Implementation of a Numerical Solution to a Two-Dimensional Advection-Diffusion Problem

1 Introduction

Advection and diffusion are two fundamental phenomena in fluid mechanics - advection describes the transportation of a quantity by fluid motion, and diffusion describes the redistribution of a quantity across an initial concentration gradient. This report presents a numerical method that simulates two-dimensional advection-diffusion as governed by the equation

$$\frac{\delta u}{\delta t} + v_x \frac{\delta u}{\delta x} + v_y \frac{\delta u}{\delta y} = D(\frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2}) \tag{1}$$

Where u is the quantity of interest, and x and y are the spatial dimensions of interest. v_x and v_y are the components of the velocity field for the fluid in the given dimensions. Finally, D is the coefficient of diffusion, a property of the system governing the rate of diffusion.

The numerical method is developed in the first second section by discretizing the equation (1) and applying appropriate boundary conditions. It is initially implemented with a predetermined exact solution of the form $u_e = e^{(-1)t}(sin(x) + sin(y))$ over an interval $[-1, 1]x[-1, 1]$ and with the diffusive constant $D = 0.0005$. Finally, the velocity field of the fluid is set to be the following.

$$\begin{aligned} v_x &= (-1)cos^2(\frac{\pi x}{2})sin(\pi y) \\ v_y &= (-1)cos^2(\frac{\pi y}{2})sin(\pi x) \end{aligned} \tag{2}$$

Secondly, the method is applied to a specific cooling system (u reflects temperature) with 100 nodes in each spatial dimension and with a time-step equal to $0.5\Delta x$, where $\$Deltax$ is the distance between two nodes. This specific system is defined with initial condition $u = 15x + 75$, and the boundary conditions are set to either reflect a fixed temperature or convective cooling. Lastly, the results are analyzed to examine the accuracy and efficacy of the derived numerical method.

2 Methods

The development of a numerical method for the advection-diffusion equation consists of three main stages. Firstly, the domain is discretized and the differential terms are expressed at each node as first-order Taylor expansions with respect to the adjacent grid points. This method defines the quantity of u at the subsequent time step for all the interior points; however, it breaks down on the boundaries since the adjacent points fall out of the domain. To address

this concern, 4 boundary conditions are imposed. Two of the boundary conditions in this setting are dirichlet boundary conditions (at $x = -1$ and $x = 1$). A Dirichlet boundary condition is simply an exact solution that is imposed on the points that lie on the domain's boundary. In the case of the cooling system, this is the analog for a fixed temperature value. The other boundary conditions are Robin boundary conditions (at $y = -1$ and $y = 1$). This is an expression that relates the diffusive or advective flux in the domain to a boundary-specific function. For the cooling system, a Robin boundary condition represents convective cooling of the fluid at the edge of the domain. The relationship between the flux and the boundary function provides an additional equation to resolve the value of u at the end of the domain.

2.1 Internal Points

The first step in deriving the numerical method is defining the discrete expressions for the differential terms in (1)

$$\frac{\delta u}{\delta t} = \frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} \tag{3}$$

$$v_x \frac{\delta u}{\delta x} = v_x \frac{u_{i+1,j}^{n} - u_{i,j}^{n}}{\Delta x} \tag{4}$$

$$v_y \frac{\delta u}{\delta y} = v_y \frac{u_{i,j}^{n} - u_{i,j-1}^{n}}{\Delta y} \tag{5}$$

$$D\frac{\delta^2 u}{\delta x^2} = D\frac{\frac{u_{i+1}^{n+1} - u_{i}^{n+1}}{\Delta x} - \frac{u_{i}^{n+1} - u_{i-1}^{n+1}}{\Delta x}}{\Delta x} \tag{6}$$

$$= D\frac{u_{i+1}^{n+1} - 2u_{i}^{n+1} + u_{i-1}^{n+1}}{\Delta x^2} \tag{7}$$

$$D\frac{\delta^2 u}{\delta y^2} = D\frac{\frac{u_{i,j+1}^{n+1} - u_{i,j}^{n+1}}{\Delta x} - \frac{u_{i,j}^{n+1} - u_{i,j-1}^{n+1}}{\Delta x}}{\Delta x} \tag{8}$$

$$= D\frac{u_{i,j+1}^{n+1} - 2u_{i,j}^{n+1} + u_{i,j-1}^{n+1}}{\Delta y^2} \tag{9}$$

where n is the temporal index, i is the spatial index in the direction x, and j is the spatial index in the direction y. Here, a key point must be noted regarding the discretization of the advective term. This discrete term is a first-order upwind scheme, and is therefore dependent on the velocity of the fluid. If the velocity is greater than zero, the difference term along the x-direction should be expressed as $v_x \frac{u_{i,j}^{n} - u_{i-1,j}^{n}}{\Delta x}$, since the fluid passes node i after node i-1. In contrast, a negative velocity would require the term $v_x \frac{u_{i+1,j}^{n} - u_{i,j}^{n}}{\Delta x}$ since, by similar reasoning, the node i is passed after the node i+1. A similar phenomenon must be accounted for along the other spatial directions. The equation presented above uses the velocity signs that are present in the first quadrant, as given by equation (2). This derivation follows the expression for this quadrant of the domain, and the results for all quadrants are presented at the end.

However, it is imperative to appreciate the fact that this derivation is performed separately for each quadrant and implemented individually through conditional statements, as seen in the appendix with the MATLAB code. Finally, a similar phenomenon will be observed during the implementation of the Robin boundary condition since the differential equation is discretized once again.

At this stage, the differential equation can be assembled from the discrete terms. An additional source term S is included in the equation - this is a matter of convenience, as the inclusion of the source terms allows the manipulation of the equation to fit the exact solution that will be implemented in the first set of simulations.

$$\begin{aligned} &\frac{u_{i,j}^{n+1}-u_{i,j}^{n}}{\Delta t}+v_x\frac{u_{i+1,j}^{n}-u_{i,j}^{n}}{\Delta x}+v_y\frac{u_{i,j}^{n}-u_{i,j-1}^{n}}{\Delta y} \\ &=D(\frac{u_{i+1}^{n+1}-2u_{i}^{n+1}+u_{i-1}^{n+1}}{\Delta x^2}+\frac{u_{i,j+1}^{n+1}-2u_{i,j}^{n+1}+u_{i,j-1}^{n+1}}{\Delta y^2})+S_{i,j}^{n} \end{aligned} \tag{10}$$

At this time, it is worth noting that the numerical method consists of several first-order accurate schemes, so it is, itself, a first-order accurate scheme in time and space. With algebraic manipulation of the expressions above (9), the following expression is introduced for a given node:

$$\begin{aligned} (1+2D\frac{\Delta t}{\Delta x^2}+2D\frac{\Delta t}{\Delta y^2})u_{i,j}^{n+1}-D\frac{\Delta t}{\Delta x^2}u_{i+1,j}^{n+1}-D\frac{\Delta t}{\Delta x^2}u_{i-1,j}^{n+1} &- D\frac{\Delta t}{\Delta y^2}u_{i,j+1}^{n+1}-D\frac{\Delta t}{\Delta x^2}u_{i,j-1}^{n+1} \\ &= u_{i,j}^{n}-v_x\frac{\Delta t}{\Delta x}(u_{i+1,j}^{n}-u_{i,j}^{n}) \\ &- v_y\frac{\Delta t}{\Delta y}(u_{i,j}^{n}-u_{i,j-1}^{n})+\Delta t S_{i,j}^{n} \\ Cu_{i,j}^{n+1}+Ru_{i+1,j}^{n+1}+Lu_{i-1,j}^{n+1}+Tu_{i,j+1}^{n+1}+Bu_{i,j-1}^{n+1} &= u_i^n-v\frac{\Delta t}{\Delta x}(u_{i+1}^{n}-u^{n})+\Delta t S_i^n \\ C &= 1+2D\frac{\Delta t}{\Delta x^2}+2D\frac{\Delta t}{\Delta y^2} \\ R &= -D\frac{\Delta t}{\Delta x^2} \\ L &= -D\frac{\Delta t}{\Delta x^2} \\ T &= -D\frac{\Delta t}{\Delta y^2} \\ T &= -D\frac{\Delta t}{\Delta y^2} \end{aligned} \tag{11}$$

Finally, each term in the resulting system of equations has the terms L, R, B, T and C and we can express the equation for the interior points in terms of an n by n matrix (where

n is the number of interior points) composed of 0, L, R, B, T and C. Because the coefficients all modify n+1 terms, resolving the equation formed by the matrix and the right hand side gives the value of u at the subsequent time step. At this stage, it is necessary to address the convention by which the equations are assembled into this matrix. Arbitrarily, the matrix is formed by moving along each row before progressing to the next column of nodes. This can be expressed as $p = (j-1)n_x + i$ and provides a mapping between the index of a node and its position in the matrix (n_x and n_y denote the number of nodes in each dimension). For each row, corresponding to a node equation, term C is placed at the column that has the same index p as the row. Terms L and R are predictably placed to the left and right of this column. Finally, terms T and B must correspond to the index of the nodes directly above or below the gridpoint of interest - this is accomplished by adding or subtracting n_x to or from the p-index.

With the matrix assembled, the vector on the right-hand of the equation consists of $u^n_{i,j} - v_x\frac{\Delta t}{\Delta x}(u^n_{i+1,j} - u^n_{i,j}) - v_y\frac{\Delta t}{\Delta y}(u^n_{i,j} - u^n_{i,j-1}) + \Delta t S^n_{i,j}$. Referring back to the discussion of the difference between the quadrants, it can be seen that all upwind terms are gathered at the right hand side, so this is what distinguishes the matrix equation for each region of the domain. For each quadrant, the right hand side can be denoted as follows:

$$
\begin{aligned}
Quadrant1 &: u^n_{i,j} - v_x\frac{\Delta t}{\Delta x}(u^n_{i+1,j} - u^n_{i,j}) - v_y\frac{\Delta t}{\Delta y}(u^n_{i,j} - u^n_{i,j-1}) + \Delta t S^n_{i,j} \\
Quadrant2 &: u^n_{i,j} - v_x\frac{\Delta t}{\Delta x}(u^n_{i+1,j} - u^n_{i,j}) - v_y\frac{\Delta t}{\Delta y}(u^n_{i,j+1} - u^n_{i,j}) + \Delta t S^n_{i,j} \\
Quadrant3 &: u^n_{i,j} - v_x\frac{\Delta t}{\Delta x}(u^n_{i,j} - u^n_{i-1,j}) - v_y\frac{\Delta t}{\Delta y}(u^n_{i,j+1} - u^n_{i,j}) + \Delta t S^n_{i,j} \\
Quadrant4 &: u^n_{i,j} - v_x\frac{\Delta t}{\Delta x}(u^n_{i,j} - u^n_{i-1,j}) - v_y\frac{\Delta t}{\Delta y}(u^n_{i,j} - u^n_{i,j-1}) + \Delta t S^n_{i,j}
\end{aligned} \tag{12}
$$

Lastly, the matrix equation can be solved in MATLAB or using a comparable computational platform to progress the numerical method through a time-step. For the simulation with the exact solution, the source term is found via partial differentiation:

$$S(t,x,y) = e^{(-1)t}(-(sin(x) + cos(y)) + v_x cos(x) + v_y(cos(y)) + D(sin(x) + sin(y))) \tag{13}$$

For the custom cooling system, the source term is set to 0 since the fluid does not generate heat. It must be noted, however, that points $i = 1$, $i = n_x$, $j = 1$, $j = n_y$ are not accounted for since the adjacent points for these nodes have unknown values of u. The types of boundary conditions imposed in this problem were discussed above - the next sections provide a more detailed mathematical analysis of the boundary conditions.

2.2 Dirichlet Boundary Condition

The Dirichlet boundary condition is relatively straightforward - an exact solution denoted $exact(t^{n+1}, y)$ is imposed on the left and right boundary conditions ($x = 1$ and $x = -1$) . To

reiterate, this boundary condition is implying that a fixed concentration of u is supplied at the fluid inlet. For the second simulation, this is simply a fixed temperature at the boundary. In the matrix denoting the system of linear equations, a value of 1 would be located in the first column (left boundary) or the last column (left boundary) for the rows corresponding to boundary layers. This configuration simply means that the value of u at the next time step is dependent on the exact term. Finally, the terms corresponding to the nodes are set equal to $exact(t^{n+1}, y)$ in the right hand vector.

In the implementation of the numerical method with an exact solution, the boundary condition is the same as the pre-determined exact solution. In the second implementation, the values of temperature are set to $u_a = 60$ and $u_b = 90$.

2.3 Robin Boundary Condition

As briefly described above, the Robin boundary condition is a statement that relates flux terms to a boundary function. This additional relationship can be used to replace the temperature values of adjacent cells in equation (11) which fall out of the domain for the boundary nodes. To do so, the boundary condition equation must be discretized in space as well. In this case, this boundary condition is applied at the upper and lower edges of the domain and represents convective cooling. Because the heat equation has a diffusive form, the boundary function only impacts the diffusive flux and can be expressed as the following expressions for the two boundaries. More specifically, due to the position of the boundary conditions only the diffusive flux in the y-direction is impacted by convective cooling. This is a reasonable assumption if the left and right walls are defined to have a fixed temperature.

$$\begin{aligned}
D\frac{\delta u}{\delta y} &= f_c(t^{n+1}, x); y = -1 \\
D\frac{u^{n+1}_{i,j+1} - u^{n+1}_{i,j-1}}{2\Delta y} &= f(t^{n+1}, x) \\
u^{n+1}_{i,j-1} &= u^{n+1}_{i,j+1} - 2\frac{\Delta y}{D} f(t^{n+1}, a) \\
-D\frac{\delta u}{\delta y} &= f_d(t^{n+1}, x); y = 1 \\
D\frac{u^{n+1}_{i,j+1} - u^{n+1}_{i,j-1}}{2\Delta y} &= f(t^{n+1}, x) \\
u^{n+1}_{i,j+1} &= u^{n+1}_{i,j-1} - 2\frac{\Delta y}{D} f(t^{n+1}, a) \qquad (14)
\end{aligned}$$

More specifically, due to the position of the boundary conditions only the diffusive flux in the y-direction is impacted by convective cooling. This is a reasonable assumption if the left and right walls are assumed to have a fixed temperature.

Firstly, it can be observed that the boundary condition is used to resolve the unknown values of the top node at the upper boundary, and the bottom node at the lower boundary. Secondly,

it is noted that the right-hand-side expressions of equations (12) include the same undefined terms; for these equations, the substitutions will be made with the n+1 time-step replaced with the n time-step. With these considerations, the resolved equations can be assembled for the nodes lying at the indicated edges. It must be pointed out, however, that the dependence of the upwind scheme for advection on fluid velocity necessitates the use of two equations for each boundary. The right hand side for each equation aligns with the corresponding quadrant from the equations (12).

These equations have the same form as (11) and therefore populate the matrix in a similar fashion. The only difference is the absence of a top or bottom coefficient, depending on the boundary, which is replaced with the Robin expression.

For the exact solution $u_e = e^{(-1)t}(sin(x) + sin(y))$, the value of f must be found by taking partial derivatives. This results in the following functions:

$$\begin{aligned} D\frac{\delta u}{\delta y} &= De^{(-1)t}(cos(y)) \\ -D\frac{\delta u}{\delta y} &= -De^{(-1)t}(cos(y)) \end{aligned} \tag{15}$$

For the second solution, the boundary function depends on the heat transfer coefficient of the system and is given as $f_c = 0.1$ and $f_d = 0.2$. With all four boundary conditions imposed, the matrix equation introduced earlier is fully defined and can be resolved. This completes the description of the numerical method for two-dimensional advection-diffusion.

3 Results and Conclusions

As highlighted several times in the previous sections, the numerical method is implemented through two simulations. The first set of results correspond to the simulation that is performed with an exact solution $u_e = e^{(-1)t}(sin(x) + sin(y))$. These figures aim to examine the accuracy of the method and to test if it is first-order accurate with respect to space. The second set of results predict the response of a cooling system that has a circular fluid flow and features a pair of fixed-temperature boundaries and two convective-cooled boundaries.

3.1 Exact Solution

The error associated with a first-order accurate scheme is divided by two if the number of nodes is multiplied by two across all directions. To test this postulate, the first simulation is performed with 20, 40, 80, 160 nodes in all directions and the maximum error is selected for each run. The results are tabulated below. Each simulation is terminated after 1 second and follows the specifications provided in the previous sections.

The results strongly support the accuracy of the simulation - the maximum errors are small, and divide by half every time the number of nodes is doubled. The figure below provides a visual

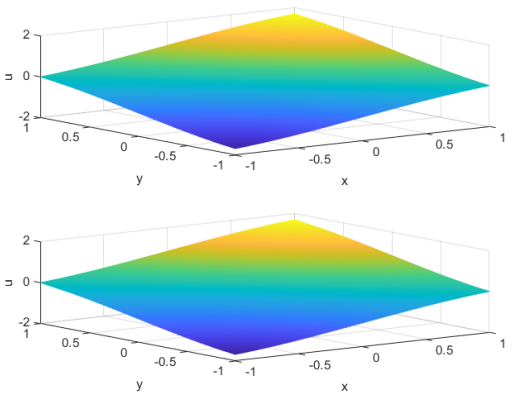
Number of Nodes Per Dimension	Maximum Error	Ratio of Errors
20	0.02800402	
40	0.01404786	0.501637265
80	0.00703762	0.500974526
160	0.00352987	0.501571554

Table 1: Table of Maximum Errors for Different Numbers of Grid points.

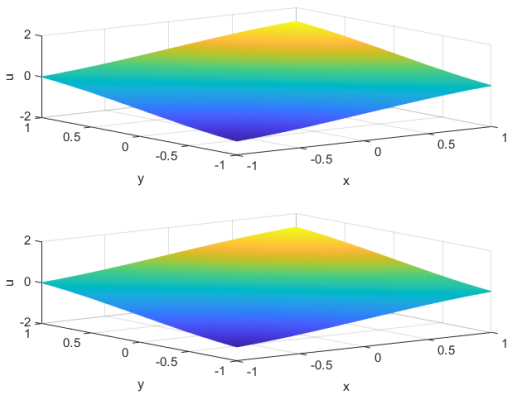
overview of the projected solution and the exact solution at selected time-step to demonstrate the accuracy of the numerical method.

3.2 Custom Solution

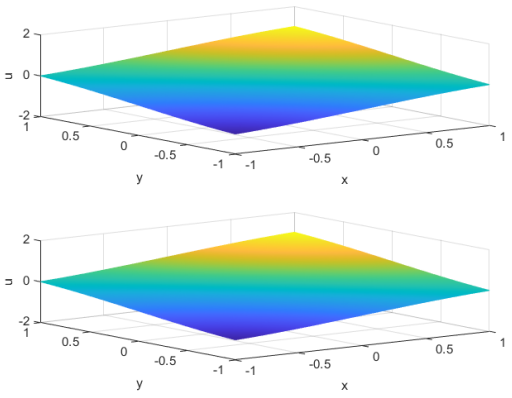
Finally, the cooling system that has been laid out in some detail is simulated with the numerical method. The images below visualize the progression of the system over time. As anticipated, the left wall acts as a heat sink and the right wall acts as a heat source. The initial condition is a linear statement, but evolves into a spiral pattern as dictated by the velocity of the fluid. The convective cooling is apparent at the top and bottom boundaries, with the temperature falling more rapidly at the top boundary because of the greater heat transfer coefficient. Finally, the system converges towards a steady-state rather quickly, taking on an increasingly uniform temperature field that persists until the end of the 10 second simulation and tends towards the temperature of the heat source. These results highlight the practical value of this numerical model for advection-diffusion and, combined with the accuracy demonstrated earlier, confirm the robustness of the proposed treatment of the system.



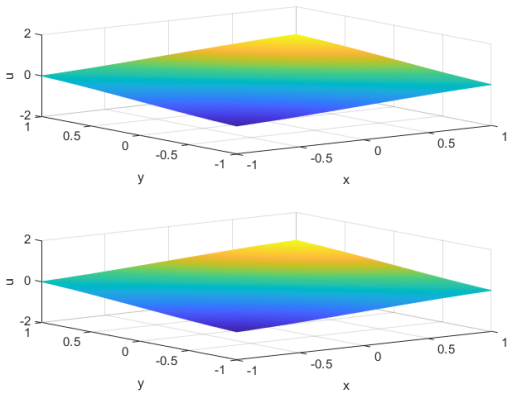
(a) $t = 0.01s$



(b) $t = 0.25s$

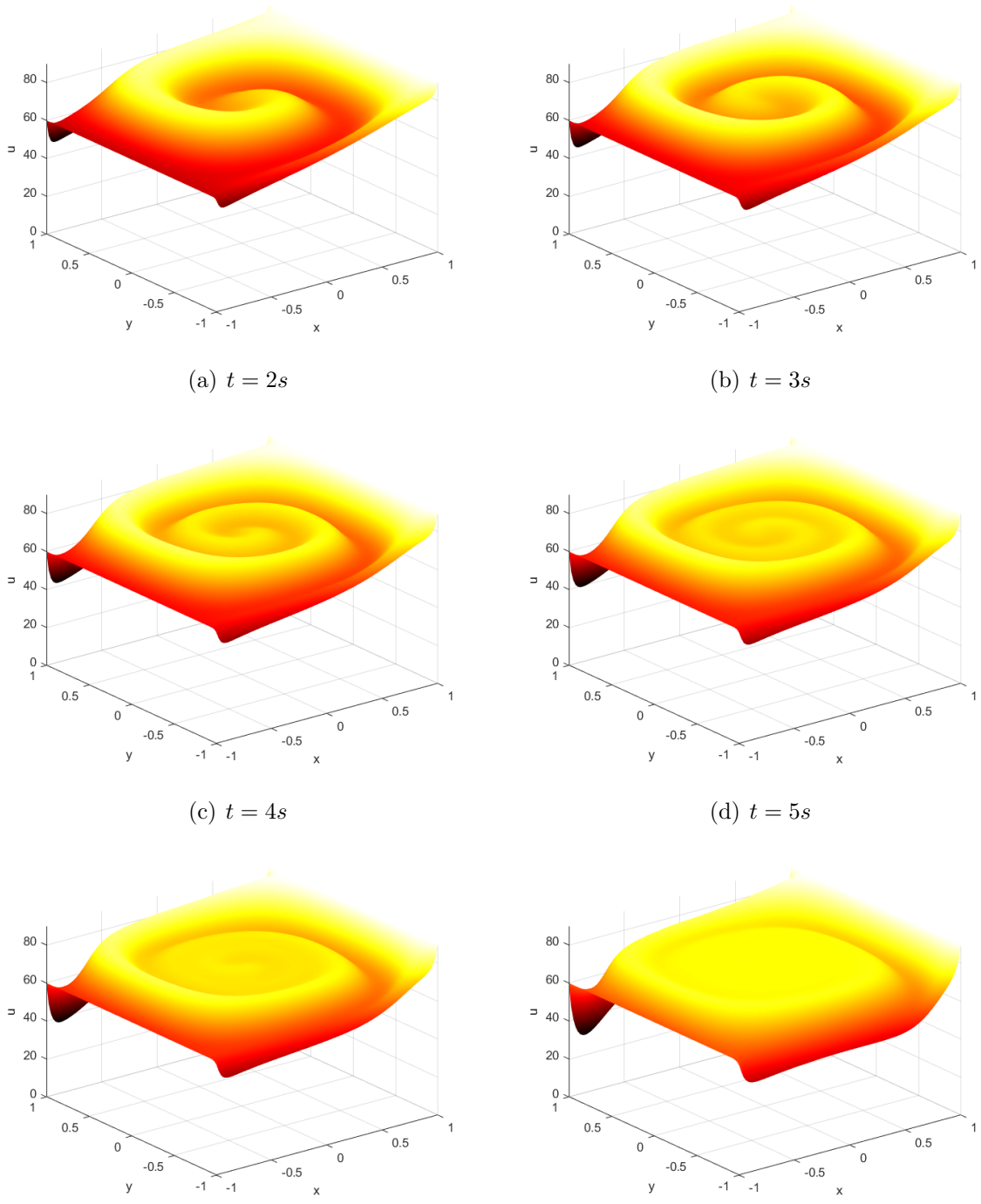


(c) $t = 0.50s$



(d) $t = 1.00s$

Figure 1: Plots of the Numerical Solution (Up) and Exact Solution (Down) to the Advection-Diffusion Equation with $u_e = e^{-t}cos(x)$ over 1 second.



(a) $t = 2s$ (b) $t = 3s$

(c) $t = 4s$ (d) $t = 5s$

(e) $t = 6s$ (f) $t = 10s$

Figure 2: Plots of the Numerical Solution to the Advection-Diffusion Equation for Custom Cooling System.

A MATLAB Implementation for Exact Solution

```
% Simulate the temperatures of a cooling system via a 2-dimensional
% advection diffusion equation.
%\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"

% Solving the advection-diffusion equation:
% du/dt + Vx du/dx + Vy du/dy = D d^2u/dx^2 + D d^2u/dy^2 in [a, b; c, d]
% uInitial = e^-t(sin(x) + sin(y))
% D is the diffusion coefficient that is given
% We use Dirichlet and robin boundary conditions:
% Dirichlet BC: exact solution imposed.
% u = u_exact(t,x,y) at x = -1, x = 1.
% Robin BC: Diffusive flux along y depends on cooling at the boundary.
% D du/dy = f_c at y = -1; -D du/dy = f_d at y = 1.

clear variables; clc;

% Define the simulation parameters.
tFinal = 1; t = 0;  % Time information
a = -1; b = 1; c = -1; d = 1;   % Domain
D = 0.0005; % Diffusion coefficient
nX = 80; nY = 80;   % Number of nodes
vX = @(x, y) -cos(pi.*x./2).^2 .* sin(pi.*y); % Velocity
vY = @(x, y)  cos(pi.*y./2).^2 .* sin(pi.*x);

% Define the controlling functions for the system.
exactU = @(t, x, y) exp(-t).*(sin(x)+sin(y));
S      = @(t, x, y) exp(-t).*(-(sin(x) + sin(y))+(vX(x,y) ...
    .*(cos(x))) + (vY(x,y).*(cos(y))) + D.*(sin(x)+sin(y)));
BC1     = @(t, x, y) exp(-t).*(sin(x)+sin(y));
BC2     = @(t, x, y) exp(-t).*(sin(x)+sin(y));
fC     = @(t, x, y) D.*exp(-t).*(cos(y));
fD     = @(t, x, y) -D.*exp(-t).*(cos(y));

%%%

% Discretize time and 2 spatial dimensions.
x = linspace(a, b, nX); dx = x(2)-x(1);
y = linspace(c, d, nY); dy = y(2)-y(1);
dt = 0.5.*dx;
```

```
% Allocate memory storage.
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);

% Determine initial values from exact solution
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:
subplot(2,1,1);
mesh(x, y, un');
s = sprintf('Solution at t=%2.2f', t);
title(s); xlabel('x'); ylabel('y'); zlabel('u');
axis([a b c d -2 2]);

subplot(2,1,2);
mesh(x, y, uExact');
s = sprintf('Exact at t=%2.2f', t);
title(s); xlabel('x'); ylabel('y'); zlabel('u');
axis([a b c d -2 2]);
pause(0.1);

dtOverDx2 = dt/dx/dx;
dtOverDy2 = dt/dy/dy;
C = (1 + 2*D*dtOverDx2 + 2*D*dtOverDy2);
L = -D*dtOverDx2; R = L;
B = -D*dtOverDy2; T = B;

while t < tFinal
    if t+dt > tFinal
        dt = tFinal - t;
        dtOverDx2 = dt/dx/dx;
        dtOverDy2 = dt/dy/dy;
```

```
    C = (1 + 2*D*dtOverDx2 + 2*D*dtOverDy2);
    L = -D*dtOverDx2; R = L;
    B = -D*dtOverDy2; 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) = BC1(t+dt, x(i), y(j));
end

% Right wall:
for j= 1:nY
    i = nX;
    p = (j-1)*nX + i;
    A(p,p) = 1; rhs(p) = BC2(t+dt, x(i), y(j));
end

% Top wall, Right half:
for i= nX/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;
    rhs(p) = un(i,j) - vX(x(i),y(j)).*dt./dx.*(un(i+1,j) - un(i,j))...
        - vY(x(i),y(j)).*dt./dy.*(un(i,j) - un(i,j-1))...
        + dt.*S(t,x(i),y(j))...
        + 2.*dy.*T./D.*(fD(t+dt,x(i),y(j)));
end

% Top wall, Left half:
for i=2:(nX/2)-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;
    rhs(p) = un(i,j) - vX(x(i),y(j)).*dt./dx.*(un(i+1,j) - un(i,j))...
        - vY(x(i),y(j)).*dt./dy.*(un(i,j-1)...
        - 2.*dy./D.*fD(t,x(i),y(j))...
        - un(i,j)) +  dt.*S(t,x(i),y(j))+ ...
        2.*dy.*T./D.*(fD(t+dt,x(i),y(j)));
end

% Bottom wall, left half:
for i= 2:(nX/2)-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;
    rhs(p) = un(i,j) - vX(x(i),y(j)).*dt./dx.*(un(i,j) - un(i-1,j))...
        - vY(x(i),y(j)).*dt./dy.*(un(i,j+1) - un(i,j))...
        + dt.*S(t,x(i),y(j))...
        + 2.*dy.*B./D.*(fC(t+dt,x(i),y(j)));
end

% Bottom wall, Right half:
for i=(nX/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;
    rhs(p) = un(i,j) - vX(x(i),y(j)).*dt./dx.*(un(i,j) - un(i-1,j))...
        - vY(x(i),y(j)).*dt./dy.*(un(i,j) - un(i,j+1) ...
        + 2.*dy./D.*fC(t,x(i),y(j))) ...
        +  dt.*S(t,x(i),y(j))+ ...
        2.*dy.*B./D.*(fC(t+dt,x(i),y(j)));
end

% Interior points: divided into four domains by the upwind scheme.
% Quadrant 1
for i= nX/2:nX-1
    for j= nY/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;
        rhs(p) = un(i,j) - vX(x(i),y(j)).*dt./dx.*...
            (un(i+1,j) - un(i,j))- vY(x(i),y(j)).*dt./dy.*...
            (un(i,j) - un(i,j-1)) + dt.*S(t,x(i),y(j));
    end
end

% Quadrant 2
for i= 2:(nX/2)-1
    for j= nY/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;
        rhs(p) = un(i,j) - vX(x(i),y(j)).*dt./dx.*...
            (un(i+1,j) - un(i,j)) - vY(x(i),y(j)).*dt./dy.*...
            (un(i,j+1) - un(i,j)) + dt.*S(t,x(i),y(j));
    end
end

% Quadrant 3
for i= 2:(nX/2)-1
    for j= 2:(nY/2)-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;
        rhs(p) = un(i,j) - vX(x(i),y(j)).*dt./dx.*...
```

```
            (un(i,j) - un(i-1,j))- vY(x(i),y(j)).*dt./dy.*(un(i,j+1)...
            - un(i,j)) + dt.*S(t,x(i),y(j));
    end
end

% Quadrant 4
for i= (nX/2):nX-1
    for j= 2:(nY/2)-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;
        rhs(p) = un(i,j) - vX(x(i),y(j)).*dt./dx.*...
            (un(i,j) - un(i-1,j))- vY(x(i),y(j)).*dt./dy.*(un(i,j) ...
            - un(i,j-1)) + dt.*S(t,x(i),y(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

% Update variables for the next time step:
t = t+dt;
un = unp1;

% Display the results:
subplot(2,1,1); cla;
mesh(x, y, un');
s = sprintf('Solution at t=%2.2f', t);
title(s); xlabel('x'); ylabel('y'); zlabel('u');
axis([a b c d -2 2]);
```

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

    %plot:
    subplot(2,1,2); cla;
    mesh(x, y, uExact');
    s = sprintf('Exact at t=%2.2f', t);
    title(s); xlabel('x'); ylabel('y'); zlabel('u');
    axis([a b c d -2 2]);
    pause();
end

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

B MATLAB Implementation for Cooling System

```
% Simulate the temperatures of a cooling system via a 2-dimensional
% advection diffusion equation.
%\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"

% Solving the advection-diffusion equation:
% du/dt + Vx du/dx + Vy du/dy = D d^2u/dx^2 + D d^2u/dy^2 in [a, b; c, d]
% uInitial = e^-t(sin(x) + sin(y))
% D is the diffusion coefficient that is given
% We use Dirichlet and robin boundary conditions:
% Dirichlet BC: exact solution imposed.
% u = u_exact(t,x,y) at x = -1, x = 1.
% Robin BC: Diffusive flux along y depends on cooling at the boundary.
% D du/dy = f_c at y = -1; -D du/dy = f_d at y = 1.

clear variables; clc;

% Define the simulation parameters.
tFinal = 10; t = 0;  % Time information
a = -1; b = 1; c = -1; d = 1;   % Domain
```

```
D = 0.0005; % Diffusion coefficient
nX = 100; nY = 100;   % Number of nodes
vX = @(x, y) -cos(pi.*x./2).^2 .* sin(pi.*y); % Velocity
vY = @(x, y)  cos(pi.*y./2).^2 .* sin(pi.*x);

% Define the controlling functions for the system.
exactU = @(t, x, y) 15.*x + 75;
S      = @(t, x, y) 0;
BC1     = @(t, x, y) 60;
BC2     = @(t, x, y) 90;
fC     = @(t, x, y) 0.1;
fD     = @(t, x, y) 0.2;

%%%

% Discretize time and 2 spatial dimensions.
x = linspace(a, b, nX); dx = x(2)-x(1);
y = linspace(c, d, nY); dy = y(2)-y(1);
dt = 0.5.*dx;

% Allocate memory storage.
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);

% Determine initial values from exact solution
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:
cla;
surf(x, y, un');
s = sprintf('Solution for Convective Cooling System t=%2.2f', t);
title(s); xlabel('x'); ylabel('y'); zlabel('u');
axis([-1 1 -1 1 0 90]); colormap('hot'); shading interp;
```

```
pause(0.1);

dtOverDx2 = dt/dx/dx;
dtOverDy2 = dt/dy/dy;
C = (1 + 2*D*dtOverDx2 + 2*D*dtOverDy2);
L = -D*dtOverDx2; R = L;
B = -D*dtOverDy2; T = B;

while t < tFinal
    if t+dt > tFinal
        dt = tFinal - t;
        dtOverDx2 = dt/dx/dx;
        dtOverDy2 = dt/dy/dy;
        C = (1 + 2*D*dtOverDx2 + 2*D*dtOverDy2);
        L = -D*dtOverDx2; R = L;
        B = -D*dtOverDy2; 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) = BC1(t+dt, x(i), y(j));
    end

    % Right wall:
    for j= 1:nY
        i = nX;
        p = (j-1)*nX + i;
        A(p,p) = 1; rhs(p) = BC2(t+dt, x(i), y(j));
    end

    % Top wall, Right half:
    for i= nX/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;
```

```
    rhs(p) = un(i,j) - vX(x(i),y(j)).*dt./dx.*(un(i+1,j) - un(i,j))...
        - vY(x(i),y(j)).*dt./dy.*(un(i,j) - un(i,j-1))...
        + dt.*S(t,x(i),y(j))...
        + 2.*dy.*T./D.*(fD(t+dt,x(i),y(j)));
end

% Top wall, Left half:
for i=2:(nX/2)-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;
    rhs(p) = un(i,j) - vX(x(i),y(j)).*dt./dx.*(un(i+1,j) - un(i,j))...
        - vY(x(i),y(j)).*dt./dy.*(un(i,j-1)...
        - 2.*dy./D.*fD(t,x(i),y(j))...
        - un(i,j)) +  dt.*S(t,x(i),y(j))+ ...
        2.*dy.*T./D.*(fD(t+dt,x(i),y(j)));
end

% Bottom wall, left half:
for i= 2:(nX/2)-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;
    rhs(p) = un(i,j) - vX(x(i),y(j)).*dt./dx.*(un(i,j) - un(i-1,j))...
        - vY(x(i),y(j)).*dt./dy.*(un(i,j+1) - un(i,j))...
        + dt.*S(t,x(i),y(j))...
        + 2.*dy.*B./D.*(fC(t+dt,x(i),y(j)));
end

% Bottom wall, Right half:
for i=(nX/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;
    rhs(p) = un(i,j) - vX(x(i),y(j)).*dt./dx.*(un(i,j) - un(i-1,j))...
        - vY(x(i),y(j)).*dt./dy.*(un(i,j) - un(i,j+1) ...
        + 2.*dy./D.*fC(t,x(i),y(j))) ...
        +  dt.*S(t,x(i),y(j))+ ...
        2.*dy.*B./D.*(fC(t+dt,x(i),y(j)));
end

% Interior points: divided into four domains by the upwind scheme.
% Quadrant 1
for i= nX/2:nX-1
    for j= nY/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;
        rhs(p) = un(i,j) - vX(x(i),y(j)).*dt./dx.*(un(i+1,j)...
            - un(i,j))...
            - vY(x(i),y(j)).*dt./dy.*(un(i,j) - un(i,j-1))...
            + dt.*S(t,x(i),y(j));
    end
end

% Quadrant 2
for i= 2:(nX/2)-1
    for j= nY/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;
        rhs(p) = un(i,j) - vX(x(i),y(j)).*dt./dx.*(un(i+1,j)...
            - un(i,j))...
            - vY(x(i),y(j)).*dt./dy.*(un(i,j+1) - un(i,j))...
            + dt.*S(t,x(i),y(j));
```

```
    end
end

% Quadrant 3
for i= 2:(nX/2)-1
    for j= 2:(nY/2)-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;
        rhs(p) = un(i,j) - vX(x(i),y(j)).*dt./dx.*...
            (un(i,j) - un(i-1,j))...
            - vY(x(i),y(j)).*dt./dy.*(un(i,j+1) - un(i,j))...
            + dt.*S(t,x(i),y(j));
    end
end

% Quadrant 4
for i= (nX/2):nX-1
    for j= 2:(nY/2)-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;
        rhs(p) = un(i,j) - vX(x(i),y(j)).*dt./dx.*(un(i,j)...
            - un(i-1,j))...
            - vY(x(i),y(j)).*dt./dy.*(un(i,j) - un(i,j-1))...
            + dt.*S(t,x(i),y(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

    % Update variables for the next time step:
    t = t+dt;
    un = unp1;

    % Display the results:
    cla;
    surf(x, y, un');
    s = sprintf('Solution for Convective Cooling System at t=%2.2f', t);
    title(s); xlabel('x'); ylabel('y'); zlabel('u');
    axis([-1 1 -1 1 0 90]); colormap('hot'); shading interp;
    pause(0.1);
end
```