MAE 5315 NUMERICAL METHODS FOR PDE
FINAL EXAM

Max Lê

ID: 901223283

September 21, 2018

Contents

List of Figures

1 INTRODUCTION

1.1 Problem Statement

In this problem, we are required to solve a 2D heat conduction equation:

$$u_t = u_{xx} + u_{yy} + f(x, y, t) \tag{1}$$

on the domain $x \in [0, 1]$ and $y \in [0, 1]$ with the source function defined as:

$$f(x, y, t) = e^{-t} sin(\pi x) sin(\pi y)(2\pi^2 - 1) \tag{2}$$

The solution is 0 on the boundary, in other words:

$$u(0, y, t) = u(1, y, t) = 0$$
$$u(x, 0, t) = u(x, 1, t) = 0$$

The initial condition is given as follow:

$$u(x, y, 0) = sin(\pi x) sin(\pi y) \tag{3}$$

We are required to solve the PDE using the Alternate Direction Implicit (ADI) scheme using a $dx = 1/64$ and up to t = 1.0

1.2 Analytical Solution

At first, the analytical solution is calculated. We start with a homogeneous PDE (f(x,y,t) = 0) and assume the solution $u(x, y, t) = X(x)Y(y)T(t)$, and substitute into the PDE, we get the following system of eigenvalue problems.

$$\frac{T'(t)}{T(t)} = \frac{X'(x)}{X(x)} + \frac{Y'(y)}{Y(y)} = -\lambda$$

The sum between X and Y can also be decomposed into a sub-eigenvalue problems, we will call this $-\mu$. Putting together, we have:

$$T'(t) + \lambda T = 0 \text{ where } T(0) = f(x, y, 0)$$
$$Y"(y) + \mu Y(y) = 0 \text{ where } Y(0) = Y(1) = 0$$
$$X"(x) + (\lambda - \mu)X(x) = 0 \text{ where } X(0) = X(1) = 0$$

Solving the Y equation first, for the case when $\lambda = 0$:

$$Y(y) = C1 + C2y$$

Substituting the boundary conditions for Y will yield both C1 = C2 = 0, which is a trivial solution and we do not want this. Now for the case when $\lambda = m^2 > 0$:

$$Y(y) = C1cos(my) + C2sin(my)$$

Again, plugging in the boundary conditions for Y yield C1 = 0; but this time we have the option of not allowing C2 to be 0. This implies m = kπ, for k = 1,2... Therefore, the first eigenvalue is known and the equation for Y is:

$$Y(y) = sin(m\pi y), m = 1, 2, ..$$

Substitute λ into the X equation, we can follow the similar procedure and get the solution as follow:

$$X(x) = sin(n\pi x), n = 1, 2, ..$$

Then our general solution will look like this:

$$u(x, y, t) = \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} sin(n\pi x)sin(m\pi y)B_{mn}(t)$$

Next, we expand the source function, f(x,y,t), into a Fourier series and compare

$$\begin{aligned} f(x, y, t) &= e^{-t}sin(\pi x)sin(\pi y)(2\pi^2 - 1) \\ &= \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} F_{mn}(t)sin(n\pi x)sin(m\pi y) \end{aligned}$$

We note that: when m = n = 1 => $F_{mn}(t) = e^{-t}(2\pi^2 - 1)$. Else, $F_{mn}(t) = 0$. Now, we use the general solution in the form of the Fourier series and get $u_t, u_x x, u_y y$ by taking derivatives. This yields the following:

$$\begin{aligned} u_t(x, y, t) &= \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} sin(n\pi x)sin(m\pi y)B'_{mn}(t) \\ u_{xx}(x, y, t) &= -\sum_{n=1}^{\infty}\sum_{m=1}^{\infty} (n^2\pi^2)sin(n\pi x)sin(m\pi y)B_{mn}(t) \\ u_{yy}(x, y, t) &= -\sum_{n=1}^{\infty}\sum_{m=1}^{\infty} (m^2\pi^2)sin(n\pi x)sin(m\pi y)B_{mn}(t) \end{aligned}$$

Equating these along with the Fourier expansion of the source term into the original PDE, doing calculation for each n and m, and simplifying, we get a first order ODE.

$$B'_{mn}(t) = -(m^2 + n^2)\pi^2 B_{mn}(t) + F_{mn}(t)$$

For values of $m \neq n \neq 1, B_{mn}(t) = 0$ because: we will not be able to get the initial condition $u(x, y, 0) = sin(\pi x)sin(\pi y)$. Therefore, for different values of m, n: $B_{mn}(t) = 0$ and from before: $F_{mn}(t) = 0$. The only value left is m = n = 1, this gives us the following ODE:

$$B'_{11}(t) = -(2\pi^2)B_{11}(t) + e^{-t}(2\pi^2 - 1) \tag{4}$$

Using the integration factor method, we can solve this equation as follow, dropping subscript 11:

$$e^{2\pi^2 t}B'(t) + e^{2\pi^2 t}B(t)(2\pi^2) = e^{(2\pi^2-1)-1}(2\pi^2 - 1)$$

$$\int_0^t \frac{d}{dt}\left(Be^{2\pi^2 t}\right) dt = \int_0^t e^{(2\pi^2-1)-1}(2\pi^2 - 1)$$

$$\boxed{B(t) = e^{-t}}$$

Substitute this into the general form of the solution, we get the following exact solution:

$$\boxed{u(x, y, t) = e^{-t} sin(\pi x) sin(\pi y)} \tag{5}$$

1.3 Numerical Solution

For our numerical solution, the basic idea is to apply a second order central difference discretization like this:

$$u_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} + O(\Delta x^2) \tag{6}$$

The stencil used in this problem is a node centered grid, the spacing, $\Delta x = \Delta y$, is used.

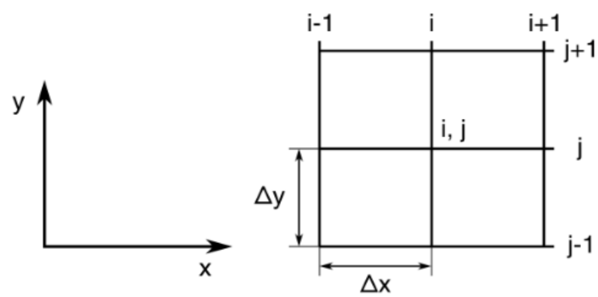

Figure 1: Numerical stencil

Let U_{ij}^n be the numerical solution at location (i,j) at time n. For the ADI scheme, we apply second order central difference discretization to advance solution from n to n+1/2:

$$\frac{U_{ij}^{n+1/2} - U_{ij}^n}{\Delta t/2} = \left[\frac{U_{i+1,j}^{n+1/2} - 2U_{ij}^{n+1/2} + U_{i-1,j}^{n+1/2}}{\Delta x^2} + \frac{U_{i,j+1}^n - 2U_{ij}^n + U_{i,j-1}^n}{\Delta y^2}\right] + f_i j^{n+1/4} \tag{7}$$

Then, the solution obtained at n+1/2 is then used to advance to final solution at time n+1.

$$\frac{U_{ij}^{n+1} - U_{ij}^{n+1/2}}{\Delta t/2} = \left[\frac{U_{i+1,j}^{n+1/2} - 2U_{ij}^{n+1/2} + U_{i-1,j}^{n+1/2}}{\Delta x^2} + \frac{U_{i,j+1}^{n+1} - 2U_{ij}^{n+1} + U_{i,j-1}^{n+1}}{\Delta y^2}\right] + f_{ij}^{n+3/4} \tag{8}$$

Both equation 6 and 7, when written out, will result in a tri-diagonal system. Rewriting the systems (putting everything we know at time n to the right hand side), and denoting the diffusion number as: $D_x = \dfrac{\alpha \Delta t}{\Delta x^2}$ and $D_y = \dfrac{\alpha \Delta t}{\Delta y^2}$, we get:

$$(1+D_x)U_{ij}^{n+1/2} + \left(\frac{D_x}{2}U_{i+1,j}^{n+1/2}\right) + \left(\frac{D_x}{2}\right)U_{i-1,j}^{n+1/2} = (1+D_y)U_{ij}^{n} + \left(\frac{D_x}{2}U_{i,j+1}^{n}\right) + \left(\frac{D_y}{2}\right)U_{i,j-1}^{n} + \frac{\Delta t}{2} f_i j^{n+1/4} \tag{9}$$

$$(1+D_y)U_{ij}^{n+1} + \left(\frac{D_y}{2}U_{i,j+1}^{n+1}\right) + \left(\frac{D_y}{2}\right)U_{i,j-1}^{n+1} = (1+D_x)U_{ij}^{n+1/2} + \left(\frac{D_x}{2}U_{i+1,j}^{n+1/2}\right) + \left(\frac{D_x}{2}\right)U_{i-1,j}^{n+1/2} + \frac{\Delta t}{2} f_{ij}^{n+3/4} \tag{10}$$

The tri-diagonal system looks like this:

$$\begin{bmatrix} (1-D_x) & (D_x/2) & 0 & \dots & 0 \\ (D_x/2) & (1-D_x) & (D_x/2) & \dots & 0 \\ 0 & (D_x/2) & (1-D_x) & (D_x/2) & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} u_1^{n+1/2} \\ u_2^{n+1/2} \\ u_3^{n+1/2} \\ \dots \\ u_n^{n+1/2} \end{bmatrix} = \begin{bmatrix} f_1^n \\ f_2^n \\ f_3^n \\ \dots \\ f_n^n \end{bmatrix}$$

Figure 2: X sweep tridiagonal system

$$\begin{bmatrix} (1-D_y) & (D_y/2) & 0 & \dots & 0 \\ (D_y/2) & (1-D_y) & (D_y/2) & \dots & 0 \\ 0 & (D_y/2) & (1-D_y) & (D_y/2) & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \\ \dots \\ u_n^{n+1} \end{bmatrix} = \begin{bmatrix} f_1^{n+1/2} \\ f_2^{n+1/2} \\ f_3^{n+1/2} \\ \dots \\ f_n^{n+1/2} \end{bmatrix}$$

Figure 3: Y sweep tridiagonal system

where f1,..,fn denote the known value at each node, and the superscript denotes the time level (n,n+1/2, or n+1). The right hand side is evaluated at each time step and at each node. In order to solve this system efficiently, we use Thomas Algorithm.

1.4 Thomas Algorithm Overview

Denoting the diagonal of the matrix as b,the subdiagonal as a and the superdiagonal as c, the solution vector as u and the right hand side vector as d, we can do the following:

$$\begin{bmatrix} b1 & c1 & 0 & \dots & 0 \\ a1 & b2 & c2 & \dots & 0 \\ 0 & a2 & b3 & c3 & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \dots \\ d_n \end{bmatrix}$$

Figure 4: Thomas parameters matrix

The ith equation in the sytem may be written as:

$$a_i u_{i-1} + b_i u_i + c_i u_{i+1} = d_i$$

where $a_1 = 0$ and $c_n = 0$. Looking at the system of equations, we see that ith unknown can be expressed as in terms of the (i+1)th unknown:

$$u_i = P_i U_{i+1} + Q_i$$
$$u_{i-1} = P_{i-1} i U_i + Q_{i-1}$$

If the all equations are written out like this, then the coefficient matrix would form an upper triangular matrix. Plugging these equations into the ith equation of the tri-diagonal matrix, we get the following expressions for P and Q:

$$P_i = \frac{-c_i}{b_i + a_i P_{i-1}}$$
$$Q_i = \frac{d_i - a_i Q_{i-1}}{b_i + a_i P_{i-1}}$$

These recursive relations show that ith unknown can be calculated when i-1 unknown is available. At i = 1, we have:

$$P_1 = \frac{-c1}{d1}$$
$$Q_1 = \frac{d1}{b1}$$

Below is a Fortran implementation for this algorithm, for our case, this will coded in Matlab.

```
      program TDMA
      implicit doubleprecision(a-h,o-z)
      parameter (nd = 100)
      doubleprecision A(nd), B(nd), C(nd), D(nd), X(nd), P(0:nd), Q(0:nd)
c
      A(1) = 0
      C(n) = 0
c
c     forward elimination
      do i = 1, n
         denom = B(i) + A(i)*P(i-1)
         P(i) = -C(i) /denom
         Q(i) = (D(i) - A(i)*Q(i-1)) /denom
      enddo
c
c     back substitution
      do i = n, 1, -1
         X(i) = P(i)*X(i+1) + Q(i)
      enddo
      stop
      end
```

Figure 5: Thomas Algorithm in Fortran

The basic idea for this ADI algorithm is to do:

1. X sweep
   - Use solution at n to solve X implicitly using and Y explicitly
   - Obtain solution at n+1/2
2. Y sweep
   - Use solution at n+1/2 to solve X implicitly using and Y explicitly
   - Obtain solution at n+1
3. Repeat until reach a specified time.

2 RESULTS

Below are the results obtained from this method:

2.1 Analytical and Numerical Results

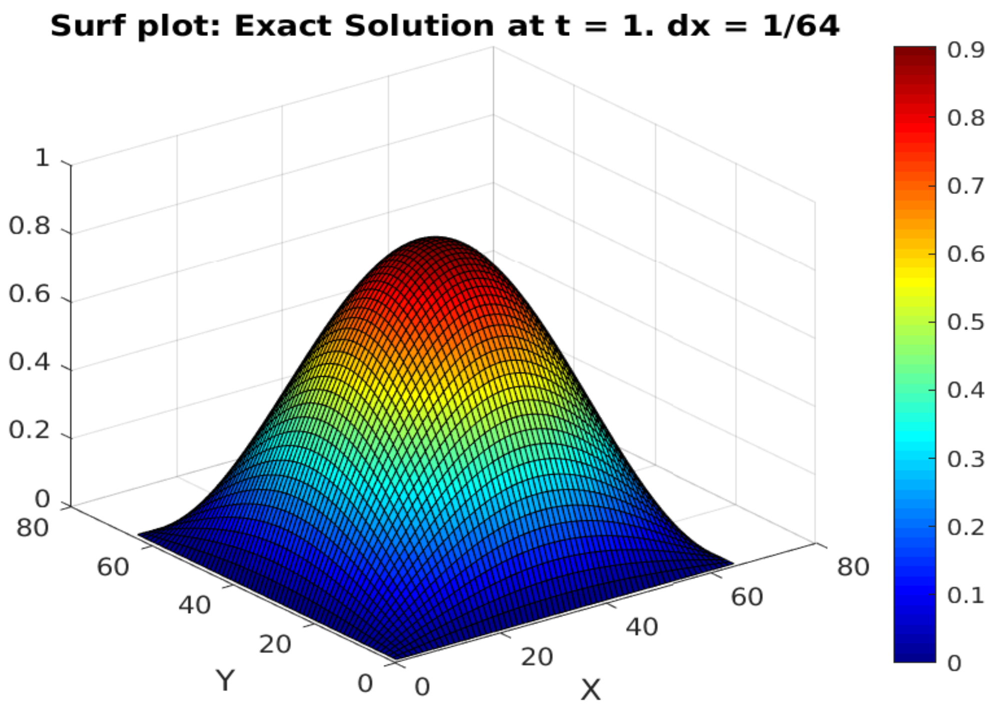

Figure 6: Analytical solution 3D plot

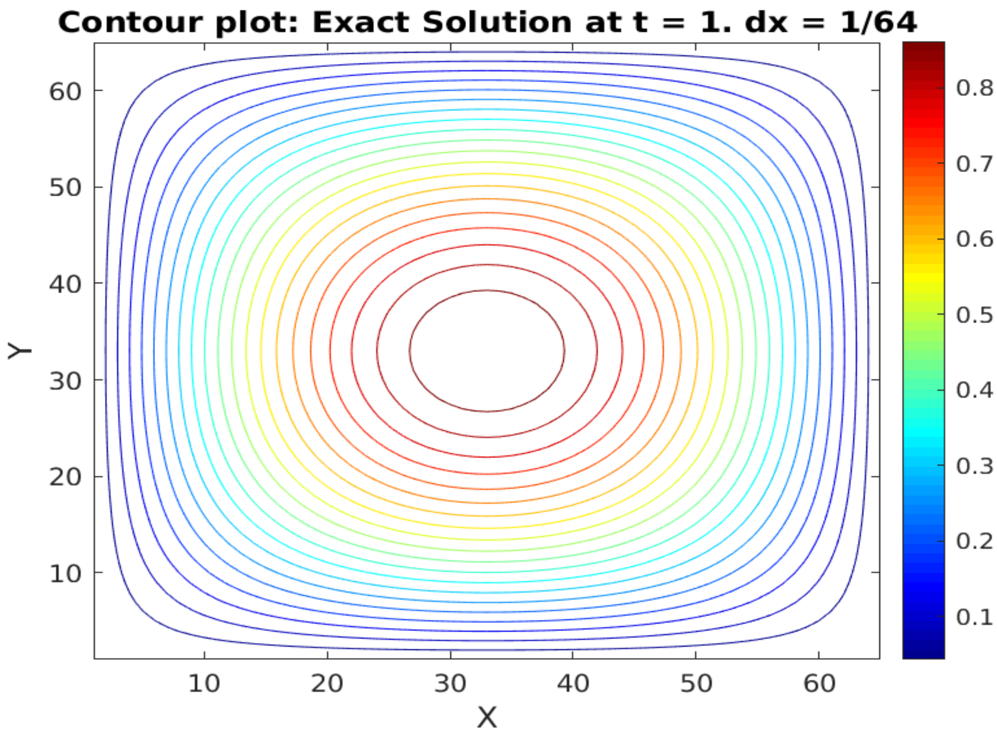

Figure 7: Analytical solution 2D plot

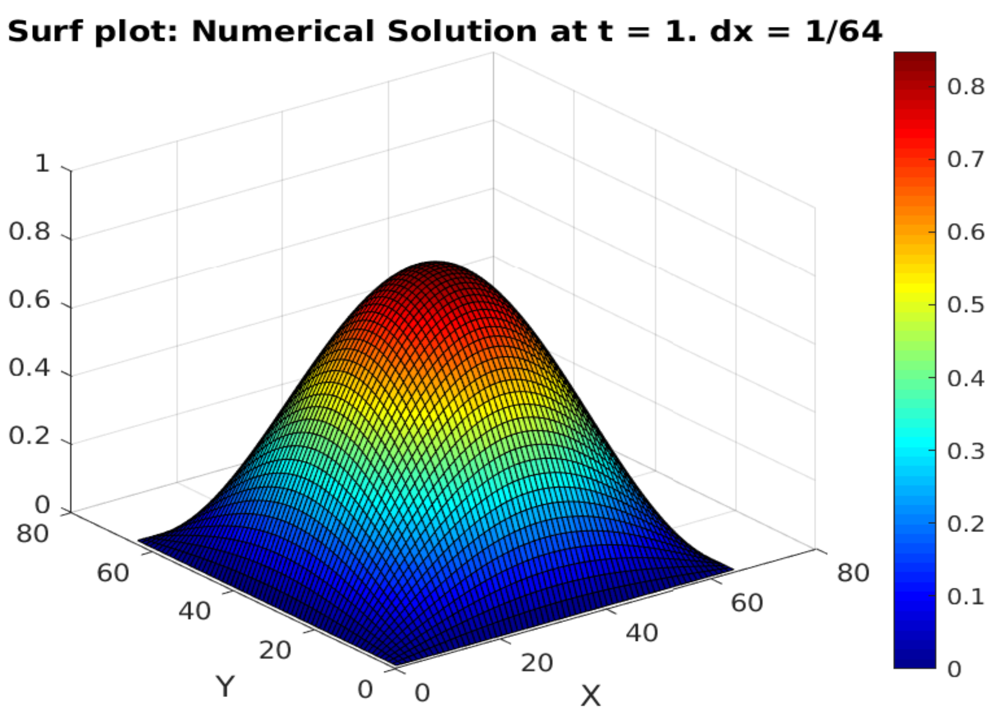

Figure 8: Numerical solution 3D plot

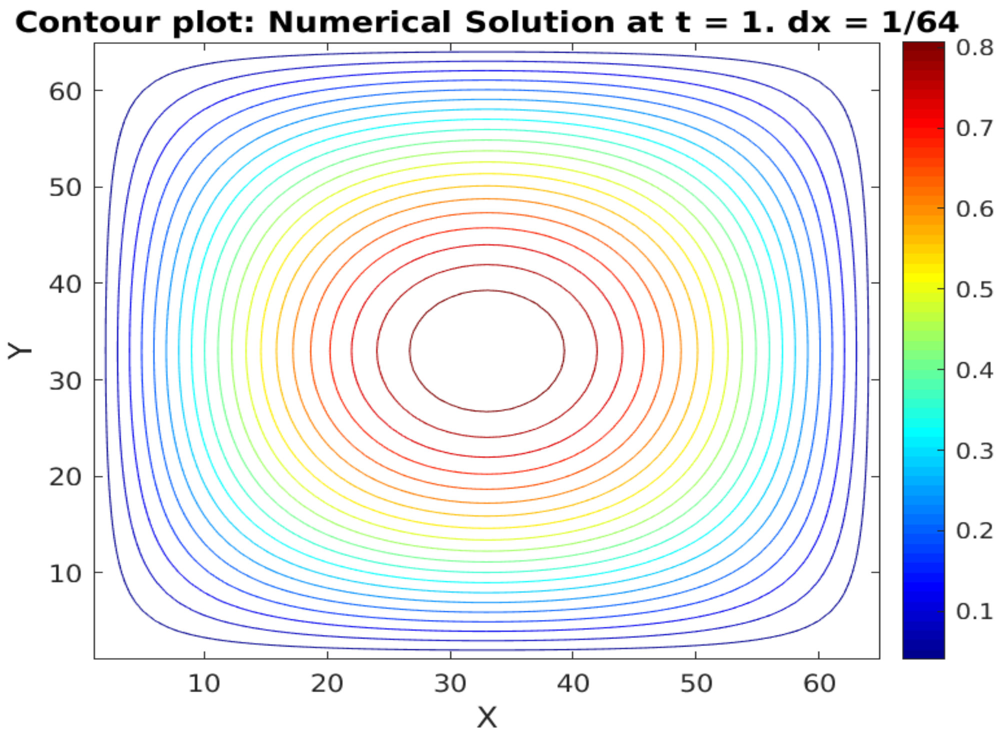

Figure 9: Numerical solution 2D plot

We can see that the numerical result is very close to the analytical one. At first glance, both solutions have the same shape. We can still very clearly the tip of the cone. Although, the colorbar shows that the maximum point at the tip is slightly off (0.9 for analytical vs 0.8 for numerical to 2 decimal places). Our original problem is a 2D heat conduction equation whose source term is controlled via a sinusoidal ($f(x, y, t) = e^{-t} sin(\pi x) sin(\pi y)(2\pi^2 - 1)$); therefore, the end result is also a sinusoidal, but because of superposition principle, we have a sinusoidal in 3D.

2.2 Error and Convergence Analysis

In other to investigate the error and convergence, we first need to plot to see how far away from the analytical solution is the numerical solution. To do that, we subtract the numerical from the analytical and then plot the error.

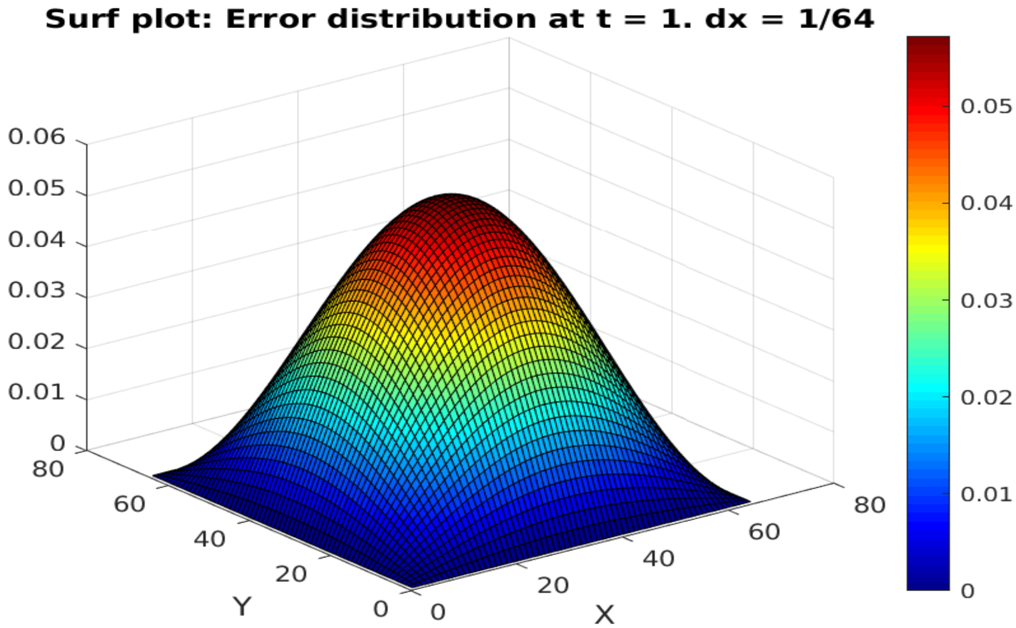

Figure 10: Error distribution 3D plot

We can see that the sides of the cone have very low error, between 0.01 to almost 0. The tips, although are the most different between the numerical and analytical, have errors of around 0.05. To see this in details, we look at the contour plot:

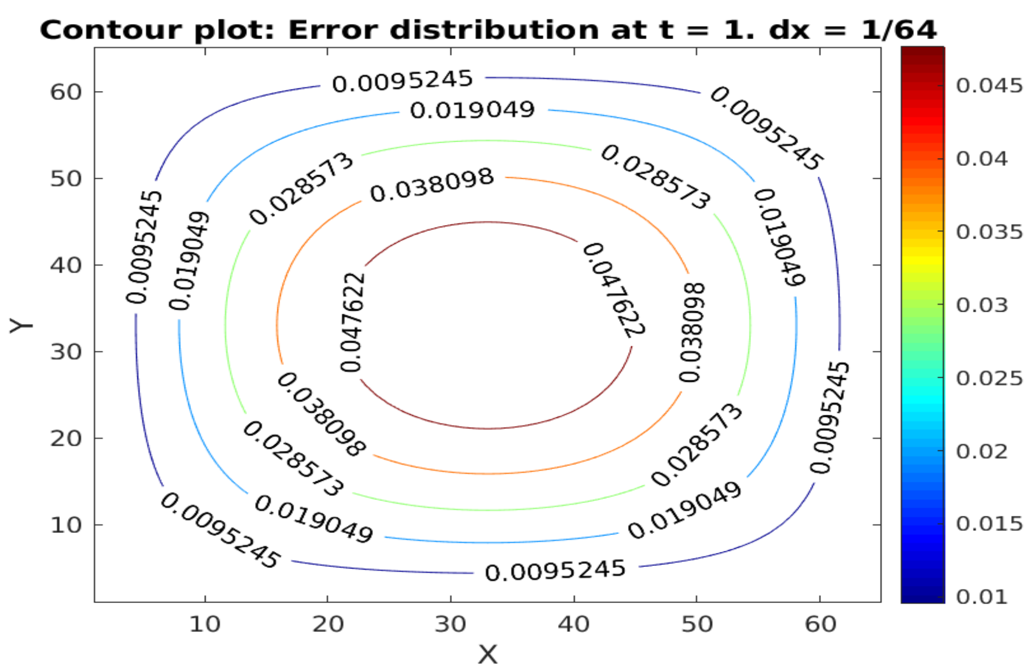

Figure 11: Error distribution 2D plot

The contour plot gives us a much better detailed view. We can see that the lowest error is around 0.009, on the outer shell, while the highest error is 0.047. If we try to plot the error with $dx = 1/128$, which is double the previous grid size, then:

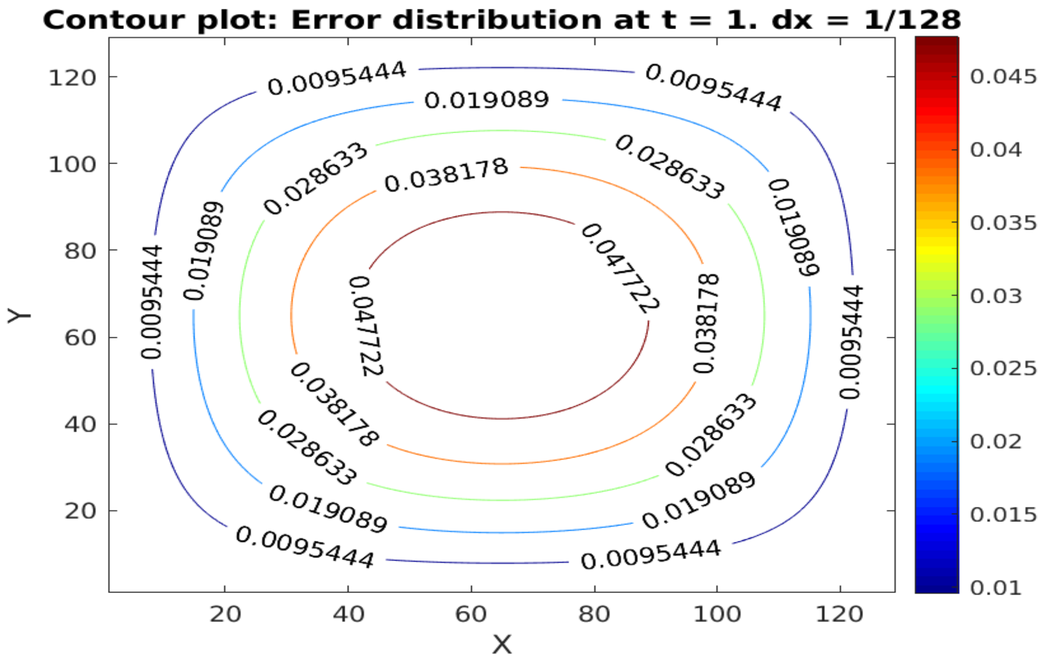

Figure 12: Error distribution 2D plot at dx = 1/128

Still, the errors are still small, the largest error is around 0.04. Therefore, half dx, which doubles the grid size, does not influence the error very much. If we try to increase the time, let the solution runs till a much later time, then we see this:

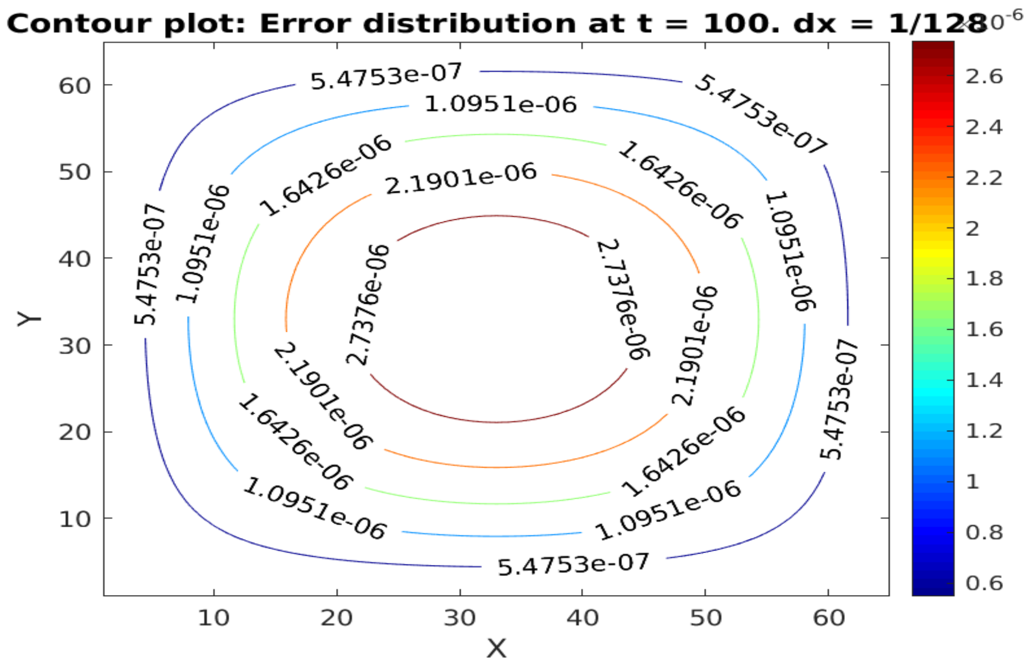

Figure 13: Error distribution 2D plot at dx = 1/128

The error drastically reduces to order of 10^{-6}. Therefore, at longer time, the solution does not get smeared or diverged from the analytical solution. In order to show this effectively, as part of the assignment, the following shows the plot between dx and the 2 norm of the error.

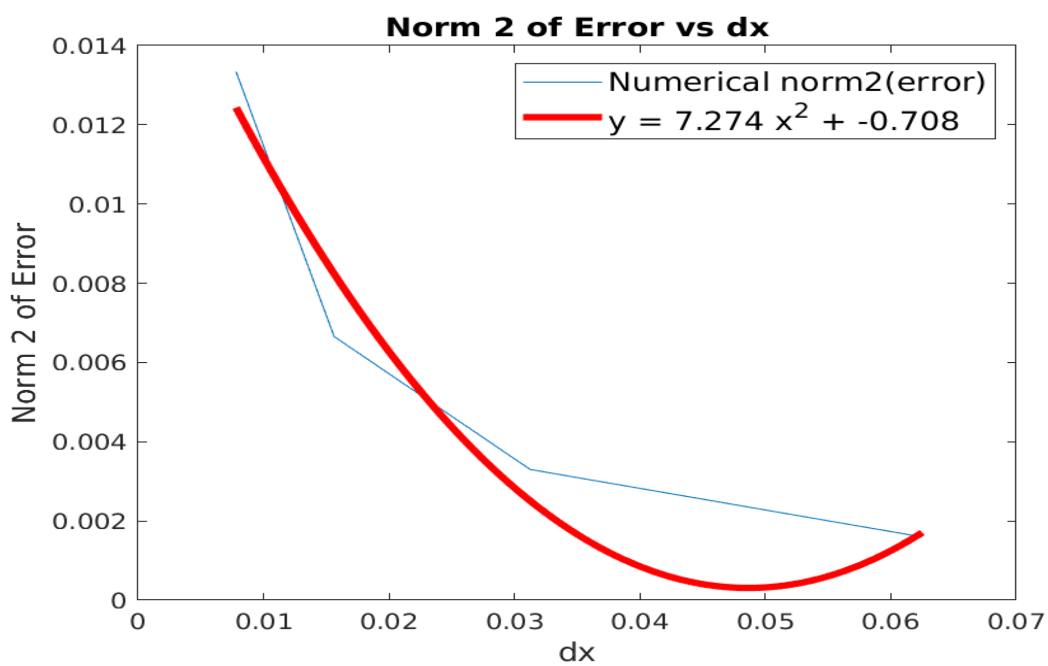

Figure 14: dx vs Norm 2 of Error

The figure is to be interpreted as follow: smaller dx means bigger grid size (very coarse grid), bigger dx means smaller grid size (fine grid). Therefore, we can see that as the grid size increases (dx gets smaller), our error goes up. The rate at which this error increases can be modeled by fitting a best fit curve. It turns out that a polynomial of 2nd order (x^2) best describes the plot. This makes sense because we use a 2nd order central difference to approximate the derivative, u_{xx}, u_{yy}, and this contains a truncation error of $O(\Delta x^2)$,$O(\Delta y^2)$. Therefore, our error increases with grid size. The 2-norm in this plot is calculated as follow:

$$||e||_2 = \left(\Delta x \Delta y \sum_{i}^{imax} \sum_{j}^{jmax} |e_{ij}|^2 \right)^{1/2}. \tag{11}$$

2.3 Stability Analysis

Recall Equation 7 and 9, where we derive the ADI scheme at n+1/2 and then n+1,denote δ_x^2 and δ_y^2 to be the second order central difference discretizations, we have the modified equations:

$$(1 - \frac{1}{2} D_x \delta_x^2) U_{ij}^{n+1/2} = (1 - \frac{1}{2} D_y \delta_y^2) U_{ij}^{n} \tag{12}$$

$$(1 - \frac{1}{2} D_y \delta_y^2) U_{ij}^{n+1} = (1 - \frac{1}{2} D_x \delta_x^2) U_{ij}^{n+1/2} \tag{13}$$

We want to apply the Fourier analysis: $F(\vec{u}) = \frac{1}{2\pi} \sum_{ij}^{\infty} e^{-Ij\xi - Ij\eta} U_{ij}$. Then for the second order central discretization part, we get:

$$\begin{aligned} F(\delta_x^2 U_{ij}^n) &= F(U_{i+1,j}^n + U_{i-1,j}^n - U_{i,j}^n) \\ &= \hat{U}^n \left[e^{I\xi} + e^{-I\xi} - 2 \right] \end{aligned}$$

Then, apply our two modified equations in similar way. For n to n+1/2:

$$\left(1 + 2D_x sin^2 \left(\frac{\xi}{2}\right)\right) \hat{U}^{n+1/2} = \left(1 - 2D_y sin^2 \left(\frac{\eta}{2}\right)\right) \hat{U}^n \tag{14}$$

This yields the amplification factor:

$$\begin{aligned} \rho_1(\xi, \eta) &= \frac{\hat{U}^{n+1/2}}{\hat{U}^n} \\ &= \frac{1 - 2D_y sin^2 \left(\frac{\eta}{2}\right)}{1 + 2D_x sin^2 \left(\frac{\xi}{2}\right)} \end{aligned}$$

Likewise, for n+1/2 to n+1:

$$\left(1 - 2D_x sin^2 \left(\frac{\xi}{2}\right)\right) \hat{U}^{n+1/2} = \left(1 + 2D_y sin^2 \left(\frac{\eta}{2}\right)\right) \hat{U}^{n+1} \tag{15}$$

This yields the amplification factor:

$$\begin{aligned}\rho_2(\xi,\eta) &= \frac{\hat{U}^{n+1}}{\hat{U}^{n+1/2}} \\ &= \frac{1-2D_x sin^2\left(\dfrac{\xi}{2}\right)}{1+2D_y sin^2\left(\dfrac{\eta}{2}\right)}\end{aligned}$$

Assuming the overall amplification factor is the multiplication of each individual factor:

$$\begin{aligned}\rho(\xi,\eta) &= \rho_1(\xi,\eta)\rho_2(\xi,\eta) \\ &= \left[\frac{1-2D_y sin^2\left(\dfrac{\eta}{2}\right)}{1+2D_x sin^2\left(\dfrac{\xi}{2}\right)}\right]\left[\frac{1-2D_x sin^2\left(\dfrac{\xi}{2}\right)}{1+2D_y sin^2\left(\dfrac{\eta}{2}\right)}\right]\end{aligned}$$

The terms: $2D_x sin^2\left(\dfrac{\xi}{2}\right)$ its similar variations (η) are always positive due to square. The denominator is always positive because two positive, square parts multiply together. The numerator, we have 1- a positive number, this makes it negative but when multiply by its other counterpart (the other numerator), they become positive. Mathematically, 1 - a square number will be less than 1 + a square number; therefore, the numerator is always less than the denominator: $\rightarrow \rho(\xi,\eta) \leq 1$. This shows that this ADI scheme is unconditionally stable

3 CONCLUSIONS

In conclusion, the ADI algorithm solves the parabolic PDE semi-implicitly. First, it solves the X-direction implicitly and Y-direction explicitly. After that, the solution at the half time step is used to solve X-direction explicitly and Y-direction implicitly. The numerical result shows very good similarity with the analytical one. Even if the simulation is left to run until large time such as 100, Figure 13, the numerical solution is still very close to the analytical solution. The error convergence plot shows that as grid size increases (dx gets smaller), the error follows a parabolic path, due to the local truncation error of O(Δx^2). Lastly, the Fourier analysis shows that this ADI scheme is in fact unconditionally stable for arbitrary D_x, D_y or arbitrary grid size and time step.

4 REFERENCE

1. MTH5315 Numerical Methods for PDE Lecture Notes. Jian,Du. Spring 2018
2. http://www4.ncsu.edu/ zhilin/TEACHING/MA584/MATLAB/ADI/adi.m. Web. May 5 2018
3. Tridiagonal Matrix Algorithm. Indian Institute of Space Science and Technology, Thiruvananthapuram. Web. May 5 2018

5 MATLAB CODE

```
%% MAIN CODE- BASIC PLOTTING

clear all
clc
clf

%basic properties
∆X = 1./64;
∆Y = ∆X;
dt = 0.1;
tmax = 1;
xmin = 0;
xmax = 1.;
imax = ((xmax-xmin)/∆X)+1;
jmax = imax;
alpha = 1.;

x = 0:∆X:1;
y = 0:∆Y:1;


%solution vectors
u = zeros(imax,jmax);
udumb = zeros(imax,jmax);
dx = dt/∆X^2; %diffusion number in X
dy = dx; %diffusion number in Y
alpha = 1;


%% THOMAS CONSTANTS X
%define Thomas vectors

ax = zeros(imax,1); %above
bx = zeros(imax,1); %below
cx = zeros(imax,1); %rhs
diagonalX = zeros(imax,1); %diagonal

for i = 1:imax
    ax(i) = -dx/2;
    bx(i) = -dx/2;
    diagonalX(i) = (1+dx);
end

%% THOMAS CONSTANTS Y
%define Thomas vectors

ay = zeros(imax,1); %above
by = zeros(imax,1); %below
cy = zeros(imax,1); %rhs
diagonalY = zeros(imax,1); %diagonal
```

```


%fill out thomas vectors

for i = 1:imax
    ay(i) = -dx/2;
    by(i) = -dx/2;
    diagonalY(i) = (1+dx);
end


%% INPUT INITIAL CONDITION


for i = 1:imax
    for j = 1:jmax
        u(i,j) = initial(x(i),y(j));
    end
end

% contour(u)
udumb(:,:) = u(:,:);

u_analytical = zeros(imax,jmax);

%Begin ADI

for t = 1:tmax
    %ANALYTICAL SOLN
    for i = 1:imax
        for j = 1:jmax
            u_analytical(i,j) = uexact(t*dt,x(i),y(j));
        end
    end
    % X sweep
    for j = 2:jmax-1
        for i = 2:imax-1

            cx(i) = (1-dx)*u(i,j)+(dx/2)*(u(i,j+1)) + (dx/2)*(u(i,j-1)) ...
                + (dt/2)*source(t*dt,x(i),y(j));
        end
        udumb(:,j) = thomas(ax,diagonalX,cx);
    end

    udumb2 = udumb;

    %Y sweep
    for i = 2:imax-1
        for j = 2:jmax-1

            cy(j) = (1-dx)*udumb2(i,j)+(dx/2)*(udumb2(i+1,j)) + ...
                (dx/2)*(udumb2(i-1,j)) + ...
                (dt/2)*source((t+0.5)*dt,x(i),y(j));
        end
```

```
        udumb(i,:) = thomas(ay,diagonalY,cy);
    end

    u = udumb;

    err = u - u_analytical.';
end

figure(1)
surf(u)
colormap jet
colorbar
title('Surf plot: Numerical Solution at t = 1. dx = 1/64', 'FontSize',12)
ylabel('Y','FontSize',12)
xlabel('X','FontSize',12)
saveas(figure(1),'numsurf.png')

figure(2)
contour(u,20)
colormap jet
colorbar
title('Contour plot: Numerical Solution at t = 1. dx = 1/64', ...
    'FontSize',12)
ylabel('Y','FontSize',12)
xlabel('X','FontSize',12)
saveas(figure(2),'numcontour.png')


figure(3)
surf(u_analytical)
colormap jet
colorbar
title('Surf plot: Exact Solution at t = 1. dx = 1/64', 'FontSize',12)
ylabel('Y','FontSize',12)
xlabel('X','FontSize',12)
saveas(figure(3),'exactsurf.png')


figure(4)
contour(u_analytical,20)
colormap jet
colorbar
title('Contour plot: Exact Solution at t = 1. dx = 1/64', 'FontSize',12)
ylabel('Y','FontSize',12)
xlabel('X','FontSize',12)
saveas(figure(4),'exactcontour.png')


figure(5)
contour(abs(err),5,'ShowText','on')
colormap jet
colorbar
title('Contour plot: Error distribution at t = 1. dx = 1/64', ...
    'FontSize',12)
```

```
ylabel('Y','FontSize',12)
xlabel('X','FontSize',12)
saveas(figure(5),'contour_error.png')

figure(6)
surf(abs(err))
colormap jet
colorbar
title('Surf plot: Error distribution at t = 1. dx = 1/64', 'FontSize',12)
ylabel('Y','FontSize',12)
xlabel('X','FontSize',12)
saveas(figure(6),'surf_error.png')









%% CONVERGENCE ANALYSIS PLOT

%{


clear all
clc


grid_vector = [1./16 1./32 1./64 1./128];
imax_vector = (1./grid_vector)+1;
l = length(grid_vector);
err_vector = zeros(1,l);
sum_store = zeros(1,l);


%% MAIN CODE
for k = 1:l

    %basic properties
    ΔX = grid_vector(k);
    ΔY = ΔX;
    dt = 0.1;
    tmax = 10;
    xmin = 0;
    xmax = 1.;
    imax = ((xmax-xmin)/ΔX)+1;
%     imax = 65;
    jmax = imax;
    alpha = 1.;

    x = 0:ΔX:1;
    y = 0:ΔY:1;
```

```
    %solution vectors
    u = zeros(imax,jmax);
    udumb = zeros(imax,jmax);
    dx = dt/ΔX^2; %diffusion number in X
    dy = dx; %diffusion number in Y
    alpha = 1;


    %% THOMAS CONSTANTS X
    %define Thomas vectors

    ax = zeros(imax,1); %above
    bx = zeros(imax,1); %below
    cx = zeros(imax,1); %rhs
    diagonalX = zeros(imax,1); %diagonal

    %fill out thomas vectors

    for i = 1:imax
        ax(i) = -dx/2;
        bx(i) = -dx/2;
        diagonalX(i) = (1+dx);
    end



    %% THOMAS CONSTANTS Y
    %define Thomas vectors

    ay = zeros(imax,1); %above
    by = zeros(imax,1); %below
    cy = zeros(imax,1); %rhs
    diagonalY = zeros(imax,1); %diagonal


    %fill out thomas vectors

    for i = 1:imax
        ay(i) = -dx/2;
        by(i) = -dx/2;
        diagonalY(i) = (1+dx);
    end



    %% INPUT INITIAL CONDITION


    for i = 1:imax
        for j = 1:jmax
            u(i,j) = initial(x(i),y(j));
        end
```

```
    end

    % contour(u)
    udumb(:,:) = u(:,:);

    u_analytical = zeros(imax,jmax);

    %% Begin ADI

    for t = 1:tmax

        %ANALYTICAL SOLN

        for i = 1:imax
            for j = 1:jmax
                u_analytical(i,j) = uexact(t*dt,x(i),y(j));
            end
        end


        % X sweep
        for j = 2:jmax-1
            for i = 2:imax-1

                cx(i) = (1-dx)*u(i,j)+(dx/2)*(u(i,j+1)) + ...
                    (dx/2)*(u(i,j-1)) + (dt/2)*source(t*dt,x(i),y(j));
            end
            udumb(:,j) = thomas(ax,diagonalX,cx);
        end

        udumb2 = udumb;

        %Y sweep
        for i = 2:imax-1
            for j = 2:jmax-1

                cy(j) = (1-dx)*udumb2(i,j)+(dx/2)*(udumb2(i+1,j)) + ...
                    (dx/2)*(udumb2(i-1,j)) + ...
                    (dt/2)*source((t+0.5)*dt,x(i),y(j));
            end
            udumb(i,:) = thomas(ay,diagonalY,cy);
        end

        u = udumb;

        err = u - u_analytical.';

        err_square = err.^2;

        %SUM DOWN ALL COLUMN.
        for i = 1:imax
            sumcol = sum(err_square);
        end

```

```
        %SUM ACROSS ALL ROW
        sumrow = sum(sumcol);
        sum_store(k) = sumrow;

    end

end

norm2_grid_vector = (ΔX*ΔY*sum_store).^(0.5);

%% PLOTTING NORM(ERROR) VS GRID SIZE
figure(1)
plot(grid_vector,norm2_grid_vector);
title('Norm 2 of Error vs dx','FontSize',12)
xlabel('dx','FontSize',12)
ylabel('Norm 2 of Error','FontSize',12)
hold on

coeffs = polyfit(grid_vector, norm2_grid_vector, 2);
fittedX = linspace(min(grid_vector), max(grid_vector),200);
fittedY = polyval(coeffs, fittedX);
hold on
plot(fittedX, fittedY, 'r-', 'LineWidth', 3);

theString = sprintf('y = %.3f x^{2} + %.3f', coeffs(1), coeffs(2));
leg = legend('Numerical norm2(error)',theString);
leg.FontSize = 12;
% saveas(figure(1),'convergence.png')

%}
```

```
%% FUNCTIONS



function exact = uexact(t,x,y)
  exact = exp(-t).*sin(pi.*x).*sin(pi.*y);
end


function f = source(t,x,y)
    f = exp(-t)*sin(pi.*x).*sin(pi.*y)*(2*(pi^2)-1);
end


function f0 = initial(x,y)
    f0 = sin(pi.*x).*sin(pi.*y);
end



function z = thomas(a,dia,f)

    n = length(dia);
    z = zeros(n,1);
    p = zeros(n,1);
    q = zeros(n,1);

    q(2) = f(2)./dia(2);
    p(2) = a(2)./(dia(2));

    for i = 3:n-1
        denom = dia(i)-a(i)*p(i-1);
        p(i) = a(i)./(denom);
        q(i) = (f(i)-a(i)*q(i-1))./denom;
    end


    z(n) = q(n);
    for i = n-1:-1:1
        z(i) = -p(i)*z(i+1)+q(i);
    end


end
```