21MAT204

Mathematics For Intelligent Systems-3

KALYANA SUNDARAM
CB.EN.U4AIE21120

Step 1

The wave equation is a second-order partial differential equation that describes the propagation of waves through a medium. In this case, the medium is a 1D domain with length 2. The wave equation in 1D can be written as:

du/dt = c * d^2u/dx^2

where u is the displacement of the wave at position x and time t, c is the wave speed, and dx and dt are the spatial and temporal steps, respectively.

the second derivative is approximated with the following finite difference equation:

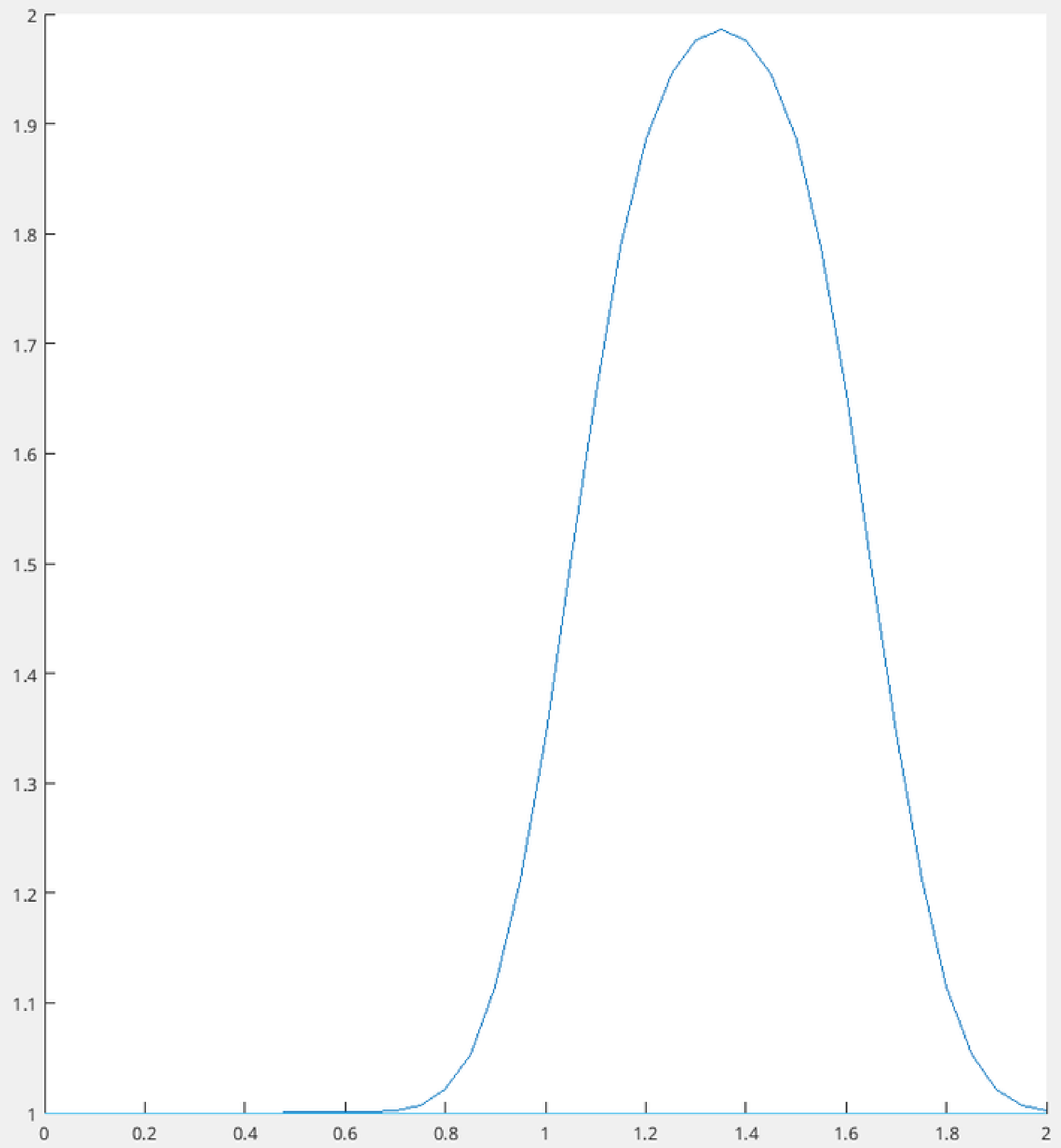
du^2/dt = (u(i) - u(i-1)) / dx

This equation is used to update the displacement of the wave u at each time step t.

Using a loop to iterate over the number of time steps **nt**. At each time step, it calculates the new displacement **u** at each spatial position using the finite difference equation mentioned above.

```
clear all;clc;
nx = 41;
dx = 2/(nx-1);
nt = 25;
dt = .025;
c = 1;
%creating one array of size nx with 1 as elements
u = ones(nx);
%setting u = 2 between 0.5 and 1 as per our I.C.s
u(0.5/dx:1/dx+1) = 2;


un = ones(nx);
for n = 1:nt
    un = u;
    for i = 2:nx
        u(i) = un(i) - c*dt/dx*(un(i)-un(i-1));
    end
end
hold on
plot(linspace(0,2,nx),u);
```

Code



Plot

Step 2

The wave equation is a second-order partial differential equation that describes the propagation of waves through a medium. In this case, the medium is a 1D domain with length 2. The wave equation in 1D can be written as:

du/dt = c * d^2u/dx^2

where u is the displacement of the wave at position x and time t, c is the wave speed, and dx and dt are the spatial and temporal steps, respectively.

the second derivative is approximated with the following finite difference equation:

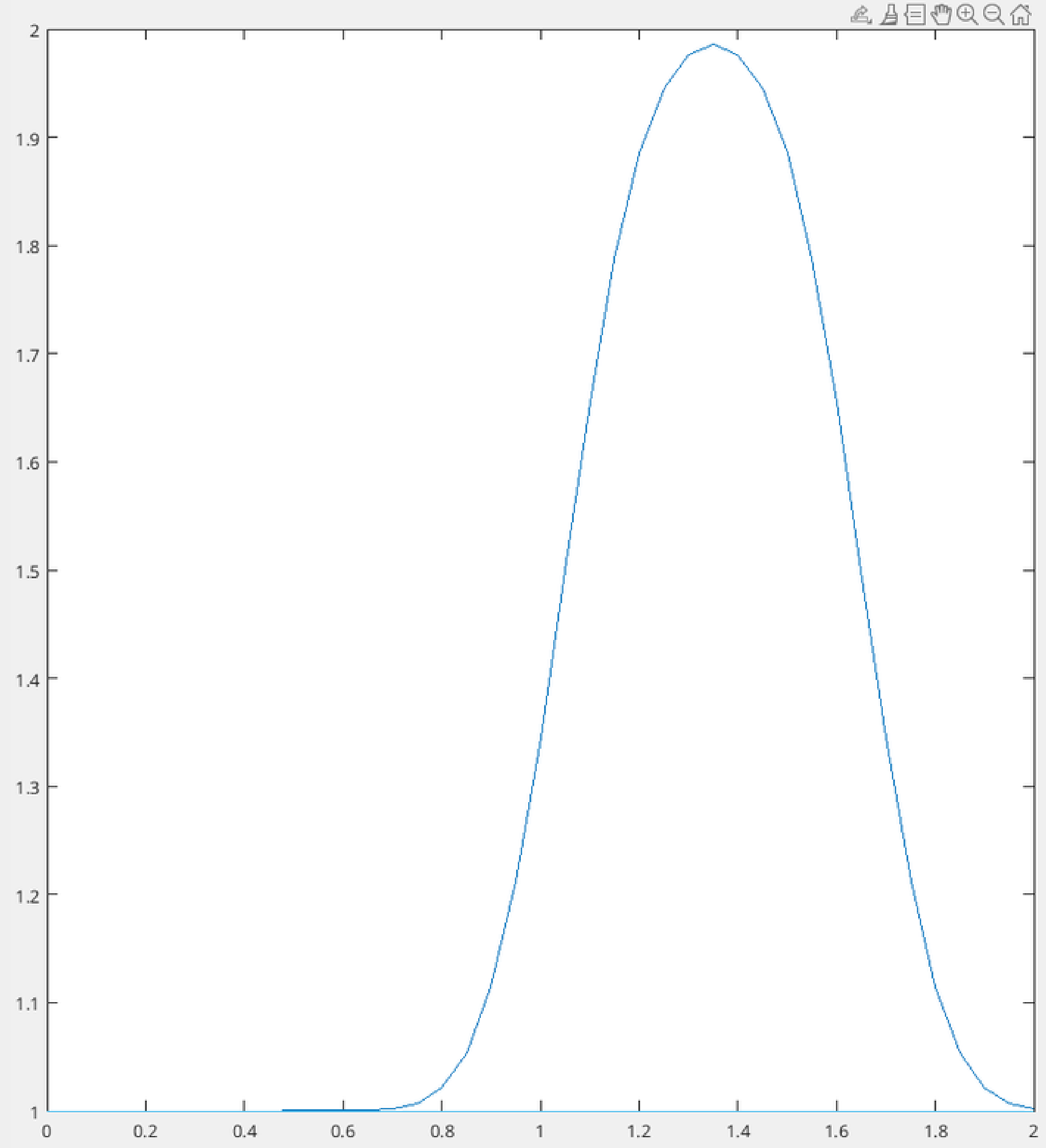
du^2/dt = (u(i) - u(i-1)) / dx

This equation is used to update the displacement of the wave **u** at each time step **t**.

Finally, the code uses a loop to iterate over the number of time steps **nt**. At each time step, it calculates the new displacement **u** at each spatial position using the finite difference equation mentioned above. The code plots the final displacement of the wave after all time steps have been completed.

```
clear all;clc;
nx = 41;
dx = 2/(nx-1);
nt = 25;
dt = .025;
c = 1;
%creating one array of size nx with 1 as elements
u = ones(nx);
%setting u = 2 between 0.5 and 1 as per our I.C.s
u(0.5/dx:1/dx+1) = 2;

un = ones(nx);

for n = 1:nt
    un = u;
    for i = 2:nx
        u(i) = un(i) - c*dt/dx*(un(i)-un(i-1));
    end
end

plot(linspace(0,2,nx),u)
```

Code



Plot

Step 3

The wave equation is a second-order partial differential equation that describes the propagation of waves through a medium. In this case, the medium is a 1D domain with length 2. The wave equation in 1D can be written as:

du/dt = c * d^2u/dx^2

where u is the displacement of the wave at position x and time t, c is the wave speed, and dx and dt are the spatial and temporal steps, respectively.

the second derivative is approximated with the following finite difference equation:

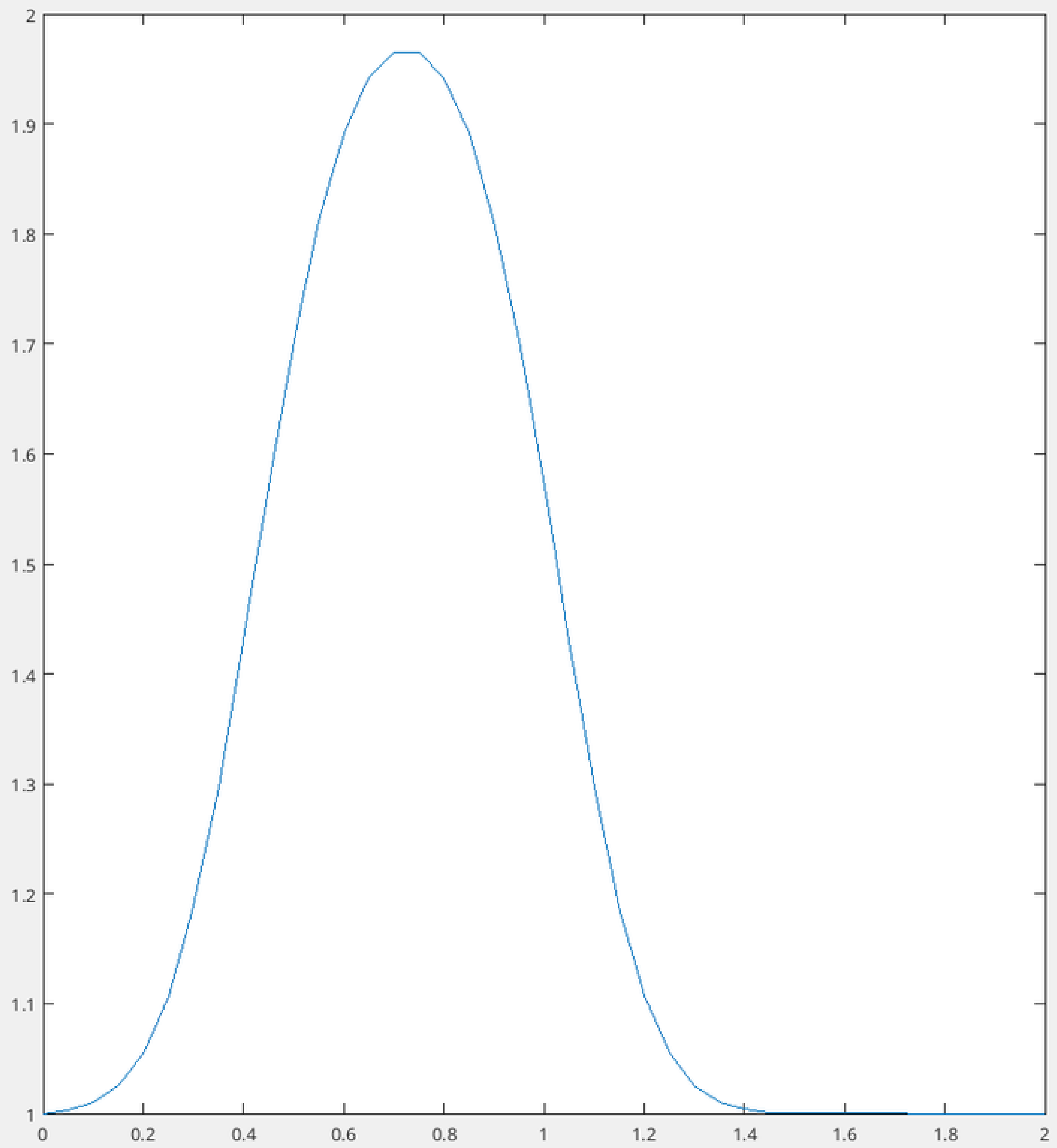
du^2/dt = (u(i) - u(i-1)) / dx

This equation is used to update the displacement of the wave u at each time step t.

Using a loop to iterate over the number of time steps **nt**. At each time step, it calculates the new displacement **u** at each spatial position using the finite difference equation mentioned above.

```
clear all;close all;

nx = 41;
dx = 2/(nx-1);
nt = 20;
nu = 0.3;
sigma = 0.2;
dt = sigma*dx^2/nu;

u = ones(nx,1);
u(0.5/dx:1/dx+1) = 2;

un = ones(nx,1);

for n = 1:nt
    un = u;
    for i = 2:nx-1
        u(i) = un(i) + nu*dt/dx^2*(un(i+1)-2*un(i)+un(i-1));
    end
end

plot(linspace(0,2,nx),u)
```

Code



Plot

Step 4

The wave equation is a second-order partial differential equation that describes the propagation of waves through a medium. In this case, the medium is a 1D domain with length 2. The wave equation in 1D can be written as:

du/dt = c * d^2u/dx^2

where u is the displacement of the wave at position x and time t, c is the wave speed, and dx and dt are the spatial and temporal steps, respectively.

the second derivative is approximated with the following finite difference equation:

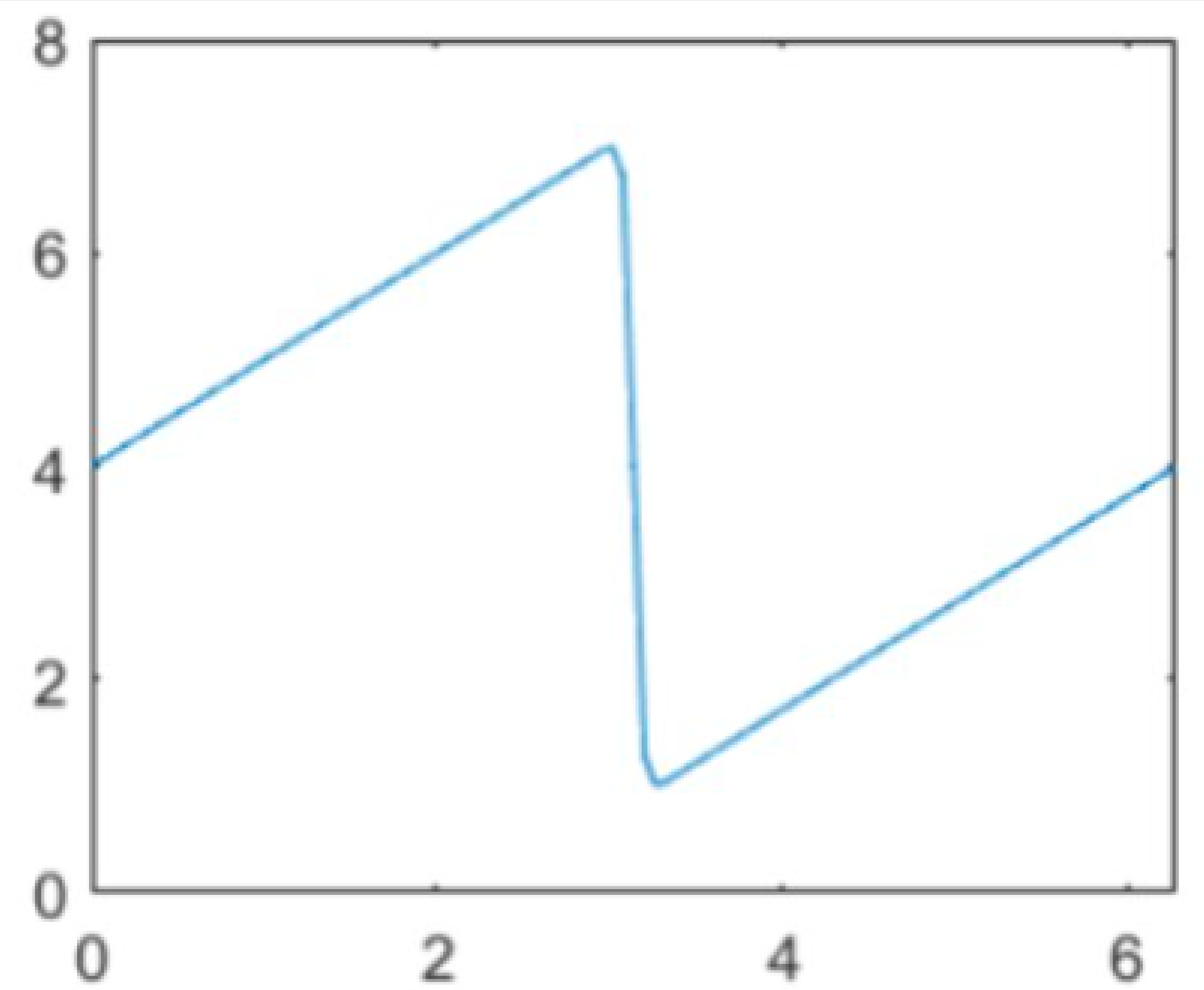
du^2/dt = (u(i) - u(i-1)) / dx

This equation is used to update the displacement of the wave u at each time step t.

Using a loop to iterate over the number of time steps **nt**. At each time step, it calculates the new displacement **u** at each spatial position using the finite difference equation mentioned above.

```
clear all;clc;

%creating symbols

syms x y t
phi =exp(-(x-4*t)^2/(4*nu*(t+1)))+exp(-(x-4*t-2*sym(pi)^2/(4*nu*(t+1))));
dphix = diff(phi,x);
fn = -2 *nu * dphix/phi +4 ; fn1 = subs(fn,t,0);
fn2 = subs(fn1,nu,0.07);

nx = 101;

xrange = linspace(0,2*pi,nx);
u = eval(subs(fn2,x,xrange));
plot(xrange,u,'k-','linewidth',2);
fn1 = subs(fn,t,0.07);
nt = 100;dx = 2*pi/(nx-1); nu = 0.07; dt = dx*nu;
p = plot(nan,nan)
p.XData = xrange;
p.YData = u;

for i=1:nt
    tval = i*dt;
    fn1 = subs(fn,t,tval);
    u = eval(subs(fn1,x,xrange));
    p.YData = u;
    drawnow
end
```

Code



Plot

Step 5

defines a 1D heat conduction problem and uses a finite difference method to approximate the solution.

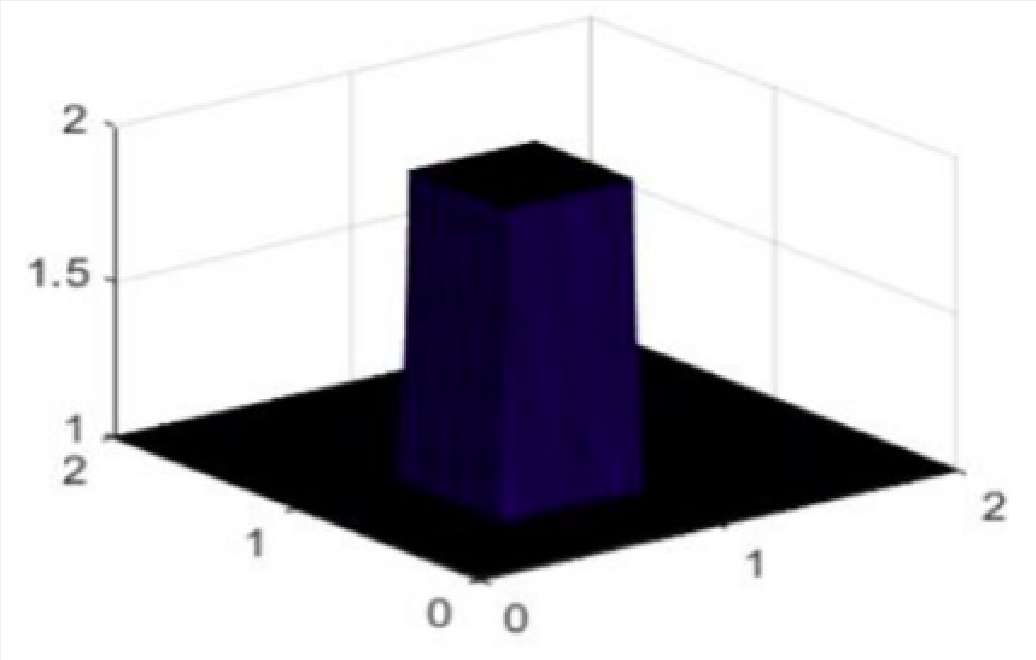
The problem domain is a rod of length 2 meters, divided into **nx** intervals of equal size **dx**. The temperature at each point in the rod is represented by a scalar value **u**. The rod is initially held at a uniform temperature of 1 degree Celsius, except for a section in the middle which is held at 2 degrees Celsius.

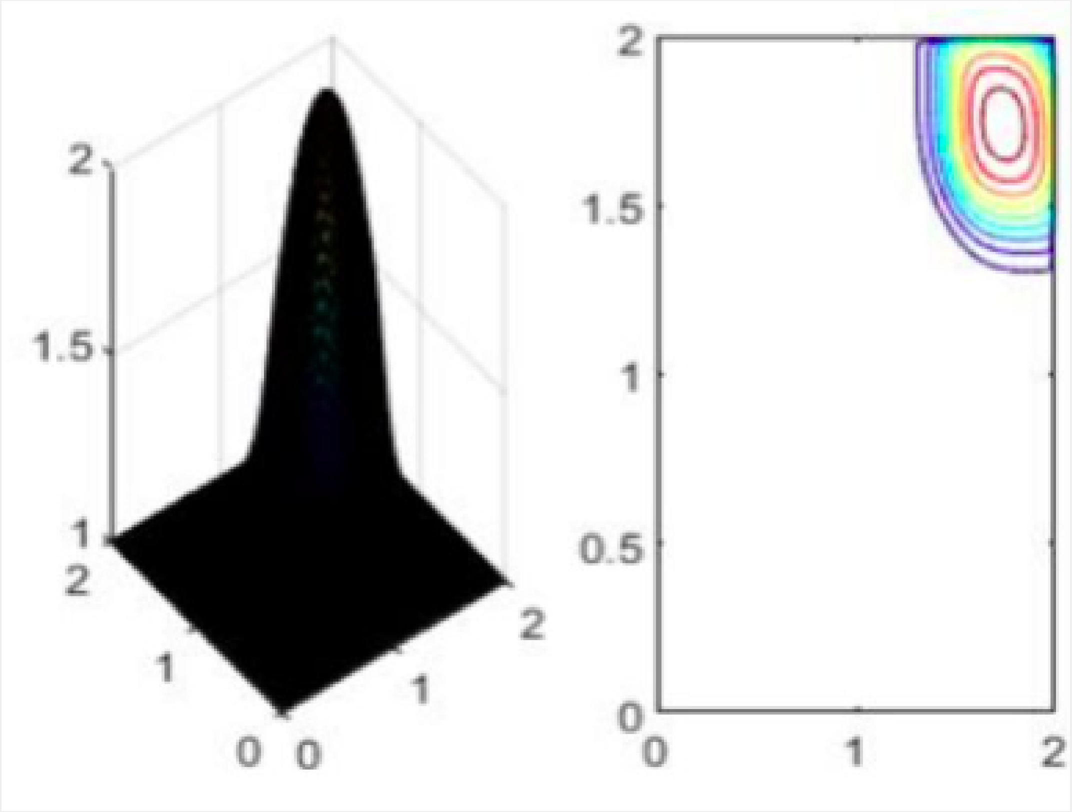
The temperature of the rod is evolved in time using a finite difference method. The time step dt is chosen such that the dimensionless number sigma = dt*nu/dx^2 is equal to 0.2, where nu is the thermal diffusivity of the rod. At each time step, the temperature at each point i in the rod is updated according to the following finite difference equation:

u(i) = un(i) + nudt/dx^2(un(i+1)-2*un(i)+un(i-1))

```
clear all;clc;
nx = 81;ny = 81;nt = 100;nit = 50;c = 1;
dx = 2/(nx-1);dy = 2/(ny-1);
sigma = .2;dt = sigma*dx;

x = linspace(0,2,nx);
y = linspace(0,2,ny);
%plot
[X,Y] = meshgrid(x,y);
colormap(jet(256));
subplot(1,2,3);
surf(X,Y,u);
set(gca,'fontsize',16);
v = VideoWriter('2DConvection.avi');
open(v);
[row col] = size(u);

for n = 1:nt
    un = u;
    for j = 2 row-1
        for i = 2 col-1
            u(j,i) = un(j,i) - c*dt/dx*(un(j,i)-un(j,i-1)) -
            c*dt/dy*(un(j,i)-un(j-1,i));
        end
    end
    u(1,:) = 1;
    u(end,:) = 1;
    u(:,1) = 1;
    u(:,end) = 1;
    colormap(jet);
    subplot(1,2,3);
    surf(X,Y,u);
    subplot(1,2,2);
    contourf(X,Y,u);
    frame = getframe(gcf);
    writeVideo(v,frame);
    pause(0.1);
end
close(v);
```

Code





Plot

Step 6

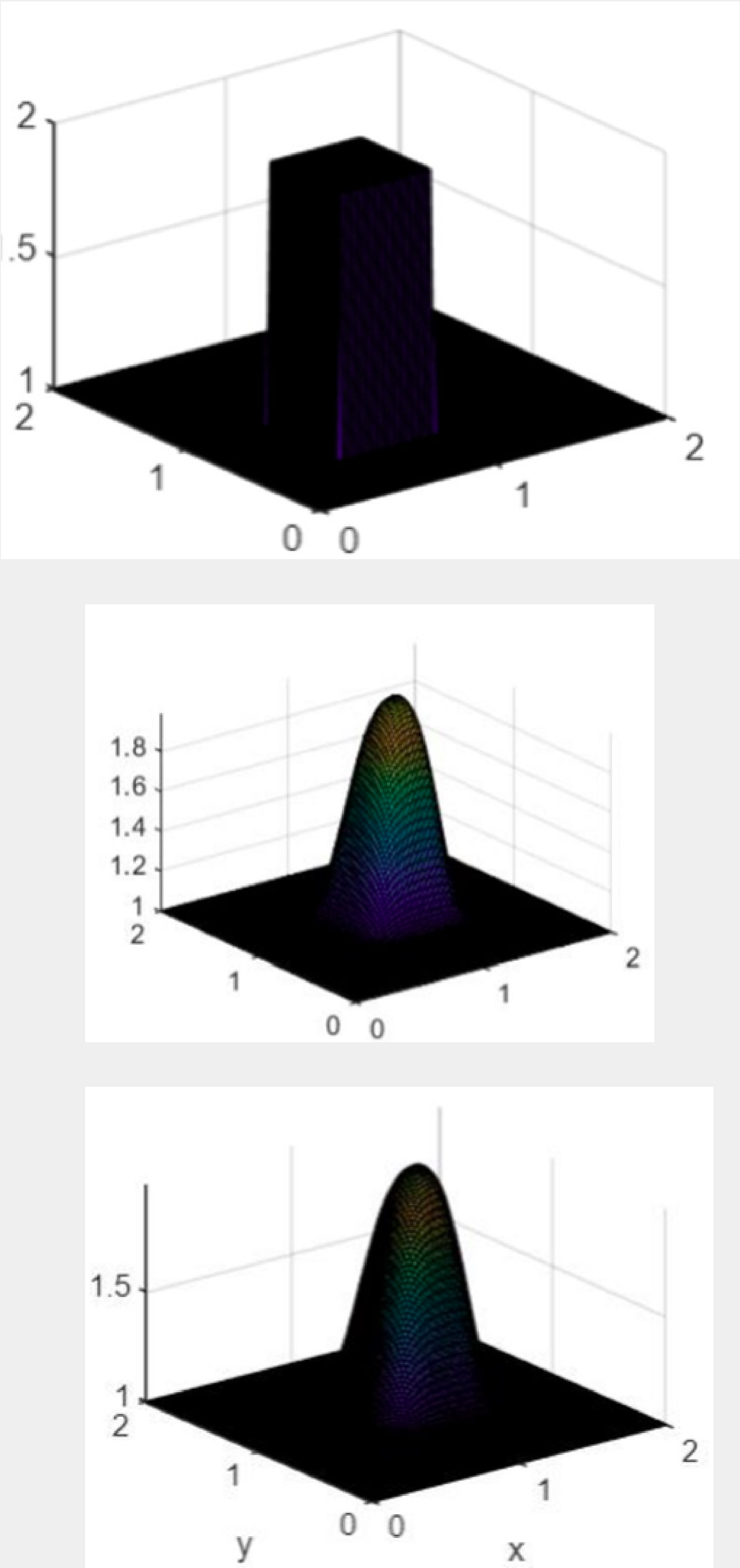
The problem domain is a square grid with dimensions 2 meters by 2 meters, divided into nx by ny intervals of equal size dx by dy. The temperature at each point in the grid is represented by a scalar value u. The velocity field in the grid is given by the vector field v, which has components v_x and v_y. The grid is initially held at a uniform temperature of 1 degree Celsius, except for a square region in the middle which is held at 2 degrees Celsius. The temperature and velocity field in the grid are evolved in time using a finite difference method. The time step dt is chosen such that the dimensionless number sigma = dt*c/dx is equal to 0.2, where c is a constant coefficient. At each time step, the temperature and velocity field at each point (i,j) in the grid are updated according to the following finite difference equations:

$$u(i,j) = un(i,j) - un(i,j)dt/dx(un(i,j)-un(i,j-1)) - vn(i,j)dt/dy(un(i,j)-un(i-1,j))$$

$$v(i,j) = vn(i,j) - un(i,j)dt/dx(vn(i,j)-vn(i,j-1)) - vn(i,j)dt/dy(vn(i,j)-vn(i-1,j))$$

```
nx = 101
ny = 101
nt = 80
c = 1
dx = 2 / (nx - 1)
dy = 2 / (ny - 1)
sigma = .2
dt = sigma * dx

x = linspace(0, 2, nx)
y = linspace(0, 2, ny)


u = ones(nx,1);
v = ones(nx,1);
un = ones(nx,1);
vn = ones(nx,1);

%Assign initial conditions
u(0.5/dx:1/dx+1) = 2;
v(0.5/dx:1/dx+1) = 2;

%Plot the initial conditions
figure(1)
plot(x,u(50,:))
hold on
plot(x,v(50,:))
legend('u','v')
title('Initial conditions')

%Loop through time
for n = 1:nt+1
    un = u
    vn = v
    u(2:end-1,2:end-1) = un(2:end-1,2:end-1) - un(2:end-1,2:end-1) * dt / dx * (un(2:end-1,2:end-1) - un(2:end-1,1:end-2)) - vn(2:end-1,2:end-1) * dt / dy * (un(2:end-1,2:end-1) - un(1:end-2,2:end-1))
    v(2:end-1,2:end-1) = vn(2:end-1,2:end-1) - un(2:end-1,2:end-1) * dt / dx * (vn(2:end-1,2:end-1) - vn(2:end-1,1:end-2)) - vn(2:end-1,2:end-1) * dt / dy * (vn(2:end-1,2:end-1) - vn(1:end-2,2:end-1))
end

%Plot the final conditions
figure(2)
plot(x,u(50,:))
hold on
plot(x,v(50,:))
legend('u','v')
title('Final conditions')
```

Code



Plot

Step 7

The problem domain is a square grid with dimensions 2 meters by 2 meters, divided into nx by ny intervals of equal size dx by dy. The velocity field in the grid is given by the vector field u, which has components u_x and u_y. The grid is initially held at a uniform velocity of (1,1), except for a square region in the middle which is held at (2,2).

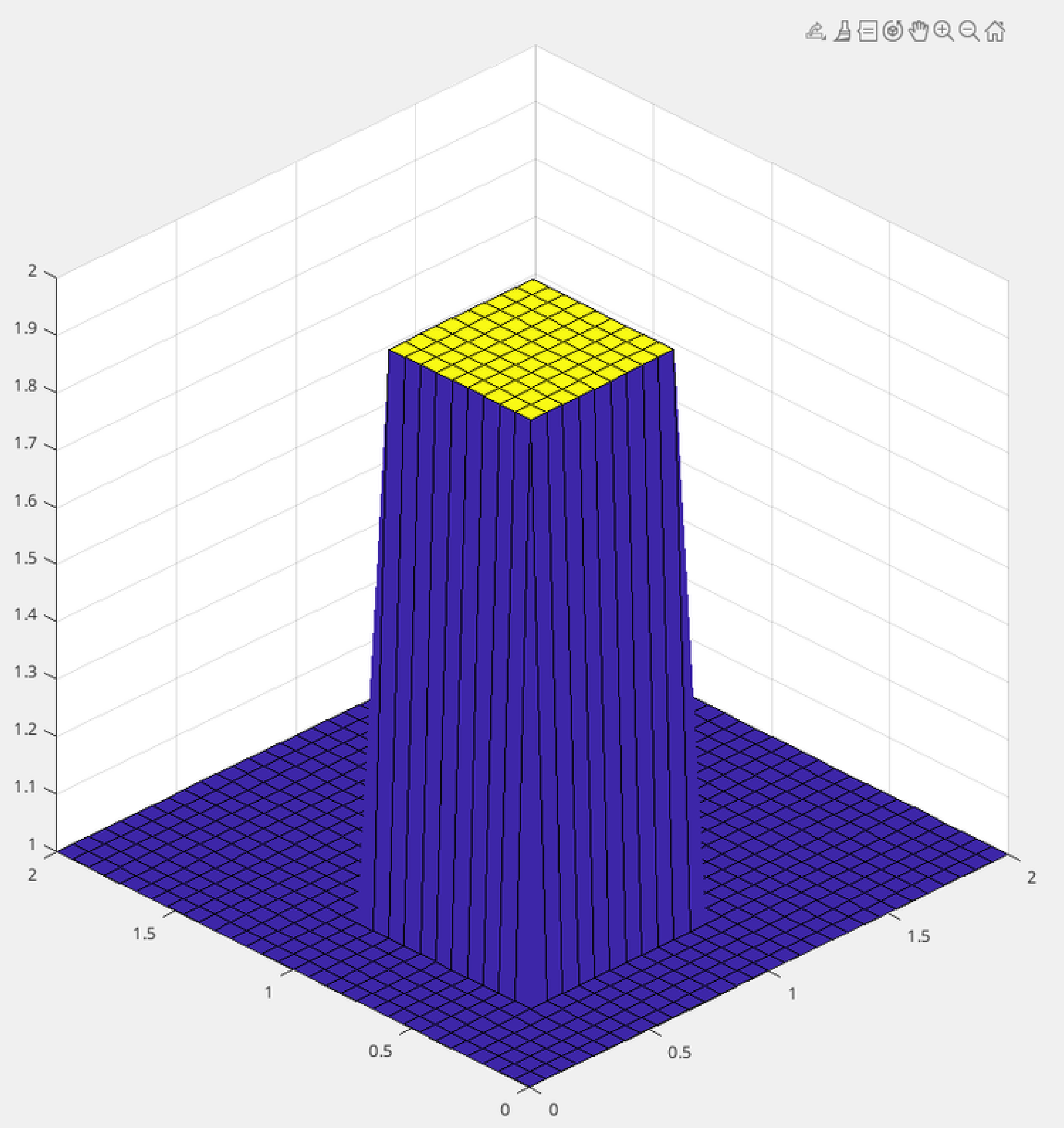
The velocity field in the grid is evolved in time using a finite difference method. The time step dt is chosen such that the dimensionless number sigma = dt*nu*(1/dx^2+1/dy^2) is equal to 0.25, where nu is the kinematic viscosity of the fluid. At each time step, the velocity field at each point (i,j) in the grid is updated according to the following finite difference equations:

un(i,j) = u(i,j) - dt/dxu(i,j)(u(i,j)-u(i-1,j)) - dt/dyv(i,j)(u(i,j)-u(i,j-1)) + nudt/dx^2(u(i+1,j)-2u(i,j)+u(i-1,j)) + nudt/dy^2*(u(i,j+1)-2*u(i,j)+u(i,j-1))

vn(i,j) = v(i,j) - dt/dxu(i,j)(v(i,j)-v(i-1,j)) - dt/dyv(i,j)(v(i,j)-v(i,j-1)) + nudt/dx^2(v(i+1,j)-2v(i,j)+v(i-1,j)) + nudt/dy^2*(v(i,j+1)-2*v(i,j)+v(i,j-1))

```
dy = 2 / (ny - 1)
sigma = .25
dt = sigma * dx * dy / nu

x = linspace(0, 2, nx)
y = linspace(0, 2, ny)

u = ones(ny,nx);
un = ones(ny,nx);
un = ones(ny,nx);
vn = ones(ny,nx);

u(floor(.5 / dy):floor(1 / dy + 1), floor(.5 / dx):floor(1 / dx + 1)) = 2;
v(floor(.5 / dy):floor(1 / dy + 1), floor(.5 / dx):floor(1 / dx + 1)) = 2;

figure;
surf(x,y,u);
```

Code



Plot

Step 8

The problem domain is a square grid with dimensions 2 meters by 2 meters, divided into nx by ny intervals of equal size dx by dy. The velocity field in the grid is given by the vector field u, which has components u_x and u_y. The grid is initially held at a uniform velocity of (1,1), except for a square region in the middle which is held at (2,2). The temperature at each point in the grid is given by the scalar field v.

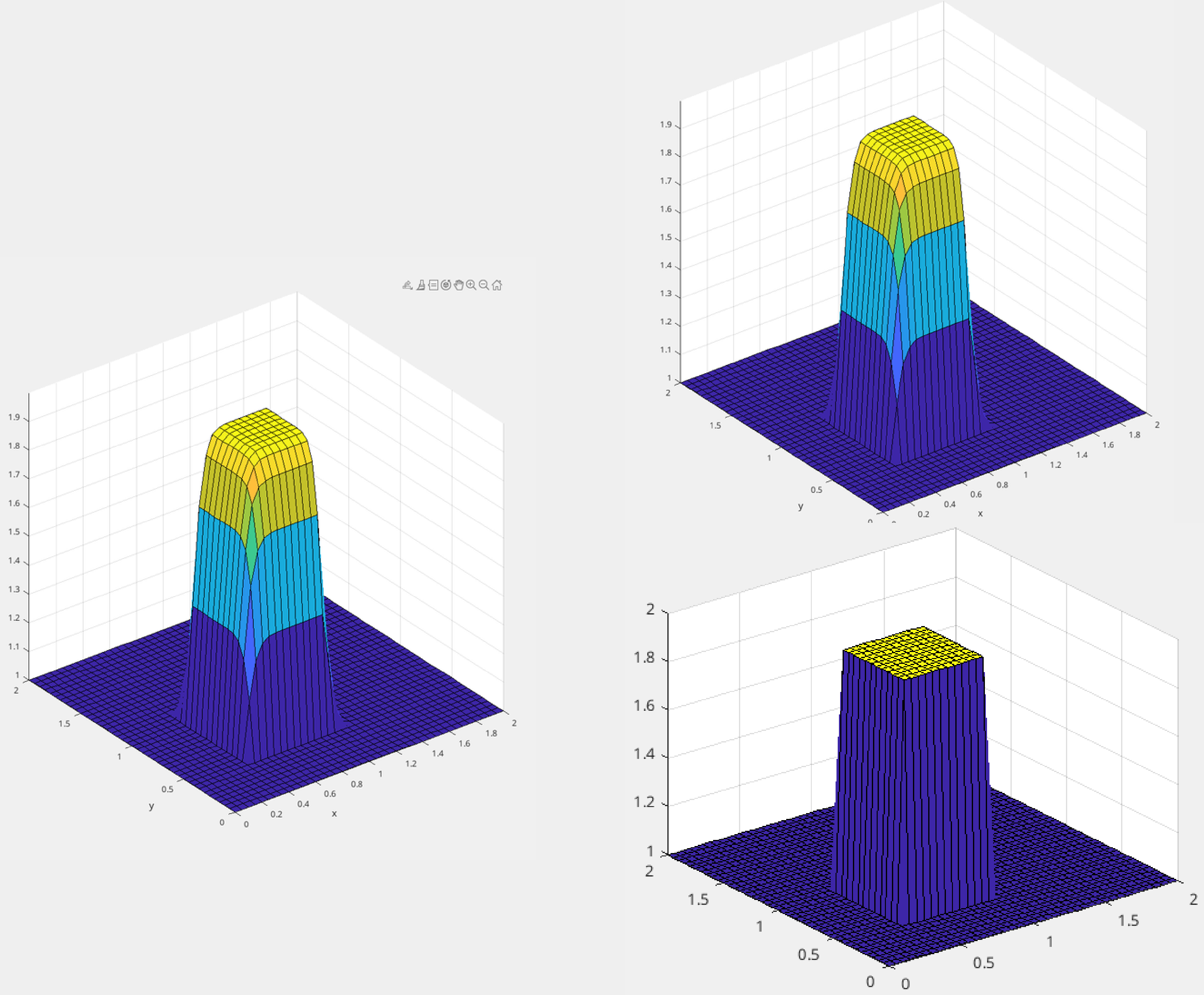
The velocity field and temperature field in the grid are evolved in time using a finite difference method. The time step dt is chosen such that the dimensionless number sigma = dt*nu*(1/dx^2+1/dy^2) is equal to 0.0009, where nu is the kinematic viscosity of the fluid. At each time step, the velocity field and temperature field at each point (i,j) in the grid are updated according to the following finite difference equations:

un(i,j) = u(i,j) - dt/dxu(i,j)(u(i,j)-u(i-1,j)) - dt/dyv(i,j)(u(i,j)-u(i,j-1)) + nudt/dx^2(u(i+1,j)-2u(i,j)+u(i-1,j)) + nudt/dy^2*(u(i,j+1)-2*u(i,j)+u(i,j-1))

vn(i,j) = v(i,j) - dt/dxu(i,j)(v(i,j)-v(i-1,j)) - dt/dyv(i,j)(v(i,j)-v(i,j-1)) + nudt/dx^2(v(i+1,j)-2v(i,j)+v(i-1,j)) + nudt/dy^2*(v(i,j+1)-2*v(i,j)+v(i,j-1))

```
% Set the grid size and time step size
nx = 41;
ny = 41;
nt = 120;
c = 1;
dx = 2 / (nx - 1);
dy = 2 / (ny - 1);
sigma = 0.0009;
nu = 0.01;
dt = sigma * dx * dy / nu;
% Set the grid coordinates
x = linspace(0, 2, nx);
y = linspace(0, 2, ny);
% Initialize the solution arrays
u = ones(ny, nx); % Create a 1xn vector of 1's
v = ones(ny, nx);
un = ones(ny, nx);
vn = ones(ny, nx);
comb = ones(ny, nx);
% Set the initial conditions
% Set hat function I.C. : u(.5<=x<=1 && .5<=y<=1 ) is 2
u(floor(.5 / dy):floor(1 / dy + 1), floor(.5 / dx):floor(1 / dx + 1)) = 2;
% Set hat function I.C. : v(.5<=x<=1 && .5<=y<=1 ) is 2
v(floor(.5 / dy):floor(1 / dy + 1), floor(.5 / dx):floor(1 / dx + 1)) = 2;
% Set up a 3D plot
figure;
surf(x, y, u)
% Use vectorized operations to update u and v at each time step
for n = 0:nt
    un = u;
    vn = v;
    u(2:end, 2:end) = (un(2:end, 2:end) - (un(2:end, 2:end) .* c .* dt ./ dx .* (un(2:end, 2:end) - un(2:end, 1:end-1))) - ...
        vn(2:end, 2:end) .* c .* dt ./ dy .* (un(2:end, 2:end) - un(1:end-1, 2:end)));
    v(2:end, 2:end) = (vn(2:end, 2:end) - (un(2:end, 2:end) .* c .* dt ./ dx .* (vn(2:end, 2:end) - vn(2:end, 1:end-1))) - ...
        vn(2:end, 2:end) .* c .* dt ./ dy .* (vn(2:end, 2:end) - vn(1:end-1, 2:end)));

    % Set the boundary conditions
    u(1, :) = 1;
    u(end, :) = 1;
    u(:, 1) = 1;
    u(:, end) = 1;

    v(1, :) = 1;
    v(end, :) = 1;
    v(:, 1) = 1;
    v(:, end) = 1;
end
% Create the plot
figure;
surfc(x, y, u);
xlabel('x');
ylabel('y');
% Create the plot
figure;
surfc(x, y, v);
xlabel('x');
ylabel('y');
```

Code



Plot

Step 9

The code begins by setting the grid size (nx and ny) and the spacing between grid points (dx and dy). It then generates the coordinates of the grid points (x and y). Next, it initializes the solution p to be a 2D array of all zeros, except for the boundary conditions, which are set as follows:

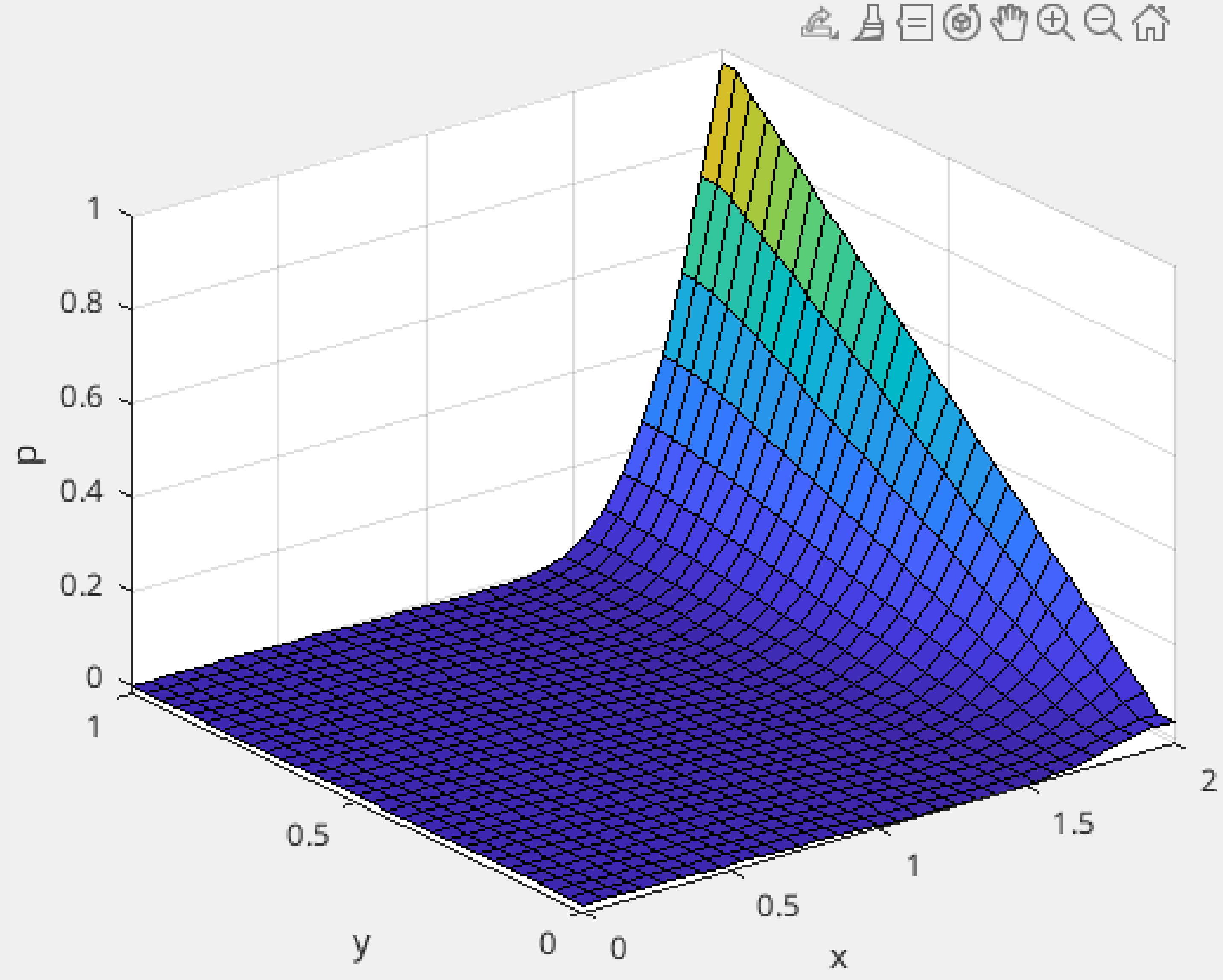
- p = 0 at x = 0
- p = y at x = 2
- dp/dy = 0 at y = 0
- dp/dy = 0 at y = 1

The code then enters a loop that iteratively updates the solution using the finite difference approximation of the Laplace equation. At each iteration, the previous solution (pn) is stored, and the current solution (p) is updated using the following equation:

```
p(2:end-1, 2:end-1) = ((dy^2 * (pn(1:end-2, 2:end-1) + pn(3:end, 2:end-1)) + dx^2 * (pn(2:end-1, 1:end-2) + pn(2:end-1, 3:end))) / (2 * (dx^2 + dy^2)) - ... (y(2:end-1).^2 .* dx^2 .* dy^2) / (2 * (dx^2 + dy^2))) / (1 + dx^2 * dy^2 / (2 * (dx^2 + dy^2)));
```

This equation uses finite differences to approximate the second partial derivatives of p with respect to x and y. The equation is applied to all interior points of the grid (i.e., points with indices 2 to end-1 in both dimensions). The boundary conditions are then applied to the corresponding points of the grid.

```
% Set grid size and boundary conditions
nx = 31;
ny = 31;
dx = 2 / (nx - 1);
dy = 2 / (ny - 1);
x = linspace(0,2,nx);
y = linspace(0,1,ny);


p = zeros(ny, nx);  % Initialize grid with all zeros
p(:, 1) = 0;  % p = 0 at x = 0
p(:, end) = y;  % p = y at x = 2
p(1, :) = p(2, :);  % dp/dy = 0 at y = 0
p(end, :) = p(end-1, :);  % dp/dy = 0 at y = 1

% Set initial error and convergence criterion
l2norm = 1;
l2_target = 1e-4;

% Iterate until error is below the convergence criterion
while l2norm > l2_target
    pn = p;  % Store previous iteration
    % Update values using the Laplace equation
    p(2:end-1, 2:end-1) = ((dy^2 * (pn(1:end-2, 2:end-1) + pn(3:end, 2:end-1)) + dx^2 * (pn(2:end-1, 1:end-2) + pn(2:end-1, 3:end))) / (2 * (dx^2 + dy^2)) - ...
                          (y(2:end-1).^2 .* dx^2 .* dy^2) / (2 * (dx^2 + dy^2))) / (1 + dx^2 * dy^2 / (2 * (dx^2 + dy^2)));
    % Apply boundary conditions
    p(:, 1) = 0;  % p = 0 at x = 0
    p(:, end) = y;  % p = y at x = 2
    p(1, :) = p(2, :);  % dp/dy = 0 at y = 0
    p(end, :) = p(end-1, :);  % dp/dy = 0 at y = 1
    % Compute error
    l2norm = sum((p(:) - pn(:)).^2) / sum(pn(:).^2);
end

% Plot the solution
surf(x, y, p)
xlabel('x')
ylabel('y')
zlabel('p')
```



Code

Plot

Step 10

The code begins by setting the grid size (nx and ny), the number of time steps (nt), and the grid spacing (dx and dy). It also sets the minimum and maximum values of the x- and y-coordinates (xmin, xmax, ymin, ymax). It then initializes the solution p and the temporary solution pd to be 2D arrays of all zeros, as well as the source term b to be a 2D array of all zeros. The source term represents the forcing function in the Poisson equation, which drives the evolution of the solution. The code sets the values of b at two points in the grid to be +100 and -100, respectively, to create a source term with two opposing charges.

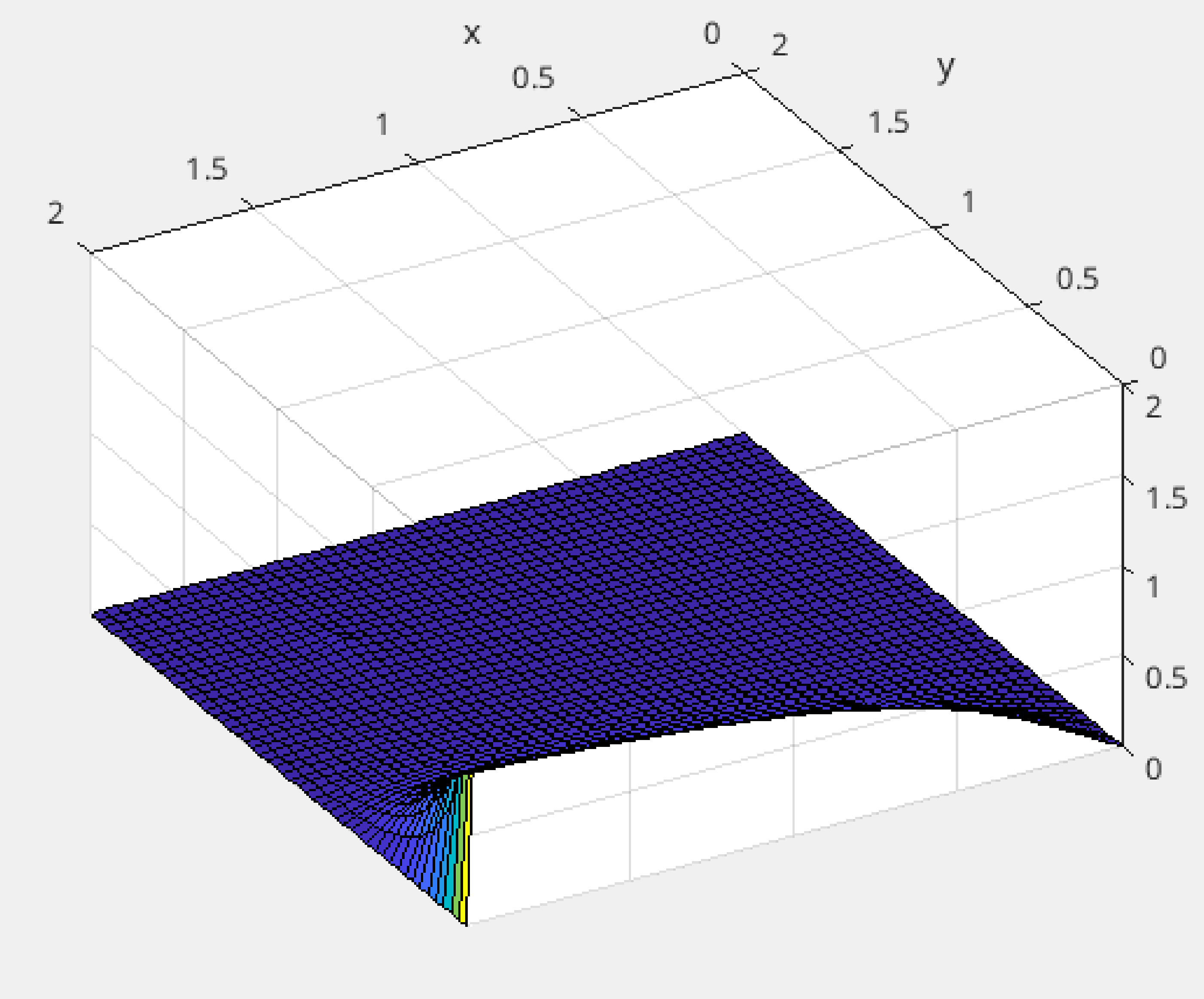
The code then enters a loop that iteratively updates the solution using the finite difference approximation of the Poisson equation. At each iteration, the previous solution (pd) is stored, and the current solution (p) is updated using the following equation:

```
p(2:end-1, 2:end-1) = (((pd(2:end-1, 3:end) + pd(2:end-1, 1:end-2)) * dy^2 + ... (pd(3:end, 2:end-1) + pd(1:end-2, 2:end-1)) * dx^2 - ... b(2:end-1, 2:end-1) * dx^2 * dy^2) / ... (2 * (dx^2 + dy^2)));
```

This equation uses finite differences to approximate the second partial derivatives of p with respect to x and y. The equation is applied to all interior points of the grid (i.e., points with indices 2 to end-1 in both dimensions).

```matlab
% Parameters
nx = 50;
ny = 50;
nt = 100;
xmin = 0;
xmax = 2;
ymin = 0;
ymax = 1;

dx = (xmax - xmin) / (nx - 1);
dy = (ymax - ymin) / (ny - 1);

% Initialization
p = zeros(ny, nx);
pd = zeros(ny, nx);
b = zeros(ny, nx);
x = linspace(xmin, xmax, nx);
y = linspace(xmin, xmax, ny);

% Source
b(round(ny / 4), round(nx / 4)) = 100;
b(round(3 * ny / 4), round(3 * nx / 4)) = -100;

for it = 1:nt
    pd = p;

    p(2:end-1, 2:end-1) = (((pd(2:end-1, 3:end) + pd(2:end-1, 1:end-2)) * dy^2 + ...
        (pd(3:end, 2:end-1) + pd(1:end-2, 2:end-1)) * dx^2 - ...
        b(2:end-1, 2:end-1) * dx^2 * dy^2) / ...
        (2 * (dx^2 + dy^2)));

    % Modify boundary conditions
    p(1, :) = y;  % p = y at y = 0
    p(end, :) = 0;  % p = 0 at y = 1
    p(:, 1) = 0;  % p = 0 at x = 0
    p(:, end) = 0;  % p = 0 at x = 2
end

plot2D(x, y, p)

function plot2D(x, y, p)
figure;
[X, Y] = meshgrid(x, y);
surf(X, Y, p);
xlabel('x');
ylabel('y');
view(30, 225);
end
```

Code



Plot

THANK YOU