DIFFUSION EQUATION Bc1-4

FINAL REPORT

Professors:
Andrea Prosperetti
Amit Amritkar

By**:** Hung Vu
ID:1152030

MECE 5397: Scientific Computing in Mechanical Engineering
Department of Mechanical Engineering
University of Houston

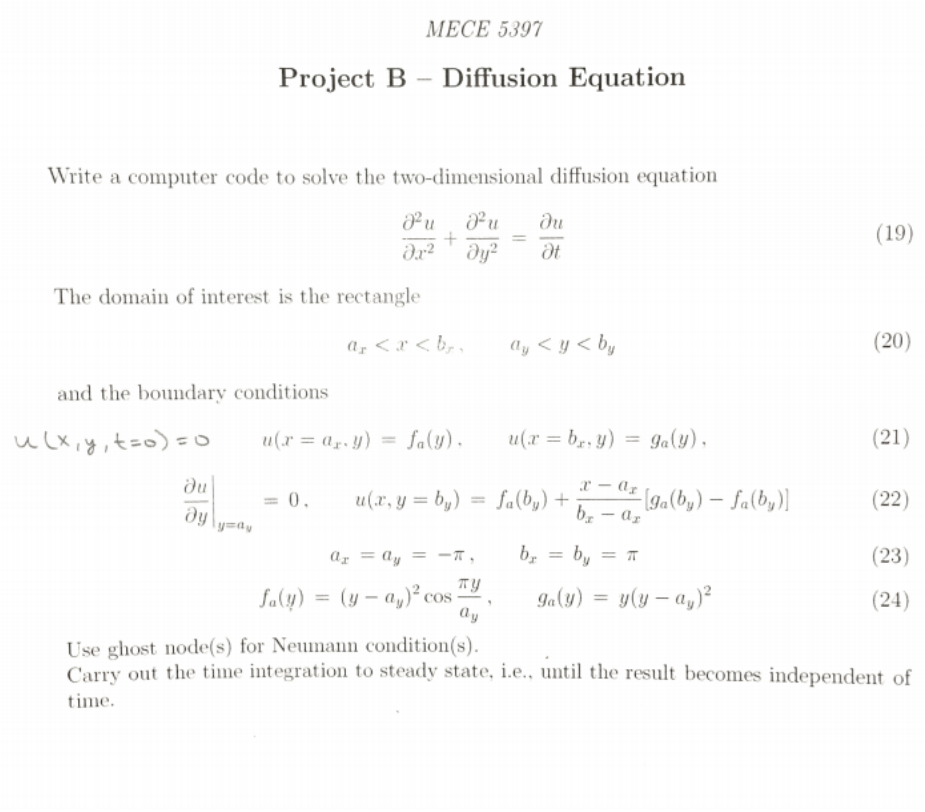
Spring 2018
May 9, 2018

Abstract

A diffusion equation was provided by the professors of MECE 5397 at the University of Houston. Two methods that were elected are the explicit method and Alternating Direction Implicit scheme. The explicit method was selected because the method is fast. However, the stability of the explicit method requires that $\lambda = \frac{D\Delta t}{\Delta x^2} < \frac{1}{2}$. In other words, the pseudo code could blow up unless that condition is met. The explicit form has a second order error. In the other case, the ADI scheme however, is unconditionally stable. The time and space will have a second order accuracy. This method uses two steps that each involve a tri-diagonal matrix. Each step is written explicitly and implicitly. This is the reason why it's called the "alternating direction." A pseudo code will be created to execute and analyze the diffusion equation that is provided from the professors. In this report, a description of the numerical method and results will be shown.

Mathematical statement of the problem

A diffusion equation was provided by the professors of MECE 5397 at the University of Houston. As shown in Figure 1, the diffusion is a 2D problem with 4 different boundary conditions. One of the four boundary conditions is a Neumann condition which ghost nodes will be used during MATLAB set-up.



MECE 5397

Project B – Diffusion Equation

Write a computer code to solve the two-dimensional diffusion equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{\partial u}{\partial t} \tag{19}$$

The domain of interest is the rectangle

$$a_x < x < b_x, \qquad a_y < y < b_y \tag{20}$$

and the boundary conditions

$$u(x = a_x, y) = f_a(y), \qquad u(x = b_x, y) = g_a(y), \tag{21}$$

$$\left.\frac{\partial u}{\partial y}\right|_{y=a_y} = 0, \qquad u(x, y = b_y) = f_a(b_y) + \frac{x - a_x}{b_x - a_x}[g_a(b_y) - f_a(b_y)] \tag{22}$$

$$a_x = a_y = -\pi, \qquad b_x = b_y = \pi \tag{23}$$

$$f_a(y) = (y - a_y)^2 \cos\frac{\pi y}{a_y}, \qquad g_a(y) = y(y - a_y)^2 \tag{24}$$

Use ghost node(s) for Neumann condition(s).
Carry out the time integration to steady state, i.e., until the result becomes independent of time.

Figure 1: Diffusion Equation provided by professors

Discretized version of the equations

Discretization is the process of changing an equation into a form that can be suitable for computers to identify. As shown in Figure 2, an explicit discretization method was used to convert the diffusion equation that was given.

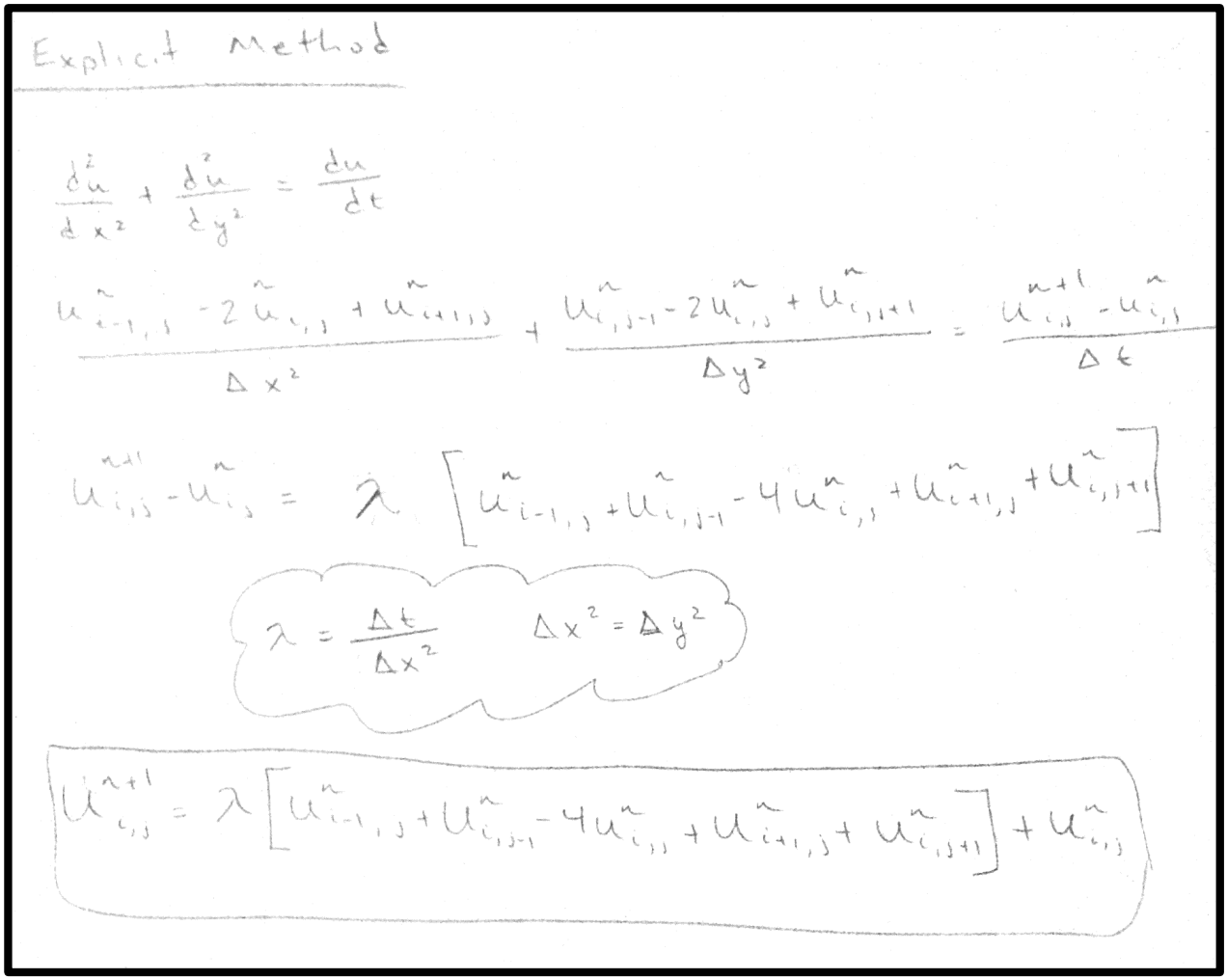


Figure 2:Explicit Discretization

The second method that was used to solve the diffusion equation was the Alternating Direction Implicit scheme. This method uses two steps, each involving a tri-diagonal matrix, to solve the problem. As shown in Figure 3, the first step is explicit in y and implicit in x. In figure 4 however, is the second step that is explicit in x and implicit in y. This is the reason why it's called the "alternating direction."

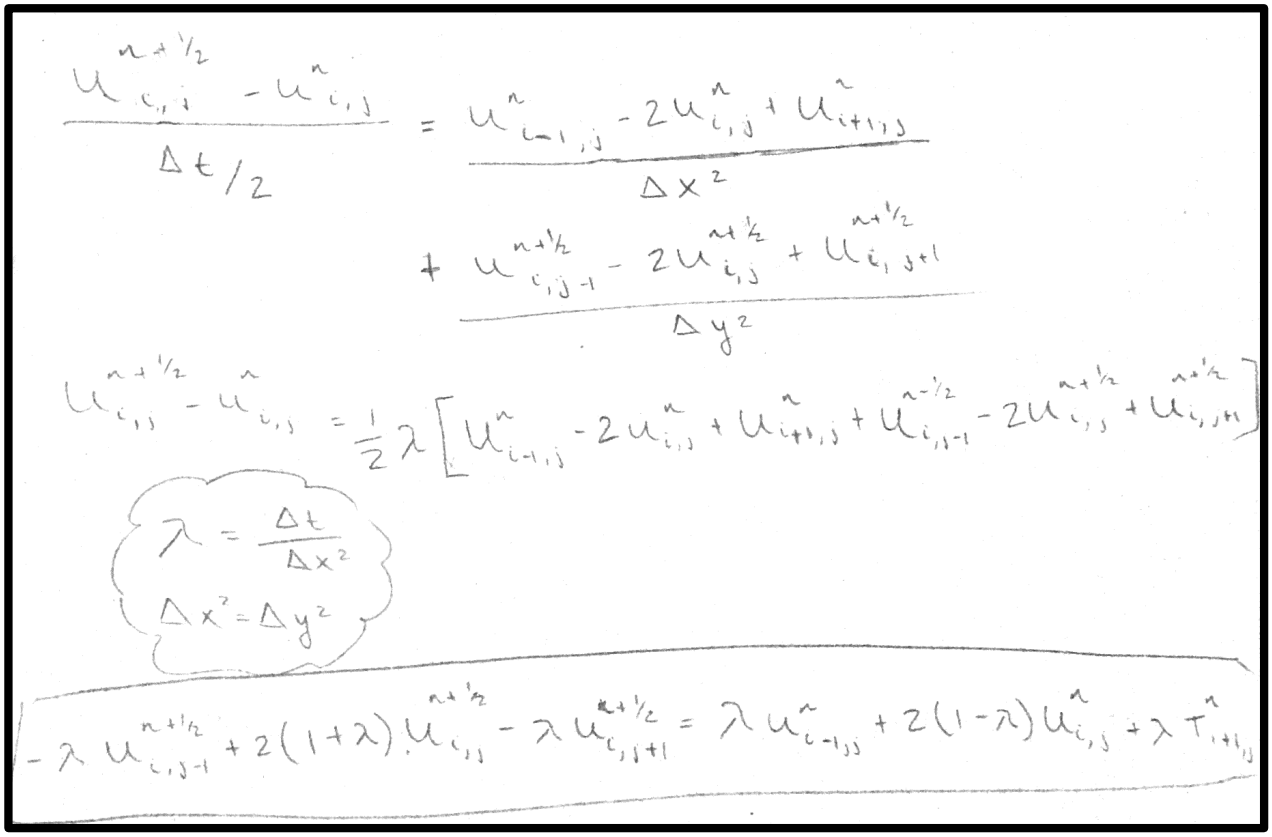


Figure 3: ADI Discretization (explicit part)

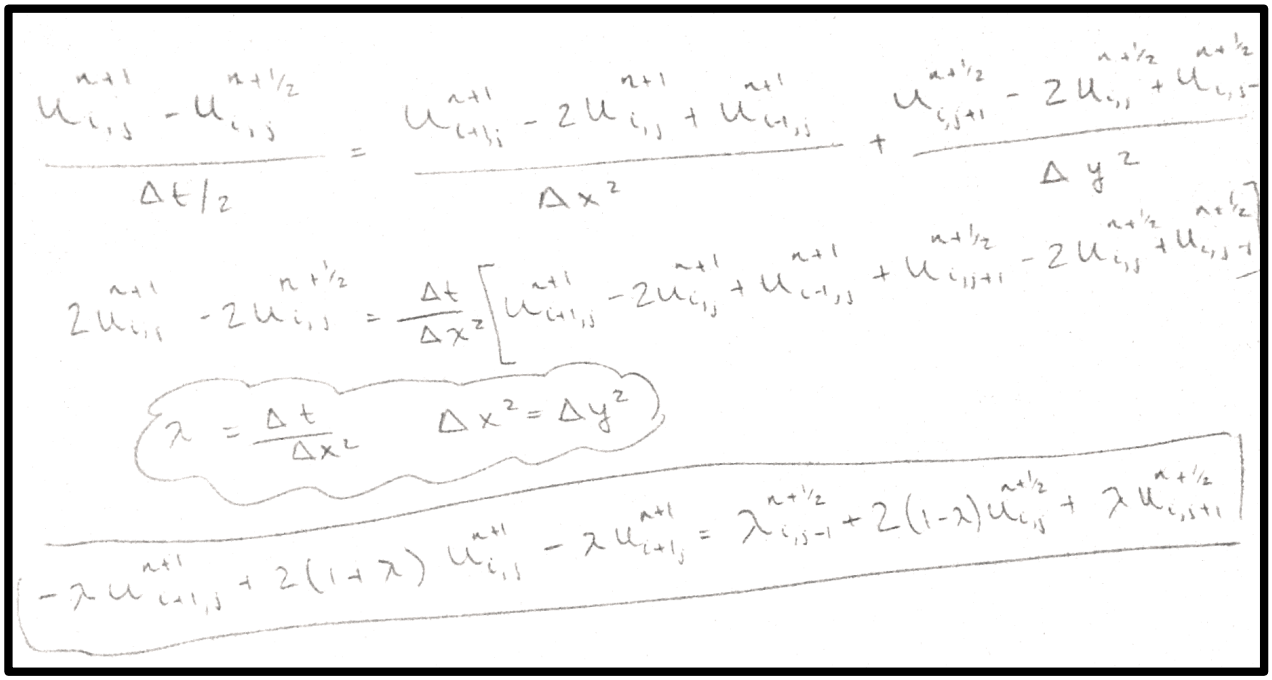


Figure 4:ADI Discretization (implicit part)

Description of the numerical method

An explicit pseudo code was created based on the discretization that was created in Figure 2. Research was conducted and based off youtube [1], a code was created. The first step of creating the code is adding in all the parameters that were given. X and Y matrix from –Pi to Pi were created for additional use. A U matrix the size of Nodes X Nodes were created so we can start adding in our diffusion equation.

```
27 - for k=1:Poi
28 -     U_New=zeros(Nx,Ny); %Setting new u matrix for diffusion
29 -     for j=2:Ny-1
30 -         for i=2:Nx-1
31 -             U_New(i,j)=(lambda*(-4*U(i,j)+U(i,j-1)+U(i-1,j)+U(i+1,j)+U(i,j+1)))+U(i,j); %Explicit
32 -         end
33 -     end
34
35       %Adding boundary conditions
36 -     U_New(:,1)=fliplr((y-ay).^2.*cos(pi*y/ay)); %Left Boundary Conditions
37 -     U_New(:,Ny)=fliplr(y.*(y-ay).^2); %Right Boundary Conditions
38 -     faby=(by-ay)^2*cos(pi*by/ay); %Given for the top boundary condition
39 -     gaby=by*(by-ay)^2; %Given for the top boundary condition
40 -     U_New(1,:)=faby+(x-ax)./(bx-ax)*(gaby-faby);  %Top Boundary Conditions
41
42       %Adding ghost node(s) for Neumann condition
43 -     for i=2:Nx-1
44 -        U_New(Nx,i)=(lambda*(U(i,j-1)+2*U(i-1,j)+U(i,j+1)))+(1-2*lambda-2*lambda)*U(i,j); %Explicit
45 -     end
```

Figure 5:Diffusion Equation and Boundary conditions

As shown in Figure 5, a for loop was created. On line 31, the discretization diffusion equation was added. Lines 36 thru 40 are all the boundary conditions that were given. Line 44, however, is the Newman Condition that uses ghost nodes. Time was then added by delta T so the time can be identified. A mesh plot was added into the pseudo code so to display visualization of the diffusive system. See Appendix A for the entire pseudo code.

Another code was created based off the pseudo code that was previous mentioned. This new code was created to identify the errors associate with the number of iterations. See Appendix A for the entire pseudo code.

The other method that was created is the ADI. This code was much longer and harder than the explicit method. As mentioned earlier, the first step is to identify all the parameters of the project based on the discretization of the ADI. This time, the boundary condition is added outside the loop as shown in Figure 6. This will speed up the process of the code.

```
24      %Boundary Conditions
25 -    faby=(by-ay)^2*cos(pi*by/ay); %Given for the top boundary condition
26 -    gaby=by*(by-ay)^2; %Given for the top boundary condition
27 -    Left=(y-ay).^2.*cos(pi*y/ay); %Left Boundary Conditions
28 -    Right=y.*(y-ay).^2; %Right Boundary Conditions
29 -    Top=(faby+(x-ax)./(bx-ax)*(gaby-faby)); %Top Boundary Conditions
30 -    Bottom=zeros(1,length(x)); % Bottom Boundary Condition
```

Figure 6:Boundary condition for ADI

Each step of this method uses a tridiagonal matrix. Therefore, one would have to create this matrix to execute the program. This code is seen in Figure 7 and the overall matrix is seen in Figure 8

```
37      %Tridiagonal matrix algorithm setup
38 -    dia1=2*(1+lambda)*ones(1,L1); %Creating a row to put in the diagonal
39 -    dia2=-lambda*ones(1,L1-1); %Creating a row to put in the diagonal
40 -    Mat=diag(dia1); %Adding diagonal into matrix
41 -    Mat=diag(dia2,1)+Mat; %Adding diagonal into matrix
42 -    Mat=diag(dia2,-1)+Mat; %Adding diagonal into matrix
43 -    mat=Mat; %Same matrix
44 -    mat(L1,L1+1)=Mat(1,2); %Adding more entries
45 -    mat(L1+1,[L1,L1+1])=[-lambda2,Mat(L1,L1)]; %Adding more entries
```

Figure 7: Tridiagonal Matrix

Variables - Mat

X | Mat

98x98 double

	1	2	3	4	5	6	7	8	9
1	2.5000	-0.2500	0	0	0	0	0	0	0
2	-0.2500	2.5000	-0.2500	0	0	0	0	0	0
3	0	-0.2500	2.5000	-0.2500	0	0	0	0	0
4	0	0	-0.2500	2.5000	-0.2500	0	0	0	0
5	0	0	0	-0.2500	2.5000	-0.2500	0	0	0
6	0	0	0	0	-0.2500	2.5000	-0.2500	0	0
7	0	0	0	0	0	-0.2500	2.5000	-0.2500	0
8	0	0	0	0	0	0	-0.2500	2.5000	-0.2500
9	0	0	0	0	0	0	0	-0.2500	2.5000
10	0	0	0	0	0	0	0	0	-0.2500
11	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0

Figure 8: Tridiagonal matrix visual

The next step is to create a true value matrix and start adding in all the entities. Since the boundary conditions for the left and right were given, codes were created to start filling in all the rows first. The steps are to create the first row, the rows in between, and then the last row. Once this was added into the matrix, the last boundary condition (bottom) was created and added. After all the rows have been completed, all the columns were created and added. Once all was completed, the true value matrix was found. The steady state time was also found. Another code was created based off the ADI code. The approximated value was found and a plot was created based on the true value that was currently found in the first code. This new code was created to identify the errors associate with the number of iterations. See Appendix B for the entire pseudo code.

Technical specifications of the computer used

Majority of the code and experiment was done at the computer lab at the University of Houston. The technical specifications of the computer are listed below:

Number of sockets - 4

CPU model name - Intel® Xeon® CPU E5620 @2.40 GHz

Number of cores/CPU - 1

Current CPU clock frequency - CPU MHz: 2394.000

Max CPU clock frequency - 2660 MHz

L1, L2, and L3 cache sizes –

-L1i cache: 32K

- L1d cache: 32K

- L2 cache: 256K

- L3 cache: 12288K

Number of memory channels - 64

Size of each DIMM - 27 bytes

Total DRAM per CPU - 27 bytes

Results

The simulations were done using mesh plotting, contour, and color bar. This well help the viewer see the different depths and heights of the graphs. As you can see in Figure 9, the graph is different colors indicating different depths and heights. A color bar was added to identify what the colors mean. As one can see, the graph shows to hit a steady state as the iterations increase. The graph doesn't change much after the 2nd plot shown. This was done over 100,000 iterations.

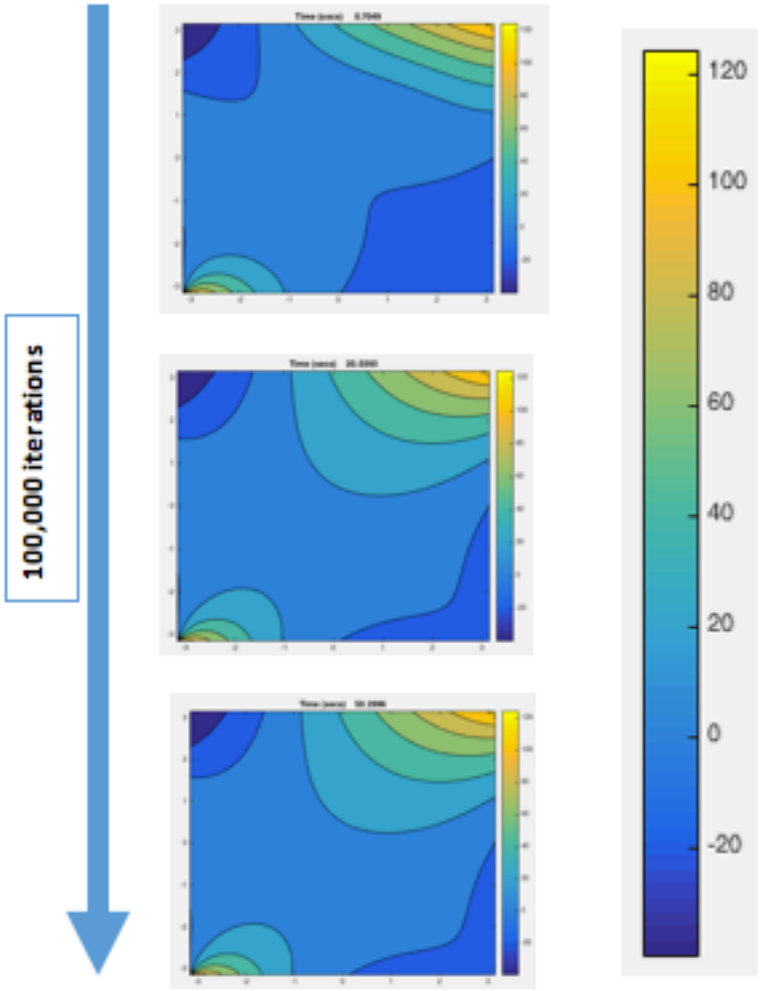


Figure 9: 100,000 iterations

As the iterations increase, the error decreased. As shown in Table 1, a pseudo code was created to see how many iterations would occur for the error to be less than 10^-10 There was only 3700 iterations, possibly indicating that there is an error in the code. Even though the code complies, there is a large variety of errors that one may not see.

Table I: Iterations Vs. Error

Iterations	Error
1000	2.1205e-04
2000	1.4443e-06
3000	9.8789e-09
3700	3.0149e-10

After the ADI code was created, a plot was created. As you can see in figure 10, the graphs don't look the same as it should. This indicates that there is an error somewhere. Either the ADI or the explicit is wrong. However, 100,000 iterations were ran and the results gave a steady state time of 100.7. The error indicated were 0.01216.

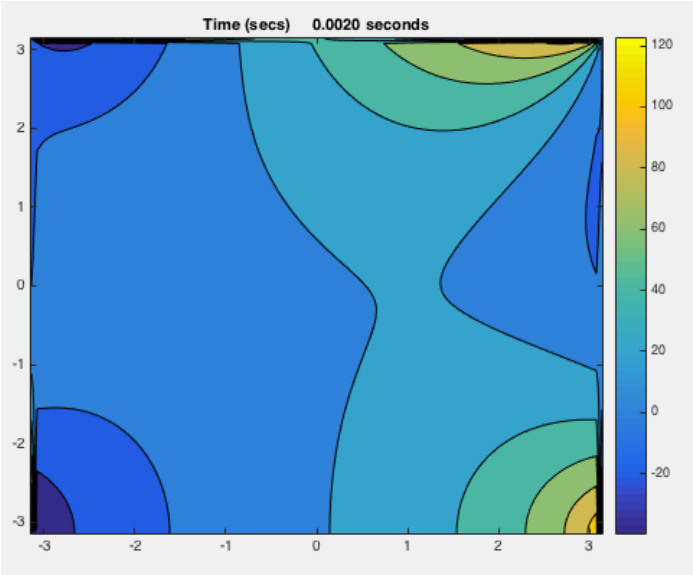


Figure 10: ADI graph

Reference

[1] “Lab10_3: Diffusion Eq 2D with Source.” YouTube, YouTube, 7 Apr. 2016, www.youtube.com/watch?v=aCRYfvh_bnY&t=523s.

Appendix A

Pseudo code for explicit method

```
%Hung Vu
%UHID: 1152030
%Project Bc1-4 with Explicit Method
clear all; clc; close all;

%domain of interest
ax=-pi; %parameter of x
bx=pi;  %parameter of x
ay=-pi; %parameter of y
by=pi;  %parameter of y

Nx=100; %Number of Nodes for x-direction
Ny=100; %Number of Nodes for y-direction
x=linspace(ax,bx,Nx);   %Creaing a matrix
y=linspace(ay,by,Ny);   %Creaing a matrix
[X,Y]=meshgrid(x,y);    %Creaing a matrix

dx=x(2)-x(1); %Change in x-direction
dy=y(2)-y(1); %Change in y-direction
dt=min([dx,dy])^2/8; %Change in t-direction with time step
lambda=dt/dx^2 %Setting lambda for better computation; Less than 0.25 stable

U=zeros(Nx,Ny);  %Creating martrix for initial u conditions
time=0; %Initial time condition
Poi=3000; % Total iterations being tested

for k=1:Poi
   U_New=zeros(Nx,Ny); %Setting new u matrix for diffusion
   for j=2:Ny-1
      for i=2:Nx-1
         U_New(i,j)=(lambda*(-4*U(i,j)+U(i,j-1)+U(i-1,j)+U(i+1,j)+U(i,j+1)))+U(i,j); %Explicit
      end
   end

   %Adding boundary conditions
   U_New(:,1)=fliplr((y-ay).^2.*cos(pi*y/ay)); %Left Boundary Conditions
   U_New(:,Ny)=fliplr(y.*(y-ay).^2); %Right Boundary Conditions
   faby=(by-ay)^2*cos(pi*by/ay); %Given for the top boundary condition
   gaby=by*(by-ay)^2; %Given for the top boundary condition
   U_New(1,:)=faby+(x-ax)./(bx-ax)*(gaby-faby);  %Top Boundary Conditions

   %Adding ghost node(s) for Neumann condition
   for i=2:Nx-1
     U_New(Nx,i)=(lambda*(U(i,j-1)+2*U(i-1,j)+U(i,j+1)))+(1-2*lambda-2*lambda)*U(i,j); %Explicit
   end

   U=U_New; %Converting the u matrix
   time=time+dt;  %Change in time
   U_New1=flipud(U_New); %Flipping the matrix for graph
```

```
    %Creating the plot
    if (mod(k,100) == 0)
        contourf(X,Y,U_New1) %Creating contour plot
        caxis([min(min(U_New1)) max(max(U_New1))]) %Creating axis values
        colorbar %adding colorbar to indicate values with color since 2D
        title(sprintf('Time (secs) %10.4f',time)) %Title
        drawnow %updates
    end

end
```

Pseudo code for explicit method error

```
%Hung Vu
%UHID: 1152030
%Project Bc1-4 with Explicit Method with Error
clear all; clc; close all;

%Domain of interest
ax=-pi; %parameter of x
bx=pi;  %parameter of x
ay=-pi; %parameter of y
by=pi;  %parameter of y

Nx=100; %Number of Nodes for x-direction
Ny=100; %Number of Nodes for y-direction
x=linspace(ax,bx,Nx);   %Creaing a matrix
y=linspace(ay,by,Ny);   %Creaing a matrix
[X,Y]=meshgrid(x,y);    %Creaing a matrix

dx=x(2)-x(1); %Change in x-direction
dy=y(2)-y(1); %Change in y-direction
dt=min([dx,dy])^2/8; %Change in t-direction with time step
lambda=dt/dx^2; %Setting lambda for better computation; Less than 0.25 stable

U=zeros(Nx,Nx);  %Setting initial temperature conditions
time=0; %Setting initial time
err=1; %Setting initial error
Poi=1; % total iterations
XX=zeros(Nx,Nx);

while err > 10^-10
    U_new=zeros(Nx,Nx);
    for j=2:Nx-1
        for i=2:Nx-1
            U_new(i,j)=(lambda*(-4*U(i,j)+U(i,j-1)+U(i-1,j)+U(i+1,j)+U(i,j+1)))+U(i,j); %Explicit
        end
    end

    % Adding boundary conditions
    U_new(:,1)=fliplr((y-ay).^2.*cos(pi*y/ay)); %Left Boundary Conditions
    U_new(:,Ny)=fliplr(y.*(y-ay).^2); %Right Boundary Conditions
    faby=(by-ay)^2*cos(pi*by/ay); %Given for the top boundary condition
```

```
    gaby=by*(by-ay)^2; %Given for the top boundary condition
    U_new(1,:)=faby+(x-ax)./(bx-ax)*(gaby-faby);  %Top Boundary Conditions
    %%
    %Adding ghost node(s) for Neumann condition
    for i=2:Nx-1
        U_new(Nx,i)=(lambda*(U(i,j-1)+2*U(i-1,j)+U(i,j+1)))+(1-2*lambda-2*lambda)*U(i,j); %Explicit
    end

    U=U_new; %Converting the u matrix
    time=time+dt; %Change in time
    U_new=flipud(U_new); %Flipping the matrix for graph

    %error check:
    a=mean(mean(U_new));
    b=mean(mean(XX));
    err=abs((a-b)/a*100);

    if (mod(Poi,1000) == 0)
        disp('Iterations:')
        disp(Poi)
        disp('Error:')
        disp(err)
    end
    Poi=1+Poi;
    XX=U_new;

end
```

Appendix B

Pseudo Code for for explicit method

```
clear all; clc; close all;

%domain of interest
ax=-pi; %parameter of x
bx=pi;  %parameter of x
ay=-pi; %parameter of y
by=pi;  %parameter of y

Nx=100; %Number of Nodes for x-direction
Ny=100; %Number of Nodes for y-direction
x=linspace(ax,bx,Nx);   %Creaing a matrix
y=linspace(ay,by,Ny);   %Creaing a matrix

dx=x(2)-x(1); %Change in x-direction
dy=y(2)-y(1); %Change in y-direction
```

```
dt=.125*(dx^2+dy^2); %Change in t-direction with Time step
lambda=dt/dx^2; %Setting lambda for better computation;
lambda2=2*lambda; %Setting lambda for better computation;
XX=2*(1-lambda); %Better outside the loop

%Boundary Conditions
faby=(by-ay)^2*cos(pi*by/ay); %Given for the top boundary condition
gaby=by*(by-ay)^2; %Given for the top boundary condition
Left=(y-ay).^2.*cos(pi*y/ay); %Left Boundary Conditions
Right=y.*(y-ay).^2; %Right Boundary Conditions
Top=(faby+(x-ax)./(bx-ax)*(gaby-faby)); %Top Boundary Conditions
Bottom=zeros(1,length(x)); % Bottom Boundary Condition

%Variables
L1=length(y)-2; %Setting up Length
L2=L1+1; %Setting up Length
Bas=2:L1+1; %Setting up Length

%Tridiagonal matrix algorithm setup
dia1=2*(1+lambda)*ones(1,L1); %Creating a row to put in the diagonal
dia2=-lambda*ones(1,L1-1); %Creating a row to put in the diagonal
Mat=diag(dia1); %Adding diagonal into matrix
Mat=diag(dia2,1)+Mat; %Adding diagonal into matrix
Mat=diag(dia2,-1)+Mat; %Adding diagonal into matrix
mat=Mat; %Same matrix
mat(L1,L1+1)=Mat(1,2); %Adding more entries
mat(L1+1,[L1,L1+1])=[-lambda2,Mat(L1,L1)]; %Adding more entries

%Adding boundary conditions into a U matrix.
U=zeros(Nx-2,Nx-2); % Initial condition with boundary condition
TRU=zeros(Nx,Ny); %Creating matrix with 0's
TRU(L1+2:-1:1,1)=Left; %Adding left boundary condition
TRU(1,1:1:L1+2)=Top; %Adding Top boundary condition
TRU(L1+2:-1:1,Nx)=Right; %Adding left boundary condition

Poi=1;
while Poi < 2000

    % Creating the first row with boundary conditions
    A(1,1:L1) = lambda*Top(Bas)+XX*U(1,1:L1)+lambda*U(2,1:L1);
    A(1) = A(1) + lambda*Left(2);
    A(L1) = A(L1) + lambda*Right(2);
    U(1,:) = Mat\A';

    % Creating the rest of the rows with boundary conditions
    AA(1:L1-2,1:L1) = lambda*U(1:L1-2,1:L1)+XX*U(2:L1-1,1:L1)+lambda*U(3:L1,1:L1);
    AA(:,1) = AA(:,1) + lambda*Left(3:L1)';
    AA(:,L1) = AA(:,L1) + lambda*Right(3:L1)';
    U(2:L1-1,:) = (Mat\AA')';

    % Creating the last row with boundary conditions
    A(1,1:L1) = lambda*U(L1-1,1:L1)+XX*U(L1,1:L1)+lambda*Bottom(Bas);
    A(1) = A(1) + lambda*Left(L2);
    A(L1) = A(L1) + lambda*Right(L2);
```

```
    U(L1,:) = Mat\A';

    % Creating the bottom condtion
    A(1,1:L1) = lambda2*U(L1,1:L1)+XX*Bottom(Bas);
    A(1) = A(1) + lambda*Left(L2);
    A(L1) = A(L1) + lambda*Right(L2);
    Bottom(2:L1+1) = Mat\A';

    % Creating the first column with boundary conditions
    B(1,1:L1) = lambda*Left(Bas)'+XX*U(1:L1,1)+lambda*U(1:L1,2);
    B(1) = B(1) + lambda*Top(2);
    B(L2) = lambda*Left(L1+2)+XX*Bottom(2)+lambda*Bottom(3);
    T_dummy = (mat\B')';
    U(:,1) = T_dummy(1:L1);
    Bottom(2) = T_dummy(L2);

    % Creating the rest of the columns with boundary conditions
    j = 2:L1-1;
    BB(1:L1,1:L1-2) = lambda*U(1:L1,1:L1-2)+XX*U(1:L1,j)+lambda*U(1:L1,3:L1);
    BB(1,:) = BB(1,:) + lambda*Top(3:L1);
    BB(L1+1,:) = lambda*Bottom(2:L1-1)+XX*Bottom(3:L1)+lambda*Bottom(4:L1+1);
    T_dummy = mat\BB;
    U(:,2:L1-1) = T_dummy(1:L1,1:L1-2);
    Bottom(3:L1) = T_dummy(L2,:);

    % Creating the last column with boundary conditions
    B(1,1:L1) = lambda*U(1:L1,L1-1)+XX*U(1:L1,L1)+lambda*Right(Bas)';
    B(1) = B(1) + lambda*Top(L1+1);
    B(L2) = lambda*Bottom(L1)+XX*Bottom(L2)+lambda*Right(L1+2);
    T_dummy = (mat\B')';
    U(:,L1) = T_dummy(1:L1);
    Bottom(L2) = T_dummy(L2);

    %Updating the true values
    TRU(Nx,L1+2:-1:1) = Bottom;
    TRU(2:L1+1,L1+1:-1:2) = U;

    if (mod(Poi,10^4) == 0)
        disp('Iteration passed:')
        disp(Poi)
    end

    Poi = Poi+1; % Going to the next iterations
end

clc
Time_SS = dt*Poi %Finding the steady state value


%% 5% True value test
clearvars -except TRU time_SS
%%
%domain of interest
ax=-pi; %parameter of x
```

```
bx=pi;  %parameter of x
ay=-pi; %parameter of y
by=pi;  %parameter of y

Nx=100; %Number of Nodes for x-direction
Ny=100; %Number of Nodes for y-direction
x=linspace(ax,bx,Nx);   %Creaing a matrix
y=linspace(ay,by,Ny);   %Creaing a matrix
[X,Y]=meshgrid(-x,-y); %Creating a mesh

dx=x(2)-x(1); %Change in x-direction
dy=y(2)-y(1); %Change in x-direction
dt=.125*(dx^2+dy^2); %Change in t-direction
lambda=dt/dx^2; %Setting lambda for better computation;
lambda2=2*lambda; %Setting lambda for better computation;
XX=2*(1-lambda); %Better outside the loop

% Inputing Data:

faby=(by-ay)^2*cos(pi*by/ay); %Given for the top boundary condition
gaby=by*(by-ay)^2; %Given for the top boundary condition
Left=fliplr((y-ay).^2.*cos(pi*y/ay)); %Left Boundary Conditions
Right=fliplr(y.*(y-ay).^2); %Right Boundary Conditions
Top=(faby+(x-ax)./(bx-ax)*(gaby-faby)); %Top Boundary Conditions
Bottom=zeros(1,length(x)); % right side

% varaibles used outside while loop to speed up process:
L1=length(y)-2; %Setting up Length
L2=L1+1; %Setting up Length
Bas=2:L1+1; %Setting up Length

%Tridiagonal matrix algorithm setup
dia1=2*(1+lambda)*ones(1,L1); %Creating a row to put in the diagonal
dia2=-lambda*ones(1,L1-1); %Creating a row to put in the diagonal
Mat=diag(dia1); %Adding diagonal into matrix
Mat=diag(dia2,1)+Mat; %Adding diagonal into matrix
Mat=diag(dia2,-1)+Mat; %Adding diagonal into matrix
mat=Mat; %Same matrix
mat(L1,L1+1)=Mat(1,2); %Adding more entries
mat(L1+1,[L1,L1+1])=[-lambda2,Mat(L1,L1)]; %Adding more entries

%creating a U matrix for an approximation value.
APX=zeros(Nx,Ny); %Creating matrix with 0's
APX(L1+2:-1:1,1)=Left; %Adding left boundary conditions
APX(1,1:1:L1+2)=Top; %Adding top boundary conditions
APX(L1+2:-1:1,Nx)=Right; %Adding right boundary conditions
U=zeros(Nx-2,Nx-2); % inital condition

Poi=1;
error=inf;
while error > 5

    % Creating the first row with boundary conditions
    A(1,1:L1) = lambda*Top(Bas)+XX*U(1,1:L1)+lambda*U(2,1:L1);
```

```
A(1) = A(1) + lambda*Left(2);
A(L1) = A(L1) + lambda*Right(2);
U(1,:) = Mat\A';

% Creating the rest of the rows with boundary conditions
AA(1:L1-2,1:L1) = lambda*U(1:L1-2,1:L1)+XX*U(2:L1-1,1:L1)+lambda*U(3:L1,1:L1);
AA(:,1) = AA(:,1) + lambda*Left(3:L1)';
AA(:,L1) = AA(:,L1) + lambda*Right(3:L1)';
U(2:L1-1,:) = (Mat\AA')';

% Creating the last row with boundary conditions
A(1,1:L1) = lambda*U(L1-1,1:L1)+XX*U(L1,1:L1)+lambda*Bottom(Bas);
A(1) = A(1) + lambda*Left(L2);
A(L1) = A(L1) + lambda*Right(L2);
U(L1,:) = Mat\A';

% Creating the bottom boundary condition
A(1,1:L1) = lambda2*U(L1,1:L1)+XX*Bottom(Bas);
A(1) = A(1) + lambda*Left(L2);
A(L1) = A(L1) + lambda*Right(L2);
Bottom(2:L1+1) = Mat\A';

% Creating the first column with boundary conditions
B(1,1:L1) = lambda*Left(Bas)'+XX*U(1:L1,1)+lambda*U(1:L1,2);
B(1) = B(1) + lambda*Top(2);
B(L2) = lambda*Left(L1+2)+XX*Bottom(2)+lambda*Bottom(3);
T_dummy = (mat\B')';
U(:,1) = T_dummy(1:L1);
Bottom(2) = T_dummy(L2);

% Creating the rest of the columns with boundary conditions
j = 2:L1-1;
BB(1:L1,1:L1-2) = lambda*U(1:L1,1:L1-2)+XX*U(1:L1,j)+lambda*U(1:L1,3:L1);
BB(1,:) = BB(1,:) + lambda*Top(3:L1);
BB(L1+1,:) = lambda*Bottom(2:L1-1)+XX*Bottom(3:L1)+lambda*Bottom(4:L1+1);
T_dummy = mat\BB;
U(:,2:L1-1) = T_dummy(1:L1,1:L1-2);
Bottom(3:L1) = T_dummy(L2,:);

% Creating the last column with boundary conditions
B(1,1:L1) = lambda*U(1:L1,L1-1)+XX*U(1:L1,L1)+lambda*Right(Bas)';
B(1) = B(1) + lambda*Top(L1+1);
B(L2) = lambda*Bottom(L1)+XX*Bottom(L2)+lambda*Right(L1+2);
T_dummy = (mat\B')';
U(:,L1) = T_dummy(1:L1);
Bottom(L2) = T_dummy(L2);

% Updating the approximated values
APX(Nx,L1+2:-1:1) = Bottom;
APX(2:L1+1,L1+1:-1:2) = U;

error = abs(sum(sum(TRU-APX)))/Nx^2*100; %calulating the errors

% Creating a blot
```

```
    drawnow %updates
    Time = Poi*dt; %calulating the Time
    if ( mod(Poi,100) == 0)
        contourf(X,Y,APX)%Creating contour plot
        caxis( [ min(min(APX)), max(max(APX)) ] ) %Creating axis values
        colorbar %adding colorbar to indicate values with color since 2D
        title(sprintf('Time (secs) %10.4f seconds',Time)) %Title
        drawnow %updates
    end

    Poi = Poi+1; % next iteration

end


%% final plot
drawnow %updates
Time = Poi*dt; %calulating the Time
time_apprx = Time; %calulating the Time
contourf(X,Y,APX) %calulating the Time
caxis( [ min(min(APX)), max(max(APX)) ] ) %Creating axis values
colorbar %adding colorbar to indicate values with color since 2D
title(sprintf('Time (secs) %10.4f seconds',Time)) %Title
drawnow %updates

disp('Approximate time it took:')
disp(time_apprx)
disp('Error:')
disp(error)
```