Simulation of a Cooling System

NOTE: I confirm that I did not use codes from the web or from past years' assignments and that the work I submit is my own and my own only.

1 Introduction

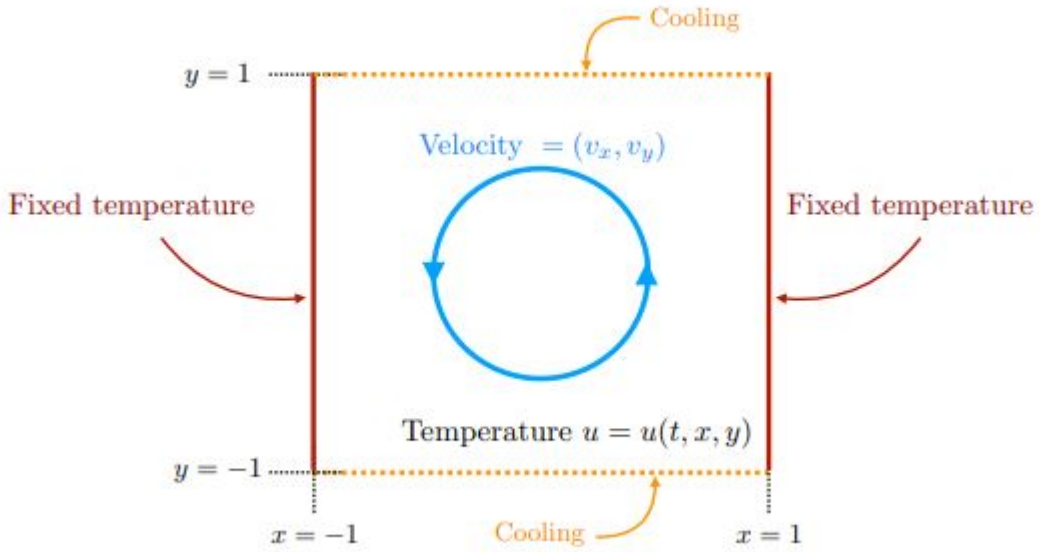


Figure 1: Schematic of a cooling system.
The goal of this final project is to simulate the cooling system depicted in figure 1.

2 Algorithm

In order to do this, we first need to consider a fluid with velocity (v_x, v_y) convecting the temperature $u = u(t, x, y)$ in the domain $[-1, 1] \times [-1, 1]$. The temperature variable satisfies the advection-diffusion equation in two different spatial dimensions:

$$\frac{\partial u}{\partial t} + v_x \frac{\partial u}{\partial x} + v_y \frac{\partial u}{\partial y} = D \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

where D is the diffusion coefficient. The fluid velocity is defined by a non-laminar, turbulent circular motion and is defined as the following:

$$v_x = -\cos^2\left(\frac{\pi x}{2}\right) \sin(\pi y) \quad v_y = \cos^2\left(\frac{\pi y}{2}\right) \sin(\pi x).$$

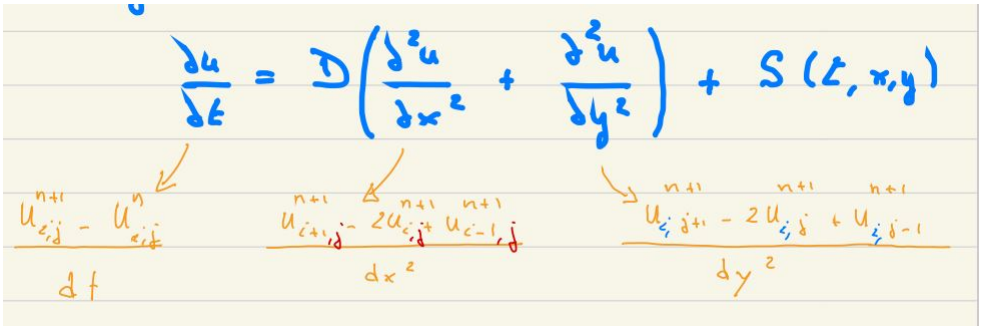
Before doing this, we need to declare some boundary conditions. We will impose the following conditions for the variable of temperature:

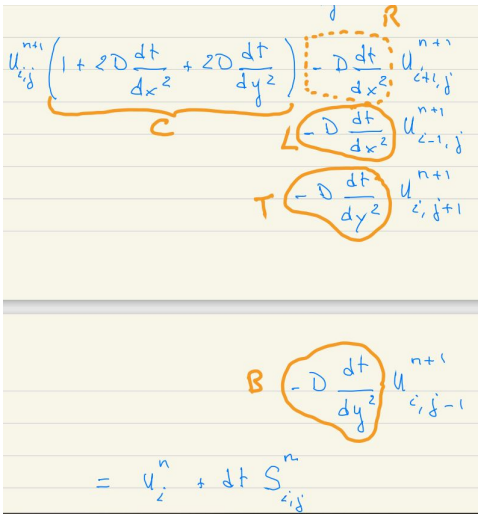
$$\begin{aligned} u &= u_a(t,y) && \text{at } x=-1 \\ u &= u_b(t,y) && \text{at } x=1 \\ D\frac{\partial u}{\partial y} &= f_c(t,x) && \text{at } y=-1 \\ -D\frac{\partial u}{\partial y} &= f_d(t,x) && \text{at } y=1 \end{aligned}$$

where the functions u_a, u_b, f_c and f_d are given.

Physically speaking, the boundary conditions mean physically that these are the edges of the walls in which the fluid will circulate at the velocity "un" (refer to code for referenced variable).

In terms of the discretizations of the partial derivatives in equation (1), we use the following to make our approximations:





The discretizations for imposing the boundary conditions given in equation (2) are as follows:

```
% Boundary points:

% Left Wall (i.e i=1)
  for j = 1:J
     i = 1;
     P = (j-1)*I + i;
     A(P,P) = 1;
     rhs(P) = BC(E^n, x(i), y(j));
  end
```

```
% Right Wall (i.e i=I)
  for j = 1:J
     i = I;
     P = (j-1)*I + i;
     A(P,P) = 1;
     rhs(P) = BC(E^n, x(i), y(j));
  end

% Bottom Wall (i.e j=1)
  for i = 1:I
     j = 1;
     P = (j-1)*I + i;
     A(P,P) = 1;
     rhs(P) = BC(E^n, x(i), y(j));
  end
```

```
% Top Wall (i.e j=J)
  for i = 1:I
     j = J;
     P = (j-1)*I + i;
     A(P,P) = 1;
     rhs(P) = BC(E^n, x(i), y(j));
  end
```

From here, we need to implement in Matlab your numerical method and compare to the exact solution $u_e = e^{-t}(\sin(x) + \sin(y))$. For that purpose, we momentarily modify the PDE to include a source term $S = S(t, x, y)$, and set u_a and u_b to be the exact solution.

$$\frac{\partial u}{\partial t} + v_x \frac{\partial u}{\partial x} + v_y \frac{\partial u}{\partial y} = D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + S,$$

The formulas for S, f_c and f_d corresponding to the exact solution are as follows:

$$S(t^n, x, y) = e^{-t}((D-1)sin(x) + (D-1)sin(y) + V_x cos(x) + V_y cos(y))$$

$$f_c = De^{-t}cos(y)$$

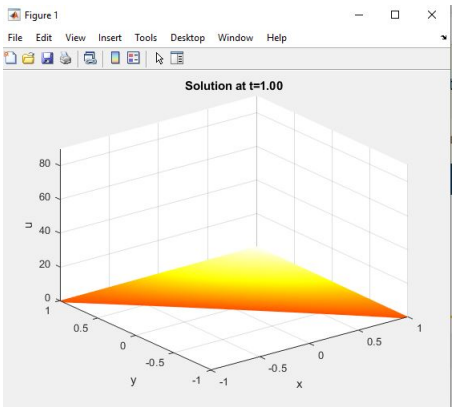
$$f_d = -De^{-t}cos(y)$$

3 Results

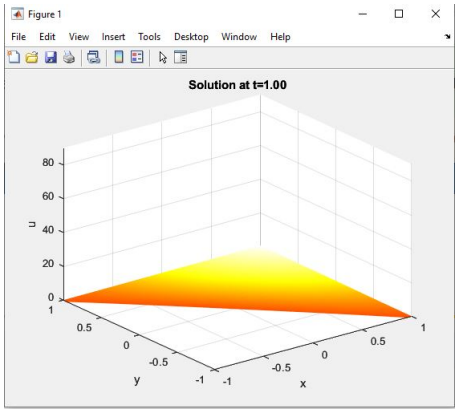
From there, we take an initial solution given by the exact solution in the interval $[-1,1]\times[-1,1]$, a final time of $t = 1$, a time step of $\Delta t = 0.5\Delta x$, a diffusion coefficient of $D = 0.0005$, and output the maximum error at the final time. This is done so that we can check that the error is roughly divided by 2 every time you double the number of grid points in each spatial dimension. Specifically, we will be considering grids with $20, 40, 80$, and 160 grid points in each spatial dimension and record the maximum error as well.

For each scenario from n=20, to n=160 he ratio of errors is close to two when comparing the one before with the one after.

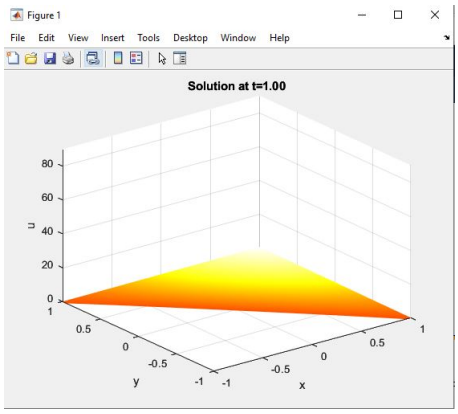
At n = 20, The maximum error is 0.02711334



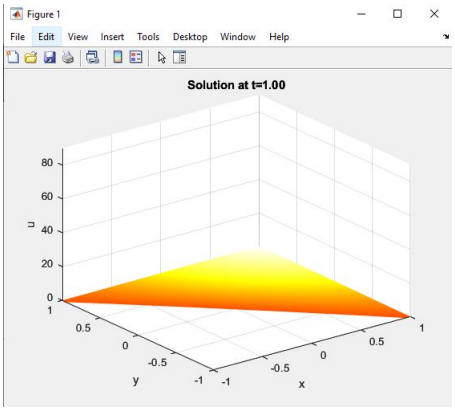
At n = 40, The maximum error is 0.01380589



At n = 80, The maximum error is 0.00698003

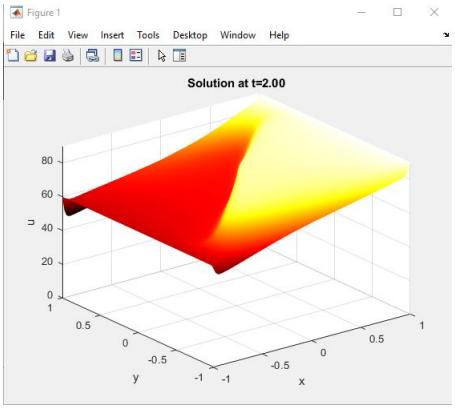


At n = 160, The maximum error is 0.00353093

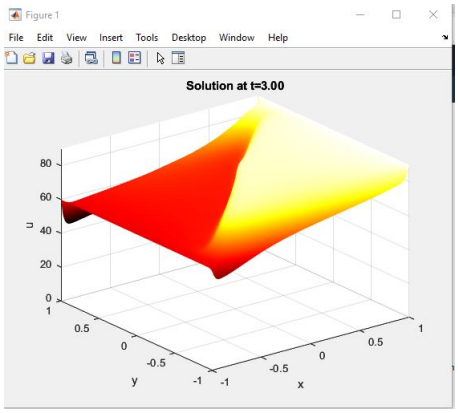


Next, I will consider the domain to be $[-1,1] \times [-1,1]$ and take $S = 0, D = 0.0005, f_c = 0.1, f_d = 0.2, u_a = 60, u_b = 90, \Delta t = .5\Delta x$ and a grid with 100 grid points in each spatial direction. Next, I will take the initial condition to be $u(t = 0) = 15x + 75$. Lastly, we will run the simulation to a final time of $t = 10$ and plot the solution at $t = 2, t = 3, t = 4, t = 5$, $t = 6$, and $t = 10$.

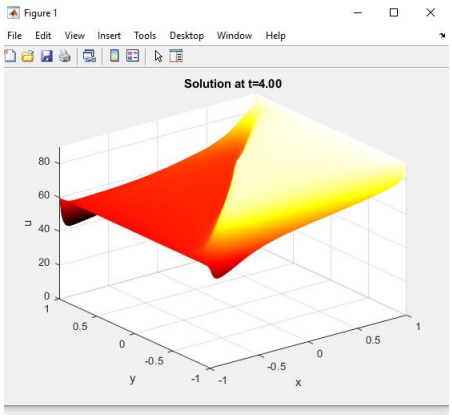
At t = 2,



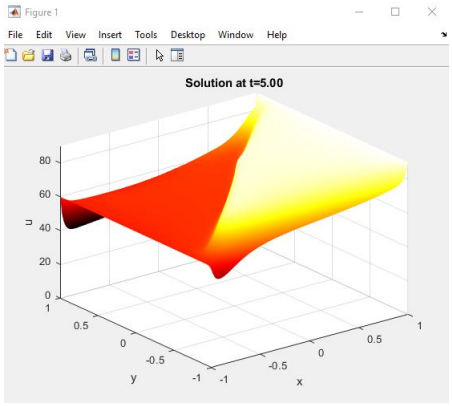
At t = 3,



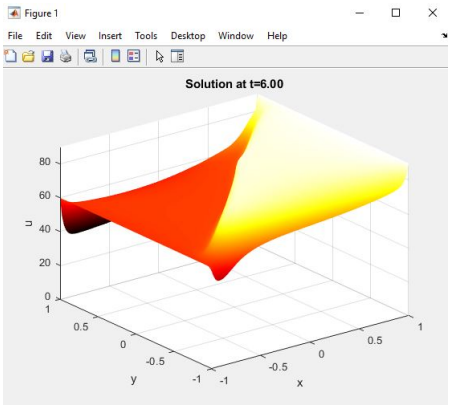
At t = 4,



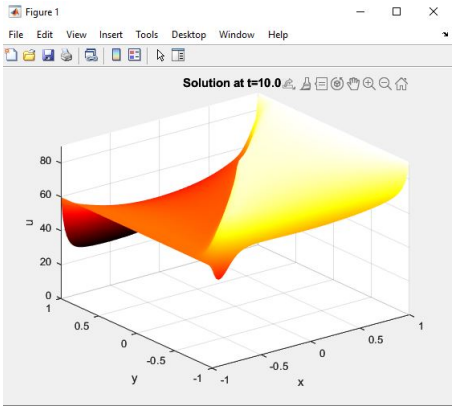
At t = 5,



At t = 6,



At t = 10,



4 Conclusion

Throughout our experiments, we have been able to conclude that the ratio of errors is close to two and that a non-steady-state can be concluded as well. This is because there is a fluctuation in velocity with respect to certain positions and not a single plane throughout the surf plot in the second experiment.

5 MATLAB Code for Number 4

```
    clear variables;
clc;

%Section 1: Setup Variables

%For time,
tFinal = 1;
```

```
t = 0;

%For Boundary Conditions
a = -1;
b = 1;
c = -1;
d = 1;

%Number of points
D = 0.0005;
nX = 160;
nY = 160;

%For Velocity Equations
vx = @(t,x,y)  cos((pi*x)/2)*cos((pi*x)/2)*sin(pi*y);
vy = @(t,x,y) -cos((pi*y)/2)*(-cos((pi*y)/2))*sin(pi*x);

%For exact differential equation, S, fc and fd
exactU = @(t, x, y)   exp(-t)*(sin(x) + sin(y));
S      = @(t, x, y) -exp(-t)*(sin(x)+sin(y)) ...
    + (cos((pi*x)/2)*cos((pi*x)/2)*sin(pi*y))*exp(-t)*cos(x) + ...
    (-cos((pi*y)/2)*(-cos((pi*y)/2))*sin(pi*x))*...
     exp(-t)*cos(y) + D*exp(-t)*sin(x) + D*exp(-t)*sin(y);
fc     = @(t, x, y)   D*exp(-t)*cos(y);
fd     = @(t, x, y)  -D*exp(-t)*cos(y);

%declaring x and y variables
x = linspace(a, b, nX);
dx = x(2)-x(1);

y = linspace(c, d, nX);
 dy = y(2)-y(1);


%Other declarations for velocity, etc.
un     = zeros(nX,     nY);
uExact = zeros(nX,     nY);
unp1   = zeros(nX   ,    nY);
rhs    = zeros(nX*nY,     1);
u      = zeros(nX*nY,     1);
A      = sparse(nX*nY, nX*nY);
dt     = 0.5*min(dx,dy);
```

```
%Section 2: Loops and Plots

%for looping statement involving exact differential and un
for i = 1:nX
    for j = 1:nY
        un(i,j)     = exactU(0, x(i), y(j));
        uExact(i,j) = exactU(0, x(i), y(j));
    end
end

%Plot for graphing results
cla;
surf(x, y, un');
s = sprintf('Solution at t=%2.2f', t);
title(s); xlabel('x'); ylabel('y'); zlabel('u');
axis([a b c d 0 90]); colormap('hot'); shading interp;
pause(.1)

%Declarations for derivatives and C,L,B,R,T
dtOverDx = dt/dx/dx;
dtOverDy = dt/dy/dy;

C = (1 + 2*D*dtOverDx + 2*D*dtOverDy);
L = -D*dtOverDx;
R = L;
B = -D*dtOverDy;
T = B;

%main while loop to run the program
while t < tFinal
    if t+dt > tFinal
        dt = tFinal - t;
        dtOverDx = dt/dx/dx;
        dtOverDy = dt/dy/dy;
        C = (1 + 2*D*dtOverDx + 2*D*dtOverDy);
        L = -D*dtOverDx;
        R = L;
        B = -D*dtOverDy;
        T = B;
    end
```

```
% Boundary conditions:
% On the Left wall:
for j=1:nY
    i = 1;
    p = (j-1)*nX + i;
    A(p,p) = 1; rhs(p) = exactU(t+dt, a, y(j));
end

% On the Right wall:
for j=1:nY
    i = nX;
    p = (j-1)*nX + i;
    A(p,p) = 1; rhs(p) =exactU(t+dt, x(i), y(j));
end
% On the Bottom wall:
for i=2:nX-1
    j = 1;
    p = (j-1)*nX + i;
        A(p,p) = C;
        A(p,p-1 ) = L;
        A(p,p+1 ) = R;
        A(p,p+nX) = B+T;
        if vx(t, x(i), y(j)) > 0
            dutimesdx = (un(i,j)-un(i-1,j))/dx;
        else
            dutimesdx = (un(i+1,j)-un(i,j))/dx;
        end
        rhs(i,j) = un(i,j) + dt*S(t, x(i), y(j))- vx(t, x(i), y(j))*dutimesdx + 2*B*
        rhs(p) = rhs(i,j);
end
% On the Top wall:
for i=2:nX-1
    j = nY;
    p = (j-1)*nX + i;
        A(p,p) = C;
        A(p,p-1 ) = L;
        A(p,p+1 ) = R;
        A(p,p-nX) = B+T;
        if vx(t, x(i), y(j)) > 0
            dutimesdx = (un(i,j)-un(i-1,j))/dx;
        else
            dutimesdx = (un(i+1,j)-un(i,j))/dx;
```

```
        end
        rhs(i,j) = un(i,j) + dt*S(t, x(i), y(j))- vx(t, x(i), y(j))*dutimesdx + 2*T*
        rhs(p) = rhs(i,j);
end

% Interior points:
for i=2:nX-1
    for j=2:nY-1
        % Index p of the row associated with the grid point
        % (i,j):
        p = (j-1)*nX + i;
        A(p,p) = C;
        A(p,p-1 ) = L;
        A(p,p+1 ) = R;
        A(p,p-nX) = B;
        A(p,p+nX) = T;

         if vx(t, x(i), y(j)) > 0
             dutimesdx = (un(i,j)-un(i-1,j))/dx;
         else
             dutimesdx = (un(i+1,j)-un(i,j))/dx;
         end

         if vy(t, x(i), y(j)) > 0
             dudy = (un(i,j)-un(i,j-1))/dy;
         else
             dudy = (un(i,j+1)-un(i,j))/dy;
         end
         rhs(i,j) = un(i,j)-vx(t, x(i), y(j))*dt*dutimesdx-vy(t, x(i), y(j))*dt*dudy
         rhs(p) = rhs(i,j);
    end
end



% Solve for the linear system:
u = A \ rhs;

for i=1:nX
    for j=1:nY
        p = (j-1)*nX + i;
        unp1(i,j) = u(p);
```

```
        end
    end


%Creates updates for time variables with respect to time step
t = t+dt;
un = unp1;

% Graphs the results
cla;
surf(x, y, un');
s = sprintf('Solution at t=%2.2f', t);
title(s); xlabel('x'); ylabel('y'); zlabel('u');
axis([a b c d 0 90]); colormap('hot'); shading interp;
pause(.1)

    for i = 1:nX
        for j = 1:nY
            uExact(i,j) = exactU(t, x(i), y(j));
        end
    end
end

maxErrorrate = max(max(abs(un - uExact)));
fprintf('The maximum error is %2.8f \n', maxErrorrate);
```

6 MATLAB Code for Number 5

```
    clear variables;
clc;

%Section 1: Setup Variables

%For time,
tFinal = 10;
t = 0;

%For Boundary Conditions
a = -1;
```

```
b = 1;
c = -1;
d = 1;

%Number of points
D = 0.0005;
nX = 100;
nY = 100;

%For Velocity Equations
Vx = @(t,x,y)  cos((pi*x)/2)*cos((pi*x)/2)*sin(pi*y);
Vy = @(t,x,y) -cos((pi*y)/2)*(-cos((pi*y)/2))*sin(pi*x);

exactU = @(t, x, y)   exp(-t)*(sin(x) + sin(y));
S      = @(t, x, y) 0
fc     = @(t, x, y)   0.1%D*exp(-t)*cos(y);
fd     = @(t, x, y)   0.2%-D*exp(-t)*cos(y);

%declaring x and y variables
x = linspace(a, b, nX);
dx = x(2)-x(1);

y = linspace(c, d, nX);
dy = y(2)-y(1);


%Other declarations for velocity, etc.
un     = zeros(nX,     nY);
uExact = zeros(nX,     nY);
unp1   = zeros(nX   ,    nY);
rhs    = zeros(nX*nY,     1);
u      = zeros(nX*nY,     1);
A      = sparse(nX*nY, nX*nY);
dt     = 0.5*dx;

%Section 2: Loops and Plots

%for looping statement involving exact differential and un
for i=1:nX
   for j=1:nY
un(i,j) = 15*x(i) + 75;
   end
```

```
end

%Plot for graphing results
cla;
surf(x, y, un');
s = sprintf('Solution at t=%2.2f', t);
title(s); xlabel('x'); ylabel('y'); zlabel('u');
axis([a b c d 0 90]); colormap('hot'); shading interp;
pause(.1)

%Declarations for derivatives and C,L,B,R,T
dtOverDx = dt/dx/dx;
dtOverDy = dt/dy/dy;

C = (1 + 2*D*dtOverDx + 2*D*dtOverDy);
L = -D*dtOverDx;
R = L;
B = -D*dtOverDy;
T = B;

while t < tFinal
    if t+dt > tFinal
        dt = tFinal - t;
        dtOverDx = dt/dx/dx;
        dtOverDy = dt/dy/dy;
        C = (1 + 2*D*dtOverDx + 2*D*dtOverDy);
        L = -D*dtOverDx;
        R = L;
        B = -D*dtOverDy;
        T = B;
    end

    % Build the linear system A*unp1 = rhs
    % Boundary conditions:
    % Left wall:
    for j=1:nY
        i = 1;
        p = (j-1)*nX + i;
        A(p,p) = 1; rhs(p) = 60 ;%exactU(t+dt, a, y(j));
    end
    % Right wall:
    for j=1:nY
```

```
    i = nX;
    p = (j-1)*nX + i;
    A(p,p) = 1; rhs(p) = 90;%exactU(t+dt, x(i), y(j));
end
% Bottom wall:
for i=2:nX-1
    j = 1;
    p = (j-1)*nX + i;
        A(p,p   ) = C;
        A(p,p-1 ) = L;
        A(p,p+1 ) = R;
        A(p,p+nX) = B+T;
        if vx(t, x(i), y(j)) > 0
            dutimesdx = (un(i,j)-un(i-1,j))/dx;
        else
            dutimesdx = (un(i+1,j)-un(i,j))/dx;
        end
        rhs(i,j) = un(i,j) + dt*S(t, x(i), y(j))- vx(t, x(i), y(j))*dutimesdx + 2*B*
        rhs(p) = rhs(i,j);
end
% Top wall:
for i=2:nX-1
    j = nY;
    p = (j-1)*nX + i;
        A(p,p   ) = C;
        A(p,p-1 ) = L;
        A(p,p+1 ) = R;
        A(p,p-nX) = B+T;
        if vx(t, x(i), y(j)) > 0
            dutimesdx = (un(i,j)-un(i-1,j))/dx;
        else
            dutimesdx = (un(i+1,j)-un(i,j))/dx;
        end
        rhs(i,j) = un(i,j) + dt*S(t, x(i), y(j))- vx(t, x(i), y(j))*dutimesdx + 2*T*
        rhs(p) = rhs(i,j);
end

% Interior points:
for i=2:nX-1
    for j=2:nY-1
        % Index p of the row associated with the grid point
        % (i,j):
```

```
            p = (j-1)*nX + i;
            A(p,p   ) = C;
            A(p,p-1 ) = L;
            A(p,p+1 ) = R;
            A(p,p-nX) = B;
            A(p,p+nX) = T;

             if vx(t, x(i), y(j)) > 0
                 dutimesdx = (un(i,j)-un(i-1,j))/dx;
             else
                 dutimesdx = (un(i+1,j)-un(i,j))/dx;
             end

             if vy(t, x(i), y(j)) > 0
                 dutimesdy = (un(i,j)-un(i,j-1))/dy;
             else
                 dutimesdy = (un(i,j+1)-un(i,j))/dy;
             end
             rhs(i,j) = un(i,j)-vx(t, x(i), y(j))*dt*dutimesdx-vy(t, x(i), y(j))*dt*duti
             rhs(p) = rhs(i,j);
        end
    end

    % Solve the linear system:
    u = A \ rhs;

    for i=1:nX
        for j=1:nY
            p = (j-1)*nX + i;
            unp1(i,j) = u(p);
        end
    end

%Creates updates for time variables with respect to time step
t = t+dt;
un = unp1

% Graphs the results
cla;
surf(x, y, un');
s = sprintf('Solution at t=%2.2f', t);
title(s); xlabel('x'); ylabel('y'); zlabel('u');
```

```
axis([a b c d 0 90]); colormap('hot'); shading interp;
pause(.1)

end

maxErrorrate = max(max(abs(un - uExact)));
fprintf('The maximum error is %2.8f \n', maxErrorrate);
```