Project for the course Applied Numerical Methods (ME685) on

Comparative Analysis of ADI and Explicit Finite Difference Method for a 2-D Block Problem

Project Report submitted by,

Devyansh Agarhari   231010021

Under the guidance of

Dr. Abhishek Sarkar

DEPARTMENT OF AEROSPACE ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

April 2024

Abstract

This study presents a comparative analysis of the ADI and Explicit scheme of Finite Difference Method in the context of heat transfer problems. The aim of the project is to evaluate the effectiveness and suitability of both methods using different numerical techniques. The implementation of numerical techniques in both FDM technique is explored through specific numerical techniques employed, such as interpolation methods, integration techniques, or differentiation methods. The implications of these findings are discussed in relation to heat transfer problems. The effectiveness of the implemented numerical techniques is evaluated through extensive simulations and comparisons. Results demonstrate (key findings related to accuracy, convergence rate, computational efficiency, etc.).

Introduction:

In the domain of computational modelling and simulation, the Finite Difference Method (FDM) scheme stand as a pillars, offering versatile approaches to solving complex mathematical problems across various disciplines. The method schemes have been extensively employed in engineering, physics, finance, and numerous other fields to tackle a wide range of challenges, from heat transfer to fluid dynamics and beyond.

The effectiveness of FDM schemes lies not only in their ability to approximate solutions to differential equations but also in their adaptability to different types of problems and geometries. However, despite their widespread use, both methods are not without their limitations. Achieving optimal accuracy, convergence, and computational efficiency often requires careful consideration of various factors, including mesh refinement, boundary conditions, and numerical stability.

The primary objectives of this study are twofold: firstly, to investigate the efficacy of different numerical techniques when applied within the contexts of FDM, and secondly, to assess their impact on the accuracy, convergence rate, and computational efficiency of the respective methods. Through a series of simulations and comparisons, I aim to identify optimal strategies for implementing numerical techniques in FDM, with implications for a wide range of engineering and scientific applications.

Understanding the thermal behavior of materials is crucial in various engineering applications. Numerical simulations provide a cost-effective means to analyze and predict temperature distributions under different conditions. The ADI method is particularly advantageous for solving parabolic partial differential equations (PDEs) like the heat equation, offering unconditional stability and efficiency in two-dimensional simulations and it gives less error for longer iteration of time step. And here I have also simulated using explicit scheme technique, explicit scheme stability condition is also defined and it has higher truncation error as compared to implicit scheme and it gives higher error for longer time steps.

By bridging the gap between theory and practice, this report contributes to the ongoing efforts to advance computational methodologies and empower researchers and practitioners with more robust tools for solving challenging problems. Through our exploration of numerical techniques within FDM, I hope to inspire further innovation and collaboration in the field of computational modelling and simulation.

Objective:

The primary objective of this study is to simulate the temperature distribution within a square block over discrete time steps, incorporating distinct boundary conditions and employing the ADI technique and explicit technique. And visualization of iso-lines of temperature and filled contour plot within the block. Plots will be utilized to analyse the temperature distribution of the block under varying time step and grid size for both method.

Problem Statement:

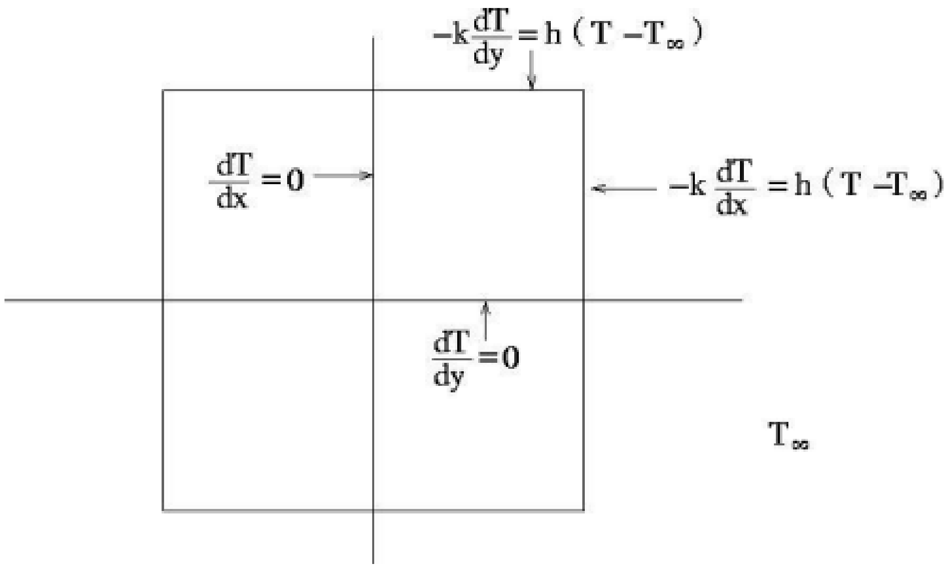
Consider a 2D square block with dimensions L x L in x-y Cartesian coordinate, where *L* represents the side length. The block's temperature evolves over time *t* due to internal heat conduction and heat transfer at the boundaries. Here I have assumed the 2D surface is a cross-section in x-y plane perpendicular to z axis.

Assumptions of this problem statement:

- In z-direction block is of infinite length and there is no variation in temperature in that direction.
- There is no variation of temperature in z direction so there is zero heat flux in and out of plane, considering it to be an adiabatic surface.
- Material's thermo physical properties are constant.
- There is volumetric heat generation.
- Ambient is at constant temperature.
- I aim to visualize the temperature distribution within the block.

Mathematical model:

I considered the heat transfer equation to describe the temperature distribution:



$-k\frac{dT}{dy} = h(T - T_\infty)$
$\frac{dT}{dx} = 0$
$-k\frac{dT}{dx} = h(T - T_\infty)$
$\frac{dT}{dy} = 0$
T_∞

Where:

***T* is the temperature.**

***ρ* is the material density.**

***c* is the specific heat capacity.**

***k* is the thermal conductivity.**

Boundary Conditions:

I impose specific boundary conditions:

- Left and Bottom Boundaries: Zero flux (adiabatic condition).
- Top and Right Boundaries: Convective heat transfer (Newton's law of cooling).

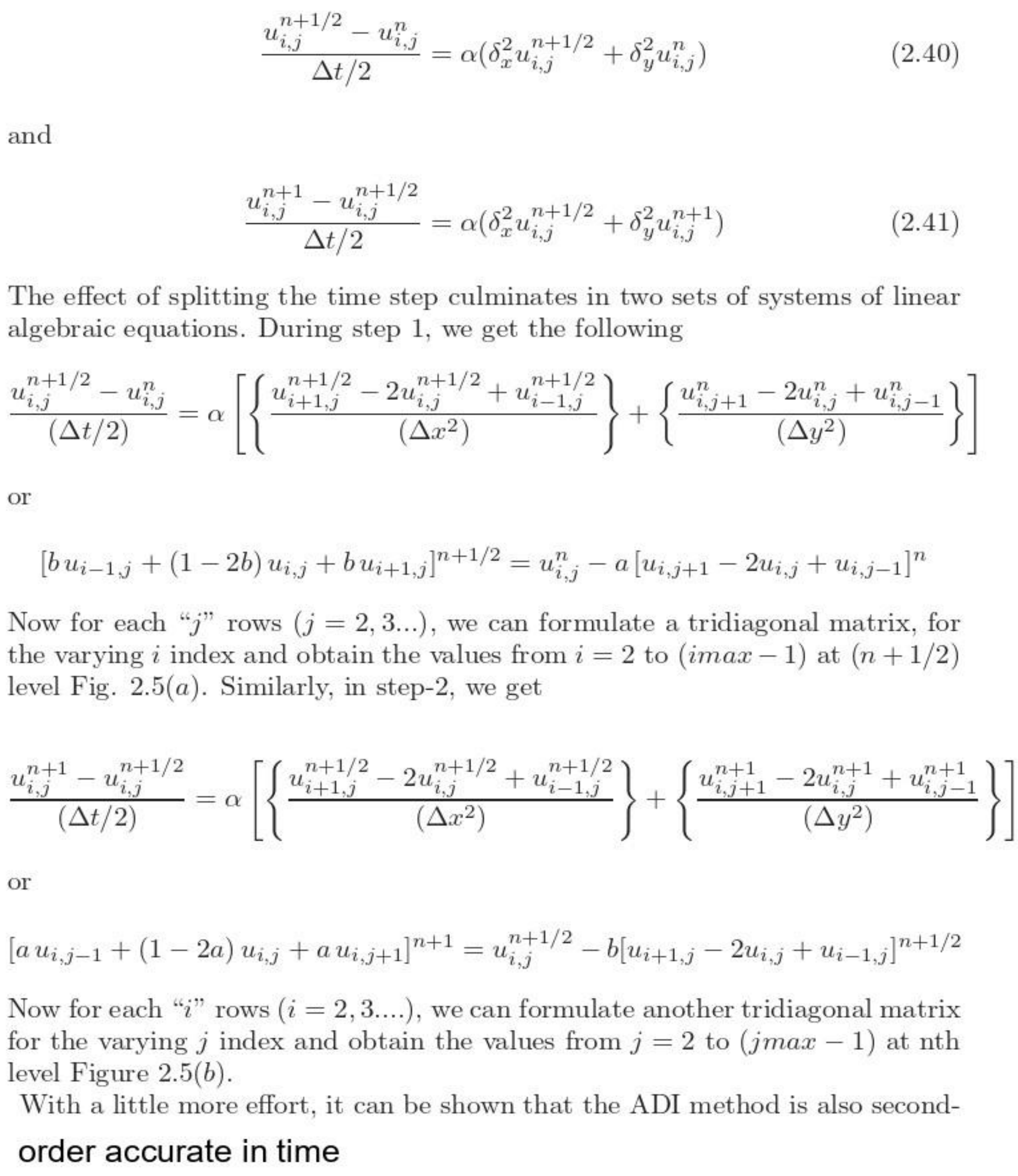
Numerical Approach:

ADI Technique

The ADI method allows us to split the 2D problem into an algebraic problem, solving them alternately. It provides numerical stability and accuracy.

** Below I have added the handout explaining complete process of applying the ADI Method with TMDA solving procedure using line by line method.**

Here temperature is represented as u.



$$\frac{u_{i,j}^{n+1/2} - u_{i,j}^{n}}{\Delta t/2} = \alpha(\delta_x^2 u_{i,j}^{n+1/2} + \delta_y^2 u_{i,j}^{n}) \tag{2.40}$$

and

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n+1/2}}{\Delta t/2} = \alpha(\delta_x^2 u_{i,j}^{n+1/2} + \delta_y^2 u_{i,j}^{n+1}) \tag{2.41}$$

The effect of splitting the time step culminates in two sets of systems of linear algebraic equations. During step 1, we get the following

$$\frac{u_{i,j}^{n+1/2} - u_{i,j}^{n}}{(\Delta t/2)} = \alpha\left[\left\{\frac{u_{i+1,j}^{n+1/2} - 2u_{i,j}^{n+1/2} + u_{i-1,j}^{n+1/2}}{(\Delta x^2)}\right\} + \left\{\frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{(\Delta y^2)}\right\}\right]$$

or

$$[b\,u_{i-1,j} + (1-2b)\,u_{i,j} + b\,u_{i+1,j}]^{n+1/2} = u_{i,j}^{n} - a\,[u_{i,j+1} - 2u_{i,j} + u_{i,j-1}]^{n}$$

Now for each "j" rows ($j = 2, 3...$), we can formulate a tridiagonal matrix, for the varying i index and obtain the values from $i = 2$ to $(imax - 1)$ at $(n + 1/2)$ level Fig. 2.5(a). Similarly, in step-2, we get

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n+1/2}}{(\Delta t/2)} = \alpha\left[\left\{\frac{u_{i+1,j}^{n+1/2} - 2u_{i,j}^{n+1/2} + u_{i-1,j}^{n+1/2}}{(\Delta x^2)}\right\} + \left\{\frac{u_{i,j+1}^{n+1} - 2u_{i,j}^{n+1} + u_{i,j-1}^{n+1}}{(\Delta y^2)}\right\}\right]$$

or

$$[a\,u_{i,j-1} + (1-2a)\,u_{i,j} + a\,u_{i,j+1}]^{n+1} = u_{i,j}^{n+1/2} - b[u_{i+1,j} - 2u_{i,j} + u_{i-1,j}]^{n+1/2}$$

Now for each "i" rows ($i = 2, 3....$), we can formulate another tridiagonal matrix for the varying j index and obtain the values from $j = 2$ to $(jmax - 1)$ at nth level Figure 2.5(b).

With a little more effort, it can be shown that the ADI method is also second-

order accurate in time

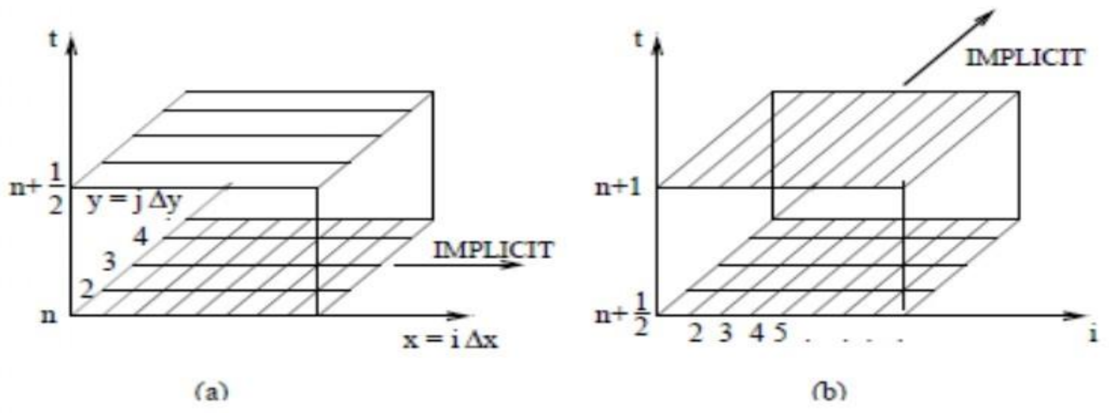


Figure 2.5: Schematic representation of ADI scheme.

Explicit method Technique:

****Handout explaining explicit method discretization.****

The two-dimensional conduction equation is given by

$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \tag{2.34}$$

Here, the dependent variable, u (temperature) is a function of space (x, y) and time (t) and α is the thermal diffusivity. If we apply the simple explicit method to the heat conduction equation, the following algorithm results

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = \alpha \left[\frac{u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{(\Delta x^2)} + \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{(\Delta y^2)} \right] \tag{2.35}$$

Error and Stability Condition:

Round-off:

This is the numerical error introduced for a repetitive number of calculations in which the computer is constantly rounding the number to some decimal points. If A = analytical solution of the partial differential equation.
D = exact solution of the finite-difference equation
N = numerical solution from a real computer with finite accuracy
Then, Discretization error = A - D = Truncation error + error introduced due to treatment of boundary condition
Round-off error = ϵ = N - D
or,

$$N = D + \epsilon \tag{2.42}$$

where, ϵ is the round-off error, which henceforth will be called "error" for convenience. The numerical solution N must satisfy the finite difference equation. Hence from Eq. (2.26)

$$\frac{D_i^{n+1} + \epsilon_i^{n+1} - D_i^n - \epsilon_i^n}{\Delta t} = \alpha \left[\frac{D_{i+1}^n + \epsilon_{i+1}^n - 2D_i^n - 2\epsilon_i^n + D_{i-1}^n + \epsilon_{i-1}^n}{(\Delta x^2)} \right] \tag{2.43}$$

By definition, D is the exact solution of the finite difference equation, hence it exactly satisfies

$$\frac{D_i^{n+1} - D_i^n}{\Delta t} = \alpha \left[\frac{D_{i+1}^n - 2D_i^n + D_{i-1}^n}{(\Delta x^2)} \right] \tag{2.44}$$

Subtracting Eq. (2.44) from Eq. (2.43)

$$\frac{\epsilon_i^{n+1} - \epsilon_i^n}{\Delta t} = \alpha \left[\frac{\epsilon_{i+1}^n - 2\epsilon_i^n + \epsilon_{i-1}^n}{(\Delta x^2)} \right] \tag{2.45}$$

From Equation 2.45, we see that the error ϵ also satisfies the difference equation.

If errors ϵ_i are already present at some stage of the solution of this equation, then the solution will be stable if the ϵ_i's shrink, or at least stay the same, as the solution progresses in the marching direction, i.e from step n to $n+1$. If the ϵ_i's grow larger during the progression of the solution from step n to $n+1$, then the solution is unstable. Finally, it stands to reason that for a solution to be stable, the mandatory condition is

$$\left| \frac{\epsilon_i^{n+1}}{\epsilon_i^n} \right| \leq 1 \tag{2.46}$$

For Eq. (2.26), let us examine under what circumstances Eq. (2.46) holds good.

Assume that the distribution of errors along the $x-$axis is given by a Fourier series in x, and the time-wise distribution is exponential in t, i.e,

$$\epsilon(x,t) = e^{at} \sum_m e^{Ik_m x} \tag{2.47}$$

where I is the unit complex number and k the wave number [1]Since the difference is linear, when Eq. (2.47) is substituted into Eq. (2.45), the behaviour of each term of the series is the same as the series itself. Hence, let us deal with just one term of the series, and write

$$\epsilon_m(x,t) = e^{at}e^{Ik_m x} \tag{2.48}$$

Substitute Eq. (2.48) into (2.45) to get

$$\frac{e^{a(t+\Delta t)}e^{Ik_m x} - e^{at}e^{Ik_m x}}{\Delta t} = \alpha \left[\frac{e^{at}e^{Ik_m(x+\Delta x)} - 2e^{at}e^{Ik_m x} + e^{at}e^{Ik_m(x-\Delta x)}}{(\Delta x)^2}\right] \tag{2.49}$$

Divide Eq. (2.49) by $e^{at}e^{Ik_m x}$

$$\frac{e^{a\Delta t} - 1}{\Delta t} = \alpha \left[\frac{e^{Ik_m\Delta x} - 2 + e^{-Ik_m\Delta x}}{(\Delta x)^2}\right]$$

or,

$$e^{a\Delta t} = 1 + \frac{\alpha(\Delta t)}{(\Delta x)^2}\left(e^{Ik_m\Delta x} + e^{-Ik_m\Delta x} - 2\right) \tag{2.50}$$

Recalling the identity

$$\cos(k_m\Delta x) = \frac{e^{Ik_m\Delta x} + e^{-Ik_m\Delta x}}{2}$$

Eq. (2.50) can be written as

$$e^{a\Delta t} = 1 + \frac{\alpha(2\Delta t)}{(\Delta x)^2}(\cos(k_m\Delta x) - 1)$$

or,

$$e^{a\Delta t} = 1 - 4\frac{\alpha(\Delta t)}{(\Delta x)^2}\sin^2[(k_m\Delta x)/2] \tag{2.51}$$

From Eq. (2.48), we can write

$$\frac{\epsilon_i^{n+1}}{\epsilon_i^n} = \frac{e^{a(t+\Delta t)}e^{Ik_m x}}{e^{at}e^{Ik_m x}} = e^{a\Delta t} \tag{2.52}$$

Combining Eqns. (2.51),(2.52) and (2.46), we have

$$\left|\frac{\epsilon_i^{n+1}}{\epsilon_i^n}\right| = |e^{a\Delta t}| = \left|1 - 4\frac{\alpha(\Delta t)}{(\Delta x)^2}\sin^2\left[\frac{(k_m\Delta x)}{2}\right]\right| \le 1 \tag{2.53}$$

[1]Let a wave travel with a velocity v. The time period "T'' is the time required for the wave to travel a distance of one wave length λ, so that λ=vT. Wave number k is defined by $k = 2\pi/\lambda$.

Eq. (2.53) must be satisfied to have a stable solution. In Eq. (2.53) the factor

$$\left|1-4\frac{\alpha(\Delta t)}{(\Delta x)^2}\sin^2\left[\frac{(k_m\Delta x)}{2}\right]\right|$$

is called the amplification factor and is denoted by G.

Evaluating the inequality in Eq. (2.53) , the two possible situations which must hold simultaneously are
(a)

$$1-4\frac{\alpha(\Delta t)}{(\Delta x)^2}\sin^2\left[\frac{(k_m\Delta x)}{2}\right]\leq 1$$

Thus,

$$4\frac{\alpha(\Delta t)}{(\Delta x)^2}\sin^2\left[\frac{(k_m\Delta x)}{2}\right]\geq 0$$

Since $\alpha(\Delta t)/(\Delta x)^2$ is always positive, this condition always holds.
(b)

$$1-4\frac{\alpha(\Delta t)}{(\Delta x)^2}\sin^2\left[\frac{(k_m\Delta x)}{2}\right]\geq -1$$

Thus,

$$\frac{4\alpha(\Delta t)}{(\Delta x)^2}\sin^2\left[\frac{(k_m\Delta x)}{2}\right]-1\leq 1$$

For the above condition to hold

$$\frac{\alpha(\Delta t)}{(\Delta x)^2}\leq\frac{1}{2} \tag{2.54}$$

Eq. (2.54) gives the stability requirement for which the solution of the difference Eq. (2.26) will be stable. It can be said that for a given Δx the allowed value of Δt must be small enough to satisfy Eq. (2.54). For $\alpha(\Delta t)/(\Delta x)^2 \leq (1/2)$ the error will not grow in subsequent time marching steps in t, and the numerical solution will proceed in a stable manner. On the contrary, if $\alpha(\Delta t)/(\Delta x)^2 > (1/2)$, then the error will progressively become larger and the calculation will be useless.

The above mentioned analysis using Fourier series is called as the Von Neumann stability analysis.

Alternating Direction Implicit (ADI) method:

The **Alternating Direction Implicit (ADI) method** is a powerful numerical technique for solving **parabolic partial differential equations (PDEs)**. It combines implicit and explicit finite difference schemes to achieve stability and accuracy. Let's explore the stability conditions for ADI methods:

ADI Method Overview:

The ADI method splits the PDE into two sub problems, each solved implicitly in one direction and explicitly in the other. It is widely used for solving time-dependent PDEs with interfaces, such as parabolic problems involving variable coefficients.

Stability of ADI Methods:

The stability of ADI methods depends on the specific problem and discretization. For parabolic problems without interfaces, the ADI method is **unconditionally stable**. This means that a large time increment can be used without compromising accuracy. However, when interfaces or variable coefficients are present, the stability conditions become more nuanced.

Stability Conditions:

The stability condition for the ADI method involves the time step size (Δt) and the spatial grid spacing (Δx).

In general, the ADI method is **conditionally stable**. The stability condition is given by:

$\Delta t \leq 0.5\Delta x^2$

If this condition is satisfied, the ADI method ensures stability during time advancement.

Practical Considerations:

While the ADI method provides stability, it's essential to choose an appropriate time step to maintain accuracy. Numerical experiments and stability analysis help determine suitable parameters for specific problems. In summary, the ADI method strikes a balance between stability and efficiency, making it a valuable tool for solving parabolic PDEs with variable coefficients and interfaces.

Explicit Finite Difference Method:

This method is commonly used for solving partial differential equations (PDEs) numerically. Specifically, we'll focus on the stability condition related to the **CFL (Courant-Friedrichs-Lewy) number**.

Explicit Finite Difference Method:

In the explicit scheme, I approximated the time derivative explicitly, which means that updating the solution at each time step based on the values at the previous time step. It's straightforward to implement but has limitations regarding stability.

Stability Condition:

The stability of the explicit finite difference method depends on the choice of time step (Δt) and spatial grid spacing (Δx).

The key parameter is the CFL number, defined as:

$CFL = \Delta t / \Delta x^2$

The explicit method is stable if the CFL number satisfies:

CFL ≤ 0.5

Interpretation:

The stability condition ensures that the numerical errors introduced at each time step do not grow uncontrollably. If the CFL number exceeds 0.5, the algorithm becomes **unstable**, leading to oscillations and unreliable results.

Practical Considerations:

To maintain stability, choose a time step (Δt) such that the CFL condition is met. Smaller time steps improve accuracy but increase computational cost. In summary, when using the explicit finite difference method, adhere to the CFL stability condition to ensure reliable and accurate solutions for time-dependent problems.

MATLAB Implementation:

I developed the MATLAB code to implement the ADI method and explicit method for solving the discretized heat equation. The code incorporates boundary conditions, discretization schemes, and iterative procedures to compute the temperature distribution at each time step.

Key steps include:

Geometry and Meshing: Define the square block geometry and create a mesh.

```
L = 8*10^(-2); % Side length of a square block

N =26; % Number of grid points in x & y-direction

t = 60; % No. of time steps

dx = L / (N - 1); % grid spacing in x direction of square mesh

dy = L / (N - 1); % grid spacing in y direction of square mesh
```

Material Properties: Specify thermal conductivity (*k*), thermal convection coefficient (h), and thermal diffusivity (alpha).

```
alpha = 1.6*10^(-5); % Thermal diffusivity

h = 400; % Convective heat transfer coefficient

k= 61; % Conductive heat transfer coefficient
```

Method's Discretization:

The 2D heat equation is discretized in both space and time using finite difference methods, allowing us to approximate the temperature distribution at discrete points within the block after certain interval of time.

ADI Technique: The ADI method involves splitting the 2D heat equation into two separate algebraic equations, each representing a half time step along one direction. This splitting allows for efficient computation and stability in solving the heat equation.

Explicit Technique: The explicit scheme involves splitting the 2D heat equation using forward in time and central in space method. And I will satisfy stability condition for convergence of solution.

****Both method's code is explained in solver section****

Boundary Conditions: Set temperatures or heat fluxes at the boundaries.

The left and bottom boundaries are set to zero flux.

The top and right boundaries are subject to convective heat transfer conditions.

****Both method's boundary condition code is explained in solver section****

Initial Conditions: Assign an initial temperature distribution.

```
Tbody_initially = 100+273.15; % Initial temperature of block

T = ones(Nx, Ny)*Tbody_initially; % Initial temperature distribution
```

Stability Conditions:

For ADI method- It is unconditionally stable.

For Explicit method- It requires to satisfy the stability condition for convergence to solution.

****Explicit method stability condition is incorporated in the main code and explained in solver section****

Solver:

Using the ADI method to solve the transient heat transfer problem:

```
n = N-1;

hx = x/(2*n); % Constant as per notes

delta_t = t/htm; % Time step size

T = ones(N)*Tbody; % Temperature distribution matrix

T1step = T; % Temperature distribution matrix for step 1 of ADI method

T2stp = T; % Temperature distribution matrix for step 2 of ADI method
```

```
%Solve for each time step
for m = 1:delta_t
        rx = (Alpa*htm)/(hx^2); % Constant as per notes
        a= -1*rx/2; % Constant as per notes
        b=1+rx; % Constant as per notes
        A=zeros(N); % Defined matrix A of Ax = B for TDMA
        A(N,N-1)=-1*k/hx; % Value assigned as per boundary condition at top surface
        A(N,N)=h+k/hx; % Value assigned as per boundary condition at top surface
        A(1,1)=-1; % Value assigned as per boundary condition at bottom surface
        A(1,2)=1; % Value assigned as per boundary condition at bottom surface
        for i = 2:N-1
                A(i, i-1)= a; % Value assigned as per ADI method
                A(i, i)= b; % Value assigned as per ADI method
                A(i, i+1)= a; % Value assigned as per ADI method
        end
        [A_inv] = Inverse_using_thomas_alg (A);
        for j=2:N-1
                da = zeros(N, 1); % Defined B matrix
                da(1) = 0; % Value assigned as per boundary condition on bottom surface
                da(N) = h*Tsurrounding; % Value assigned as per b.c on top surface
                for i=2:N-1
                        da(i)=T(j,i)+(rx/2)*(T(j+1,i)-2*T(j,i)+T(j-1,i)); % Value assigned as
per ADI method
                end
                Tj=A_inv*da; % Solving for ½ time step
                for i = 1:N
                        T1step(j,i) = Tj(i);
                end
                for i = 1:N
                        T1step(1,i) = T1step(2,i); % Applying left B.C
                T1step(N,i) = (k*T1step(N-1,i)+h*hx*Tsurrounding)/(h*hx+k); %
Applying right B.C
            end
        end
        % Repeating the same thing as above for another half time step
        for j=2:N-1
            db = zeros(N, 1);
            da(1) = 0;
            db(N) = h*Tsurrounding;
            for i=2:N-1
                db(i)=T1step(i,j)+(rx/2)*(T1step(i, j+1)-2*T1step(i,j)+T1step(i,j-1));
            end
            T_i=A_inv*db;
            for i = 1:N
                T2stp(i,j) = T_i(i);
            end
            for i = 1:N
                T2stp(i,1) = T2stp(i,2);
                T2stp(i,N) = (k*T2stp(i,N-1)+h*hx*Tsurrounding)/(h*hx+k);
            end
        end
        T=T2stp;
end
```

Using the explicit method to solve the transient heat transfer problem:

```
        %Stability condition for explicit method
if (alpha*dt)/(dx^2) <= 0.25 & (alpha*dt)/(dy^2) <= 0.25
        for k = 1:num_steps
      for i = 2:Nx-1
        for j = 2:Ny-1
```

```
            % Explicit update using finite difference
            T(i, j) = T(i, j) + alpha * dt * ((T(i+1, j) - 2*T(i, j) + T(i-1, j)) / dx^2 + ...
            (T(i, j+1) - 2*T(i, j) + T(i, j-1)) / dy^2);
         end
      end
      T(end,:) = (T(end-1,:) + (dy*h/k)*Tsurrounding)/(1 + (dy*h/k)); % Top boundary
condition
      T(:,end) = (T(:,end-1) + (dx*h/k)*Tsurrounding)/(1 + (dx*h/k)); % Right boundary
condition
      T(:,1) = T(:,2); % Left boundary condition
      T(1,:) = T(:,2); % Bottom boundary condition
         end
         T
else
           disp('System is unstable');
end
```

Visualization: Plotting functions are utilized to visualize the temperature distribution within the square block at different location on the block. Iso-lines of constant temperature contours and filled contour plot within the block are generated to facilitate analysis.

```
% Visualization code (same for both method)
```

****Code for filled contour plot****

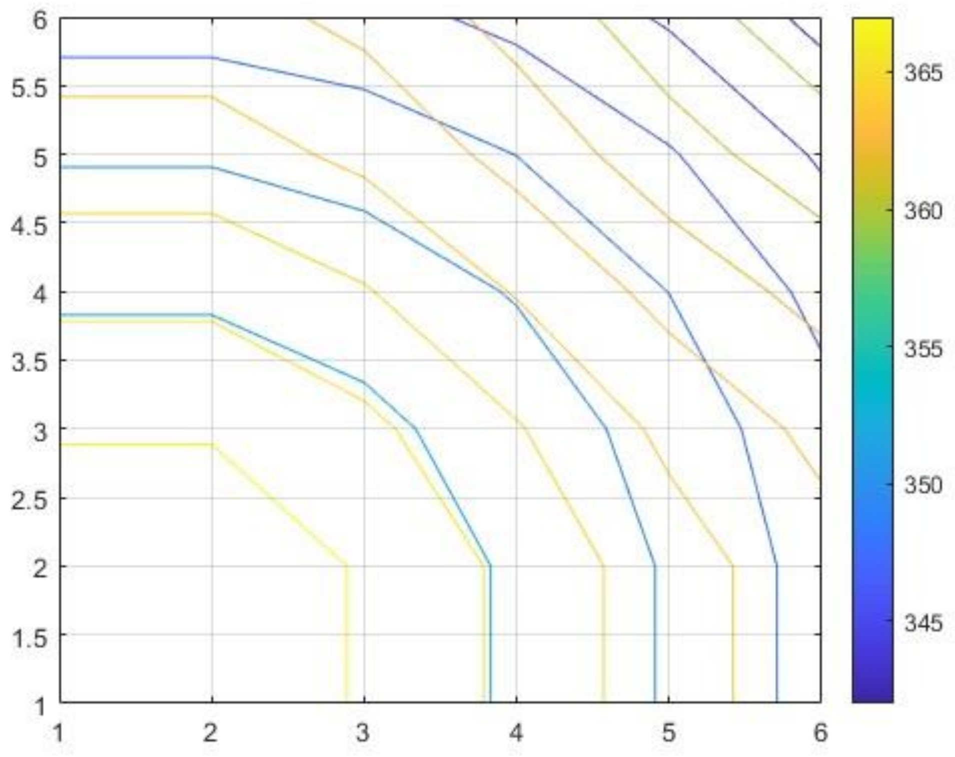
```
figure(1)
[X, Y] = meshgrid(0:dx:L, 0:dy:L);
contourf(X, Y, T, 20, 'LineColor', 'none');
colorbar;
xlabel('X (m)');
ylabel('Y (m)');
title('Temperature Distribution in the Square Block');
```

****Code for contour of iso-lines plot****

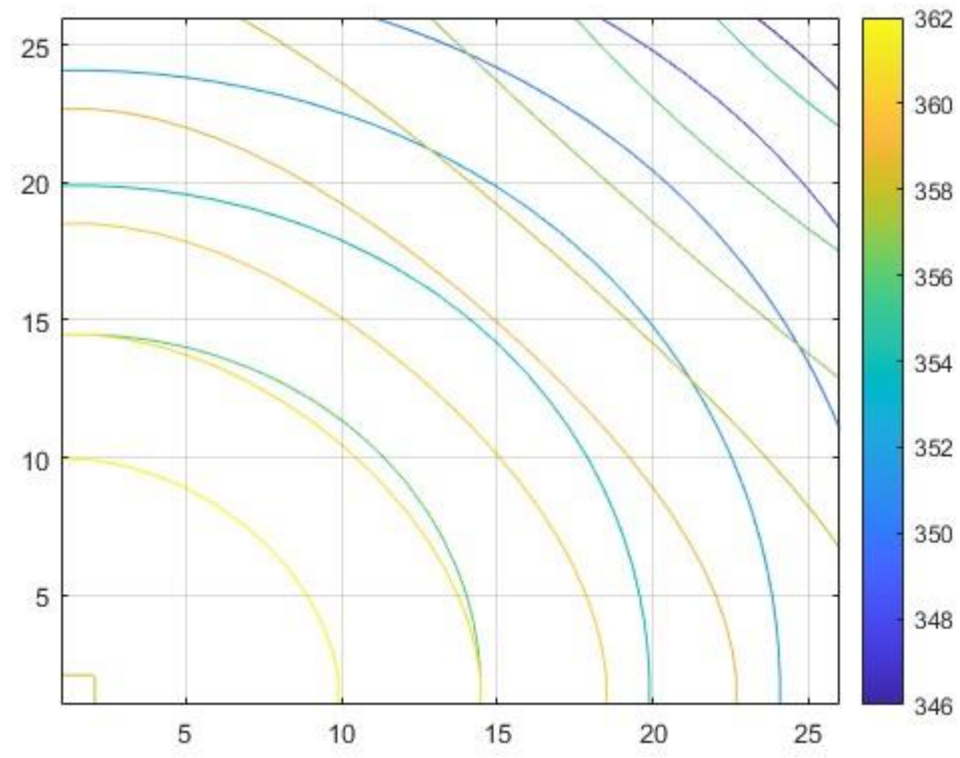
```
figure(2)
contour(T)
colorbar;
grid on;
```

Results and Discussion:

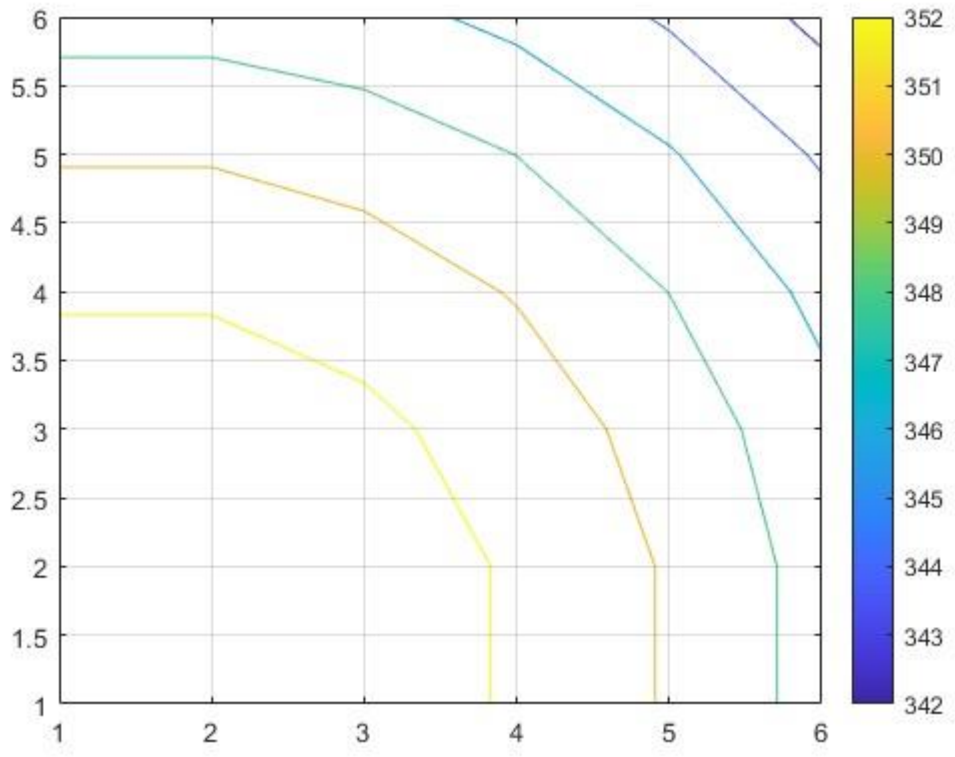
The simulation results demonstrate the evolution of temperature distribution within the square block over discrete time intervals. The iso-line plots of temperature and filled contour plot within the block provide insights into how temperature gradients develop and propagate within the block under given boundary conditions. Analysis of these results allows us to draw conclusions regarding the thermal behavior of the material and the effectiveness of the ADI method over explicit method in simulating such phenomena.



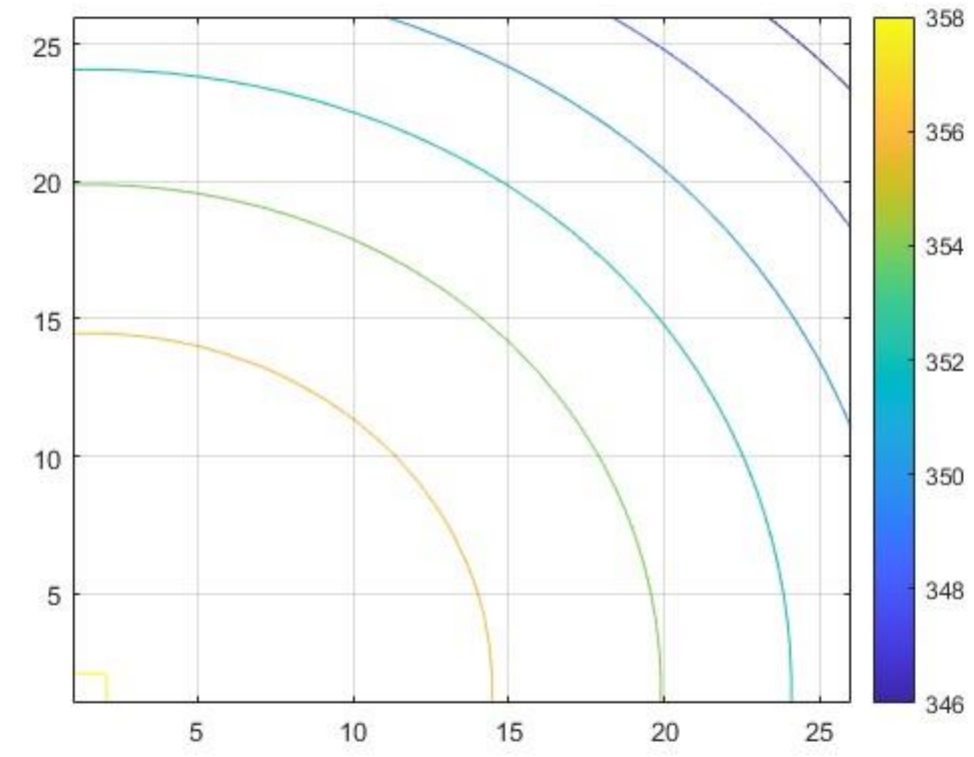
Above plot shows the difference in iso-lines between the two method with same number of grid points(6x6) and time interval(60sec).



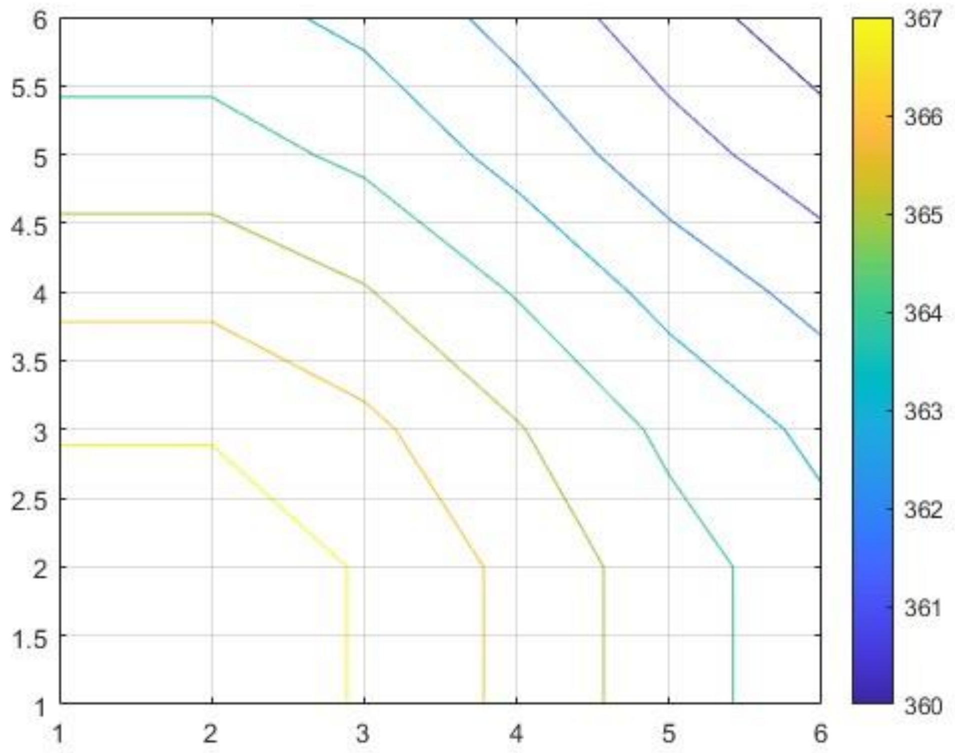
Above plot shows the difference in iso-lines between the two method with same number of grid points(26x26) and time interval(60sec).



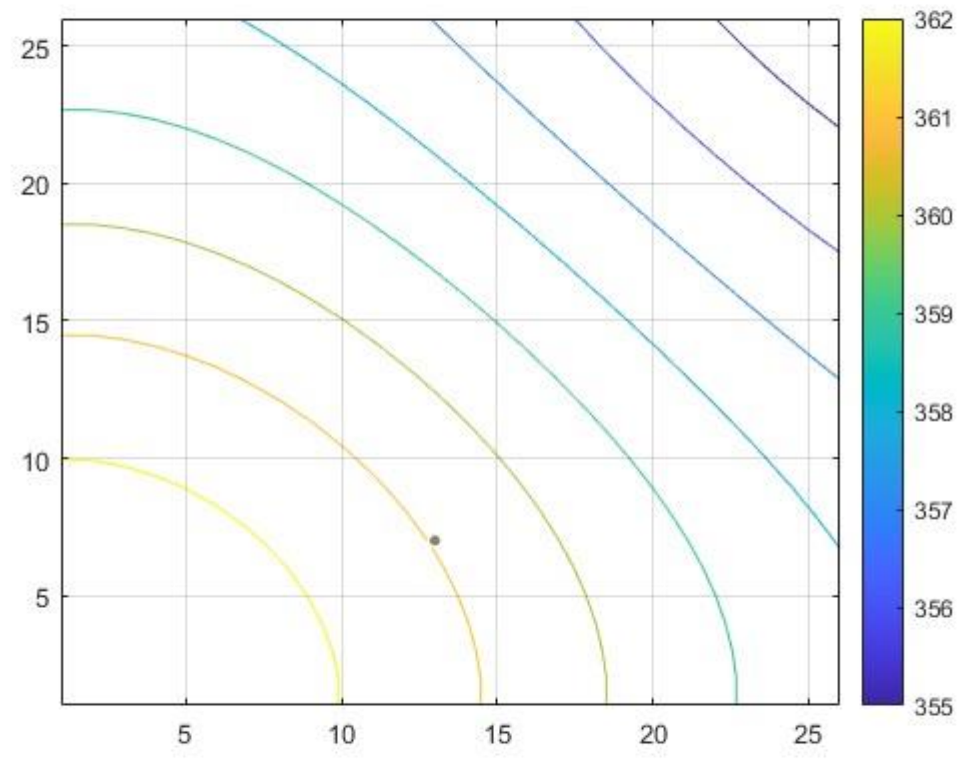
Above plot shows the different iso-lines of temperature using implicit method. Here at the top left corner and right bottom corner temperature is around 347 K, and at the top right corner it is 342 K.



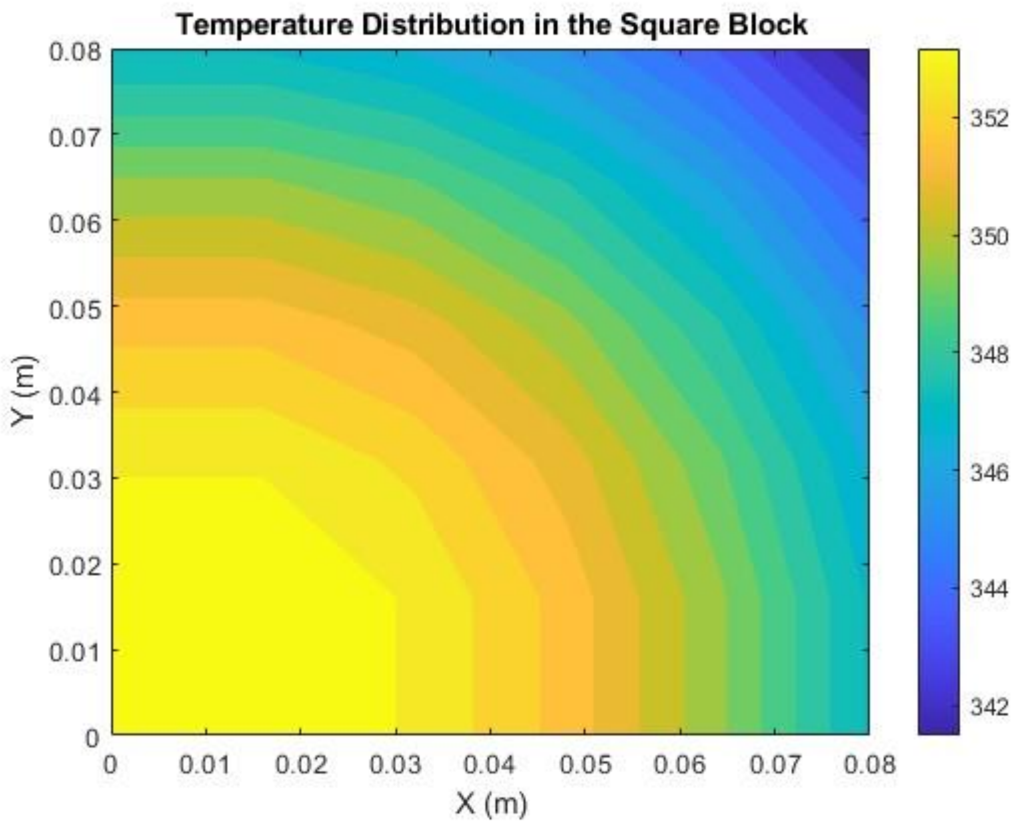
Above plot shows the different iso-lines of temperature using implicit method. Here at the top left corner and right bottom corner temperature is around 350 K, and at the top right corner it is 344 K.



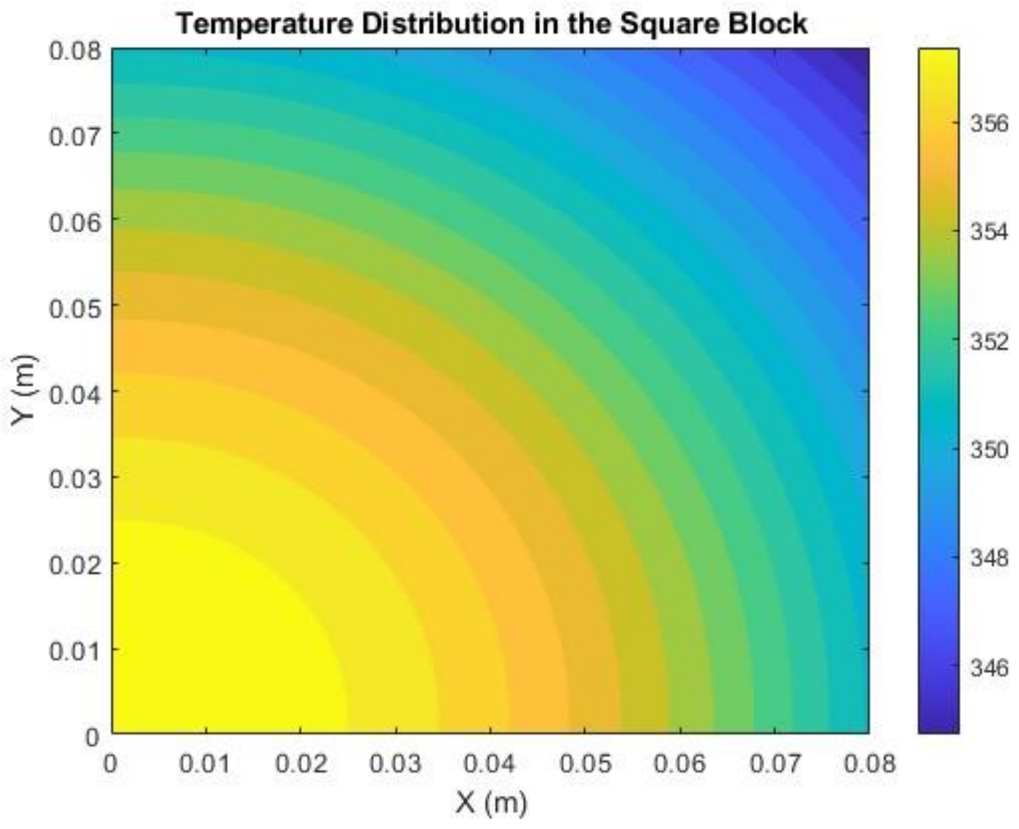
Above plot shows the different iso-lines of temperature using explicit method. Here at the top left corner and right bottom corner temperature is around 363 K, and at the top right corner it is 359 K.



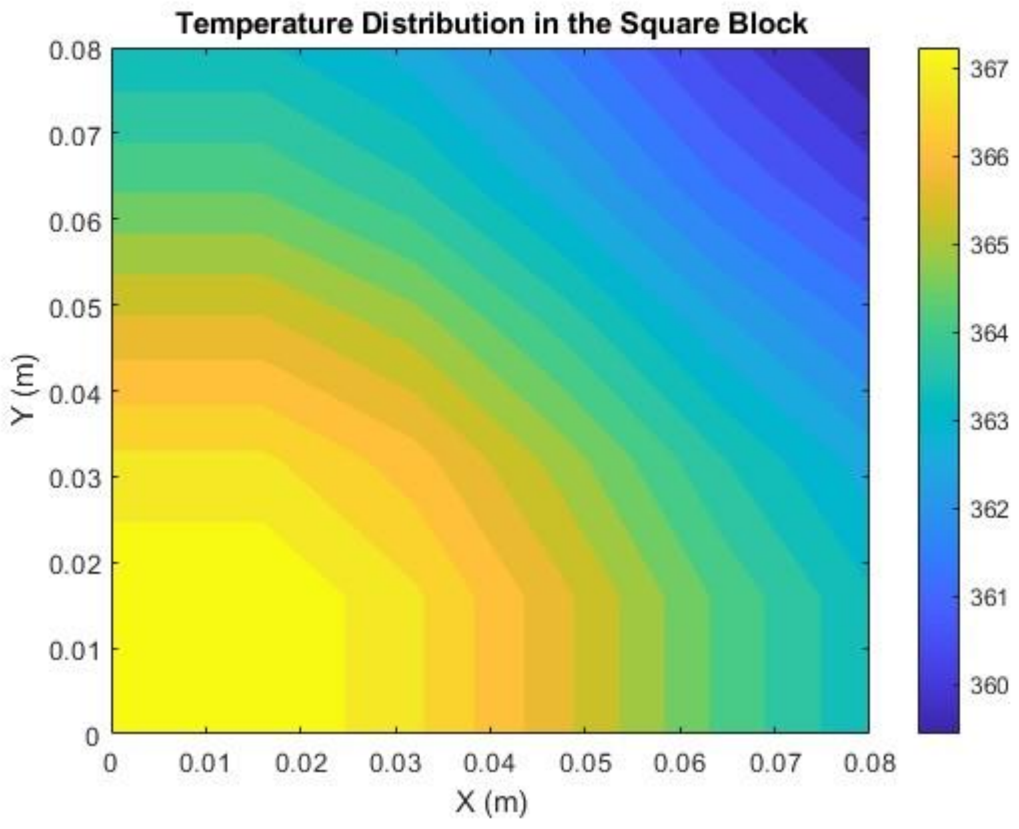
Above plot shows the different iso-lines of temperature using explicit method. Here at the top left corner and right bottom corner temperature is around 358 K, and at the top right corner it is 354 K.



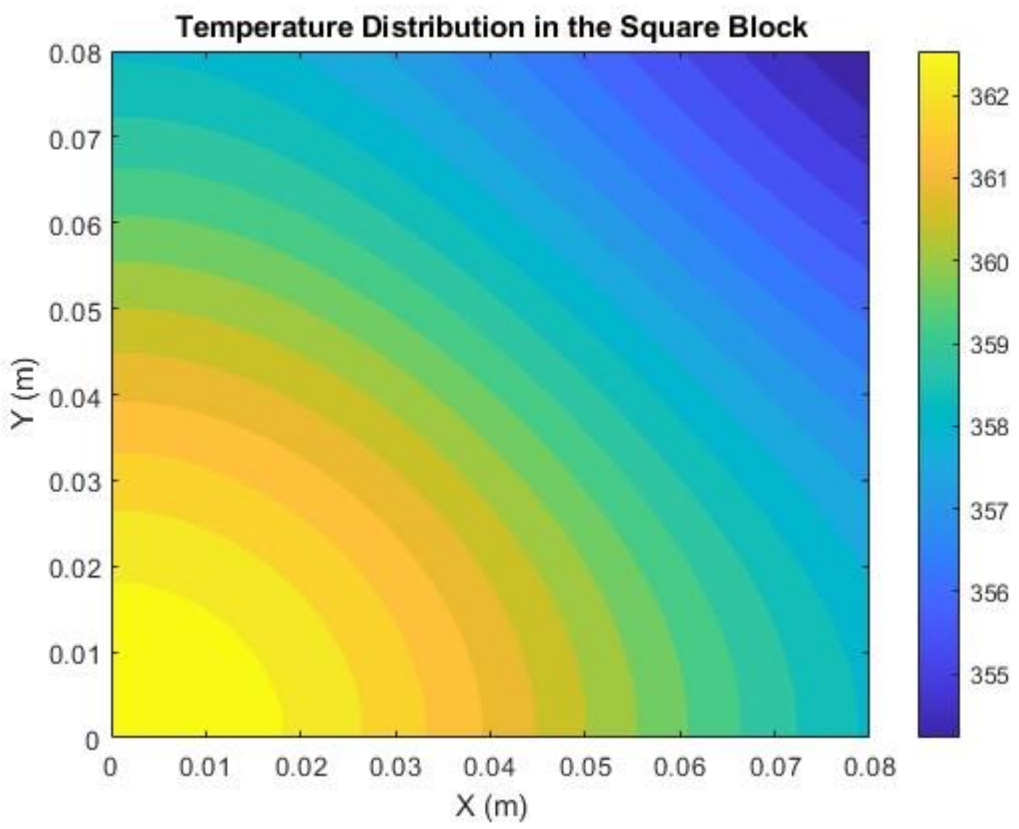
Above plot shows the filled contour plot of temperature using implicit method. Here at the top left corner and right bottom corner temperature is around 347 K, and at the top right corner it is 342 K.



Above plot shows the filled contour plot of temperature using implicit method. Here at the top left corner and right bottom corner temperature is around 350 K, and at the top right corner it is 344 K.



Above plot shows the filled contour plot of temperature using explicit method. Here at the top left corner and right bottom corner temperature is around 363 K, and at the top right corner it is 359 K



Above plot shows the filled contour plot of temperature using explicit method. Here at the top left corner and right bottom corner temperature is around 358 K, and at the top right corner it is 354 K

Observations:

- ADI Method:

In our 2D block simulation, I observe the following characteristics when using ADI:

- Accuracy: The ADI method maintains high accuracy even near interfaces or boundaries.
- Efficiency: It efficiently advances in time, making it computationally faster than fully implicit methods.
- Stability: ADI exhibits unconditional stability, ensuring robustness during simulations.

- Explicit Method:

In our 2D block simulation, I noticed the following aspects with the explicit method:

- Accuracy: The explicit method can suffer from accuracy loss near interfaces or sharp gradients.
- Efficiency: It is computationally less expensive per time step but may require smaller time steps for stability.
- Stability: Explicit methods have stability constraints (e.g., CFL condition) that limit time step sizes.

- Importance of Correct Boundary Conditions:

- Boundary conditions play a crucial role in numerical simulations. Incorrectly implemented boundary conditions can lead to erroneous results.
- Essential Boundary Conditions: These prescribe the value of the solution at the boundary (e.g., fixed temperature or heat flux).
- Natural Boundary Conditions: These relate to the derivative of the solution at the boundary (e.g., insulated or convective boundaries).

- Properly enforcing boundary conditions ensures physical realism and prevents unphysical artefacts.

➢ ADI vs. Explicit Method:

- Accuracy:

1. Over longer time steps or larger grids, ADI outperforms the explicit method.
2. ADI’s implicit treatment mitigates stability issues near interfaces, maintaining accuracy.
3. Explicit methods struggle with stability constraints, especially as time steps increase.

- Computational Efficiency:

1. ADI is more efficient due to its semi-implicit nature.
2. Explicit methods require smaller time steps for stability, increasing computational cost.

- Grid Resolution:

1. ADI handles finer grids better. It converges faster spatially and temporally.
2. Explicit methods may need excessively small grids to maintain stability.

In summary, while both methods have their merits, ADI shines when dealing with variable coefficient PDEs, longer time steps, and larger grids. However, always validate your implementation against analytical solutions or physical experiments to ensure correctness and reliability.

For further exploration, feel free to dive into the fascinating world of numerical methods and their applications!

Conclusion:

I have successfully demonstrated the simulation of temperature distribution within a square block using the ADI technique and explicit method in MATLAB. The study highlights the importance of numerical methods in analyzing thermal behavior and provides a foundation for further investigations into more complex systems and boundary conditions. The presented methodologies can be extended to study a wide range of heat transfer problems in various engineering applications.

References:

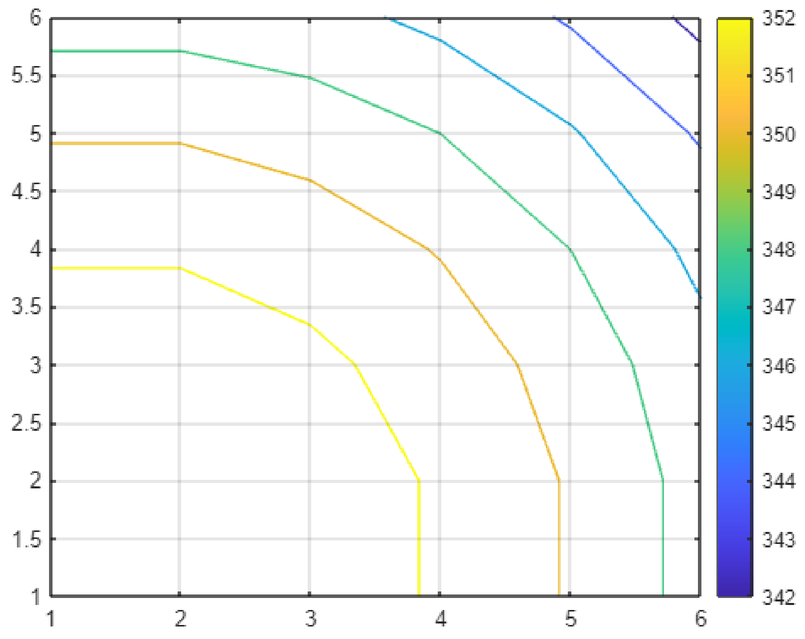
[1] Steven C. Chapra, & Raymond P. Canale (2015). Numerical Methods for Engineers. Mc Graw Hill Education.

[2] Gautam Biswas, & Somenath Mukherjee (2014). Computational Fluid Dynamics. Alpha Science International.

```
% Devyansh Agarhari (231010021)
%ANM_TERM PAPER CODE
%Comparative Analysis of ADI and Explicit Finite Difference Method for a 2-D Block Problem.

clc; clear all; close all;
% Parameters
L = 8*10^(-2); % Side length of the square block (meters)
dt = 0.15; % Time step (seconds)
alpha = 1.6*10^(-5); % Thermal diffusivity (m^2/s)
Tsurrounding = 25+273.15; % Surrounding Temperature
h = 400; % Convective heat transfer coefficient
k = 61; % Conduction heat transfer coefficient
Tbody_initially = 100+273.15;

% Defined for implicit method
N1 =6; % Number of grid points in x & y-direction
t1 = 60;% No. of time steps
dx1 = L / (N1 - 1);% Grid spacing x-direction
dy1 = L / (N1 - 1);% Grid spacing y-direction
[T1] = Heat_Transfer_Function_implicit (alpha, k, h, Tsurrounding, Tbody_initially, N1, dt, t1, L);

N2 =26;% Number of grid points in x & y-direction
t2 = 60;% No. of time steps
dx2 = L / (N2 - 1);% Grid spacing x-direction
dy2 = L / (N2 - 1);% Grid spacing y-direction
[T2] = Heat_Transfer_Function_implicit (alpha, k, h, Tsurrounding, Tbody_initially, N2, dt, t2, L);


% Defined for explicit method
Nx1 = 26; % Number of grid points in x-direction
Ny1 = 26; % Number of grid points in y-direction
dx = L / (Nx1 - 1);% Grid spacing x-direction
dy = L / (Ny1 - 1);% Grid spacing y-direction
t = 60; % Final time
[T3] = Heat_Transfer_Function_explicit ( alpha, h, Tsurrounding, Tbody_initially, dx, dy, Nx1, dt, t);

% Visualization
figure(1)
fprintf("Grid size 5x5 and time of 60s");
```

```
Grid size 5x5 and time of 60s
```

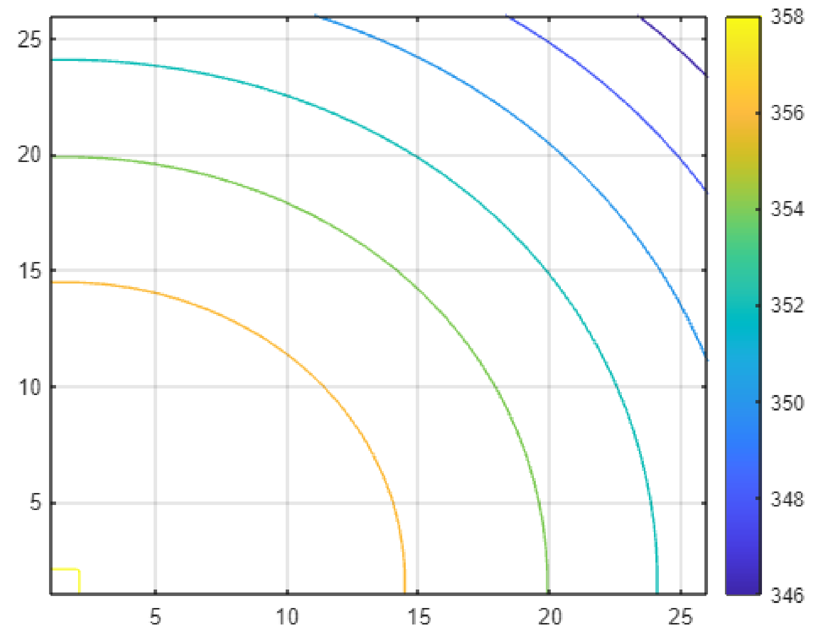
```
contour(T1)
colorbar;
grid on;
```



```
figure(2)
fprintf("Grid size 25x25 and time of 60s");
```

```
Grid size 25x25 and time of 60s
```

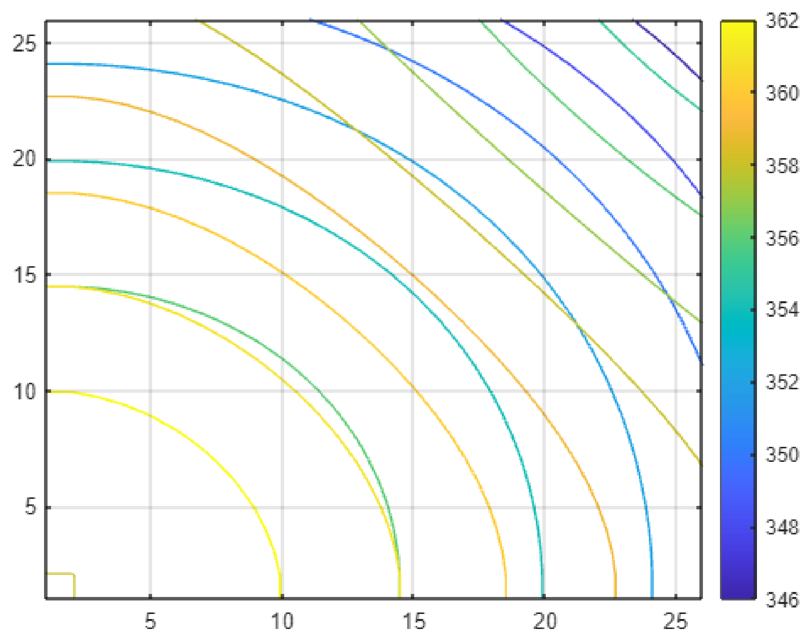
```
contour(T2)
colorbar;
grid on;
```



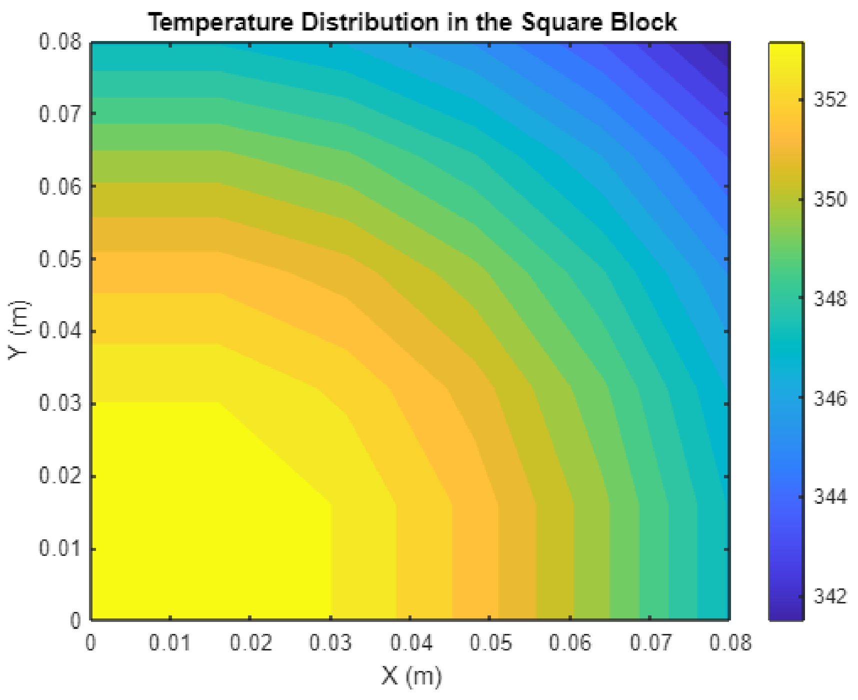
```
figure(3)
fprintf("Grid size 25x25 and time of 60s");
```

```
Grid size 25x25 and time of 60s
```

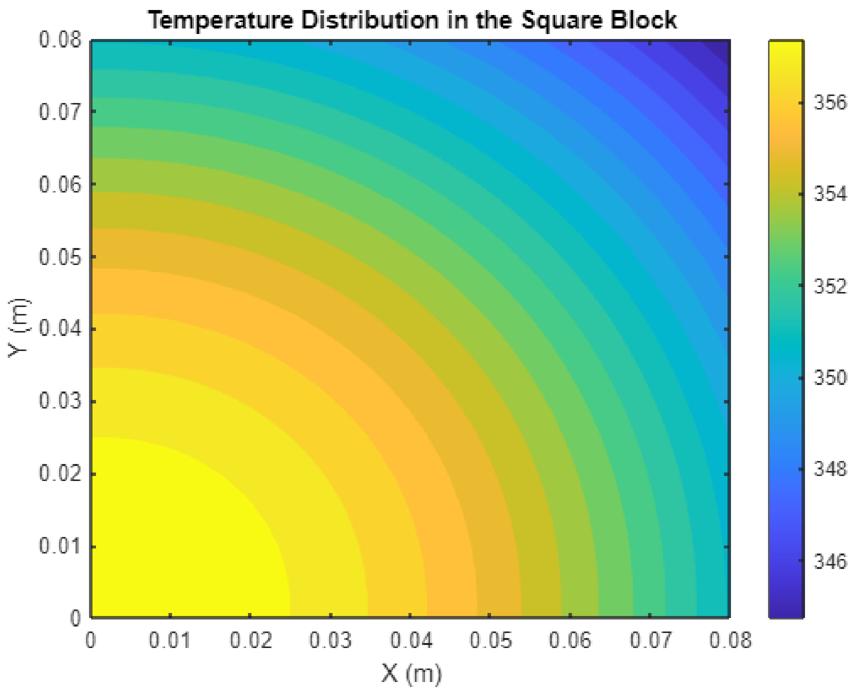
```
contour(T2)
hold on
contour(T3)
colorbar;
grid on;
hold off
```



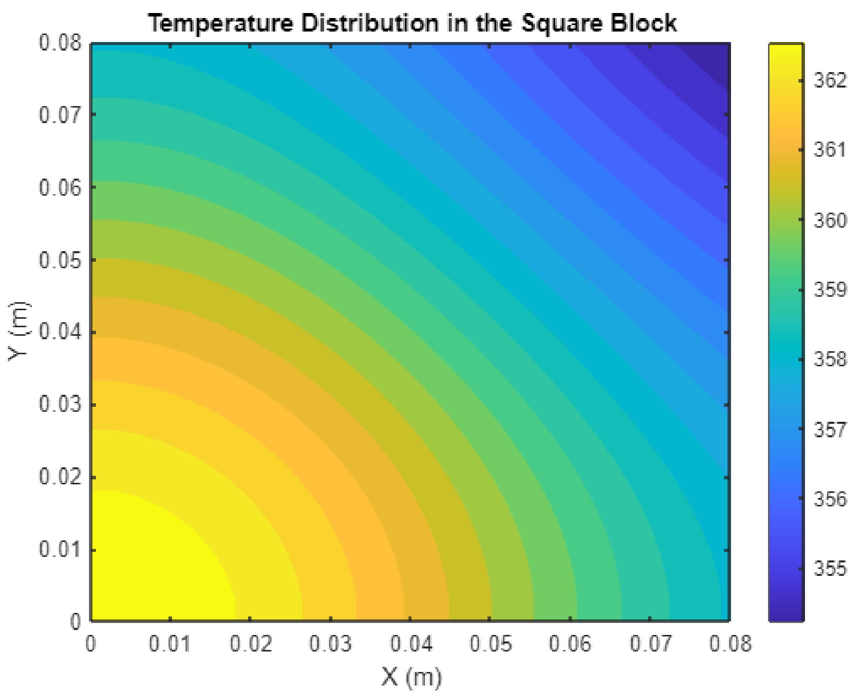
```
figure(4)
[X, Y] = meshgrid(0:dx1:L, 0:dy1:L);
contourf(X, Y, T1, 20, 'LineColor', 'none')
colorbar;
xlabel('X (m)');
ylabel('Y (m)');
title('Temperature Distribution in the Square Block');
```



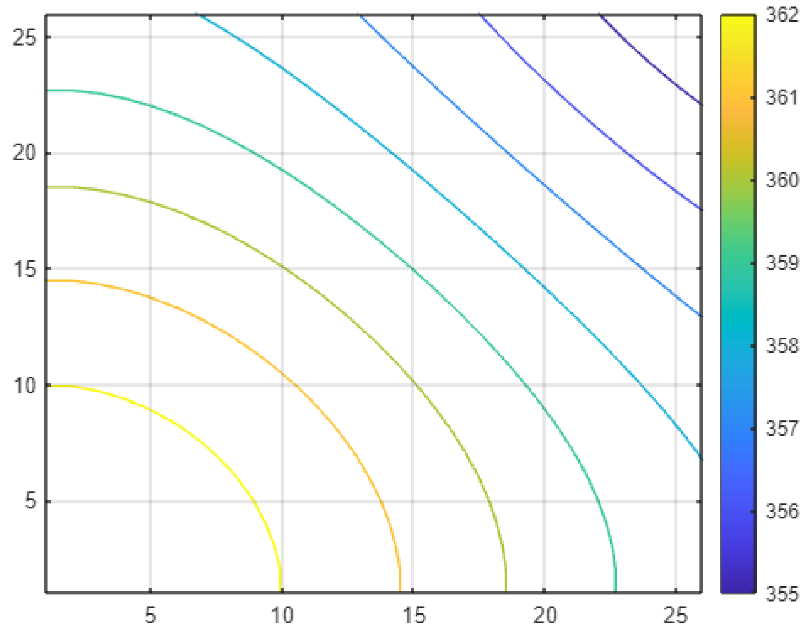
```
figure(5)
[X, Y] = meshgrid(0:dx2:L, 0:dy2:L);
contourf(X, Y, T2, 20, 'LineColor', 'none')
colorbar;
xlabel('X (m)');
ylabel('Y (m)');
title('Temperature Distribution in the Square Block');
```



```
figure(6)
[X, Y] = meshgrid(0:dx:L, 0:dy:L);
contourf(X, Y, T3, 20, 'LineColor', 'none')
colorbar;
xlabel('X (m)');
ylabel('Y (m)');
title('Temperature Distribution in the Square Block');
```



```
figure(7)
contour(T3)
colorbar;
grid on;
```



```
%% Implicit method based Function
```

```
function [T] = Heat_Transfer_Function_implicit (Alpa, k, h, Tsurrounding, Tbody, N, htm, t, x)
n = N-1;
hx = x/(2*n);
delta_t = t/htm;
T = ones(N)*Tbody;
```

```
T1step = T;
T2stp = T;


%Solve for each time step
for m = 1:delta_t
    rx = (Alpa*htm)/(hx^2);
    a=-1*rx/2;
    b=1+rx;
    A=zeros(N);
    A(N,N-1)=-1*k/hx;
    A(N,N)=h+k/hx;
    A(1,1)=-1;
    A(1,2)=1;
    for i = 2:N-1
        A(i, i-1)= a;
        A(i, i)= b;
        A(i, i+1)= a;
    end
    [A_inv] = Inverse_using_thomas_alg (A);
    for j=2:N-1
        da = zeros(N, 1);
        da(1) = 0;
        da(N) = h*Tsurrounding;
        for i=2:N-1
            da(i)=T(j,i)+(rx/2)*(T(j+1,i)-2*T(j,i)+T(j-1,i));
        end
        Tj=A_inv*da;
        for i = 1:N
            T1step(j,i) = Tj(i);
        end
        for i = 1:N
            T1step(1,i) = T1step(2,i);
            T1step(N,i) = (k*T1step(N-1,i)+h*hx*Tsurrounding)/(h*hx+k);
        end
    end
    for j=2:N-1
        db = zeros(N, 1);
        da(1) = 0;
        db(N) = h*Tsurrounding;
        for i=2:N-1
            db(i)=T1step(i,j)+(rx/2)*(T1step(i, j+1)-2*T1step(i,j)+T1step(i,j-1));
        end
        T_i=A_inv*db;
        for i = 1:N
            T2stp(i,j) = T_i(i);
        end
        for i = 1:N
            T2stp(i,1) = T2stp(i,2);
            T2stp(i,N) = (k*T2stp(i,N-1)+h*hx*Tsurrounding)/(h*hx+k);
        end
    end
    T=T2stp;
end

end




%% Explicit method based Function
function[T] = Heat_Transfer_Function_explicit ( alpha, h, Tsurrounding, Tbody_initially, dx, dy, Nx, dt, t)
Ny = Nx;
% Time-stepping loop(No. of iterations)
```

```
num_steps = t/dt; % Total number of time steps

% Initialize temperature field
T = ones(Nx, Ny)*Tbody_initially; % Initial temperature distribution

%Stability condition for explicit method
if (alpha*dt)/(dx^2) <= 0.25 & (alpha*dt)/(dy^2) <= 0.25
    for k = 1:num_steps
        for i = 2:Nx-1
            for j = 2:Ny-1
                % Explicit update using finite difference
                T(i, j) = T(i, j) + alpha * dt * ((T(i+1, j) - 2*T(i, j) + T(i-1, j)) / dx^2 + ...
                (T(i, j+1) - 2*T(i, j) + T(i, j-1)) / dy^2);
            end
        end
        T(end,:) = (T(end-1,:) + (dy*h/k)*Tsurrounding)/(1 + (dy*h/k)); % Top boundary condition
        T(:,end) = (T(:,end-1) + (dx*h/k)*Tsurrounding)/(1 + (dx*h/k)); % Right boundary condition
        T(:,1) = T(:,2); % Left boundary condition
        T(1,:) = T(:,2); % Bottom boundary condition
    end
else
    disp('System is unstable');
end
end

%% Thomas_Algorithm_for_inverse_calculation_of_matrix
function [X] = Inverse_using_thomas_alg (T)
%Finds inverse of a Tri-diagonal matrix using Thomas Algorithm

[n, ~] = size(T); %find the size of T matrix
B = eye(n); %initialize B as an identity matrix

%decomposition of T & B(identity, as inverse is needed) matrix

%row operations on T matrix
for i = 2:n
    T(i, i-1) = T(i, i-1)/T(i-1, i-1); %define ele of -1 diag ele wrt to the principal diag in the T matrix
    T(i, i) = T(i, i)- (T(i, i-1)*T(i-1, i)); %define ele of the principal diag in the T matrix
end
%row operations on T matrix

%row operations on ith row of jth column of B matrix
for j = 1:n
    for i = 2:n
        B(i, j) = B(i, j) - T(i, i-1)*B(i-1, j); %perform row operations
    end
end
%row operations on ith row of jth column of B matrix

%decomposition of T & B(identity, as inverse is needed) matrix


%Back substitution to find X which is the inverse of T matrix
X = zeros(n , n); %initialize X as a zero matrix
for p = 1:n
    X (n, p) = B (n, p)/ T (n, n); %find the X_n ele of p_th column
    for q = n-1:-1:1
        X(q, p) = (B(q,p) - T(q, q+1)*X(q+1, p))/T(q,q); %find ele of q_th row & p_th column
    end
end
%Back substitution to find X which is the inverse of T matrix

end
```

```
%% Thank you
%% Please review and suggest your feedback.
```