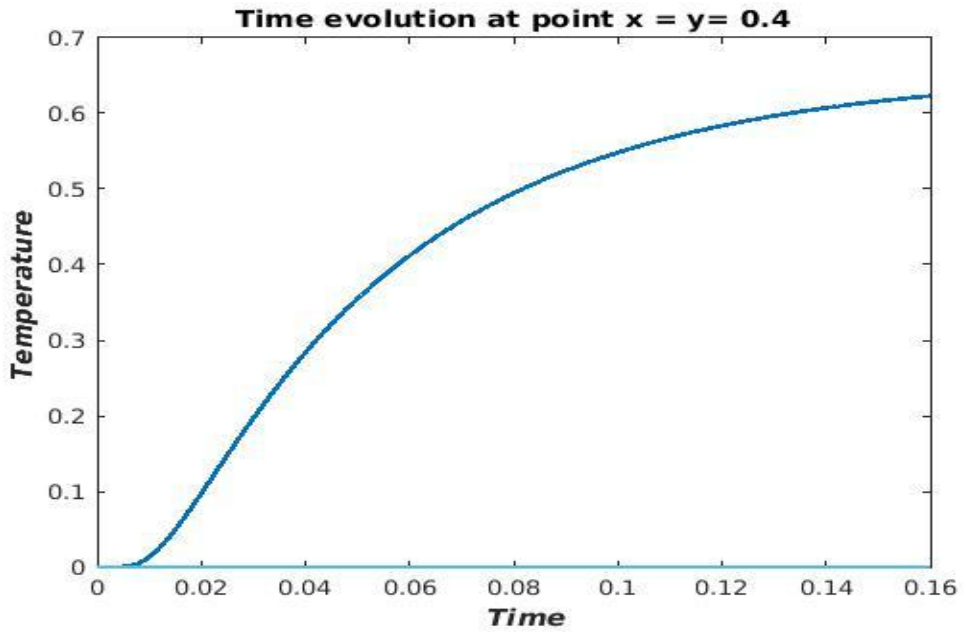

Fig 3.1.1: Time evolution of temperature at x=y=0.4 with Δt=0.0001 using Explicit Euler method

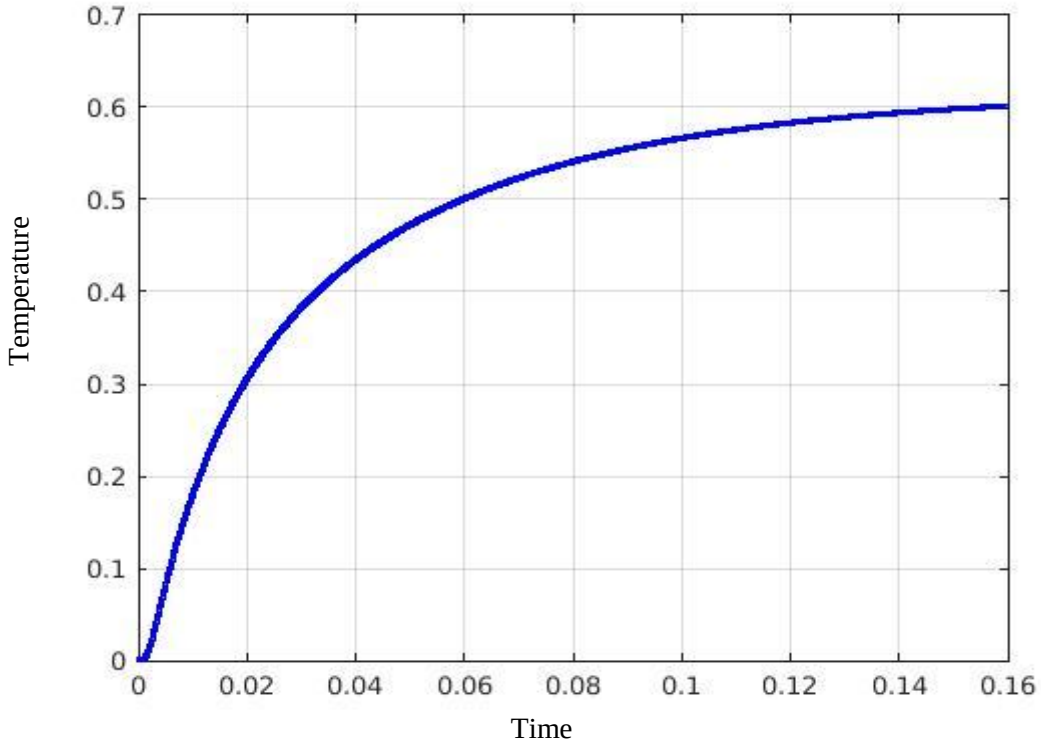

Fig 3.1.2 : Time evolution of temperature at x=y=0.4 with Δt=0.0001 using Crank-Nicolson method

3.2) Vertical temperature profile

Here we have plotted the variation of temperature for a vertical profile x=0.4. The variation is linear starting from maximum at y=0 and then reducing at constant rate and finally reaches 0 at y =1. For the both the schemes the result is more or less the same.

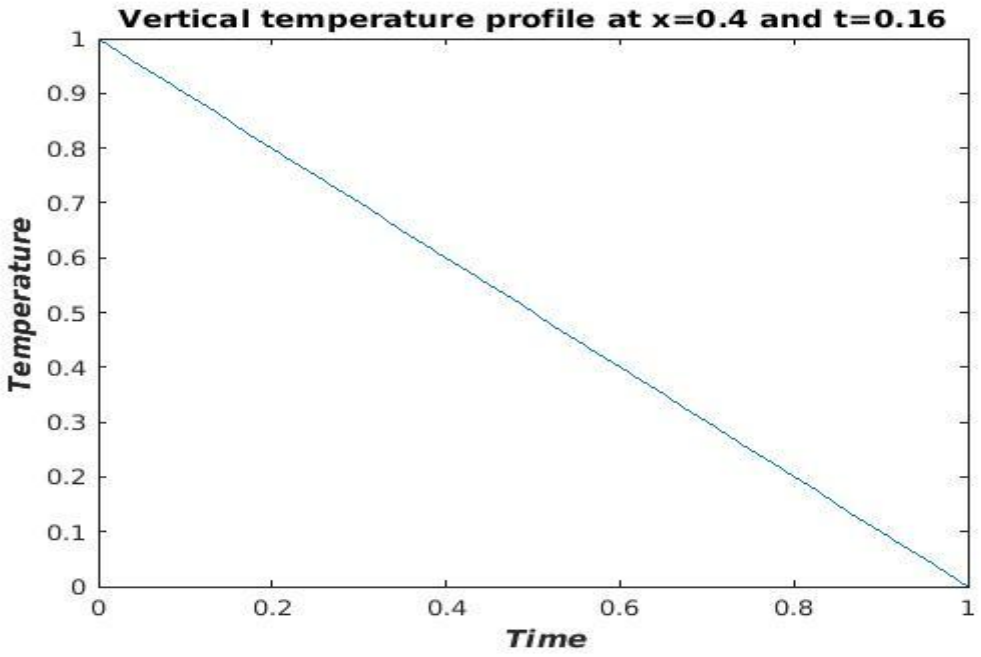

Fig 3.2.1: Vertical Temperature profile at x=0.4 and t=0.16 using Explicit Euler method

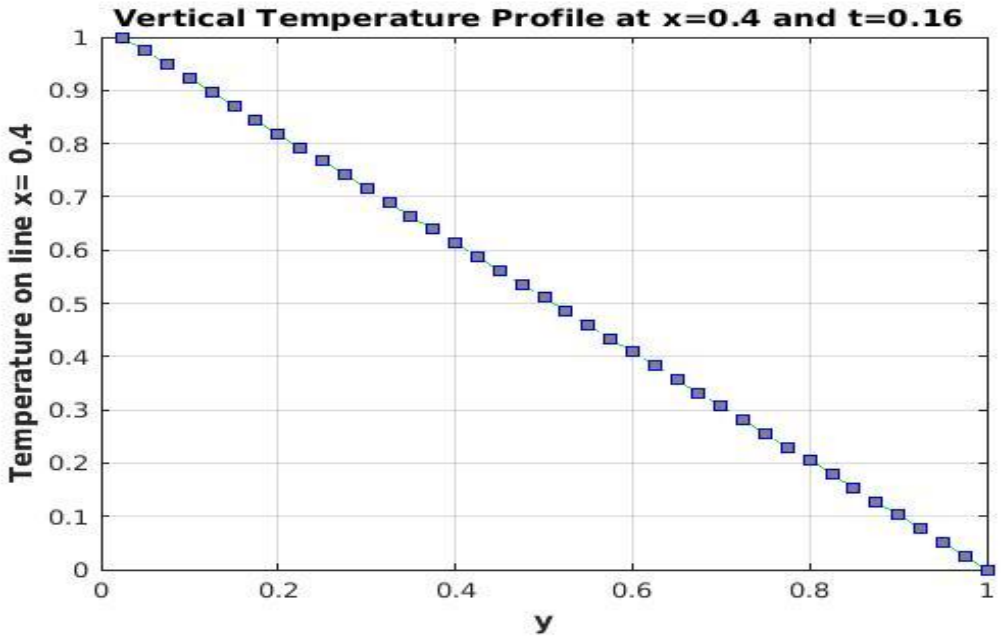

Fig 3.2.2 Vertical Temperature profile at x=0.4 and t=0.16 using Crank-Nicolson method

3.3) Performance and accuracy

If you compare both methods on the performance we can see that Crank-Nicolson method converges and maintains stability for all time steps. Whereas Explicit Euler methods converges only for particular value of Δt, which satisfies the condition$\Delta t < \Delta x^2/4$.

But considering the computational time, Crank Nicolson takes more time since it is more computational intensive compared to Explicit Euler. But this can be overcome by increasing the time step(since it is unconditionally stable). So on higher time steps the average computational timing for Crank Nicolson is more reliable than Explicit Euler.

4.) Plot of numerical solution for whole domain in progressive time

The contour lines shows the diffusion phenomena when we apply the given boundary conditions. The maximum heat is produced in bottom of the square grid and it get diffused to neighboring points satisfying 2D the heat equation. At the beginning we can see that width of the contour is short which means it's still there's a temperature gradient and the system is in transient state. But gradually the width gets larger showing that the system is approaching a steady state.

Since both the results are almost equal and for the time scale Δt =0.0001, as Euler method is more efficient(for this particular time scale only)we have provided here the result from Euler method in different time steps.

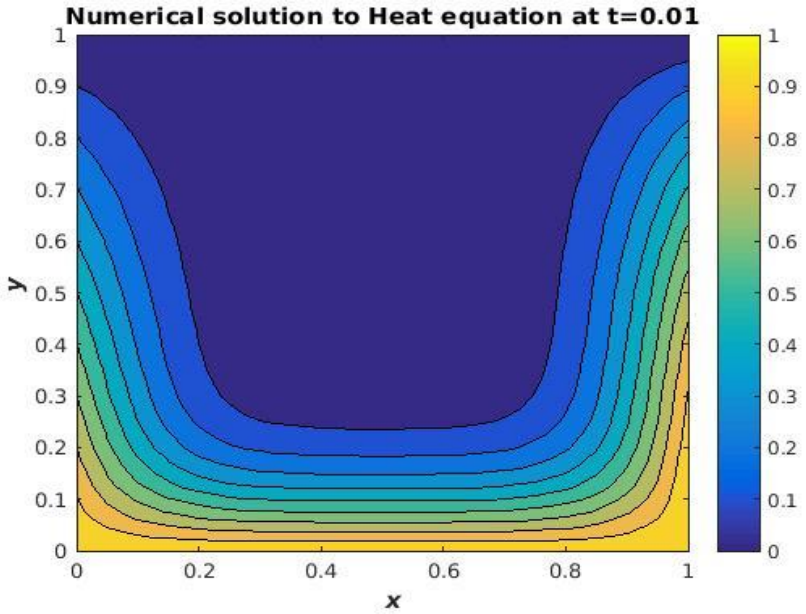

Fig 4.1: Temperature contour plot at t=0.01 using Crank-Nicolson at time step $\Delta t = 0.0001$

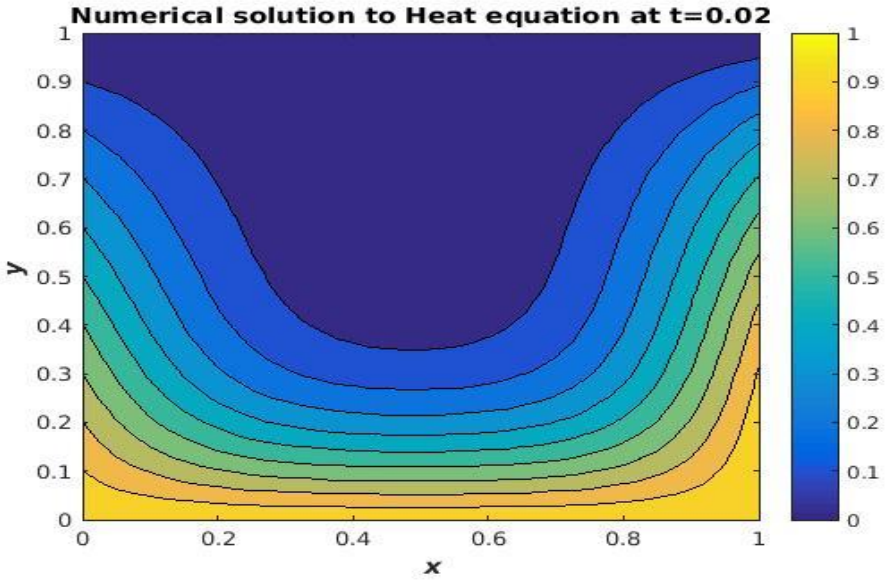

Fig 4.2: Temperature contour plot at t=0.02 using Crank-Nicolson at time step $\Delta t = 0.0001$

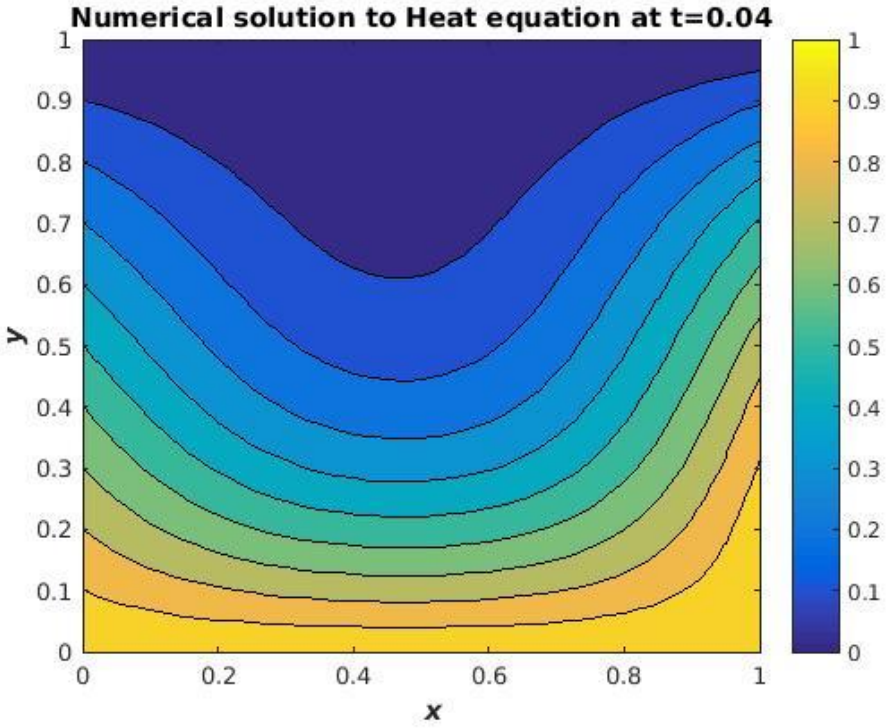

Fig 4.3: Temperature contour plot at t=0.04 using Crank-Nicolson at time step $\Delta t = 0.0001$

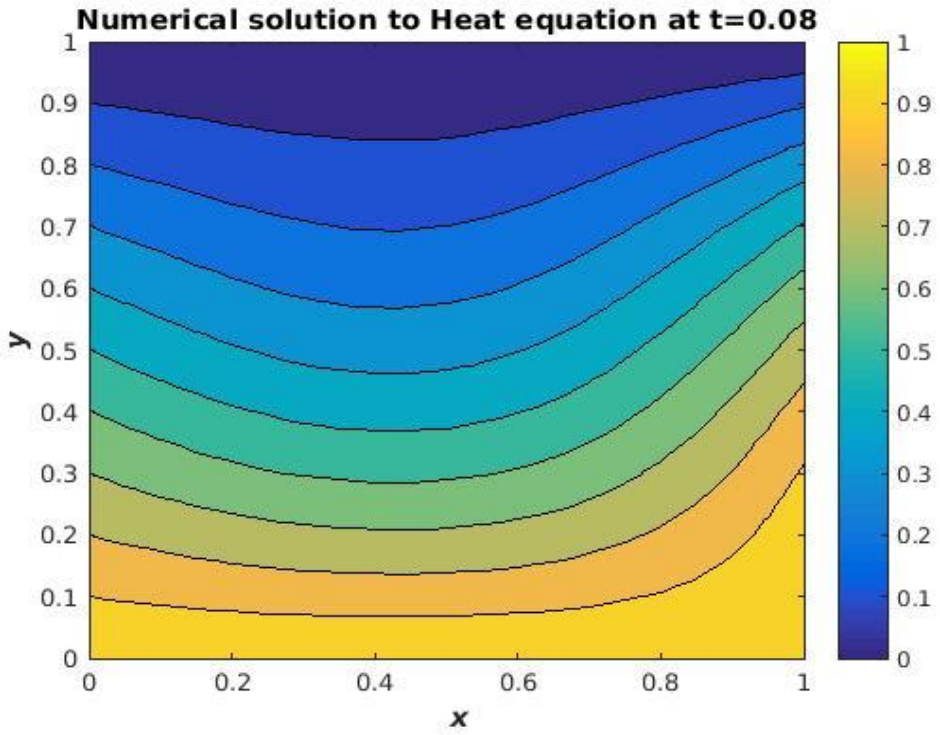

Fig 4.4: Temperature contour plot at t=0.08 using Crank-Nicolson at time step $\Delta t = 0.0001$

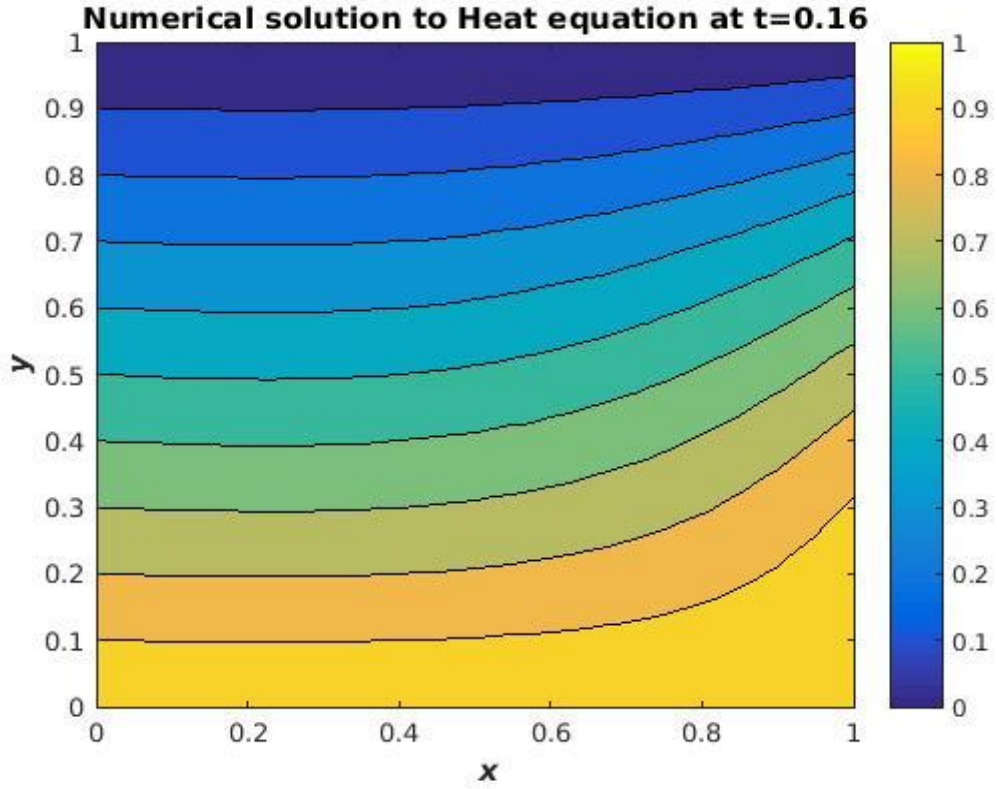

Fig 4.1: Temperature contour plot at t=0.16 using Crank-Nicolson at time step $\Delta t = 0.0001$

Conclusion

If the Δt is small enough and high accuracy is not required Explicit methods are best suited. But if we want high accuracy (truncation error is second order) and also if considerably larger Δt is required Crank Nicolson method is suited more irrespective of its high computational complexity.

APPENDIX

```
%% EULER METHOD

% Solution of Heat equation in 2d using Explicit method

%% Parameters to define heat equation and define range in space and time

% x is length, y is width , t is time
% square grid x = y =(0 to 1)
L = 1; % length of x and y coordinates

%% Parameters to solve equation by Explicit method

% Since Explicit methods are stable only in small magnitude of dt,
% given by the relation dt/(dx)^2, in the given set of time intervals
% only t=0.0001 is stable

maxt = 0.16;
dt = 0.0001 ;  % stable only when t = 0.0001
nt = maxt/dt;  % number of time step
nx = 41;       % no of grid points in x direction
ny = 41;       % no of grid points in y direction
dx = L/(nx-1);
dy = L/(ny-1);
b = dt/(dx)^2 ;

% to store the index of grid point at x = y =0.4
gridx = 0.4/dx ;
gridy = 0.4/dy ;

%% Initial and Boundary conditions

t = 0:dt:0.16;
x = linspace(0,1,41);
y = linspace(0,1,41);


w = zeros(nx,ny);       % matrix to store all grid point values
wnew = zeros(nx,ny);
w(1,:)   = 1;
w(nx,:)  = 0;
w(:,1)   = 1-y;
w(:,nx)  = 1-y.*y;

wnew = w;

wpoint = zeros(nt,1);

%% Implimentation of explicit method
       % For each node points in n+1 time step, temperature is
       % calculated from previous time steps and stored
```

```
for time=0 : dt : 0.16

    for i=2:nx-1

        for  j=2:ny-1

         wnew(i,j)=w(i,j) + b*(w(i+1,j)+w(i-1,j)+w(i,j+1)+w(i,j-1)
                      - 4* w(i,j));

         if i== gridx && j== gridy  % to get the values of point x=y=0.4

             wpoint(fix((time*10000)+1),1)=w(i,j);

         end
        end
    end
    w = wnew;

    if time==0.01       % To plot temperature profile at time = 0.01
        figure(1);
        contourf(x,y,wnew);colorbar;title('2D Diffusion Heat equation')
        xlabel(' \it \bf x','FontSize',16)
        ylabel('\it \bf y','FontSize',16)
        title('Numerical solution to Heat equation at
t=0.01','FontSize',15,'FontWeight','bold')
    end

    if time==0.02      % To plot temperature profile at time = 0.02
        figure(2);
        contourf(x,y,wnew);colorbar;
        xlabel(' \it \bf x','FontSize',16)
        ylabel('\it \bf y','FontSize',16)
        title('Numerical solution to Heat equation at
t=0.02','FontSize',15,'FontWeight','bold')
    end

    if time==0.04     % To plot temperature profile at time = 0.04
        figure(3);
        contourf(x,y,wnew);colorbar;
        xlabel(' \it \bf x','FontSize',16)
        ylabel('\it \bf y','FontSize',16)
        title('Numerical solution to Heat equation at
t=0.04','FontSize',15,'FontWeight','bold')
    end

    if time==0.08     % To plot temperature profile at time = 0.08
        figure(4);
        contourf(x,y,wnew);colorbar;
        xlabel(' \it \bf x','FontSize',16)
        ylabel('\it \bf y','FontSize',16)
        title('Numerical solution to Heat equation at
t=0.08','FontSize',15,'FontWeight','bold')
    end

    if time==0.16     % To plot temperature profile at time =0.16
            figure(5);
            contourf(x,y,wnew);colorbar;
            xlabel(' \it \bf x','FontSize',16)
            ylabel('\it \bf y','FontSize',16)
```

```
            title('Numerical solution to Heat equation at
t=0.16','FontSize',15,'FontWeight','bold')
    end

end

% Time evolution of temperature at x = y = 0.4%wpoint = zeros(nx,ny);  %
matrix to store temperature values at x=y=0.4

figure(6);
plot(t',wpoint,'linewidth',2);
xlabel( 'Time')
ylabel('Temperature')
title('Time evolution at point x = y= 0.4')

% Vertical Temperature profile at x=0.4 and t =0.16

figure(7);
plot(y,w(:,17));
xlabel(' Time')
ylabel('Temperature')
title('Vertical temperature profile at x=0.4 and t=0.16')
```

```
% Crank-Nicolson method of solution

% Solving by Alternating Direction Implicit
%two-level solution method. This approach ensures that at each time
step,
%there are no more than three unknowns to solve.


% Initial conditions

space = 0.025;
time= 0.0001;
T=0.16;
t=T/time;

% Parameters to solve equation by Crank Nicolson

No_of_Grid = 40;        % Number of grid points
N_dim = No_of_Grid^2;
p = zeros(N_dim,1);     % Column matrix to be solved
q=p;

% variable to get the co ordinates of point x=y=0.4

Data= zeros(t,1);
m=0.4/space;
n=0.4/space;

% Boundary conditions initialisation

for i = 1:No_of_Grid-1
    q(No_of_Grid*i,1) = 0;
    q(No_of_Grid*i+1,1) = 1;

end

for i = 1:No_of_Grid
    q(i,1) = 1-(i*space);
    q(N_dim-No_of_Grid+i,1) = (1-((i*space)^2));
end

inv = Matrix_A(space,time);     % Function to evaluate inverse of matrix

% Evolution with time

for i = 1:t
    p = inv * Matrix_B(space,time) * q ;
    q=p;
    Data(i,1) = p(m*n,1);
end

% Converting the solution to matrix form
c=zeros(No_of_Grid,No_of_Grid);
for i = 1:No_of_Grid
    x = ((i-1)*No_of_Grid)+1;
    y = i*No_of_Grid;
    c(:,i)= p(x:y,1);
end
```

```
% Ploting the results

 figure(1)           % Ploting the contour of heat solution
 contourf(c,13);
 colorbar('Direction','reverse');
 xlabel('spatial coordinate x','FontWeight','bold');
 ylabel ('spatial coordinate y','FontWeight','bold');
 title('2D Diffusion Heat equation')

 figure(2)           % Plotting Temperature at point x=y=0.4
 x = time:time:T;
 xlabel('Time Step','FontWeight','bold');
 ylabel ('Temperature at (0.4, 0.4)','FontWeight','bold');
 title('Time evolution of temperature at x=y=0.4 ');
 plot(x,Data,'--gs','MarkerFacecolor',[0.5 0.5
0.5],'MarkerEdgecolor','b','MarkerSize',2);
 grid on;
 hold on;

 figure(3)   % Plotting Temperature distribution on vertical line x=0.4
at t=0.16
 info1 = c(:,16);
 x1 = space:space:1;
 plot(x1,info1,'--gs','MarkerFacecolor',[0.5 0.5
0.5],'MarkerEdgecolor','b','MarkerSize',5);
 xlabel('y','FontWeight','bold');
 ylabel ('Temperature on line x= 0.4 at t=0.16','FontWeight','bold');
 title('Vertical Temperature Profile at x=0.4 and t=0.16 ');
 grid on;
 hold on;
```

% Function to evaluate matrix A

```
function [ Ainv ] = Matrix_A(space,time)
No_of_Grid = 40; %%%%%%%%%Nuber of grid points
N_dim = No_of_Grid^2;
lmd = (time/(2*space^2));

A = zeros(N_dim,N_dim);
for i = 1:No_of_Grid
A(i,i) = 1;
end
for i = N_dim-No_of_Grid+1:N_dim
A(i,i) = 1;
end
A(No_of_Grid+1,No_of_Grid+1) = (1+4*lmd);
A(No_of_Grid+1,No_of_Grid+2) = -lmd;
A(No_of_Grid+1,(2*No_of_Grid)+1) = -lmd;
A(N_dim-No_of_Grid,N_dim-(2*No_of_Grid)) = -lmd;
A(N_dim-No_of_Grid,N_dim-No_of_Grid) = (1+4*lmd);
A(N_dim-No_of_Grid,N_dim-No_of_Grid-1) = -lmd;
for i = No_of_Grid+2:N_dim-No_of_Grid-1
A(i,i-1) = -lmd;
A(i,i) = (1+4*lmd);
A(i,i+1) = -lmd;
if(No_of_Grid+1<i && i<=(N_dim-No_of_Grid))
A(i,No_of_Grid+i) = -lmd;
```

```
end
if(No_of_Grid<i && i<=(N_dim-No_of_Grid))
A(i,i-No_of_Grid) = -lmd;
end
end
for i = 1:No_of_Grid
A(No_of_Grid*i,:)=0;
A(No_of_Grid*i,No_of_Grid*i)=1;
end
for i = 0:No_of_Grid-1
A(No_of_Grid*i+1,:)=0;
A(No_of_Grid*i+1,No_of_Grid*i+1)=1;
end
Ainv = A^-1;

end
```

% function to evaluate matrix B

```
function [ Bn ] = Matrix_B(space,time )

No_of_Grid = 40; %%%%%%%%%Nuber of grid points
N_dim = No_of_Grid^2;
lmd = (time/(2*space^2));
B = zeros(N_dim,N_dim);
for i = 1:No_of_Grid
B(i,i) = 1;
end
for i = N_dim-No_of_Grid+1:N_dim
B(i,i) = 1;
end
B(No_of_Grid+1,No_of_Grid+1) = (1-4*lmd);
B(No_of_Grid+1,No_of_Grid+2) = lmd;
B(No_of_Grid+1,(2*No_of_Grid)+1) = lmd;
B(N_dim-No_of_Grid,N_dim-(2*No_of_Grid)) = lmd;
B(N_dim-No_of_Grid,N_dim-No_of_Grid) = (1-4*lmd);
B(N_dim-No_of_Grid,N_dim-No_of_Grid-1) = lmd;
for i = No_of_Grid+2:N_dim-No_of_Grid-1
B(i,i-1) = lmd;
B(i,i) = (1-4*lmd);
B(i,i+1) = lmd;
if(No_of_Grid+1<i && i<=(N_dim-No_of_Grid))
B(i,No_of_Grid+i) = lmd;
end
if(No_of_Grid<i && i<=(N_dim-No_of_Grid))
B(i,i-No_of_Grid) = lmd;
end
end
for i = 1:No_of_Grid
B(No_of_Grid*i,:)=0;
B(No_of_Grid*i,No_of_Grid*i)=1;
end
for i = 0:No_of_Grid-1
B(No_of_Grid*i+1,:)=0;
B(No_of_Grid*i+1,No_of_Grid*i+1)=1;
Bn=B;
end
```

References

-Basic fluid mechanics: *Fluid Mechanics – An Introduction to the Theory of Fluid Flows,F.Durst,Springer,2008*

-CFD: *Computational Methods for Fluid Dynamics,J.Ferziger,M.Peric,Springer,2002*