Solving the Diffusion Equation in 2D

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

The goal of this assignment is to solve the diffusion equation for the concentration $u = u(x, y, t)$:

$$\frac{\partial u}{\partial t} = D(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) + S \tag{1}$$

where D is the diffusion coefficient and S is the source term given by the following equation:

$$S(x, y, t) = (2D - 1)exp(-t)sin(x)cos(y)$$

The diffusion equation is within the domain [-1,1]x[-1,1] and solved for with the initial condition:

$$u(x, y, t = 0) = sin(x)cos(y)$$

We can approximate the diffusion equation by the following standard numerical method:

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = D(\frac{u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{\Delta x} + \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{\Delta y}) + S(x(i), y(j), t^n) \tag{2}$$

We would like to implement this method in MATLAB for two different cases: Dirichlet and the Robin case. The Dirichlet (or first-type) boundary condition is a type of boundary condition, named after Peter Gustav Lejeune Dirichlet. When this boundary condition is imposed on an ordinary or a partial differential equation, it specifies the values that a solution needs to take on along the boundary of the domain. The Dirichlet boundary conditions will be applied to the four walls of the computational domain with the equations u(x,y,t) = exp(-t)sin(x)cos(y).

Whereas the Robin (or third type) boundary condition is a type of boundary condition, named after Victor Gustave Robin. When this boundary condition is imposed on an ordinary or a partial differential equation, it is a specification of a linear combination of the values of a function and the values of its derivative on the boundary of the domain. The Robin boundary condition has the following form:

$$\alpha u \pm \frac{\partial u}{\partial x} = exp(-t)cos(y)(\alpha sin(\pm 1) - cos(\pm 1))$$
$$\alpha u \pm \frac{\partial}{\partial y} = exp(-t)sin(x)(\alpha cos(\pm 1) - sin(\pm 1))$$

with the $\pm$ sign depending on which wall you are in the computational domain.

This assignment asks the following:

a) Implement this method in MATLAB in the case of Dirichlet and Robin boundary conditions, i.e u(x,y,t) = exp(-t)sin(x)cos(y) on the four walls of the computational domain.
b) In both the Dirichlet and the Robin case, compare the results with the exact solution, which is u(x,y,t) = exp(-t)sin(x)cos(y). Perform accuracy analysis to compute the maximum error between the numerical and the exact solutions (in absolute value). Show that the error is roughly divided by 4 when the number of grid points in the x and y directions are both multiplied by 2.

Take $\Delta t = 0.2\Delta x^2$, D = 0.7, $\alpha = 2$, and $t_{final} = 0.2$ s. For the accuracy analysis, consider the grids 10x10, 20x20, 40x40, and 80x80.

2 Background

The diffusion equation in 1D is derived to be:

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + S \quad (3)$$

Where D is the diffusion constant and S is the source term. A numerical approximation can be made for this differential equation setup, which looks like the following:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = D\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} + S_i \quad (4)$$

The numerical solution can be solved for by isolating the u_i^{n+1} term. This will provide you with the following equation in **1D case**:

$$u_i^{n+1} = u_i^n + \Delta t \cdot D \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} + \Delta t \cdot S_i \tag{5}$$

This analysis can be extended into 2D, which is exactly what you will need to do in order to solve this assignment. The following equations are the derived 2D numerical equations with the respective source terms and initial conditions (as seen in the introduction):

$$\begin{aligned} u_{i,j}^{n+1} &= u_{i,j}^n + \Delta t \cdot D(\frac{u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n}{\Delta x} + \frac{u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n}{\Delta y}) + \Delta t \cdot S(x(i), y(j), t^n) \\ S(x,y,t) &= (2D-1)exp(-t)sin(x)cos(y) \\ u(x,y,t=0) &= sin(x)cos(y) \end{aligned}$$

Now that the diffusion equation for the concentration of u = u(x,y,t) is in 2D, we need to apply the Dirichlet and Robin boundary conditions.

The Dirichlet boundary conditions are applied to the numerical diffusion solution by setting the values on the four walls of the computational domain equal to the exact solution.

$$u(x,y,t) = exp(-t)sin(x)cos(y) \tag{6}$$

The Robin boundary conditions are implemented by applying the following equations at the four walls of the computational domain:

$$\begin{aligned} \alpha u - \frac{\partial u}{\partial x} &= exp(-t)cos(y)(\alpha sin(-1) - cos(-1)) \quad at\ left\ wall\ x = -1 \\ \alpha u + \frac{\partial u}{\partial x} &= exp(-t)cos(y)(\alpha sin(1) - cos(1)) \quad at\ right\ wall\ x = 1 \\ \alpha u - \frac{\partial u}{\partial y} &= exp(-t)sin(x)(\alpha cos(-1) + sin(-1)) \quad at\ bottom\ wall\ y = -1 \\ \alpha u + \frac{\partial u}{\partial y} &= exp(-t)cos(y)(\alpha cos(1) - cos(-1)) \quad at\ top\ wall\ y = 1 \end{aligned}$$

3 Algorithms

The equations for the Dirichlet boundary conditions will be used to solve for the four walls of the computational domain. Initially we would like to loop to solve for the interior part of the computational domain. This leaves us with the four walls left to solve for in the computational domain. This can be solved for by looking at each index (i,j) of our domain and solving for the exact solutions at that index and using a for loop to find the rest of the points. Doing this method for all four walls of the computational domain will yield a solution similar to the exact solution.

Also, the equations for the Robin boundary condition will be used to solve for the four walls of the computational domain. Similar to before, this is done by looping to find all the values within the domain then looking at an inner section of wall and apply the equation for the robin boundary condition shown above in the background section. These equations will be used to create "ghost points" in order to solve for the numerically approximated solution. When solving for the non-corner parts of the computational domain, you only need to create one "ghost point" and use one equation for the Robin boundary condition shown in the Background section. When looking at the corners, one needs to consider two of the boundary conditions because a corner is the intersect on of two computational walls. This is done by creating two ghost points and then solving the numerical solution, which should yield a solution similar to the exact and Dirichlet boundary conditions solution.

This algorithm is implemented in the MATLAB code below, and the way to plot the solution depends on commenting in which boundary condition you would like to solve for. This is done for both cases, and plots of the final diagrams for both boundary condition solutions can be shown in the results section. I ended up with plots which were very similar and the maximum error followed the trend of being divided by four when the grid points are multiplied by 2.

4 Results

Dirichlet BC	Dirichlet Error	Robin BC	Robin Error
10 x 10	$4.7834 \cdot 10^{-5}$	10 x 10	$9.832 \cdot 10^{-4}$
20 x 20	$1.0102 \cdot 10^{-5}$	20 x 20	$2.2065 \cdot 10^{-4}$
40 x 40	$2.3622 \cdot 10^{-6}$	40 x 40	$5.2459 \cdot 10^{-5}$
80 x 80	$5.67355 \cdot 10^{-7}$	80 x 80	$1.2802 \cdot 10^{-5}$

The table above shows a comparison of the numerical to the exact solution for the two different boundary condition cases: Dirichlet and Robin cases. The error associated with each case is solved for by taking the absolute value of the difference between the numerical diffusion solution and the exact diffusion solution while varying the x and y grid points. It should be noted that as the grid points are increased by two the error divides by approximately 4.

An example calculation for both the Dirichlet boundary condition and the Robin boundary condition for 10 x 10 to 20 x 20 grid points is shown below.

Dirichlet Boundary Conditions:

$$\begin{aligned} m_x \; x \; m_y : \frac{10}{20} &= 2 \\ error : \quad \frac{4.7834 \cdot 10^{-5}}{1.0102 \cdot 10^{-5}} &= 4.7351 \; (\sim 4) \end{aligned}$$

Robin Boundary Conditions:

$$\begin{aligned} m_x \; x \; m_y : \frac{10}{20} &= 2 \\ error : \quad \frac{9.832 \cdot 10^{-4}}{2.2065 \cdot 10^{-4}} &= 4.45593 \; (\sim 4) \end{aligned}$$

5 Implementation in Matlab

The Matlab implementation, with comments, is given here:

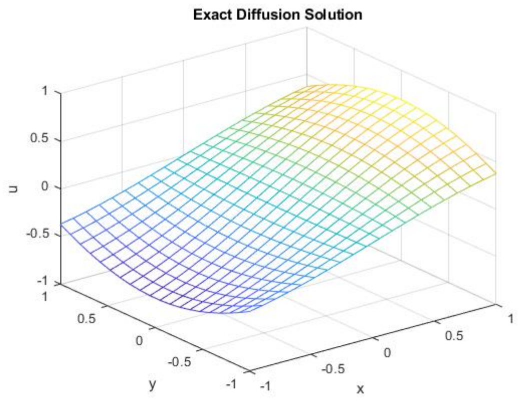
```
%%ME 17 HW5 - Alex Nguyen

clc; clear; close all;

%% Initial Data:

% Domain:
xmin = -1; xmax = 1; mx = 80;
ymin = -1; ymax = 1; my = 80;

% Discretize the and and y axis:
x = linspace(xmin, xmax, mx); dx = x(2) - x(1);
y = linspace(ymin, ymax, my); dy = y(2) - y(1);

% Initial and final time:
t = 0; tfinal = 0.2;

% Diffusion coefficient:
D = 0.7;

% Initial solution:
un = zeros(mx,my);
for i = 1:mx
```

```
    for j = 1:my
        un(i,j) = sin(x(i))*cos(y(j));
    end
end

% Source term and exact solution:
Exact = @(x,y,t)         exp(-t)*sin(x)*cos(y);
S     = @(x,y,t) (2*D-1)*exp(-t)*sin(x)*cos(y);

% Boundary conditions:
% BC = "DIRICHLET";
BC = "ROBIN";

if BC == "ROBIN"
    alpha = 2;
    gtop   = @(x,t) exp(-t)*sin(x)*(alpha*cos(ymax) - sin(xmax));
    gleft  = @(y,t) exp(-t)*cos(y)*(alpha*sin(xmin) - cos(ymin));
    gright = @(y,t) exp(-t)*cos(y)*(alpha*sin(xmax) + cos(ymax));
    gbottom   = @(x,t) exp(-t)*sin(x)*(alpha*cos(ymin) + sin(xmin));
end

%% Computation:

% Define the time step (the formula for dt is given by stability analysis,
% which is beyond the scope of this class):
dt = 0.2*dx*dx;

% Preallocation:
unp1 = zeros(mx, my);
ue   = zeros(mx, my);

% March in time
while t < tfinal
    if t + dt > tfinal
        dt = tfinal - t; %ensures tfinal is reached
    end

    % Use the update rule to go from un to unp1 for all grid indices:
```

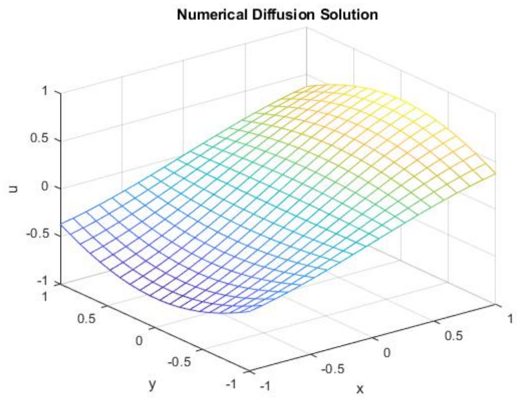
```
for i = 2:mx-1
    for j = 2:my-1

        unp1(i,j) = un(i,j) + D*dt*(un(i+1,j)-2*un(i,j)+un(i-1,j))/dx/dx ...
                    + D*dt*(un(i,j+1)-2*un(i,j)+un(i,j-1))/dy/dy...
                    + dt*S(x(i),y(j),t);

    end
end

% Impose Dirichlet BCs to Each Wall:
if     BC == "DIRICHLET"

    for j = 1:my

        unp1(1,j)  = exp(-(t+dt))*sin(xmin)*cos(y(j));  %Left wall
        unp1(mx,j) = exp(-(t+dt))*sin(xmax)*cos(y(j)); %Right wall

    end

    for i = 1:mx

        unp1(i,1)  = exp(-(t+dt))*sin(x(i))*cos(ymin);  %Bottom wall
        unp1(i,my) = exp(-(t+dt))*sin(x(i))*cos(ymax); %Top wall

    end

elseif BC == "ROBIN"

   for j = 2:my-1

       unxo = un(2,j) + 2*dx*(gleft(y(j),t) - alpha*un(1,j));  %min x ghost value
       unxm1 = un(mx-1,j) + 2*dx*(gright(y(j),t) - alpha*un(mx,j)); %max x ghost value

       %Left Wall
       unp1(1,j) = un(1,j) + D*dt*(un(2,j) - 2*un(1,j) + unxo)/dx/dx...
                   + D*dt*(un(1,j+1) - 2*un(1,j) + un(1,j-1))/dy/dy ...
                   + dt*S(x(1),y(j),t);
       %Right Wall
       unp1(mx,j) = un(mx,j) + D*dt*(unxm1 - 2*un(mx,j) + un(mx-1,j))/dx/dx...
                    + D*dt*(un(mx,j+1) - 2*un(mx,j) + un(mx,j-1))/dy/dy ...
                    + dt*S(x(mx),y(j),t);

   end

   for i = 2:mx-1

       unyo = un(i,2) + 2*dy*(gbottom(x(i),t) - alpha*un(i,1));  %min y ghost value
       unym1 = un(i,my-1) + 2*dy*(gtop(x(i),t) - alpha*un(i,my));  %max y ghost value

       %Bottom wall
       unp1(i,1) = un(i,1) + D*dt*(un(i+1,1) - 2*un(i,1) + un(i-1,1))/dx/dx...
                    + D*dt*(un(i,2) - 2*un(i,1) + unyo)/dy/dy ...
                    + dt*S(x(i),y(j),t);
       %Top wall
       unp1(i,my) = un(i,my) + D*dt*(un(i+1,my) - 2*un(i,my) + un(i-1,my))/dx/dx...
                    + D*dt*(unym1 - 2*un(i,my) + un(i,my-1))/dy/dy ...
                    + dt*S(x(i),y(j),t);

   end

   %Bottom Left Corner
   unxo = un(2,1) + 2*dx*(gleft(y(1),t) - alpha*un(1,1));
   unyo = un(1,2) + 2*dy*(gbottom(x(1),t) - alpha*un(1,1));

   unp1(1,1) = un(1,1) + D*dt*(un(2,1) - 2*un(1,1) + unxo)/dx/dx ...
                    + D*dt*(un(1,2) - 2*un(1,1) + unyo)/dy/dy ...
                    + dt*S(x(1),y(1),t);

   %Top Left Corner
   unxo = un(2,my) + 2*dx*(gleft(y(my),t) - alpha*un(1,my));
   unym1 = un(1,my-1) + 2*dy*(gtop(x(1),t) - alpha*un(1,my));

   unp1(1,my) = un(1,my) + D*dt*(un(2,my) - 2*un(1,my) + unxo)/dx/dx ...
                    + D*dt*(unym1 - 2*un(1,my) + un(1,my-1))/dy/dy ...
                    + dt*S(x(1),y(my),t);
    %Bottom Left Corner
    unxm1 = un(mx-1,1) + 2*dx*(gright(y(1),t) - alpha*un(mx,1));
    unyo = un(mx,2) + 2*dy*(gbottom(x(mx),t) - alpha*un(mx,1));

    unp1(mx,1) = un(mx,1) + D*dt*(unxm1 - 2*un(mx,1) + un(mx-1,1))/dx/dx ...
                 + D*dt*(un(mx,2) - 2*un(mx,1) + unyo)/dy/dy ...
                 + dt*S(x(mx),y(1),t);

    %Bottm Right Corner
    unxm1 = un(mx-1,my) + 2*dx*(gright(y(my),t) - alpha*un(mx,my));
    unym1 = un(mx,my-1) + 2*dy*(gtop(x(mx),t) - alpha*un(mx,my));

    unp1(mx,my) = un(mx,my) + D*dt*(unxm1 - 2*un(mx,my) + un(mx-1,my))/dx/dx ...
                 + D*dt*(unym1 - 2*un(mx,my) + un(mx,my-1))/dy/dy ...
                 + dt*S(x(mx),y(my),t);


else
    error('The boundary condition %s is not defined. ABORT. \n', BC);
end

% Update time and prepare for next iterate:
t = t + dt;
un = unp1;

% Exact solution:
for i = 1:mx
    for j = 1:my
        ue(i,j) = Exact(x(i),y(j),t);
    end
end

%Plot:
figure(1)
mesh(x,y,un')
xlabel('x'); ylabel('y'); zlabel('u');
axis([-1 1 -1 1 -1 1])
title('Numerical Diffusion Solution')
pause(dt);
```

```
    figure(2)
    mesh(x,y,ue')
    axis([-1 1 -1 1 -1 1])
    xlabel('x'); ylabel('y'); zlabel('u');
    title('Exact Diffusion Solution')
    pause(dt);

end

error = abs(un - ue);
disp(['The maximum error is ' num2str(max(max(error)))])
```

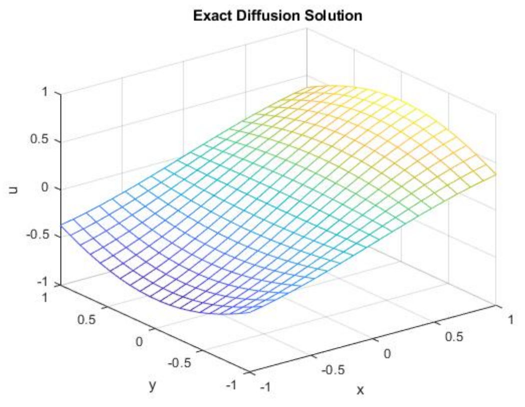
6 Boundary Condition Plots



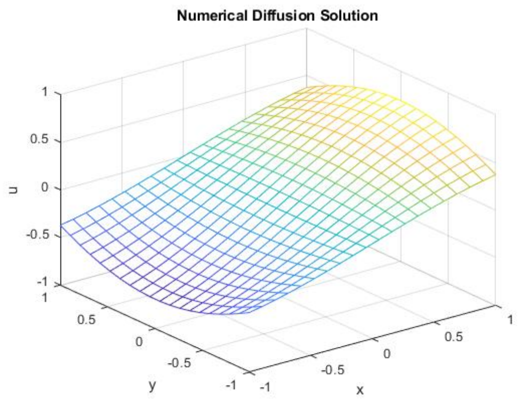
(a) Exact Dirichlet Solution



(b) Numerical Dirichlet Solution

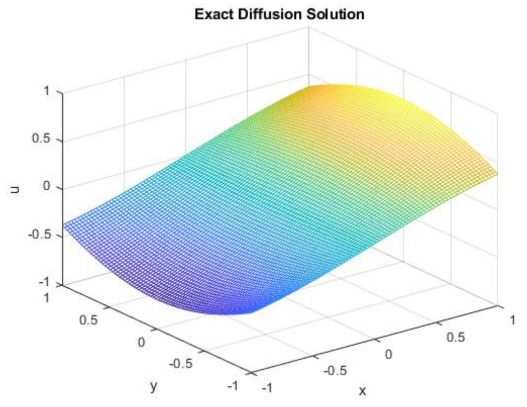


(c) Exact Robin Solution

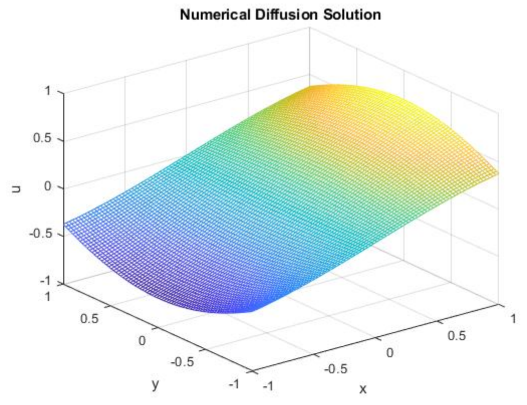


(d) Numerical Robin Solution

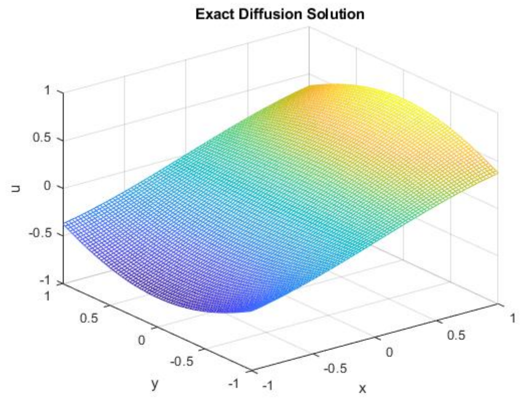
Figure 1: Boundary Condition Solutions with grid points [20x20] at $t_{final} = 0.2$ s



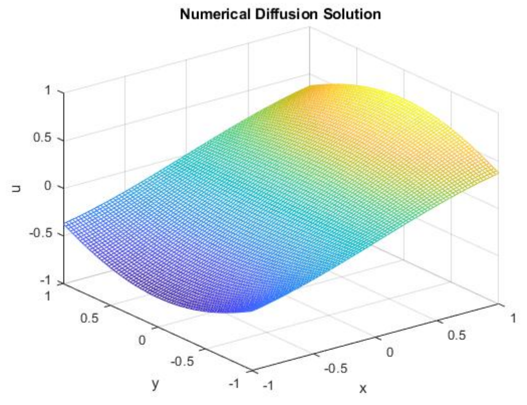
(a) Exact Dirichlet Solution



(b) Numerical Dirichlet Solution



(c) Exact Robin Solution



(d) Numerical Robin Solution

Figure 2: Boundary Condition Solutions with grid points [80x80] at $t_{final} = 0.2$s