PROJECT REPORT

ON

The Finite Element Method/Finite difference methods in Electromagnetism

By

C Pranay Reddy - 2014B4A80757P

Rasal Kumar - 2014B4A80801P

Prepared in complete fulfillment of the

MATH F376(Design Project) submitted to

Dr. Sangita Yadav

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE,PILANI

Acknowledgements

We would like to thank our Vice-Chancellor, Prof. Souvik Bhattacharya and HoD, Department of Mathematics, BITS Pilani Pilani Campus, Dr. Balram Dubey for giving us an opportunity to enroll in the course on Design Project in Mathematics(MATH F376), so that we could study the topic of my interest in BITS. We express special gratitude to Dr. Sangita Yadav for her relentless efforts in guiding us, so that the topic is clearly understood by us and making us thorough with the knowledge leaving no stone unturned so that we could implement the theory in practical world.

Abstract

The project aims to study the application and use of finite element method and finite difference methods in electromagnetism. Electromagnetism is one of the evolving fields of science with various applications in real life. We consider some of these problems in this project.

Introduction

We know the methods like Bisection method, Secant method, Newton Raphson method to find the root of an equation. But when the equation includes a differential operator i.e. a differential equation the known numerical solution methods are Picards method, Runge kutta method etc.

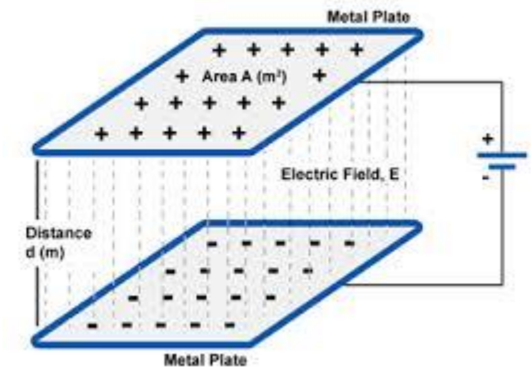
For a better solution i.e. for an approximated solution with lesser error we use various collocation methods. Hence a numerical method is being used in this project.

Here in this contest of study, we considered two problems in electromagnetism. The first problem involves the problem of finding electric field on a parallel plate capacitor with a dielectric slab which involves one dimensional Poisson's equation. The second problem involves the calculation of electric field on a Cylindrical surface involving the use of three dimensional Poisson Equation using polar coordinates.

The work done till mid semester was based on the complete solution of one dimensional parallel plate capacitor problem and illustration of how to solve 2 dimensional Poisson equation using finite element method. Then, we concentrated on solving the electromagnetic equations using finite difference methods

The project in the initial phase was aimed at solving the two problems using finite element method and show the error between the finite element solutions and the exact solutions. Then, all types of PDE's have been solved in the matlab.

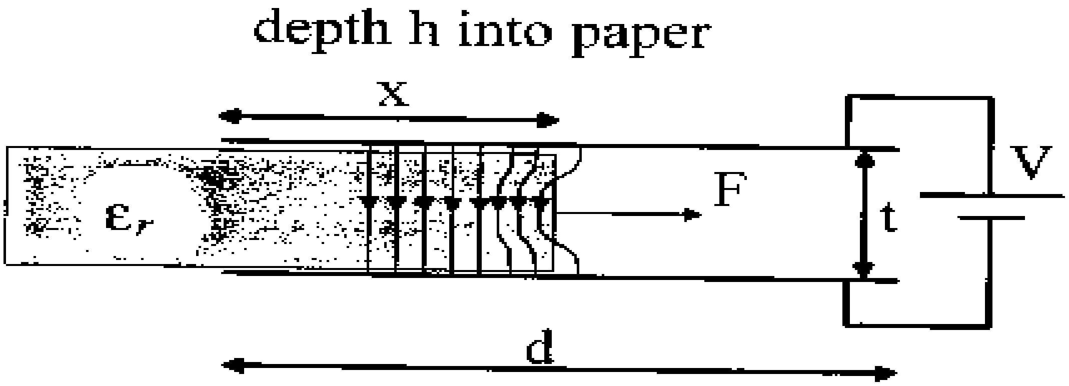
Electric Potential in a parallel plate Capacitor

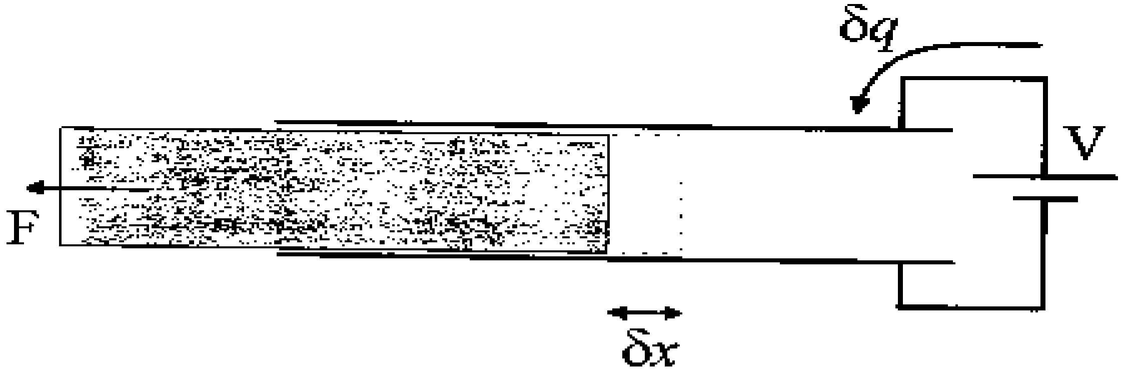


$$\frac{\partial^2 V}{\partial x^2} = 0$$

$$V = c_1 y + c_2$$

Dielectric Slab in a Capacitor





Steps to solve Boundary Value Problems:

- Solve the equation (Poisson / Laplace) in appropriate coordinate system.
- Apply boundary conditions.
- Calculate V.
- Use E=-Grad(V)
- D=eE and J=σE

For the capacitor problem let us consider a dielectric placed in z direction with a uniform charge density ρ. Therefore

$$\frac{\partial^2 V}{\partial z^2} = -\frac{\rho}{\varepsilon}$$

$$\frac{\partial V}{\partial z} = -\frac{\rho z}{\varepsilon} + c_1$$

$$V = -\frac{\rho z^2}{2\varepsilon} + c_1 z + c_2$$

z = 0 , $V = V_0$

z = d , V = 0

$$c_1 = \frac{\rho_0 d}{2\varepsilon} - \frac{V_0}{d}$$

$$c_2 = V_0$$

$$V = \frac{-\rho_0 z^2}{2\varepsilon} + \left(\frac{\rho_0 d}{2\varepsilon} - \frac{V_0}{d}\right) z + V_0$$

$$E = -\frac{\partial V}{\partial z} a_z$$

$$E = \frac{\rho_0 z}{\varepsilon} + \frac{V_0}{d} - \frac{\rho_0 d}{2\varepsilon}$$

Matlab Implementation:

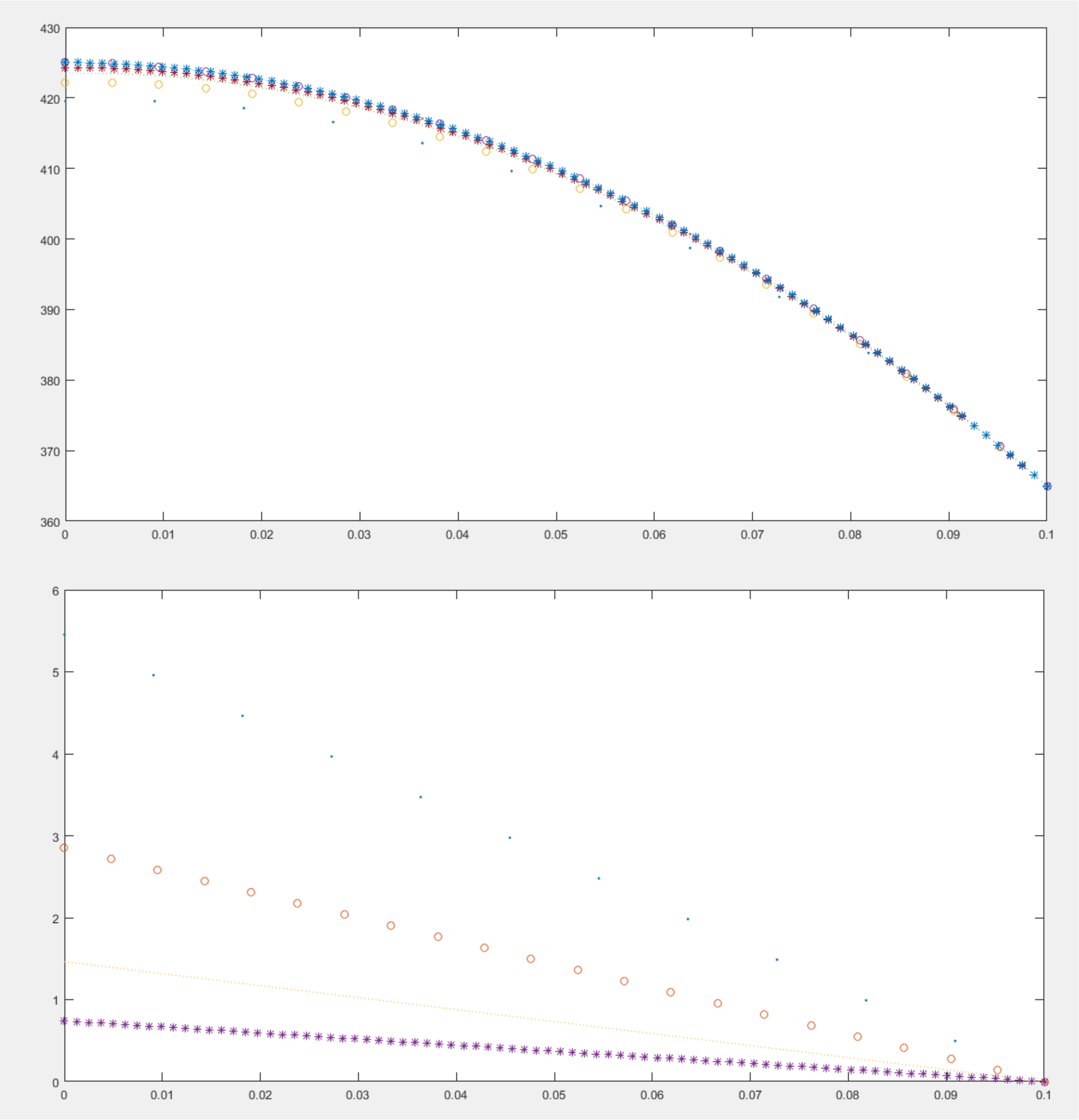
```
N = [12,22,42,82];
q = 0.3*10^(6);
k = 25;
T_s = 365;
b(1,1) =0;
T1 = zeros(82,4);
T_analyical = zeros(82,4);  % analytical/exact Potential matrix

for i =  1:4
a =[0,1,-1,1,-2,1,0,1,0,N(1,i)];
 % matrix of coefficients
b(N(1,i),1) = T_s;
b(2:(N(1,i)-1),1) = (-q/k)*((0.1/(N(1,i)-1))^(2));
TDM  = tridiagonal_mat(a);
[B,A] = res_mat(b,TDM);
T = TDM_sol(B,A);
T(1,1) = T(2,1);
T1(1:N(1,i),i) = T(:,1);
%  Potential  values  calculated  using  TDM  with  different  grid
sizes.
T_analytical(1:N(1,i),i) = T_exact(T_s,N(1,i));
% exact potential values with different grid sizes
Error(1:N(1,i),i) = T_analytical(1:N(1,i),i)-T1(1:N(1,i),i);
Average_L2_Norm(1,i)              =             sqrt((1/(N(1,i)-
1))*(sum(Error(:,i).^(2))));
end

x1 = 0:(0.1/11):0.1;
x2 = 0:(0.1/21):0.1;
x3 = 0:(0.1/41):0.1;
x4 = 0:(0.1/81):0.1;

figure % Potential Plot (both numerical and analytical)
plot(x1,T1(1:12,1),'.',x1,T_analytical(1:12,1),'.')
hold on
plot(x2,T1(1:22,2),'o',x2,T_analytical(1:22,2),'o')
hold on
plot(x3,T1(1:42,3),':',x3,T_analytical(1:42,3),':')
hold on
plot(x4,T1(1:82,4),'*',x4,T_analytical(1:82,4),'*')

figure (2) % Error Plot
plot(x1,Error(1:12,1),'.')
hold on
plot(x2,Error(1:22,2),'o')
```

```
hold on
plot(x3,Error(1:42,3),':')
hold on
plot(x4,Error(1:82,4),'*')
```

```
function [T] = TDM_sol(B,A)
N = size(B,1);
T(N,1) = B(N,1)/A(2,N);

   for i = N-1:-1:2
      T(i,1)   =   (B(i,1)   -   (A(3,i)*T(i+1,1)))/A(2,i);        %
calculating Potential matrix using Tri-diagonal matrix.
   end
end
```

```
function [T_analytical] = T_exact(T_s,N)
 q = 0.3*10^(6);
 k = 25;
    for j = 1:N
        T(1,j) = T_s + (q/(2*k))*(0.01-(((0.1/(N-1))*(j-1))^2));
    end
    T_analytical = T;

end
```

```
function[D, L, U] = trans_mat(A)

D = diag(diag(A));  % extracting the diagonal elements of A
L = -1*tril(A,-1);
U = -1*triu(A,1);

end
```


430
420
410
400
390
380
370
360
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
6
5
4
3
2
1
0
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1

The convergence criteria was 1e-08 (from the exact solution).

For N=11, the number of iterations needed for convergence were 1180.

For N=21, the number of iterations needed for convergence were 3742.

For N=41, the number of iterations needed for convergence were 12714.

For N=81, the number of iterations needed for convergence were 44801.

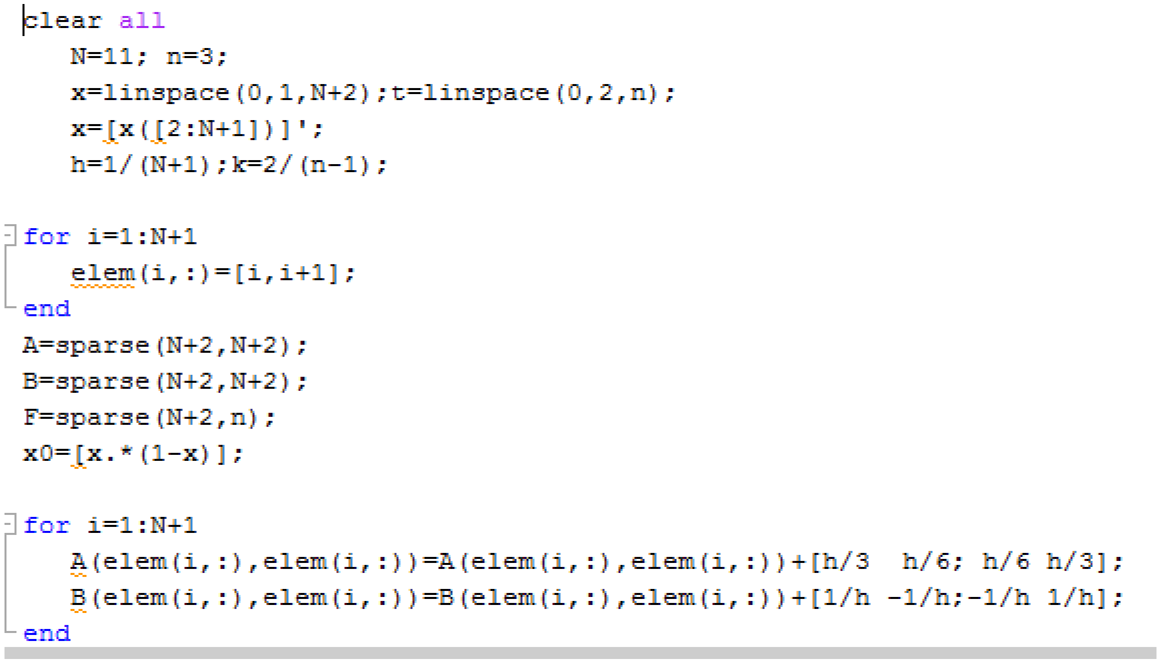
All these results are without relaxation.

Taking the boundary condition as first order played an important role in defining the temperature at other nodes as can be seen from the graph. As the number of grid points are increased the error decreases, this is because as the size of grid decreases the governing equation can be approximated with linear relation for closer points. But, as the governing equation is of second order, in gauss siedel it is found that the potential values converges even for coarse grid size of N = 11. But the L2-norm of the second order approximation increases because as the number of grid points increases the error gets accumulated.

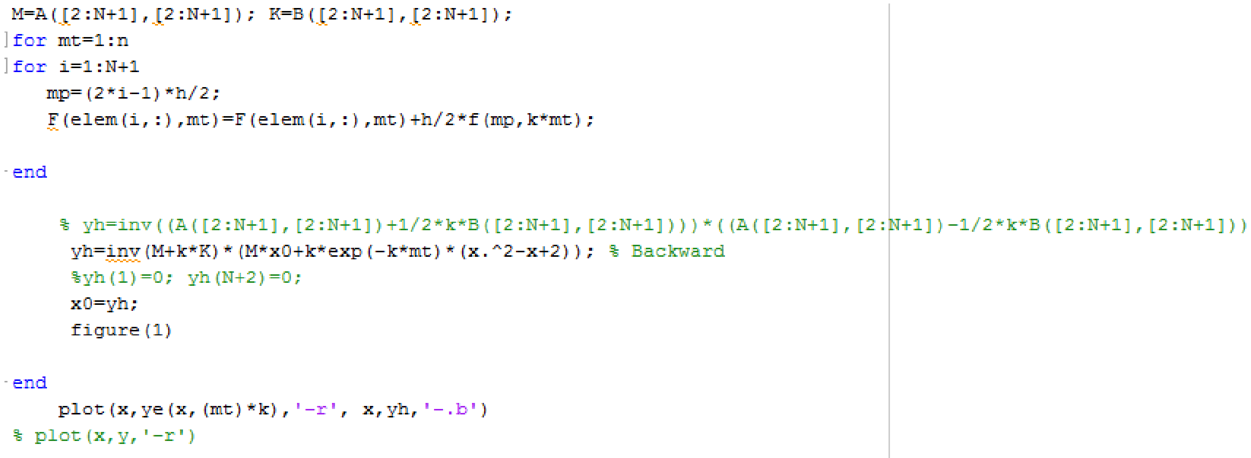
Problem for 2D Poisson Equation:

Suppose a rectangle bar in kept on Y-axis of a coordinate system. It has length l and breadth W. Assume the charge distribution to be uniform in the bar with sigma as the constant. Assume that the centre of gravity of the rod lies at the origin. Find the electric field distribution on the X axis.

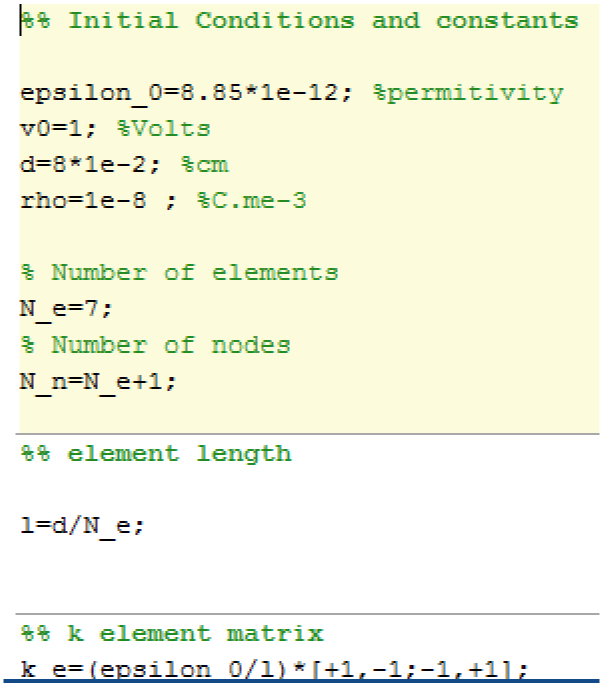
Matlab Implementation:(Poisson)



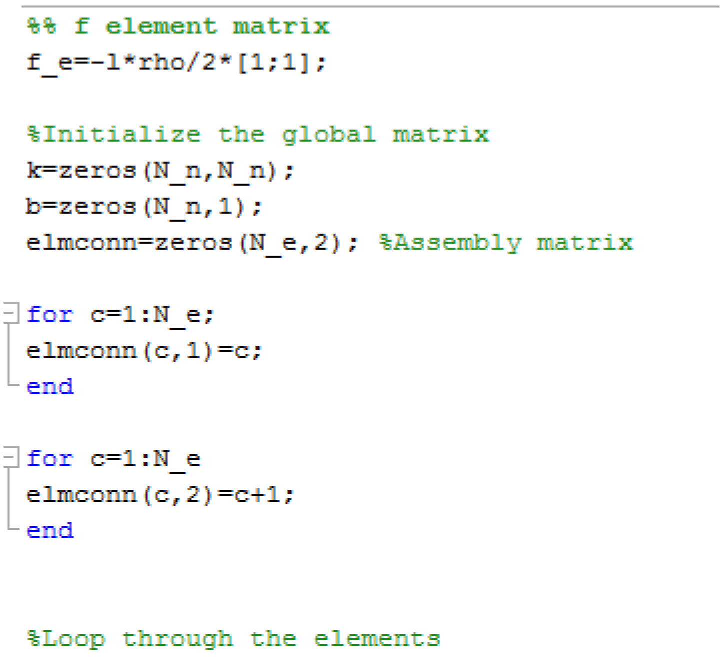
```
clear all
   N=11; n=3;
   x=linspace(0,1,N+2);t=linspace(0,2,n);
   x=[x([2:N+1])]';
   h=1/(N+1);k=2/(n-1);

for i=1:N+1
   elem(i,:)=[i,i+1];
end
A=sparse(N+2,N+2);
B=sparse(N+2,N+2);
F=sparse(N+2,n);
x0=[x.*(1-x)];

for i=1:N+1
   A(elem(i,:),elem(i,:))=A(elem(i,:),elem(i,:))+[h/3  h/6; h/6 h/3];
   B(elem(i,:),elem(i,:))=B(elem(i,:),elem(i,:))+[1/h -1/h;-1/h 1/h];
end
```



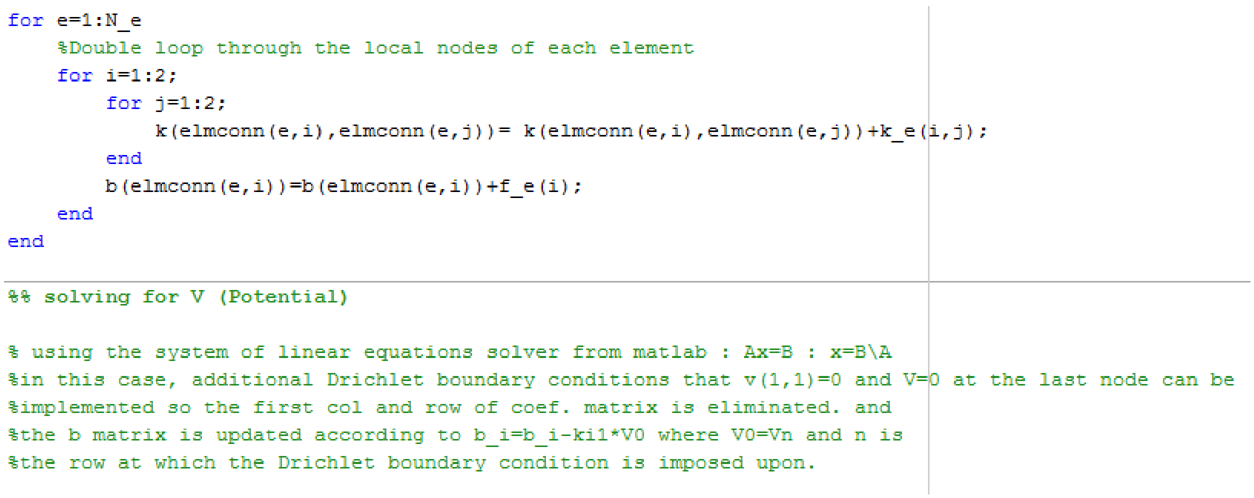
```
M=A([2:N+1],[2:N+1]); K=B([2:N+1],[2:N+1]);
for mt=1:n
for i=1:N+1
   mp=(2*i-1)*h/2;
   F(elem(i,:),mt)=F(elem(i,:),mt)+h/2*f(mp,k*mt);

end

    % yh=inv((A([2:N+1],[2:N+1])+1/2*k*B([2:N+1],[2:N+1])))*((A([2:N+1],[2:N+1])-1/2*k*B([2:N+1],[2:N+1]))
     yh=inv(M+k*K)*(M*x0+k*exp(-k*mt)*(x.^2-x+2)); % Backward
     %yh(1)=0; yh(N+2)=0;
     x0=yh;
     figure(1)

end
    plot(x,ye(x,(mt)*k),'-r', x,yh,'-.b')
% plot(x,y,'-r')
```



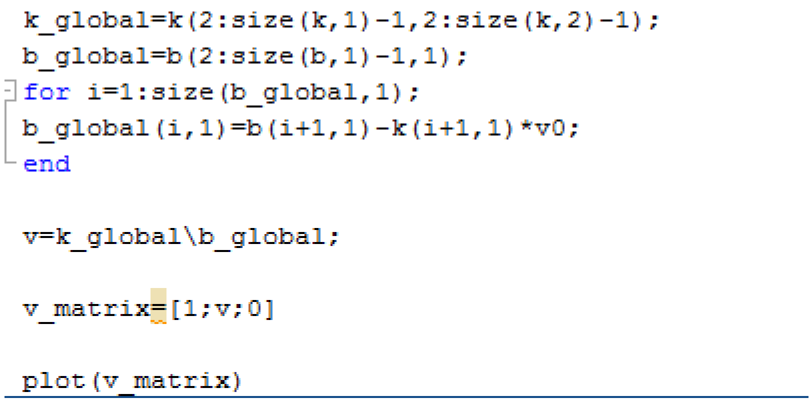
```
%% Initial Conditions and constants

epsilon_0=8.85*1e-12; %permitivity
v0=1; %Volts
d=8*1e-2; %cm
rho=1e-8 ; %C.me-3

% Number of elements
N_e=7;
% Number of nodes
N_n=N_e+1;

%% element length

l=d/N_e;


%% k element matrix
k_e=(epsilon_0/l)*[+1,-1;-1,+1];
```



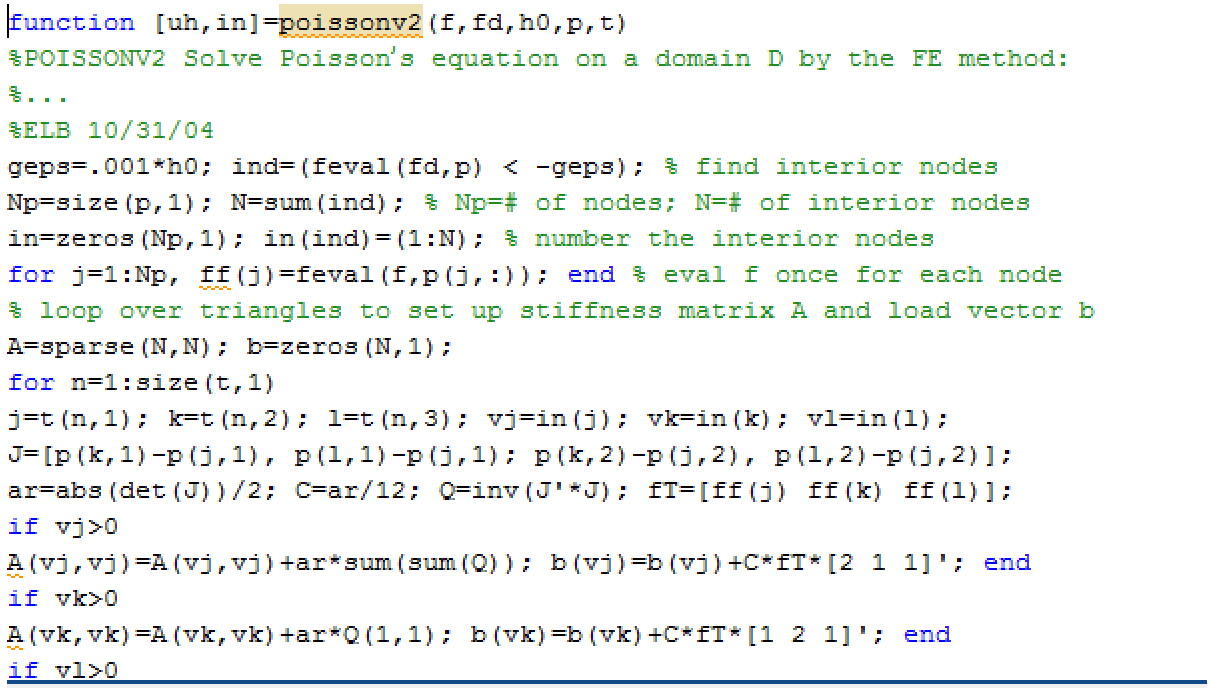
```
%% f element matrix
f_e=-1*rho/2*[1;1];

%Initialize the global matrix
k=zeros(N_n,N_n);
b=zeros(N_n,1);
elmconn=zeros(N_e,2); %Assembly matrix

for c=1:N_e;
elmconn(c,1)=c;
end

for c=1:N_e
elmconn(c,2)=c+1;
end


%Loop through the elements
```



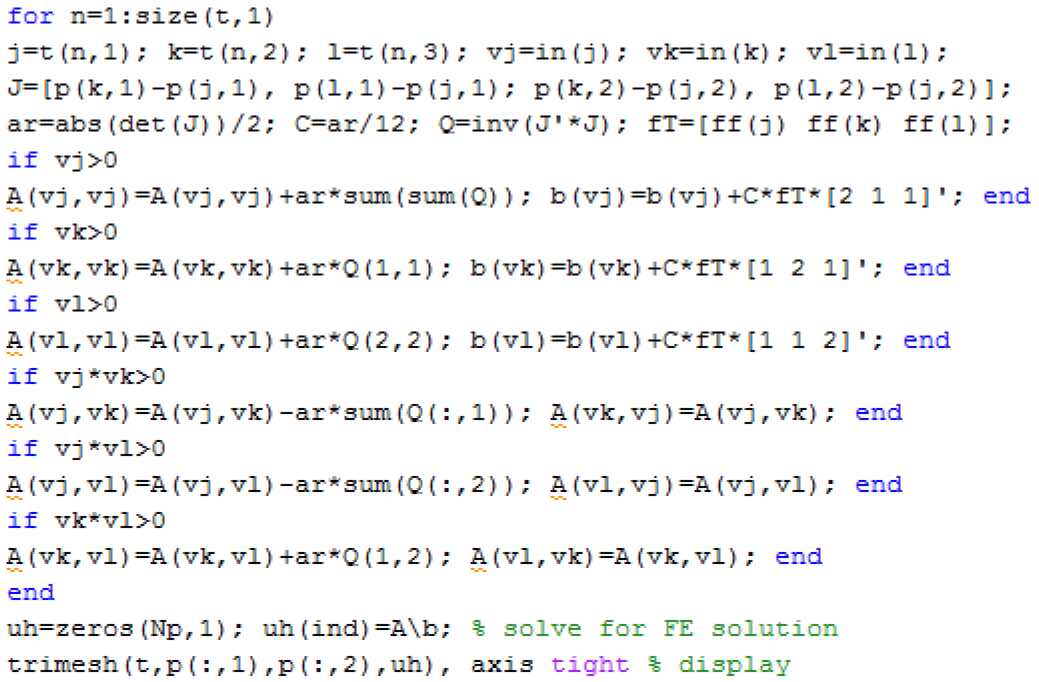
```
for e=1:N_e
    %Double loop through the local nodes of each element
    for i=1:2;
        for j=1:2;
            k(elmconn(e,i),elmconn(e,j))= k(elmconn(e,i),elmconn(e,j))+k_e(i,j);
        end
        b(elmconn(e,i))=b(elmconn(e,i))+f_e(i);
    end
end
%% solving for V (Potential)

% using the system of linear equations solver from matlab : Ax=B : x=B\A
%in this case, additional Drichlet boundary conditions that v(1,1)=0 and V=0 at the last node can be
%implemented so the first col and row of coef. matrix is eliminated. and
%the b matrix is updated according to b_i=b_i-ki1*V0 where V0=Vn and n is
%the row at which the Drichlet boundary condition is imposed upon.
```



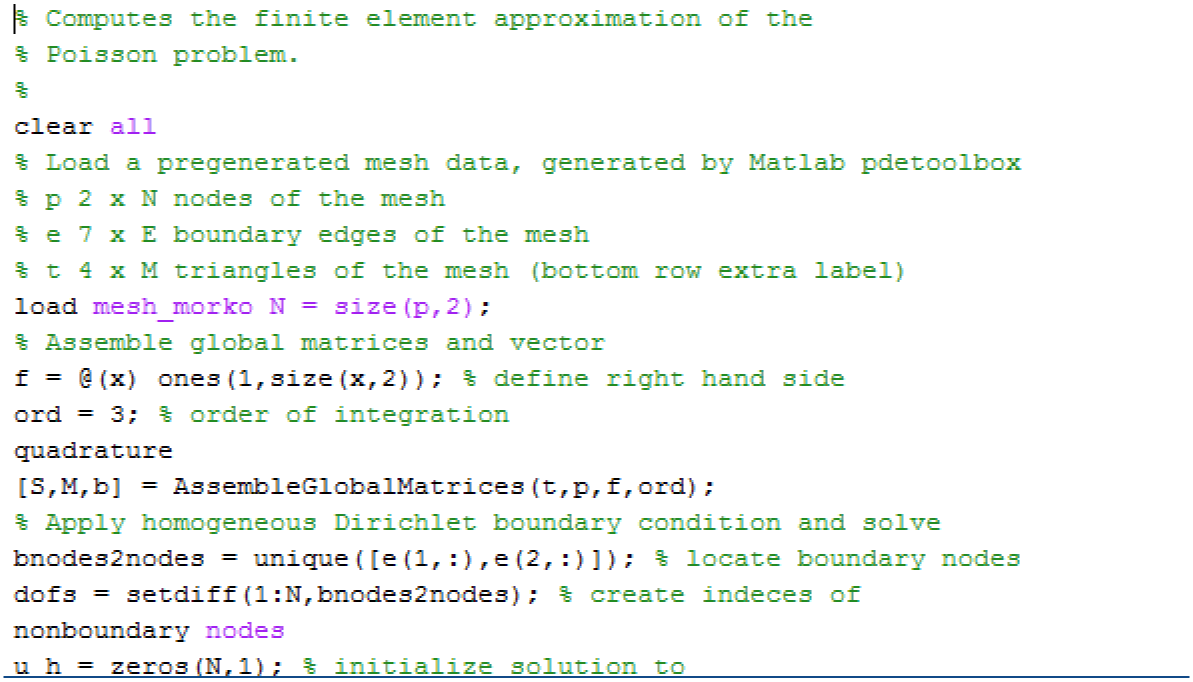
```
k_global=k(2:size(k,1)-1,2:size(k,2)-1);
b_global=b(2:size(b,1)-1,1);
for i=1:size(b_global,1);
b_global(i,1)=b(i+1,1)-k(i+1,1)*v0;
end

v=k_global\b_global;

v_matrix=[1;v;0]

plot(v_matrix)
```



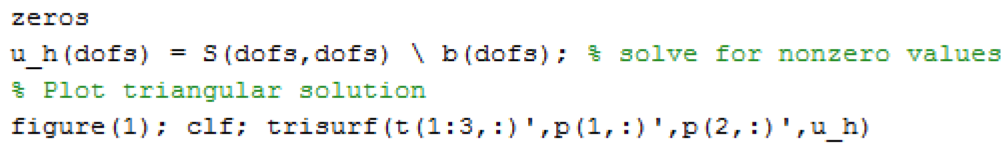
```
function [uh,in]=poissonv2(f,fd,h0,p,t)
%POISSONV2 Solve Poisson’s equation on a domain D by the FE method:
%...
%ELB 10/31/04
geps=.001*h0; ind=(feval(fd,p) < -geps); % find interior nodes
Np=size(p,1); N=sum(ind); % Np=# of nodes; N=# of interior nodes
in=zeros(Np,1); in(ind)=(1:N); % number the interior nodes
for j=1:Np, ff(j)=feval(f,p(j,:)); end % eval f once for each node
% loop over triangles to set up stiffness matrix A and load vector b
A=sparse(N,N); b=zeros(N,1);
for n=1:size(t,1)
j=t(n,1); k=t(n,2); l=t(n,3); vj=in(j); vk=in(k); vl=in(l);
J=[p(k,1)-p(j,1), p(l,1)-p(j,1); p(k,2)-p(j,2), p(l,2)-p(j,2)];
ar=abs(det(J))/2; C=ar/12; Q=inv(J'*J); fT=[ff(j) ff(k) ff(l)];
if vj>0
A(vj,vj)=A(vj,vj)+ar*sum(sum(Q)); b(vj)=b(vj)+C*fT*[2 1 1]'; end
if vk>0
A(vk,vk)=A(vk,vk)+ar*Q(1,1); b(vk)=b(vk)+C*fT*[1 2 1]'; end
if vl>0
```



```
for n=1:size(t,1)
j=t(n,1); k=t(n,2); l=t(n,3); vj=in(j); vk=in(k); vl=in(l);
J=[p(k,1)-p(j,1), p(l,1)-p(j,1); p(k,2)-p(j,2), p(l,2)-p(j,2)];
ar=abs(det(J))/2; C=ar/12; Q=inv(J'*J); fT=[ff(j) ff(k) ff(l)];
if vj>0
A(vj,vj)=A(vj,vj)+ar*sum(sum(Q)); b(vj)=b(vj)+C*fT*[2 1 1]'; end
if vk>0
A(vk,vk)=A(vk,vk)+ar*Q(1,1); b(vk)=b(vk)+C*fT*[1 2 1]'; end
if vl>0
A(vl,vl)=A(vl,vl)+ar*Q(2,2); b(vl)=b(vl)+C*fT*[1 1 2]'; end
if vj*vk>0
A(vj,vk)=A(vj,vk)-ar*sum(Q(:,1)); A(vk,vj)=A(vj,vk); end
if vj*vl>0
A(vj,vl)=A(vj,vl)-ar*sum(Q(:,2)); A(vl,vj)=A(vj,vl); end
if vk*vl>0
A(vk,vl)=A(vk,vl)+ar*Q(1,2); A(vl,vk)=A(vk,vl); end
end
uh=zeros(Np,1); uh(ind)=A\b; % solve for FE solution
trimesh(t,p(:,1),p(:,2),uh), axis tight % display
```



```
% Computes the finite element approximation of the
% Poisson problem.
%
clear all
% Load a pregenerated mesh data, generated by Matlab pdetoolbox
% p 2 x N nodes of the mesh
% e 7 x E boundary edges of the mesh
% t 4 x M triangles of the mesh (bottom row extra label)
load mesh_morko N = size(p,2);
% Assemble global matrices and vector
f = @(x) ones(1,size(x,2)); % define right hand side
ord = 3; % order of integration
quadrature
[S,M,b] = AssembleGlobalMatrices(t,p,f,ord);
% Apply homogeneous Dirichlet boundary condition and solve
bnodes2nodes = unique([e(1,:),e(2,:)]); % locate boundary nodes
dofs = setdiff(1:N,bnodes2nodes); % create indeces of
nonboundary nodes
u_h = zeros(N,1); % initialize solution to
```

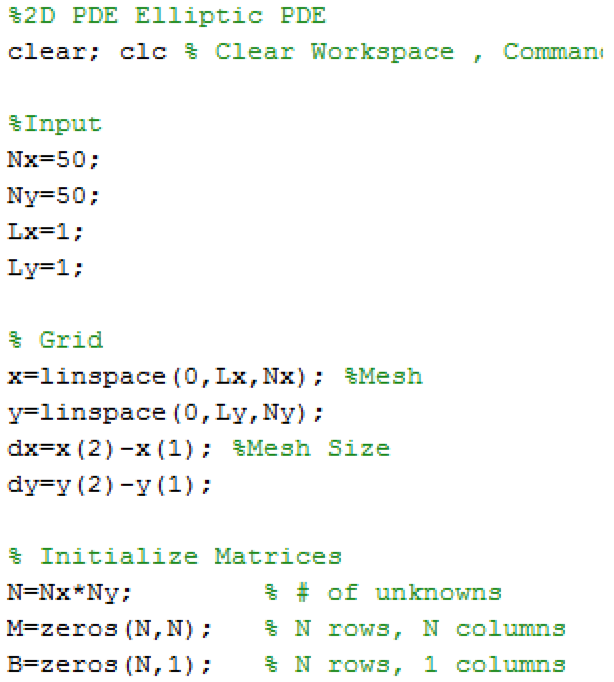


```
zeros
u_h(dofs) = S(dofs,dofs) \ b(dofs); % solve for nonzero values
% Plot triangular solution
figure(1); clf; trisurf(t(1:3,:)',p(1,:)',p(2,:)',u_h)
```

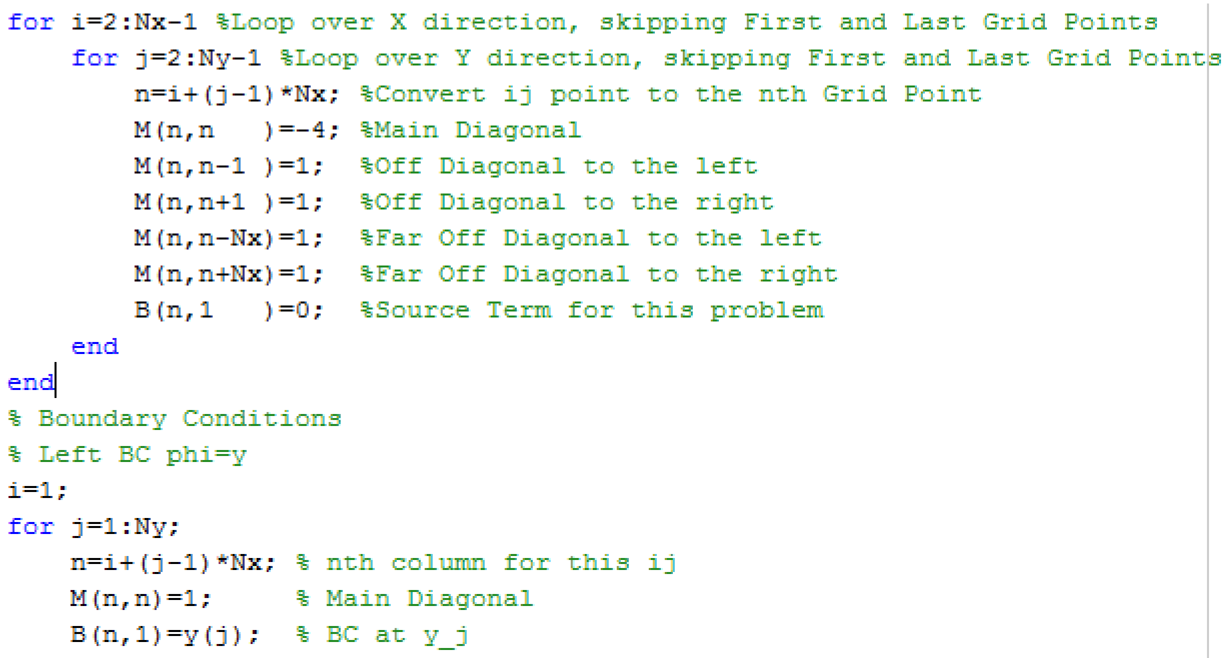
Laplace 2D Equation:

Application: Suppose there are two concentric hollow shells with charge density +Q and -Q on them respectively. Then the net electric field at any point outside the two shells will be zero. This is because the net charge enclosed by a gaussian surface including the two rings is zero. Therefore gradient of potential will be zero which is the Laplace Equation.

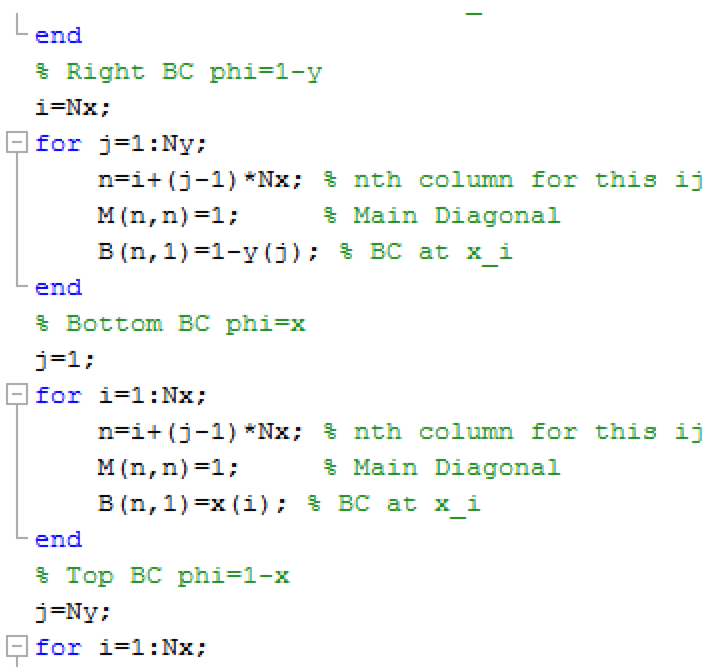
Matlab Implementation:(Laplace)



```
%2D PDE Elliptic PDE
clear; clc % Clear Workspace , Comman

%Input
Nx=50;
Ny=50;
Lx=1;
Ly=1;

% Grid
x=linspace(0,Lx,Nx); %Mesh
y=linspace(0,Ly,Ny);
dx=x(2)-x(1); %Mesh Size
dy=y(2)-y(1);

% Initialize Matrices
N=Nx*Ny;        % # of unknowns
M=zeros(N,N);   % N rows, N columns
B=zeros(N,1);   % N rows, 1 columns
```

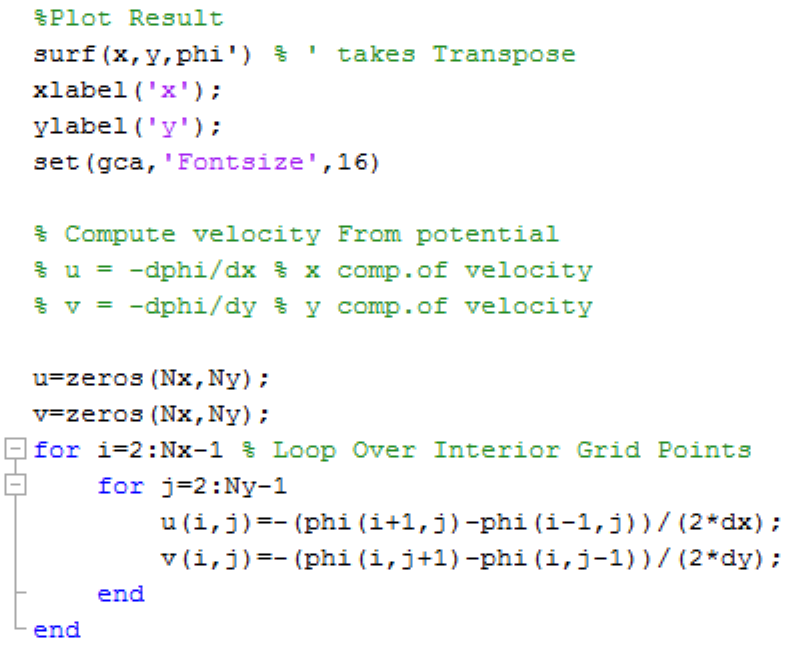


```
for i=2:Nx-1 %Loop over X direction, skipping First and Last Grid Points
    for j=2:Ny-1 %Loop over Y direction, skipping First and Last Grid Points
        n=i+(j-1)*Nx; %Convert ij point to the nth Grid Point
        M(n,n   )=-4; %Main Diagonal
        M(n,n-1 )=1;  %Off Diagonal to the left
        M(n,n+1 )=1;  %Off Diagonal to the right
        M(n,n-Nx)=1;  %Far Off Diagonal to the left
        M(n,n+Nx)=1;  %Far Off Diagonal to the right
        B(n,1   )=0;  %Source Term for this problem
    end
end
% Boundary Conditions
% Left BC phi=y
i=1;
for j=1:Ny;
    n=i+(j-1)*Nx; % nth column for this ij
    M(n,n)=1;     % Main Diagonal
    B(n,1)=y(j);  % BC at y_j
```

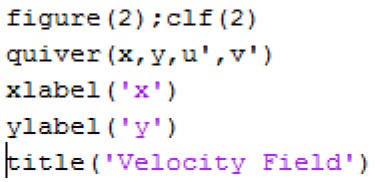


```
end
% Right BC phi=1-y
i=Nx;
for j=1:Ny;
    n=i+(j-1)*Nx; % nth column for this ij
    M(n,n)=1;      % Main Diagonal
    B(n,1)=1-y(j); % BC at x_i
end
% Bottom BC phi=x
j=1;
for i=1:Nx;
    n=i+(j-1)*Nx; % nth column for this ij
    M(n,n)=1;      % Main Diagonal
    B(n,1)=x(i); % BC at x_i
end
% Top BC phi=1-x
j=Ny;
for i=1:Nx;
```

```
    n=i+(j-1)*Nx; % nth column for this ij
    M(n,n)=1;      % Main Diagonal
    B(n,1)=1-x(i); % BC at x_i
end
%Solve for potential M*phi=B;
%phi=inv(M)*B;
phi_vec=M\B;

% Convert the Vector Solution to a 2D Array
for i=1:Nx
    for j=1:Ny
        n=i+(j-1)*Nx; % nth row of this ij
        phi(i,j)=phi_vec(n);
    end
end

%Plot Result
surf(x,y,phi') % ' takes Transpose
```



```
%Plot Result
surf(x,y,phi') % ' takes Transpose
xlabel('x');
ylabel('y');
set(gca,'Fontsize',16)

% Compute velocity From potential
% u = -dphi/dx % x comp.of velocity
% v = -dphi/dy % y comp.of velocity

u=zeros(Nx,Ny);
v=zeros(Nx,Ny);
for i=2:Nx-1 % Loop Over Interior Grid Points
    for j=2:Ny-1
        u(i,j)=-(phi(i+1,j)-phi(i-1,j))/(2*dx);
        v(i,j)=-(phi(i,j+1)-phi(i,j-1))/(2*dy);
    end
end
```



```
figure(2);clf(2)
quiver(x,y,u',v')
xlabel('x')
ylabel('y')
title('Velocity Field')
```

We ended up with the following results.

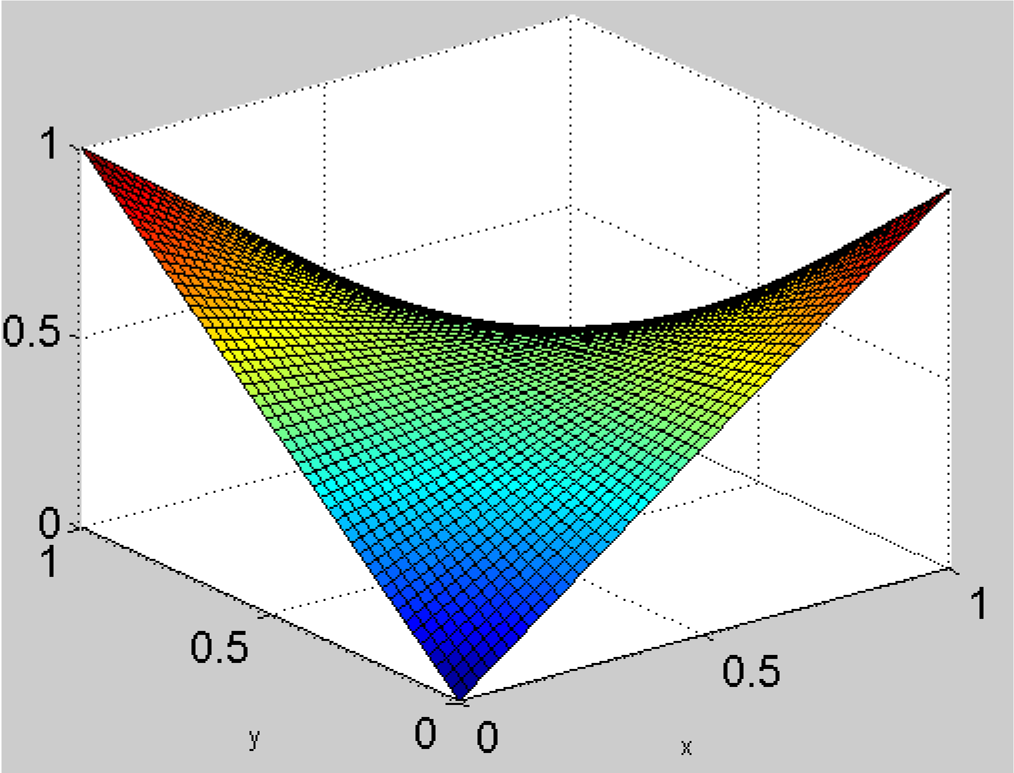
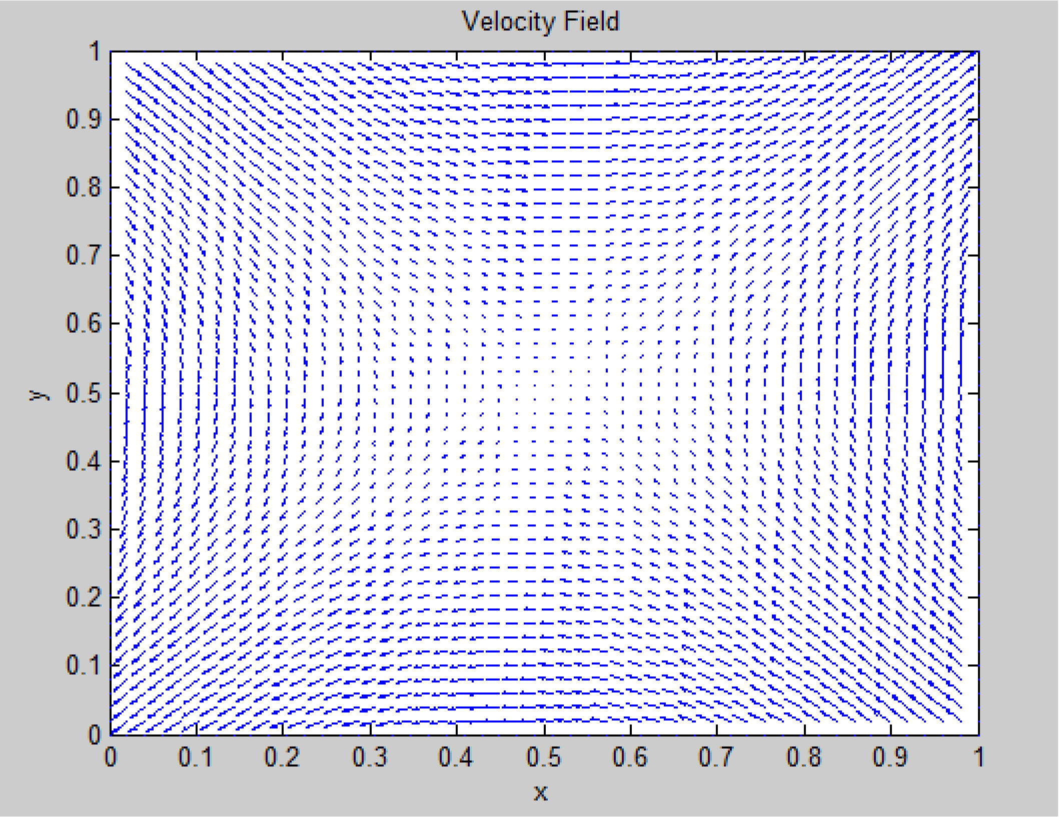


Fig. The Potential Plot.



This graph can be interpreted as suppose there are two source(+ve) charges and two sink charges at (1,0) ,(0,1) and (0,0),(1,1) . The two source charges will create field lines that are going away from them and the two sink charges will have field lines that are coming towards them. There will be a NULL point(0.5,0.5) where the field lines will cancel each other resulting in the net electric field to be zero at that point. All these things are clearly visible in the graph.

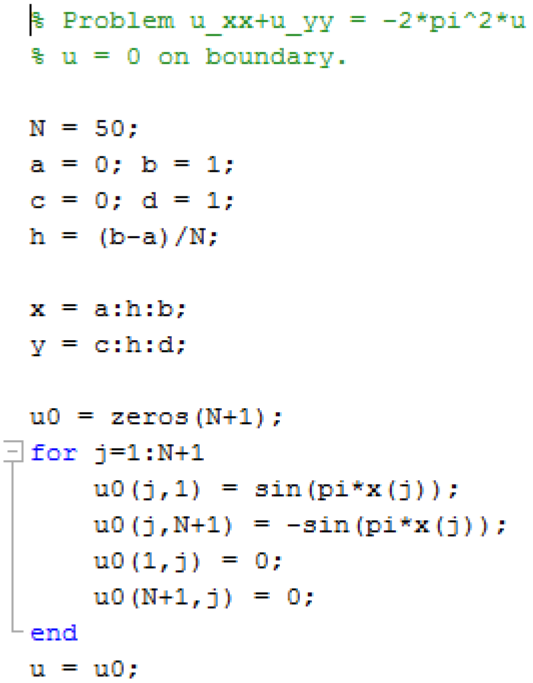
Fig. The Velocity Field.

Then, we solved one more elliptic equation on the same lines.

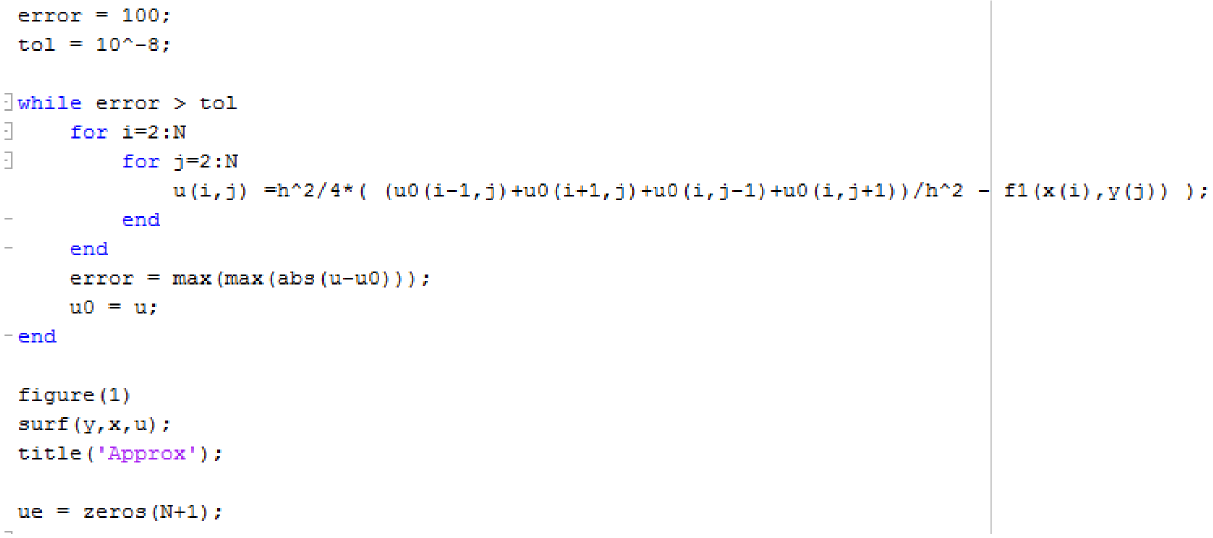
$$u_{xx} + u_{yy} = -2\pi^2 u$$

where u = 0 on boundary.

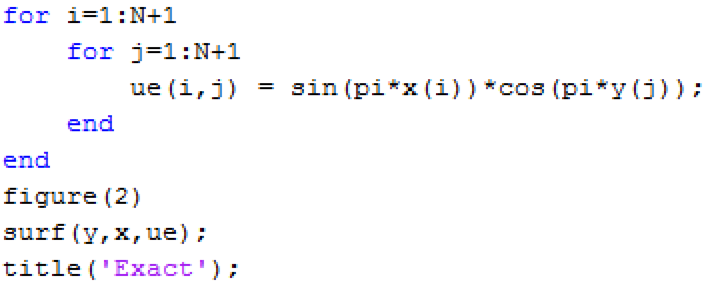
Matlab Implementation.



```
% Problem u_xx+u_yy = -2*pi^2*u
% u = 0 on boundary.

N = 50;
a = 0; b = 1;
c = 0; d = 1;
h = (b-a)/N;

x = a:h:b;
y = c:h:d;

u0 = zeros(N+1);
for j=1:N+1
    u0(j,1) = sin(pi*x(j));
    u0(j,N+1) = -sin(pi*x(j));
    u0(1,j) = 0;
    u0(N+1,j) = 0;
end
u = u0;
```

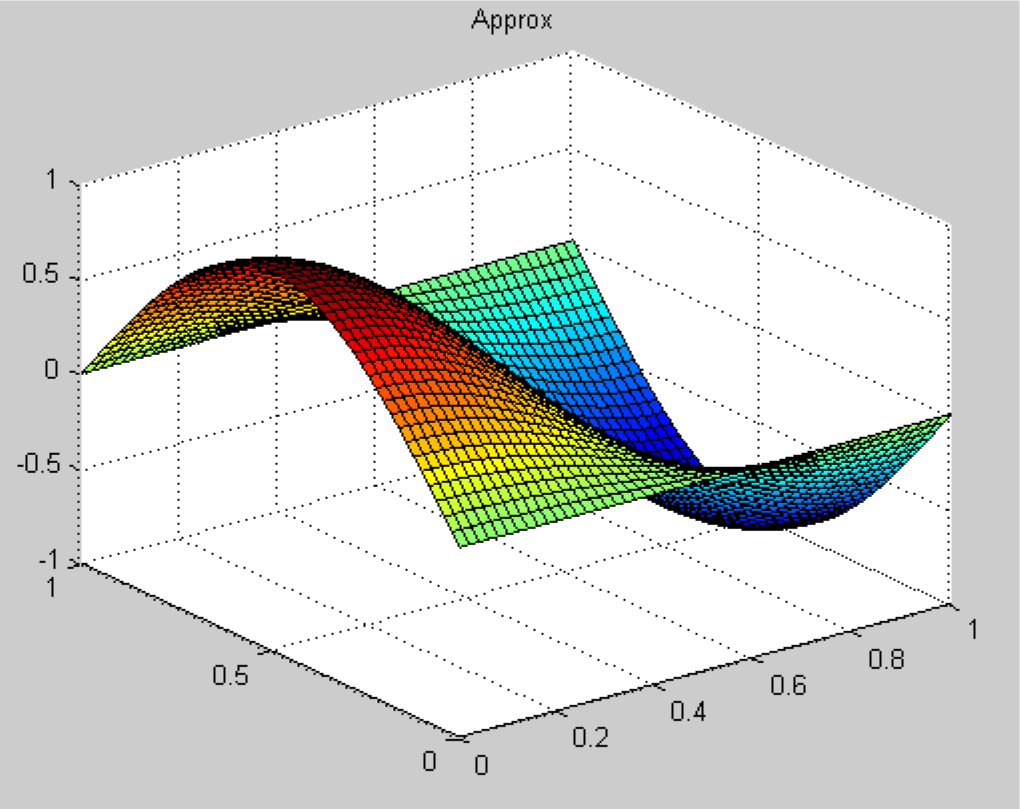


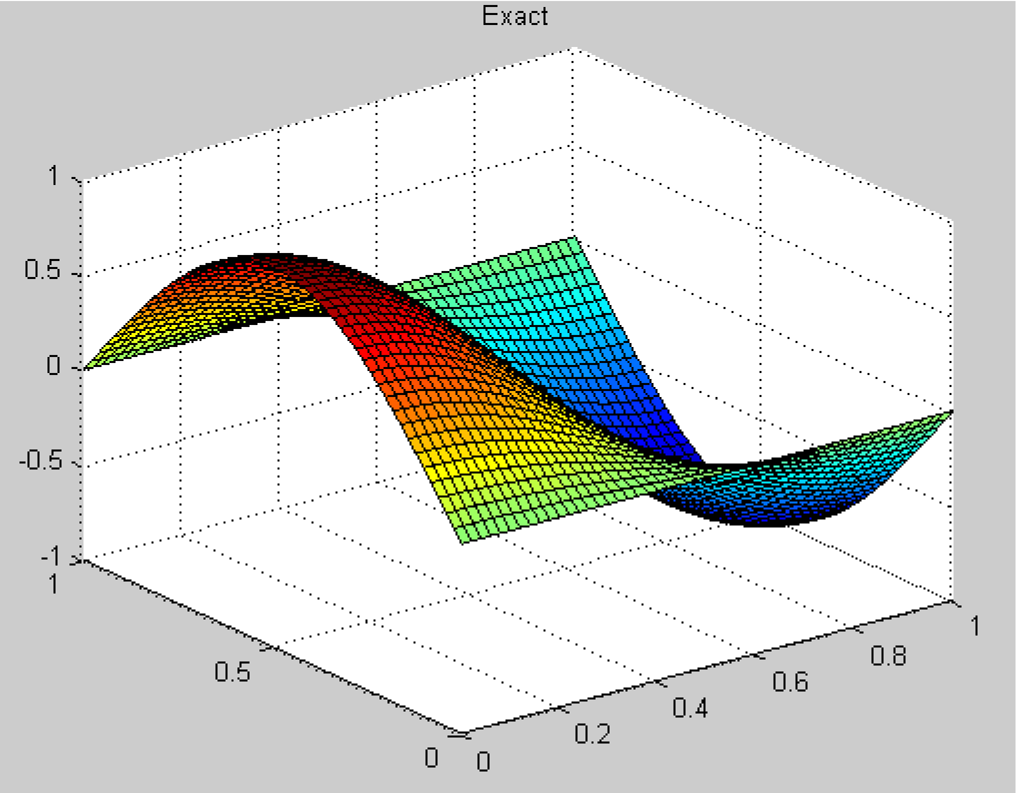
```
error = 100;
tol = 10^-8;

while error > tol
    for i=2:N
        for j=2:N
            u(i,j) =h^2/4*( (u0(i-1,j)+u0(i+1,j)+u0(i,j-1)+u0(i,j+1))/h^2 - f1(x(i),y(j)) );
        end
    end
    error = max(max(abs(u-u0)));
    u0 = u;
end

figure(1)
surf(y,x,u);
title('Approx');

ue = zeros(N+1);
```



```
for i=1:N+1
    for j=1:N+1
        ue(i,j) = sin(pi*x(i))*cos(pi*y(j));
    end
end
figure(2)
surf(y,x,ue);
title('Exact');
```

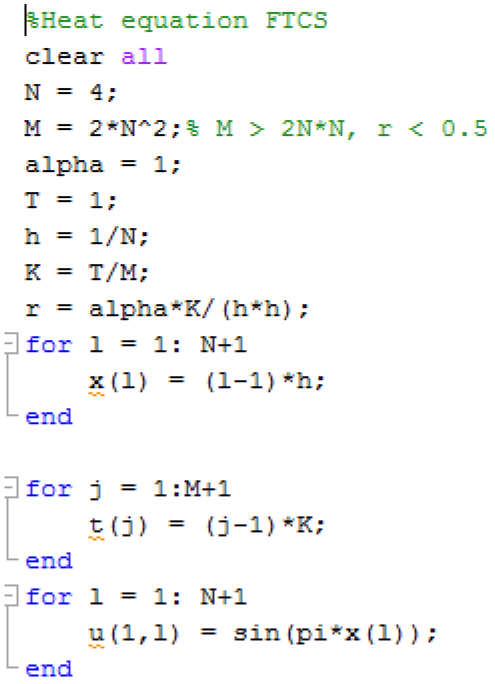
We ended up with the following results.


Approx
1
0.5
0
-0.5
-1
1
0.5
0
0
0.2
0.4
0.6
0.8
1

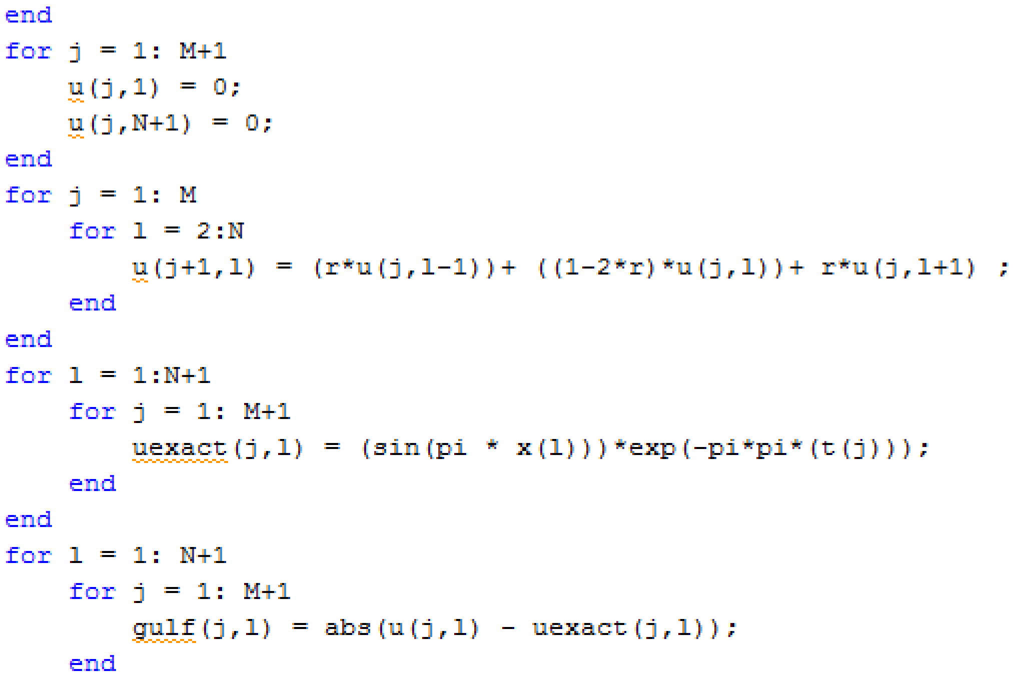

Exact
1
0.5
0
-0.5
-1
1
0.5
0
0
0.2
0.4
0.6
0.8
1

Matlab Implementation:(Heat)

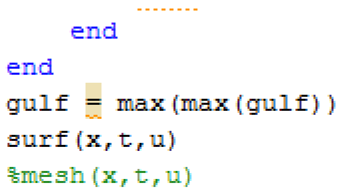
$$u_t = u_{xx}; u(x,0) = \sin(\pi x)\,; u(0,t) = \ 0; u(1,t) = \ 0$$



```
%Heat equation FTCS
clear all
N = 4;
M = 2*N^2;% M > 2N*N, r < 0.5
alpha = 1;
T = 1;
h = 1/N;
K = T/M;
r = alpha*K/(h*h);
for l = 1: N+1
    x(l) = (l-1)*h;
end

for j = 1:M+1
    t(j) = (j-1)*K;
end
for l = 1: N+1
    u(1,l) = sin(pi*x(l));
end
```

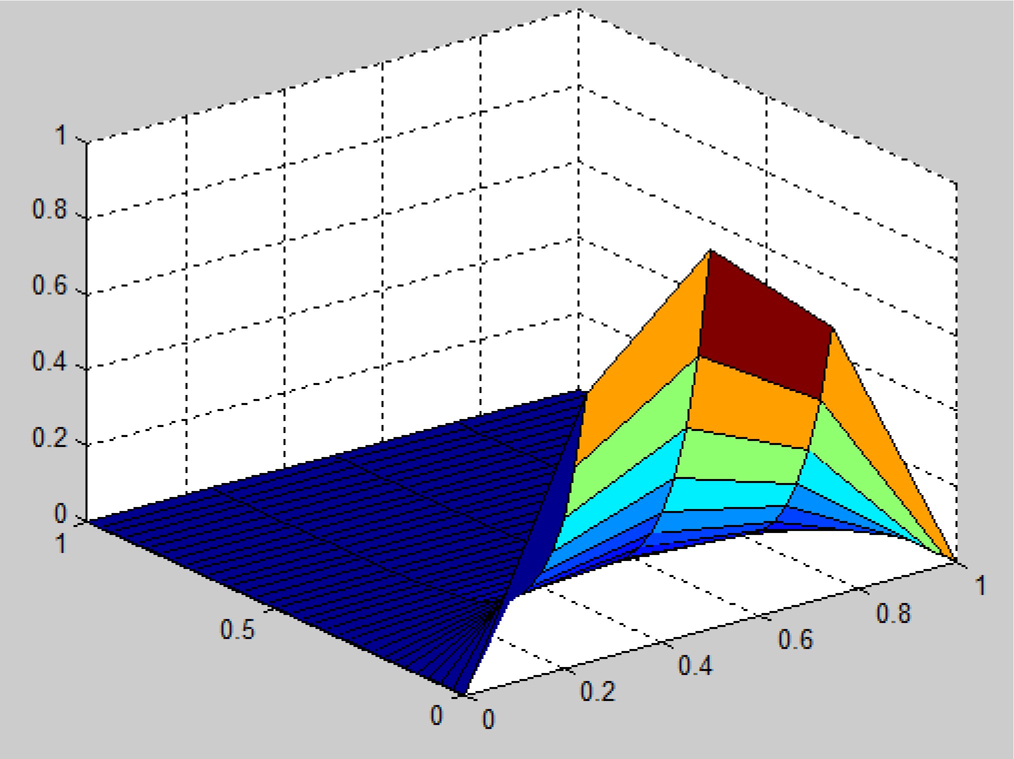


```
end
for j = 1: M+1
    u(j,1) = 0;
    u(j,N+1) = 0;
end
for j = 1: M
    for l = 2:N
        u(j+1,l) = (r*u(j,l-1))+ ((1-2*r)*u(j,l))+ r*u(j,l+1) ;
    end
end
for l = 1:N+1
    for j = 1: M+1
        uexact(j,l) = (sin(pi * x(l)))*exp(-pi*pi*(t(j)));
    end
end
for l = 1: N+1
    for j = 1: M+1
        gulf(j,l) = abs(u(j,l) - uexact(j,l));
    end
```



```
    end
end
gulf = max(max(gulf))
surf(x,t,u)
%mesh(x,t,u)
```

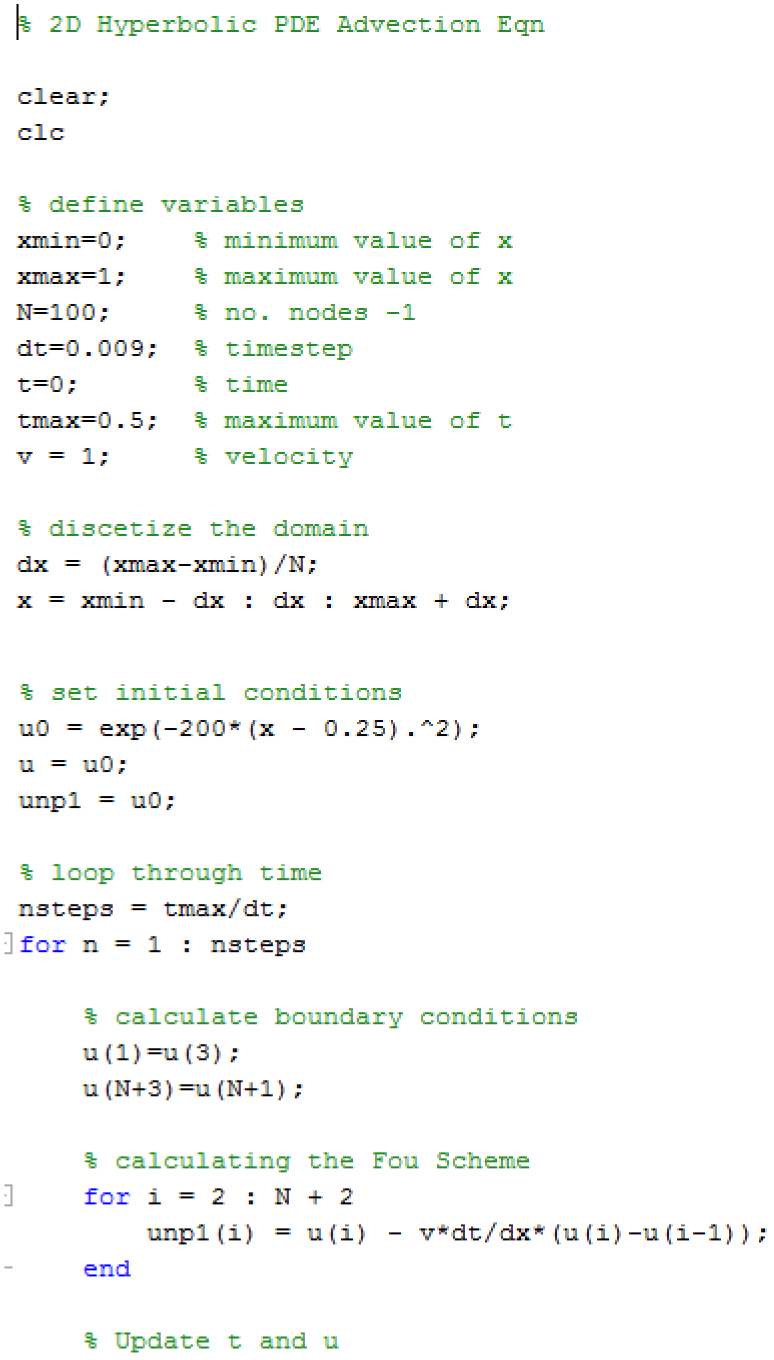
We ended up with the following results.



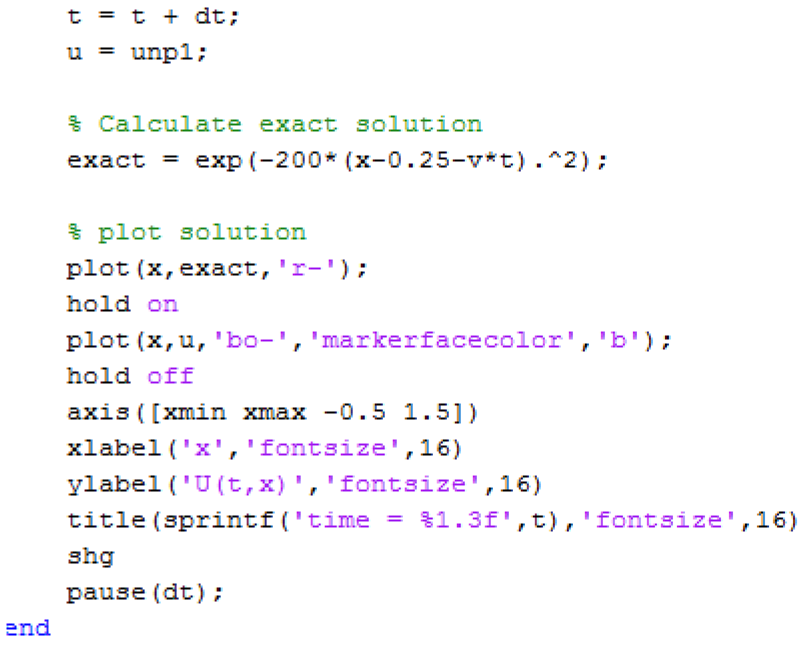
Matlab Implementation:(Advection equation)

Problem: Suppose there is a charge moving with its velocity both along with its position coordinates and along with the time axes. Find the velocity at a particular time.

$$u_t = -vu_x$$

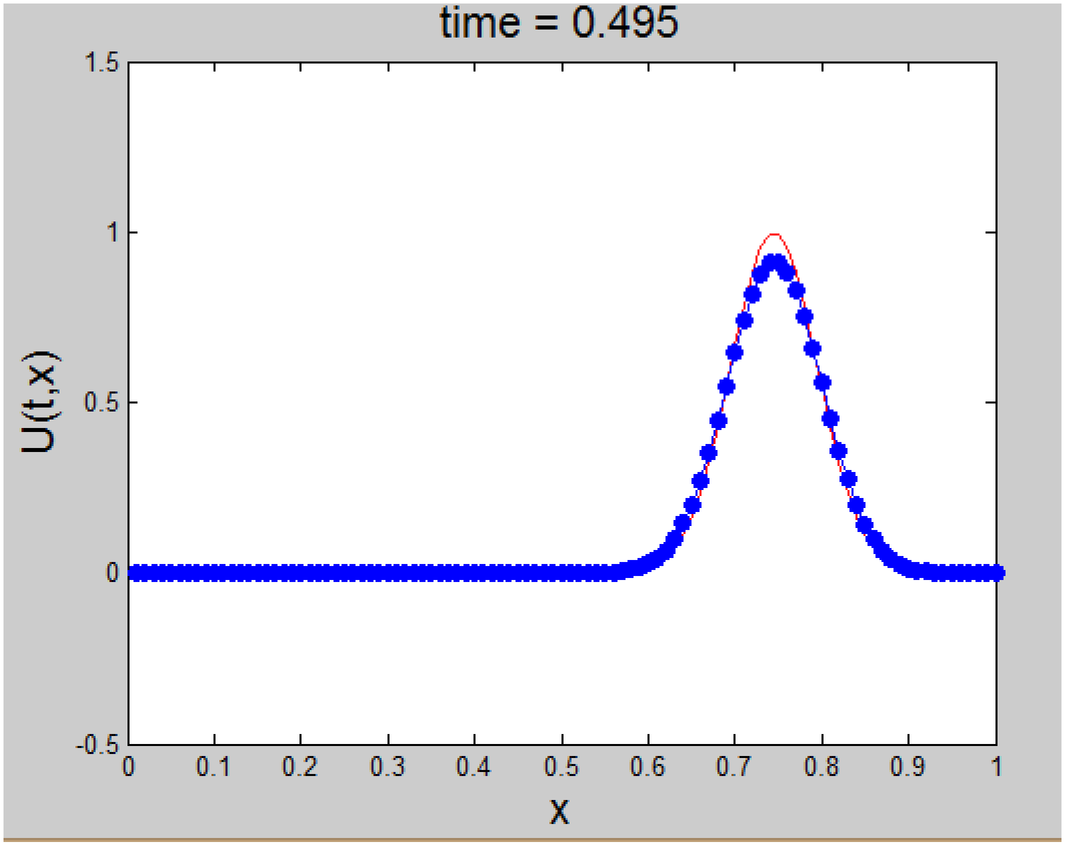


```
% 2D Hyperbolic PDE Advection Eqn

clear;
clc

% define variables
xmin=0;     % minimum value of x
xmax=1;     % maximum value of x
N=100;      % no. nodes -1
dt=0.009;   % timestep
t=0;        % time
tmax=0.5;   % maximum value of t
v = 1;      % velocity

% discetize the domain
dx = (xmax-xmin)/N;
x = xmin - dx : dx : xmax + dx;


% set initial conditions
u0 = exp(-200*(x - 0.25).^2);
u = u0;
unp1 = u0;

% loop through time
nsteps = tmax/dt;
for n = 1 : nsteps

    % calculate boundary conditions
    u(1)=u(3);
    u(N+3)=u(N+1);

    % calculating the Fou Scheme
    for i = 2 : N + 2
        unp1(i) = u(i) - v*dt/dx*(u(i)-u(i-1));
    end

    % Update t and u
```



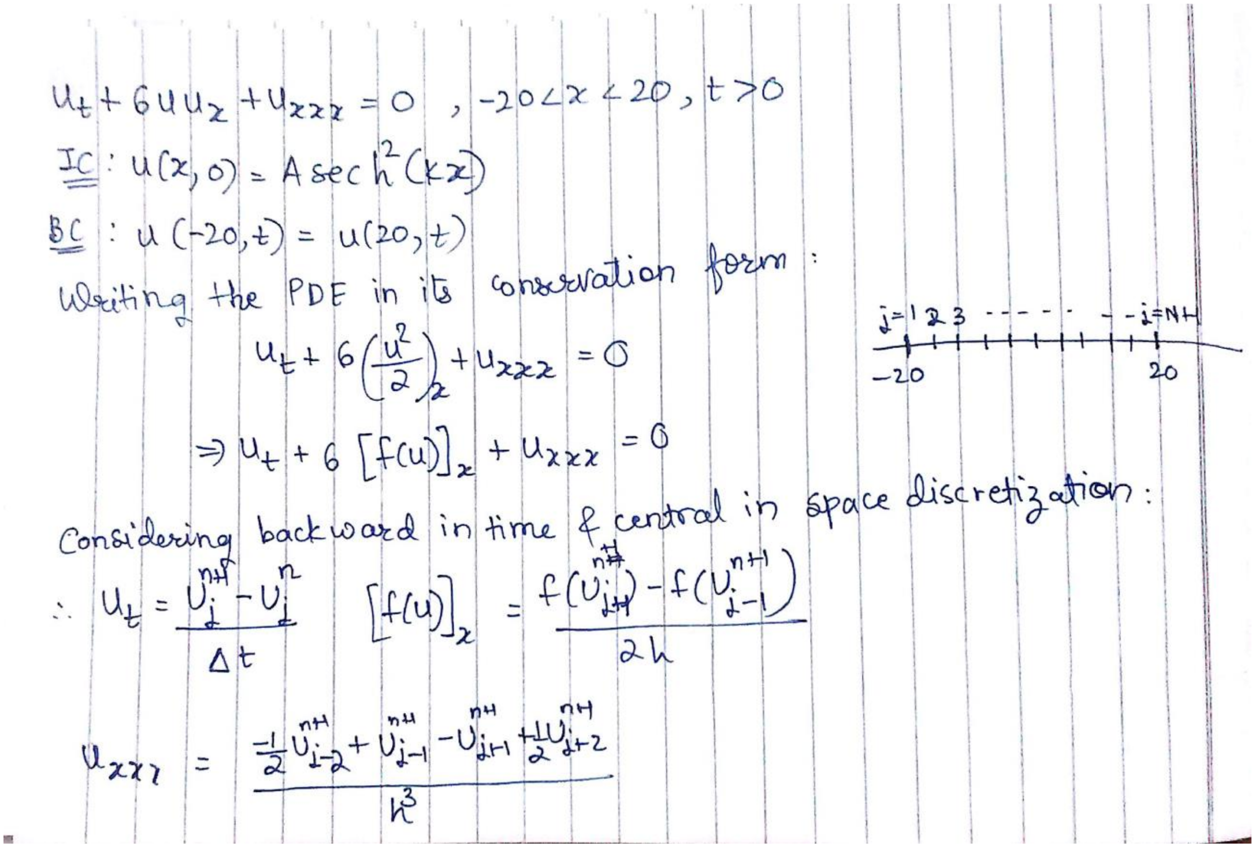
```
    t = t + dt;
    u = unp1;

    % Calculate exact solution
    exact = exp(-200*(x-0.25-v*t).^2);

    % plot solution
    plot(x,exact,'r-');
    hold on
    plot(x,u,'bo-','markerfacecolor','b');
    hold off
    axis([xmin xmax -0.5 1.5])
    xlabel('x','fontsize',16)
    ylabel('U(t,x)','fontsize',16)
    title(sprintf('time = %1.3f',t),'fontsize',16)
    shg
    pause(dt);
end
```

We ended up with the following results.



Non linear PDE

Let's suppose we have a non linear PDE given. Then the Tri diagonal solution wont work. We would have to discretize the PDE and then use Newton Ralpson method to calculated the values at various coordinates.



$u_t + 6uu_x + u_{xxx} = 0$, $-20 < x < 20$, $t > 0$

IC : $u(x,0) = A\,\mathrm{sec}h^2(kx)$

BC : $u(-20,t) = u(20,t)$

Writing the PDE in its conservation form :

$$u_t + 6\left(\frac{u^2}{2}\right)_x + u_{xxx} = 0$$

$$\Rightarrow u_t + 6\,[f(u)]_x + u_{xxx} = 0$$

j=1 2 3 - - - - - - - j=N+1
-20
20

Considering backward in time & central in space discretization :

$$\therefore\; u_t = \frac{U_j^{n+1} - U_j^{n}}{\Delta t} \qquad [f(u)]_x = \frac{f(U_{j+1}^{n+1}) - f(U_{j-1}^{n+1})}{2h}$$

$$u_{xxx} = \frac{-\frac{1}{2}U_{j-2}^{n+1} + U_{j-1}^{n+1} - U_{j+1}^{n+1} + \frac{1}{2}U_{j+2}^{n+1}}{h^3}$$

⇒ The scheme becomes:

$$\frac{U_j^{n+1}-U_j^{n}}{\Delta t} + \frac{6}{2h}\left[f(U_{j+1}^{n+1}) - f(U_{j-1}^{n+1})\right] + \frac{1}{h^3}\left[-\frac{1}{2}U_{j-2}^{n+1} + U_{j-1}^{n+1} - U_{j+1}^{n+1} + \frac{1}{2}U_{j+2}^{n+1}\right] = 0$$

→ Giving us a non-linear system of equations [as f is non linear]

So, by using Newton's method: $U^{n+1} = U^{n} - J^{-1}F$ J: Jacobian.

Now, writing:

$$F(j) = U_j^{n+1} - U_j^{n} + \frac{3\Delta t}{h}\left[f(U_{j+1}^{n+1}) - f(U_{j-1}^{n+1})\right] + \frac{\Delta t}{h^3}\left[-\frac{1}{2}U_{j-2}^{n+1} + U_{j-1}^{n+1} - U_{j+1}^{n+1} + \frac{1}{2}U_{j+2}^{n+1}\right] = 0$$

let $\boxed{\mu = \frac{3\Delta t}{h}}$, $\boxed{\tau = \frac{\Delta t}{h^3}}$

At $j=1$:

$$F(1) = U_1^{n+1} - U_1^{n} + \mu\left[f(U_2^{n+1}) - f(\overset{U_N^{n+1}}{\cancel{U_0^{n+1}}})\right] + \tau\left[-\frac{1}{2}\overset{U_{N-1}^{n+1}}{\cancel{U_{-1}^{n+1}}} + \overset{U_N^{n+1}}{\cancel{U_0^{n+1}}} - U_2^{n+1} + \frac{1}{2}U_3^{n+1}\right] = 0$$

At $j=2$:

$$F(2) = U_2^{n+1} - U_2^{n} + \mu\left[f(U_3^{n+1}) - f(U_1^{n+1})\right] + \tau\left[-\frac{1}{2}U_N^{n+1} + U_1^{n+1} - U_3^{n+1} + \frac{1}{2}U_4^{n+1}\right] = 0$$

For $j = 3 : N-1$

$$F(j) = U_j^{n+1} - U_j^{n} + \mu\left[f(U_{j+1}^{n+1}) - f(U_{j-1}^{n+1})\right] + \tau\left[-\frac{1}{2}U_{j-2}^{n+1} + U_{j-1}^{n+1} - U_{j+1}^{n+1} + \frac{1}{2}U_{j+2}^{n+1}\right] = 0$$

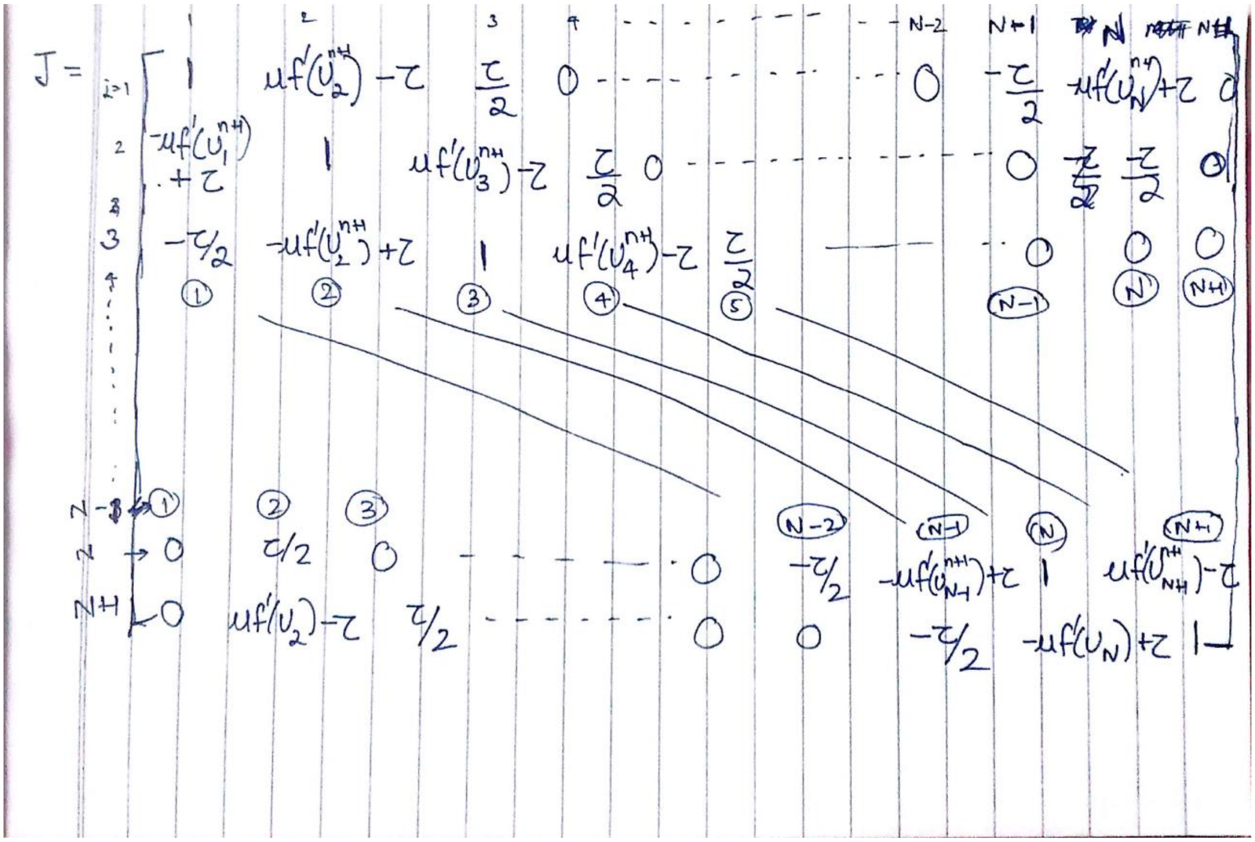
end.

At $j=N$:

$$F(N) = U_N^{n+1} - U_N^{n} + \mu\left[f(U_{N+1}^{n+1}) - f(U_{N-1}^{n+1})\right] + \tau\left[-\frac{1}{2}U_{N-2}^{n+1} + U_{N-1}^{n+1} - U_{N+1}^{n+1} + \frac{1}{2}U_2^{n+1}\right] = 0$$

At $j=N+1$:

$$F(N+1) = U_{N+1}^{n+1} - U_{N+1}^{n} + \mu\left[f(U_2^{n+1}) - f(U_N^{n+1})\right] + \tau\left[-\frac{1}{2}U_{N-1}^{n+1} + U_N^{n+1} - U_2^{n+1} + \frac{1}{2}U_3^{n+1}\right] = 0$$



$$J = \begin{array}{c|cccccccccc}
 & 1 & 2 & 3 & 4 & \cdots & & N-2 & N+1 & N & N+1 \\
i=1 & 1 & uf'(U_2^{n+1})-\tau & \frac{\tau}{2} & 0 & \cdots & & 0 & -\frac{\tau}{2} & -uf'(U_N^{n+1})+\tau & 0 \\
2 & -uf'(U_1^{n+1})+\tau & 1 & uf'(U_3^{n+1})-\tau & \frac{\tau}{2} & 0 & \cdots & & 0 & -\frac{\tau}{2} & -\frac{\tau}{2} & 0 \\
3 & -\tau/2 & -uf'(U_2^{n+1})+\tau & 1 & uf'(U_4^{n+1})-\tau & \frac{\tau}{2} & \cdots & & 0 & 0 & 0 \\
4 & \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & & & (N-1) & (N) & (N+1) \\
\vdots & & & & & & & & & & \\
N-1 & \textcircled{1} & \textcircled{2} & \textcircled{3} & & & & (N-2) & (N-1) & (N) & (N+1) \\
N & 0 & \tau/2 & 0 & \cdots & 0 & & -\tau/2 & -uf'(U_{N-1}^{n+1})+\tau & 1 & uf'(U_{N+1}^{n+1})-\tau \\
N+1 & 0 & uf'(U_2)-\tau & \tau/2 & \cdots & 0 & & 0 & -\tau/2 & -uf'(U_N)+\tau & 1
\end{array}$$

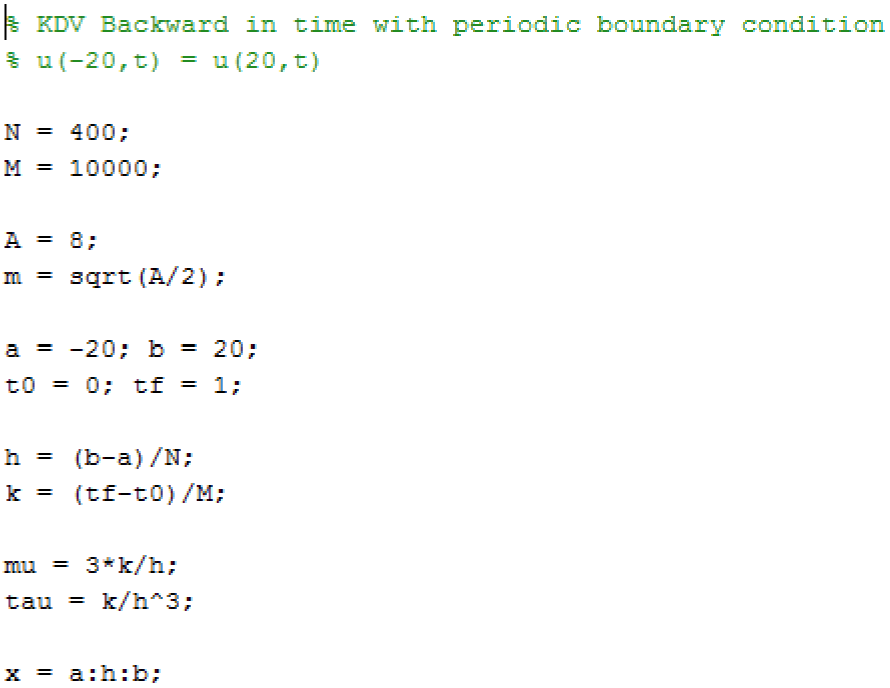
$$u_t + 6uu_x + u_{xxx} = 0;$$

$$u(x, 0) = A.\sec h^2\,(Kx);$$

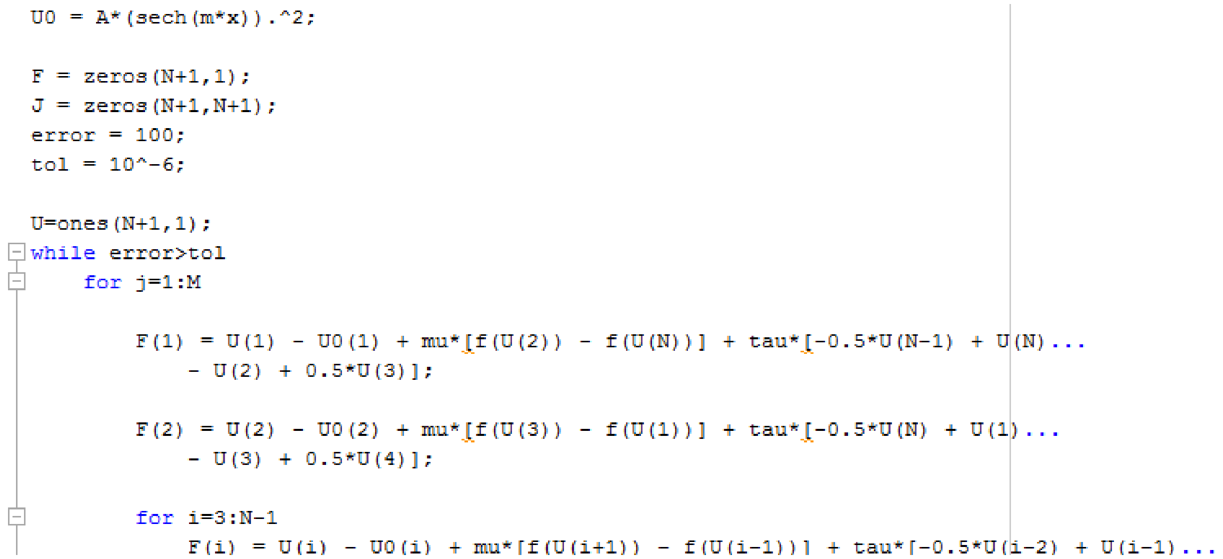
$$u(-20, t) = \ u(20, t)$$

-20 < x < 20; t > 0.

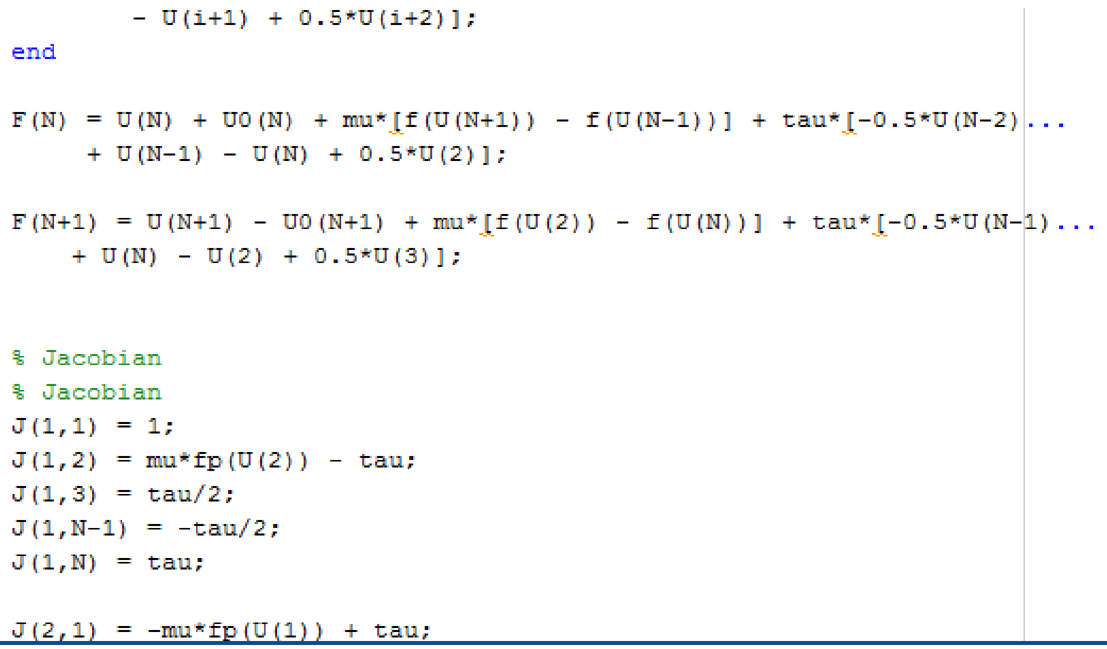
Matlab Implementation.



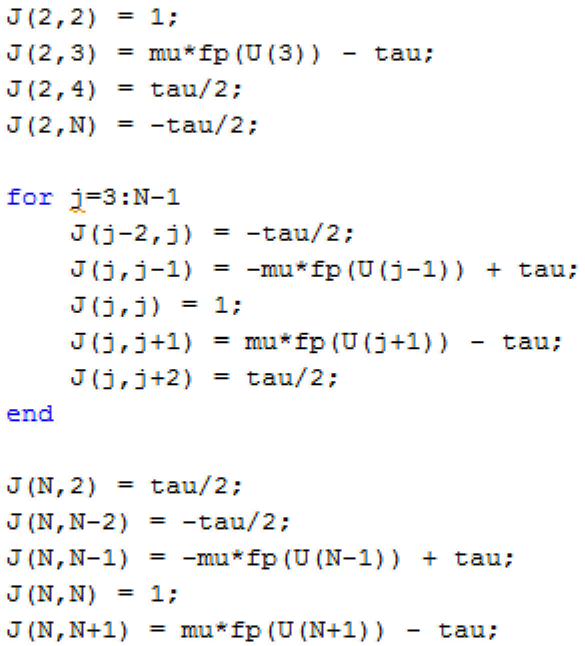
```matlab
% KDV Backward in time with periodic boundary condition
% u(-20,t) = u(20,t)

N = 400;
M = 10000;

A = 8;
m = sqrt(A/2);

a = -20; b = 20;
t0 = 0; tf = 1;

h = (b-a)/N;
k = (tf-t0)/M;

mu = 3*k/h;
tau = k/h^3;

x = a:h:b;
```



```matlab
U0 = A*(sech(m*x)).^2;

F = zeros(N+1,1);
J = zeros(N+1,N+1);
error = 100;
tol = 10^-6;

U=ones(N+1,1);
while error>tol
    for j=1:M

        F(1) = U(1) - U0(1) + mu*[f(U(2)) - f(U(N))] + tau*[-0.5*U(N-1) + U(N)...
            - U(2) + 0.5*U(3)];

        F(2) = U(2) - U0(2) + mu*[f(U(3)) - f(U(1))] + tau*[-0.5*U(N) + U(1)...
            - U(3) + 0.5*U(4)];

        for i=3:N-1
            F(i) = U(i) - U0(i) + mu*[f(U(i+1)) - f(U(i-1))] + tau*[-0.5*U(i-2) + U(i-1)...
```



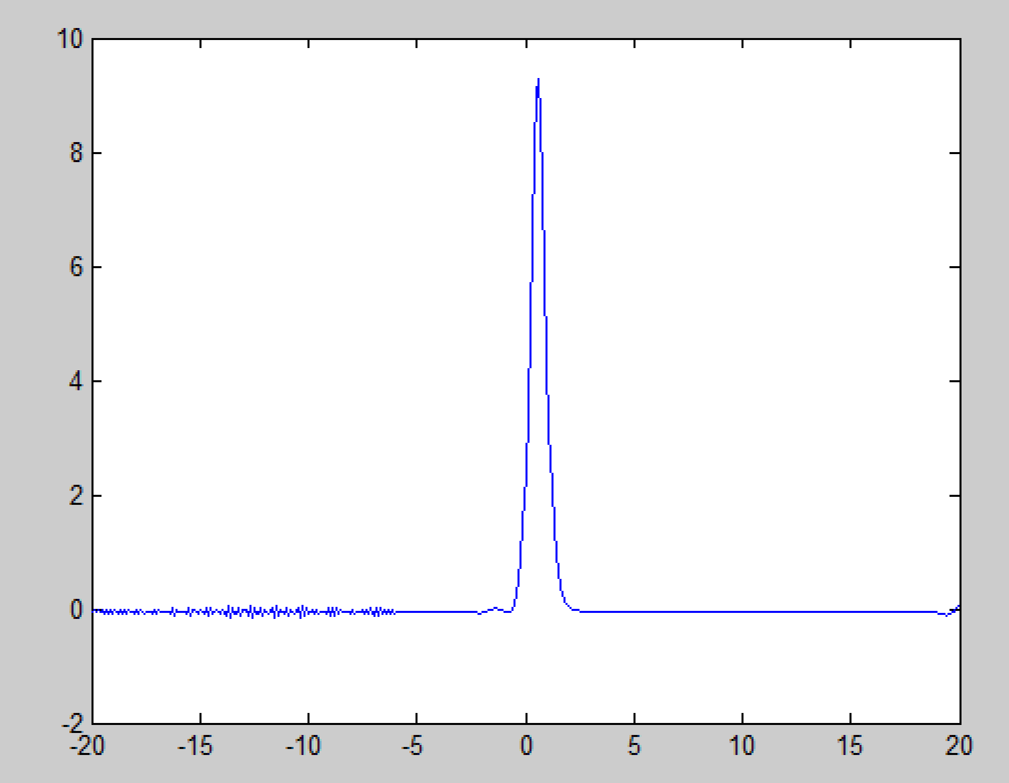
```
        - U(i+1) + 0.5*U(i+2)];
end

F(N) = U(N) + U0(N) + mu*[f(U(N+1)) - f(U(N-1))] + tau*[-0.5*U(N-2)...
     + U(N-1) - U(N) + 0.5*U(2)];

F(N+1) = U(N+1) - U0(N+1) + mu*[f(U(2)) - f(U(N))] + tau*[-0.5*U(N-1)...
    + U(N) - U(2) + 0.5*U(3)];


% Jacobian
% Jacobian
J(1,1) = 1;
J(1,2) = mu*fp(U(2)) - tau;
J(1,3) = tau/2;
J(1,N-1) = -tau/2;
J(1,N) = tau;

J(2,1) = -mu*fp(U(1)) + tau;
```



```
J(2,2) = 1;
J(2,3) = mu*fp(U(3)) - tau;
J(2,4) = tau/2;
J(2,N) = -tau/2;

for j=3:N-1
    J(j-2,j) = -tau/2;
    J(j,j-1) = -mu*fp(U(j-1)) + tau;
    J(j,j) = 1;
    J(j,j+1) = mu*fp(U(j+1)) - tau;
    J(j,j+2) = tau/2;
end

J(N,2) = tau/2;
J(N,N-2) = -tau/2;
J(N,N-1) = -mu*fp(U(N-1)) + tau;
J(N,N) = 1;
J(N,N+1) = mu*fp(U(N+1)) - tau;
```

```
        J(N+1,2) = mu*fp(U(2)) - tau;
        J(N+1,3) = tau/2;
        J(N+1,N-1) = -tau/2;
        J(N+1,N) = -mu*fp(U(N)) + tau;
        J(N+1,N+1) = 1;

        DELT = J\F;
        U1 = U - DELT;
        error = max(abs(U1-U));
        U = U1;
        U0 = U1;

        plot(x,U);
        axis([a,b,-2,10])
        pause(1E-10);

    end
end
```

Plot is as follows.



Using Crank Nicholson Scheme:

CRANK NICOLSON SCHEME:-

$$u_t + 6uu_x + u_{xxx} = 0 \quad , \quad -20 < x < 20 \, , \; t > 0$$

IC: $u(x,0) = A\,\mathrm{sech}^2(kx)$

BC: $u(-20,t) = u(20,t)$

Writing out the PDE in its conservation form:-

$$u_t + 6\left(\frac{u^2}{2}\right)_x + u_{xxx} = 0$$

$$\Rightarrow u_t + 6\,(f(u))_x + u_{xxx} = 0 \quad , \quad f(u) = \frac{u^2}{2}$$

Now, in Crank Nicolson scheme,

$$u_t = \frac{U_j^{n+1} - U_j^n}{\Delta t}$$

$$(f(u))_x = \frac{1}{2}\left(\frac{f(U_{j+1}^{n+1}) - f(U_{j-1}^{n+1})}{2h} + \frac{f(U_{j+1}^n) - f(U_{j-1}^n)}{2h} \right)$$

$$= \frac{1}{4h}\left(f(U_{j+1}^{n+1}) - f(U_{j-1}^{n+1}) + f(U_{j+1}^n) - f(U_{j-1}^n) \right)$$

$$u_{xxx} = \frac{1}{4h^3}\left(U_{j+2}^{n+1} - 2U_{j+1}^{n+1} + 2U_{j-1}^{n+1} - U_{j-2}^{n+1} + U_{j+2}^n - 2U_{j+1}^n + 2U_{j-1}^n - U_{j-2}^n \right)$$

$\Rightarrow$ The scheme becomes.

$$\frac{U_j^{n+1} - U_j^n}{\Delta t} + \frac{6}{4h}\left(f(U_{j+1}^{n+1}) - f(U_{j-1}^{n+1}) + f(U_{j+1}^n) - f(U_{j-1}^n) \right) + \frac{1}{4h^3}\left(U_{j+2}^{n+1} - 2U_{j+1}^{n+1} + 2U_{j-1}^{n+1} - U_{j-2}^{n+1} + U_{j+2}^n - 2U_{j+1}^n + 2U_{j-1}^n - U_{j-2}^n \right) = 0$$

Giving us a non- Linear system of equation :- So using Newton Raphson Method.

$$U^{n+1} = U^{n} - J^{-1}F \qquad J : \text{Jacobian of "F"}.$$

Now, writing :-

$$F(j) = U_j^{n+1} - U_j^{n} + \frac{3\Delta t}{2h}\left(f(U_{j+1}^{n+1}) - f(U_{j-1}^{n+1}) + f(U_{j+1}^{n}) - f(U_{j-1}^{n}) \right)$$
$$+ \frac{\Delta t}{4h^3}\left(\begin{array}{l} U_{j+2}^{n+1} - 2U_{j+1}^{n+1} + 2U_{j-1}^{n+1} - U_{j-2}^{n+1} + \\ U_{j+2}^{n} - 2U_{j+1}^{n} + 2U_{j-1}^{n} - U_{j-2}^{n} \end{array} \right)$$

$$\boxed{\mu = \frac{3\Delta t}{2h}} \quad \boxed{\tau = \frac{\Delta t}{4h^3}}$$

$j=1$

$$F(1) = U_1^{n+1} - U_1^{n} + \mu\left(f(U_2^{n+1}) - f(\overset{U_N^{n+1}}{\cancel{U_0^{n+1}}}) + f(U_2^{n}) - f(\overset{U_N^{n}}{\cancel{U_0^{n}}}) \right)$$
$$+ \tau\left(\begin{array}{l} U_3^{n+1} - 2U_2^{n+1} + 2U_N^{n+1} - U_{N-1}^{n+1} - \\ U_3^{n} - 2U_2^{n} + 2U_N^{n} - U_{N-1}^{n} \end{array} \right)$$

$j=2$

$$F(2) = U_2^{n+1} - U_2^{n} + \mu\left(f(U_3^{n+1}) - f(U_1^{n+1}) + f(U_2^{n}) - f(U_1^{n}) \right)$$
$$+ \tau\left(U_4^{n+1} - 2U_3^{n+1} + 2U_1^{n+1} - U_N^{n+1} + U_4^{n} - 2U_3^{n} + 2U_1^{n} - U_N^{n} \right)$$

for $j = 3 : N-1$

$$F(j) = U_j^{n+1} - U_j^{n} + \mu\left(f(U_{j+1}^{n+1}) - f(U_{j-1}^{n+1}) + f(U_{j+1}^{n}) - f(U_{j-1}^{n}) \right)$$
$$+ \tau\left(\begin{array}{l} U_{j+2}^{n+1} - 2U_{j+1}^{n+1} + 2U_{j-1}^{n+1} - U_{j-2}^{n+1} + \\ U_{j+2}^{n} - 2U_{j+1}^{n} + 2U_{j-1}^{n} - U_{j-2}^{n} \end{array} \right)$$

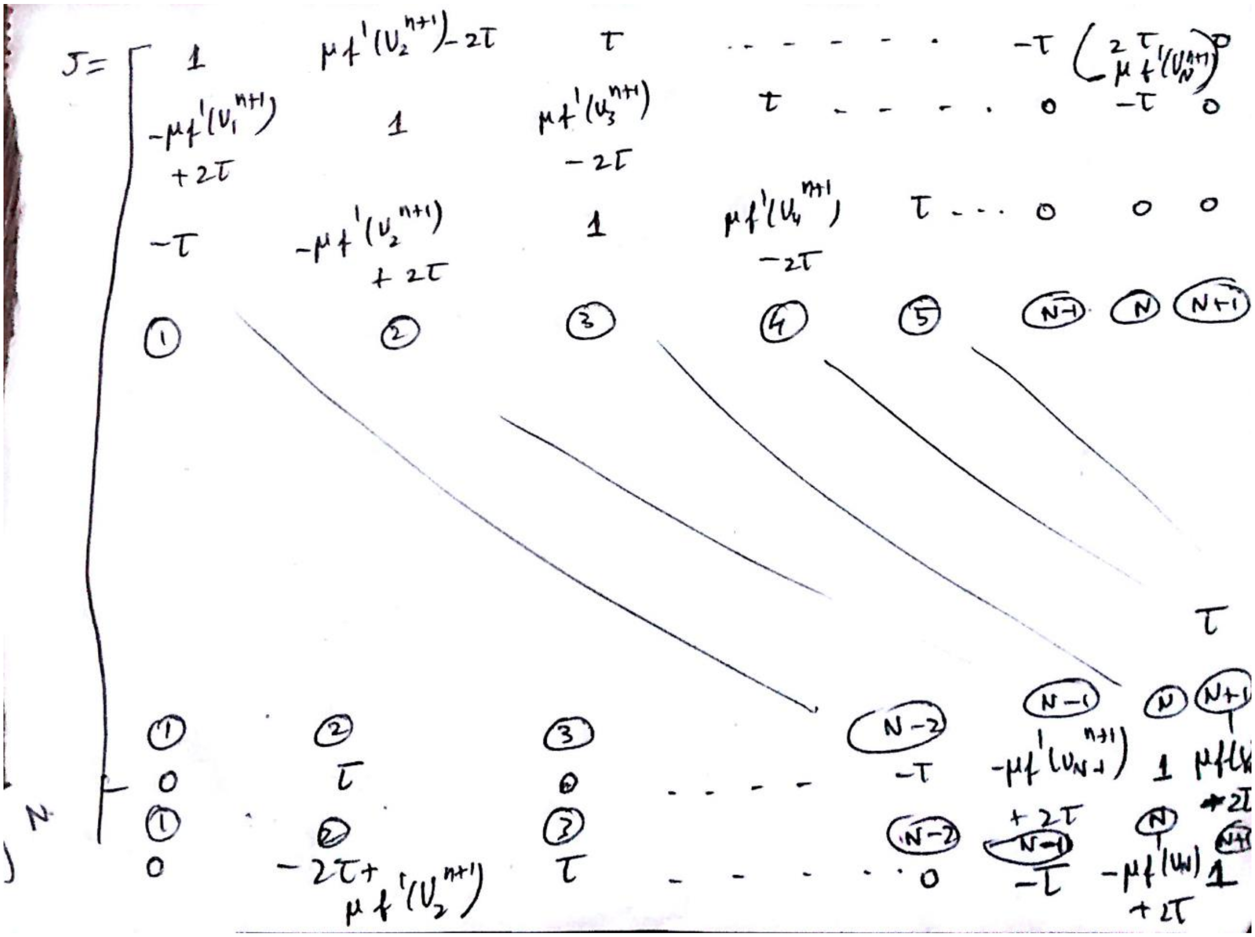
end

$j=N$

$$F(N) = U_N^{n+1} - U_N^{n} + \mu\left(f(U_{N+1}^{n+1}) - f(U_{N-1}^{n+1}) + f(U_{N+1}^{n}) - f(U_{N-1}^{n}) \right)$$
$$+ \tau\left(\begin{array}{l} \underset{U_2^{n+1}}{\cancel{U_{N+2}^{n+1}}} - 2U_{N+1}^{n+1} + 2U_{N-1}^{n+1} - U_{N-2}^{n+1} + \\ \underset{U_2^{n}}{\cancel{U_{N+2}^{n}}} - 2U_{N+1}^{n} + 2U_{N-1}^{n} - U_{N-2}^{n} \end{array} \right)$$

$j=N+1$

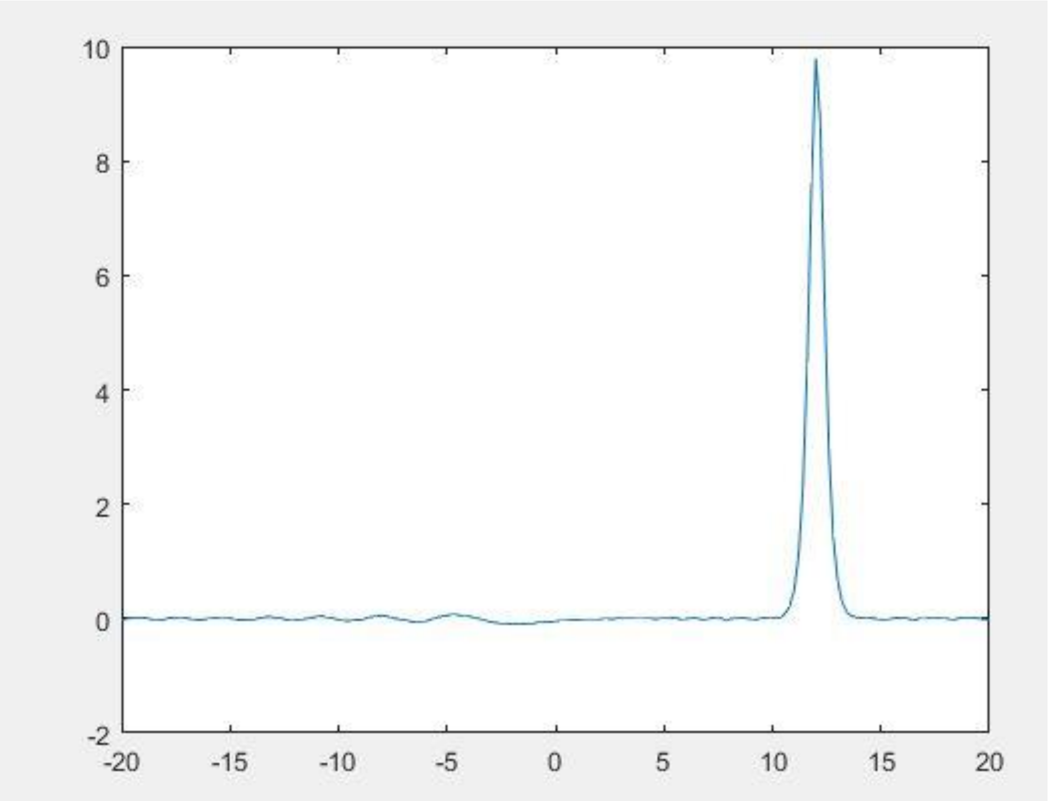
$$F(N+1) = U_{N+1}^{n+1} - U_{N+1}^{n} + \mu\left(f(U_2^{n+1}) - f(U_N^{n+1}) + f(U_2^{n}) - f(U_N^{n}) \right)$$
$$+ \tau\left(\begin{array}{l} U_3^{n+1} - 2U_2^{n+1} + 2U_N^{n+1} - U_{N-1}^{n+1} \\ + U_3^{n} - 2U_2^{n} + 2U_N^{n} - U_{N-1}^{n} \end{array} \right)$$


$J =$
1
$\mu f'(U_2^{n+1}) - 2\tau$
τ
$-\tau$
$2\tau - \mu f'(U_N^{n+1})$
0
$-\mu f'(U_1^{n+1}) + 2\tau$
1
$\mu f'(U_3^{n+1}) - 2\tau$
τ
0
$-\tau$
0
$-\tau$
$-\mu f'(U_2^{n+1}) + 2\tau$
1
$\mu f'(U_4^{n+1}) - 2\tau$
τ
0
0
0
1
2
3
4
5
N-1
N
N+1
τ
1
2
3
N-2
N-1
N
N+1
0
τ
0
$-\tau$
$-\mu f'(U_{N-1}^{n+1}) + 2\tau$
1
$\mu f'(U$
$+2\tau$
N
1
2
3
N-2
N-1
N
N+1
0
$-2\tau + \mu f'(U_2^{n+1})$
τ
0
$-\tau$
$-\mu f'(U_N) + 2\tau$
1

```
Crank Nicolson with periodic boundary condition
% u(-20,t) = u(20,t)

N = 200;
M = 1000;

A = 10;
m = sqrt(A/2);

a = -20; b = 20;
t0 = 0; tf = 1;

h = (b-a)/N;
k = (tf-t0)/M;

mu = 6*k/(4*h);
tau = k/(4*h^3);

x = a:h:b;

u0 = A*(sech(m*x)).^2;

F = zeros(N+1,1);
J = zeros(N+1);
```

```
error = 100;
tol = 10^-6;

U = ones(N+1,1);
while error > tol
    for p=1:M
        F(1) = U(1) - u0(1) + mu*(f(U(2))-f(U(N)) + f(u0(2))-f(u0(N))) + ...
            tau*(U(3)-2*U(2)+2*U(N)-U(N-1) + u0(3)-2*u0(2)+2*u0(N)-u0(N-1));

        F(2) = U(2) - u0(2) + mu*(f(U(3))-f(U(1)) + f(u0(3))-f(u0(1))) + ...
            tau*(U(4)-2*U(3)+2*U(1)-U(N) + u0(4)-2*u0(3)+2*u0(1)-u0(N));

        F(N) = U(N) - u0(N) + mu*(f(U(N+1))-f(U(N-1)) + f(u0(N+1))-f(u0(N-
1))) + ...
            tau*(U(2)-2*U(N+1)+2*U(N-1)-U(N-2) + u0(2)-2*u0(N+1)+2*u0(N-1)-
u0(N-2));

        F(N+1) = U(N+1) - u0(N+1) + mu*(f(U(2))-f(U(N)) + f(u0(2))-f(u0(N)))
+ ...
            tau*(U(3)-2*U(2)+2*U(N)-U(N-1) + u0(3)-2*u0(2)+2*u0(N)-u0(N-1));

        for j=3:N-1
            F(j) = U(j) - u0(j) + mu*(f(U(j+1))-f(U(j-1)) + f(u0(j+1))-
f(u0(j-1))) + ...
                tau*(U(j+2)-2*U(j+1)+2*U(j-1)-U(j-2) + u0(j+2)-
2*u0(j+1)+2*u0(j-1)-u0(j-2));
        end

        % Jacobian
        % Jacobian
        J(1,1) = 1;
        J(1,2) = mu*df(U(2)) - 2*tau;
        J(1,3) = tau;
        J(1,N-1) = -tau;
        J(1,N) = -mu*df(U(N))+2*tau;

        J(2,1) = -mu*df(U(1)) + 2*tau;
        J(2,2) = 1;
        J(2,3) = mu*df(U(3)) - 2*tau;
        J(2,4) = tau;
        J(2,N) = -tau;

        for j=3:N-1
            J(j,j) = 1;
            J(j,j+2) = tau;
            J(j,j-2) = -tau;
            J(j-1,j) = mu*df(U(j))-2*tau;
            J(j,j-1) = -mu*df(U(j-1))+2*tau;
        end
        J(N,2) = tau;
        J(N,N-1) = -mu*df(U(N-1))+2*tau;
        J(N,N) = 1;
        J(N,N-2) = -tau;
        J(N,N+1) = mu*df(U(N)) - 2*tau;
        J(N+1,2) = mu*df(U(2))-2*tau;
        J(N+1,3) = tau;
        J(N+1,N-1) = -tau;
        J(N+1,N) = -mu*df(U(N))+2*tau;
        J(N+1,N+1) = 1;

        DELT = J\F;
        U2 = U - DELT;
        error = max(abs(U2-U));
        U = U2;
        u0 = U2;
        plot(x,U)
        axis([a,b,-2,10])
        pause(0.001)
    end
end


function V = f(x)
    V = x^2/2;
end


function V = df(x)
    V = x;
end
```

Output:



References

1. R. Courant, "Variational methods for the solution of Problems of Equilibrium and vibrations", *Bull. Am. Math. Soc.*, vol. 49, 1943.

2. J. F. Hoburg, J. L. Davis, "A student-oriented finite element program for electromagnetism problems", *IEEE Trans. Educ.*, vol. E-26, pp. 138-142, Nov. 1983.

3. The Finite Element Method in Electromagnetics, 3rd Edition

Jian-Ming Jin, Mar 2014.

4. Wikipedia

5. Www.mathworks.com

Thank You