



***2D Diffusion Equation model using the Explicit Finite
Difference Method***

Scott Roberts

Student Number: 1451418

BEng Mechanical Engineering

27th January 2025

Computational Methods – ACMA6012

Word Count: 3286

(Excluding citations, headings, tables and figures)

Abstract

This report details the development, implementation and evaluation of a 2D heat diffusion model using the Explicit Finite Difference Method (EFDM). The model simulates the heat diffusion within an aluminium plate under Dirichlet and Neumann boundary conditions, assessing when thermal equilibrium is achieved. MATLAB was used to implement the EFDM, incorporating stability criteria and ensuring appropriate user inputs to increase accuracy and stability.

The heat equation and its application to 2D heat diffusion are introduced, followed by an explanation of the discretisation process. Taylor series approximation were used to transform the partial differential equation into an numerical solution. A walkthrough of the MATLAB implementation is provided, highlighting incorporated conditions for user input to enhance stability and accuracy of the model.

Sensitivity analysis and model validation was conducted to explore the impact of user inputs such as grid spacing, time step and code defined convergence criteria on the model's accuracy. Results demonstrated the importance of the input and also highlighted limitation of the model. A validation was performed against a provided analytical solution confirming the model's accuracy, as evidenced by the minimal temperature differences between the numerical and analytical solution matrices.

This report attempts to demonstrate the effectiveness of EFDM in modelling heat diffusion, while recognising the limitations and improvements other methods could provide.

Table of Contents

Abstract.....	2
List of Figures	5
List of Tables	5
1 Introduction	6
1.1 Aims & Objectives.....	6
1.1.1 Aim	6
1.1.2 Objectives	6
2 Constitutive Equation	7
2.1 Heat Equation	7
3 Finite Difference Method	7
3.1 Taylor Series	7
3.2 First Derivative Approximations.....	8
3.2.1 Forward Difference Approximation	8
3.2.2 Backward Difference Approximation	8
3.2.3 Central Difference Approximation.....	8
3.3 Second Derivative Approximations.....	8
3.4 Explicit Finite Difference Scheme.....	8
3.4.1 Discretise the Spatial Domain	9
3.4.2 Discretise the Temporal Domain	10
3.5 EFDM form of the Heat Equation	11
3.6 Stability Criterion.....	11
4 Initial and Boundary Conditions.....	11
5 Analytical Solution.....	12
6 Code Details and Explanations	13
6.1 Section 1: Workspace Management Commands.....	13
6.2 Section 2: Defining the Material Properties	13
6.3 Spatial Consideration	14
6.3.1 Defining plate dimensions	14
6.3.2 Defining Node Distance	14
6.4 Compute Number of Rows and Columns	14
6.5 Temperature Matrix T1	15
6.6 Temporal Consideration	15
6.7 Initialisation of Temperature Matrix.....	16
6.7.1 Implementation of the Neumann Boundary Conditions	16
6.8 Convergence Criterion Threshold	17
6.9 Calculation of T2	17

6.10	Analytical Solution.....	18
7	Code Validation.....	19
8	Sensitivity Analysis of the Code	20
8.1	Influence of the Grid Spacing.....	20
8.2	Influence of the Time Step	21
8.3	Influence of the Convergence Criterion	22
9	Limitations of the Mode	24
9.1	Round-off Error	24
9.2	Truncation Error	24
9.3	Stability	24
9.4	Complex Geometries	24
9.5	Coding Limitations	24
10	Suggested Improvements and Future Developments.....	25
10.1	Use of Alternative Finite Difference Methods (FDM).....	25
10.2	Use of Numerical Alternatives	25
10.3	Alternatives to Calculating Thermal Equilibrium	26
11	Conclusion	27
	References	28
	Appendices	30
	Appendix A: MATLAB Code for 2D Diffusion Equation Model.....	30
	Appendix B: Numerical Solution Figure Output Example.....	34
	Appendix C: Example 1 Validation Code Output	35
	Appendix D: Example 2 Validation Code Output	36

List of Figures

Figure 1: Discretising the Spatial Domain along the X Axis.	9
Figure 2: Discretising the Spatial Domain along the Y Axis.	9
Figure 3: Discretising the Temporal Domain.	10
Figure 4: Initial and boundary conditions schematic.	12
Figure 5: Image of Section 1 Workspace Management Commands.	13
Figure 6: Image of Section 2 Defining the Material Properties.	13
Figure 7: Image of Section 3 showing Xdim and Ydim inputs including the set conditions...	14
Figure 8: Image of Section 3 showing grid spacing inputs including the set conditions.....	14
Figure 9: Image of Section 4 showing calculation for rows and columns.	14
Figure 10: Image of Section 5 showing Temperature Matrix T1 Code.....	15
Figure 11: Temperature matrix T1 with applied inputs.....	15
Figure 12: Image of Section 6 User ΔT input with conditions.....	15
Figure 13: Image of Section 6 Stability Criterion Check.....	16
Figure 14: Image of re-prompt message for an unsuitable ΔT entry.....	16
Figure 15: Image of the modified EFDM equation to apply Neumann Boundary Conditions.	16
Figure 16: Final EFDM equation implemented in the code.	16
Figure 17: Image of relevant code lines to the Convergence Criterion Threshold.	17
Figure 18: Image of completed T2 matrix after code execution.	17
Figure 19: Time to reach thermal equilibrium output.	18
Figure 20: Analytical solution implemented in the code.	18
Figure 21: Update TA matrix output in the code.....	18
Figure 22: Calculation of maximum temperature difference between T2 and TA temperature distribution.	18
Figure 23: Maximum difference matrix code output.	19
Figure 24: Sensitivity Analysis Results for Varying Grid Spacing Graph.	21
Figure 25: Sensitivity Analysis Results for Varying Time Step Graph.....	22
Figure 26: Sensitivity Analysis Results for Varying Convergence Criterion Graph.	23

List of Tables

Table 1: Provided Aluminium Properties.	13
Table 2: User inputs for Example 1 code validation execution.	19
Table 3: User inputs for Example 2 code validation execution.	19
Table 4: Sensitivity Analysis Results for Varying Grid Spacing	20
Table 5: Sensitivity Analysis Results for Varying Time Step	22
Table 6: Sensitivity Analysis Results for Varying Convergence Criterion.....	23
Table 7: Comparison of ADI, Implicit FDM and EFDM	25
Table 8: Comparison of FEM, FVM and BEM.....	26
Table 9: Alternatives to Calculating Thermal Equilibrium	26

1 Introduction

This report explores the development of a 2D heat diffusion model using the Explicit Finite Difference Method (EFDM) in MATLAB. The model simulates heat diffusion in an aluminium plate over time until thermal equilibrium is reached. The MATLAB code used for this analysis is provided in Appendix A.

The report first explains the heat equation, followed by the discretisation procedure used in the model. It then details the initial and boundary conditions, before presenting information on the developed code, validation and sensitivity analysis. Finally, the report discusses the model's limitations and suggests improvements for future development.

1.1 Aims & Objectives

1.1.1 Aim

This report aims to provide an in-depth overview of the development process for the 2D heat diffusion model implemented using the Explicit Finite Difference Method (EFDM). It also evaluates the model's performance and accuracy in simulating heat diffusion within an aluminium plate over time.

1.1.2 Objectives

This report has the following objectives:

- 1) Provide an introduction into the heat equation, explaining what the heat equation is and its importance. Defining the terms and giving mathematical definition for a 2D geometry with no heat source term.
- 2) Describe the full discretisation process followed when developing the heat diffusion code based on the EFDM equation.
- 3) Provide an explanation of the code, including performing a sensitivity analysis on the code to illustrate alternative outputs in response to input variables.
- 4) Validate and analyse code results, identifying limitations with the model produced and further improvements that could be made.

2 Constitutive Equation

2.1 Heat Equation

The heat equation is a partial differential equation that describes distribution of heat in a given region over a period of time. The modern heat equation as shown in Equation (1) (Evans, 2022). Jean Baptiste Joseph Fourier presented the foundational work to the modern day formula in his published book *Théorie analytique de la chaleur* (The analytical Theory of Heat) in 1822.

$$\frac{1}{\alpha} \frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \quad (Eq. 1)$$

Where:

T : *Temperature.*

t : *Time.*

$\frac{\partial u}{\partial t}$: *Rate of change of temperature (u) with respect to time (t).*

α : *Thermal diffusivity of the material.*

$\frac{\partial^2 u}{\partial x^2}$: *Second partial derivative of temperature with respect to spatial coordinate x.*

$\frac{\partial^2 u}{\partial y^2}$: *Second partial derivative of temperature with respect to spatial coordinate y.*

3 Finite Difference Method

The finite difference method solves the heat equation by discretising both time and space, replacing derivatives with finite difference approximations derived from Taylor series expansions (Recktenwald, 2004). This transforms the partial differential equation into algebraic equations solvable based on the initial and boundary conditions.

3.1 Taylor Series

Taylor series expansions approximate functions near a point using polynomials. The series for function $f(x)$ around $x = x_0$ is shown in Equation (2) (Croft & Lilley, 1977). Truncated Taylor series are used to approximate derivatives, which are then substituted into the heat equation.

$$f(x) = f(x_0) + f'(x_0) \cdot \frac{(x - x_0)^1}{1!} + f''(x_0) \cdot \frac{(x - x_0)^2}{2!} + f'''(x_0) \cdot \frac{(x - x_0)^3}{3!} + f^{(4)}(x_0) \cdot \frac{(x - x_0)^4}{4!} \dots \quad (Eq. 2)$$

Equation (3) (Haftka & Gurdal, 2012) shows the transposed and truncated Taylor series expression for the first derivative $f'(x_0)$.

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h} \quad (Eq. 3)$$

3.2 First Derivative Approximations

3.2.1 Forward Difference Approximation

Equation (3) provides the forward difference approximation for the first derivative, as it generates the approximation by looking to the right (forward) of x_0 . This method numerically approximates $f'(x_0)$.

3.2.2 Backward Difference Approximation

The backward difference approximation approximates to the left (backwards) of x_0 . The Taylor series can be modified as shown in Equation (4) (Croft & Lilley, 1977) to reflect this.

$$f(x_0 - h) = f(x_0) - h \cdot f'(x_0) + \frac{h^2}{2!} f''(x_0) - \frac{h^3}{3!} f^{(3)}(x_0) + \dots \quad (Eq. 4)$$

After transposing the equation, the backward difference approximation can be seen in Equation (5) (Manglik, 2024).

$$f'(x_0) \approx \frac{f(x_0) - f(x_0 - h)}{h} \quad (Eq. 5)$$

3.2.3 Central Difference Approximation

The Taylor series for the forward difference approximation (3) and the backward approximation (5) can be combined, resulting in an alternative Taylor series shown in Equation (7) (Özişik, 2017) showing the central difference approximation.

$$f(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} \quad (Eq. 7)$$

3.3 Second Derivative Approximations

The second derivative approximation is obtained by adding the forward and backward difference terms, once transposed Equation (8) (Patankar, 1980) shows the second derivative expression.

$$f''(x_0) = \frac{f(x_0 + h) - 2f(x_0) + f(x_0 - h)}{h^2} \quad (Eq. 9)$$

3.4 Explicit Finite Difference Scheme

The approximations are used to replace both the temporal first derivative and the spatial second derivative in the heat equation. The assignment specifies the use of the EFDM, where the first derivative is approximated using the forward difference and the second derivative using the central difference. This method calculates the solution at the next time step directly from known values at the current step, making it simple and conditionally stable, though it requires adherence to a stability criterion for accurate results (Smith, 1985).

3.4.1 Discretise the Spatial Domain

Discretising the spatial domain involves dividing it into discrete points where function values are calculated. Neighbouring values are approximated using the numerical scheme for the second spatial derivatives. Figure 1 and 2 illustrate the discretisation of the spatial domain for calculating the second derivatives $\frac{\partial^2 T}{\partial x^2}$ and $\frac{\partial^2 T}{\partial y^2}$, where the function f represents the temperature T at a location along the X and Y axes.

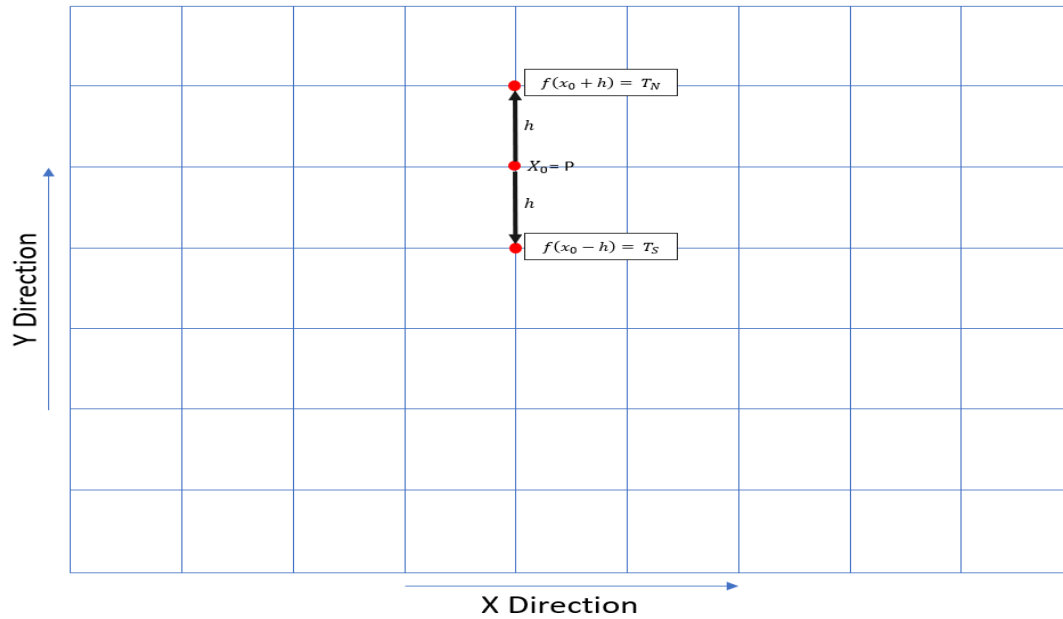


Figure 1: Discretising the Spatial Domain along the X Axis.

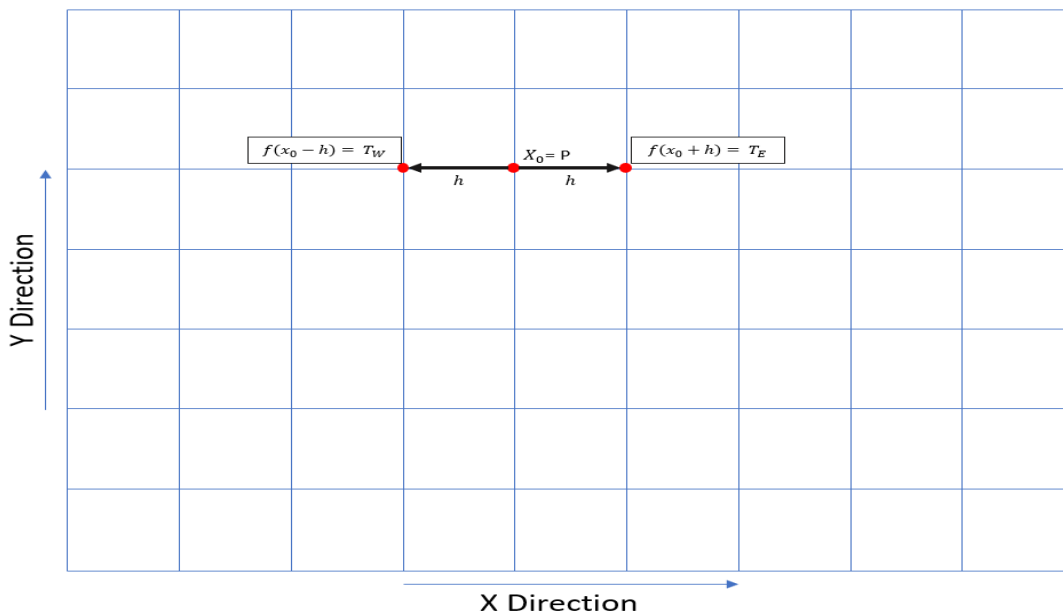


Figure 2: Discretising the Spatial Domain along the Y Axis.

As shown in Figure 1 and 2, the second derivative notations are adapted to define the function and location points within the domain. This results in Equations (12) and (13) (Causon & Mingham, 1997).

$$T_P'' \approx \frac{T_E^t - 2T_P^t + T_W^t}{h^2} \quad (\text{Eq. 12})$$

$$T_P'' \approx \frac{T_N^t - 2T_P^t + T_S^t}{h^2} \quad (\text{Eq. 13})$$

Where

T : Temperature.

P : Current node location.

h : Grid Spacing.

E : East of the current node.

W : West of the current node.

S : South of the current node.

N : North of the current node.

t : Current time.

3.4.2 Discretise the Temporal Domain

Discretising the temporal domain divides time into intervals, allowing calculations at specific time steps. Figure 3 illustrates this process for the first derivative $\frac{\partial T}{\partial t}$. The time step size Δt affects the accuracy of the numerical solution (Smith, 1985).

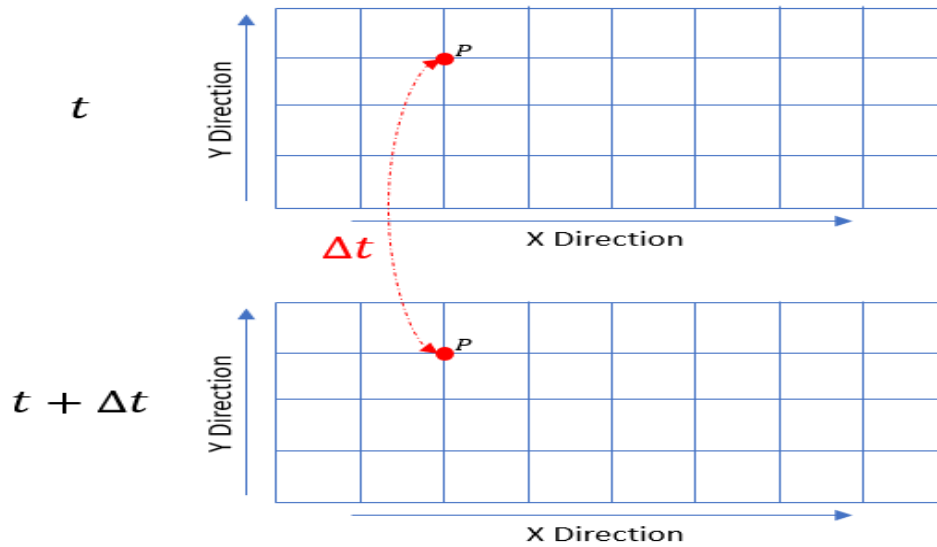


Figure 3: Discretising the Temporal Domain.

The forward difference approximation is adapted to represent the function and time steps, resulting in Equation (14) (Causon & Mingham, 1997).

$$T'_P \approx \frac{T_P^{t+\Delta t} - T_P^t}{\Delta t} \quad (\text{Eq. 14})$$

Where

T : Temperature.

P : Current node location.

t : Current time.

Δt : Time step.

3.5 EFDM form of the Heat Equation

Discretising the spatial and temporal domains transforms the heat equation into a numerical approximation solvable iteratively. Equation (15) replaces the time derivative with the forward difference and the spatial second derivatives with the central difference approximations.

$$\frac{1}{\alpha} \frac{T_P^{t+\Delta t} - T_P^t}{\Delta t} = \frac{T_E^t - 2T_P^t + T_W^t}{h^2} + \frac{T_N^t - 2T_P^t + T_S^t}{h^2} \quad (\text{Eq. 15})$$

Further manipulation introduces the Fourier number, yielding a more compact discretised form of the heat equation. The simplified EFDM is shown in Equation (16).

$$T_{i,j}^{n+1} = T_{i,j}^n + Fo(T_{i+1,j}^n + T_{i-1,j}^n + T_{i,j+1}^n + T_{i,j-1}^n - 4T_{i,j}^n) \quad (\text{Eq. 16})$$

The Fourier number Fo , characterises the relative influence of thermal diffusion on temporal and spatial scales. Its expression is shown in Equation (17).

$$Fo = \frac{\alpha * \Delta T}{h^2} \quad (\text{Eq. 17})$$

3.6 Stability Criterion

The Fourier number is critical for the stability and accuracy of the numerical solution. In the discretised heat equation (16), it influences stability and dictates the appropriate time step, ensuring it is small enough relative to the grid spacing to prevent instability. The assignment specifies maintaining a Fourier number below 0.25 to ensure a stable and accurate solution during iteration.

4 Initial and Boundary Conditions

The initial and boundary conditions outlined in the assignment are summarised in Figure 5.

The temperature at the north boundary T_0 is user defined, while the south boundary

temperature is $-T_0$. Both remain fixed throughout the simulation using Dirichlet boundary conditions, which set fixed values at the domain boundaries. This ensures the temperatures at the north and south boundaries stay constant during the simulation (Koshizuka, et al., 2018). Dirichlet conditions are commonly used when boundary values are known and must be maintained.

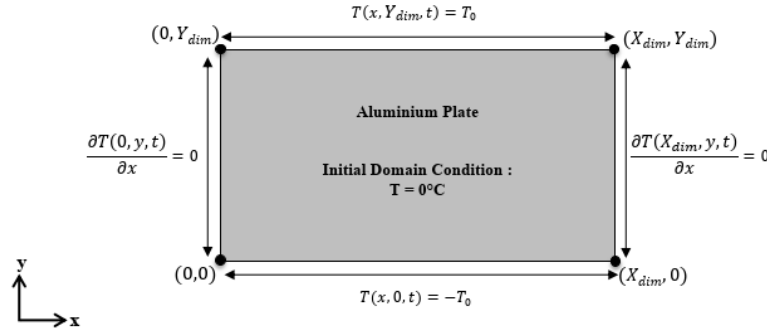


Figure 4: Initial and boundary conditions schematic.

Figure 4 illustrates the zero flux condition at the east and west boundaries, representing perfectly insulated edges enforced using the Neumann boundary condition. This condition specifies the derivative of the solution at the boundary, ensuring no heat transfer across these boundaries (Webster, 2025). To implement this, the EFDM equation is modified (see Section 6.7.1).

5 Analytical Solution

The assignment brief stated an analytical solution of the temperature at thermal equilibrium as shown in Equation (18) (Chang & Ma, 2001).

$$T(x, y) = -T_0 + \frac{2 \cdot T_0}{Y_{dim}} \cdot y + \sum_{k=1}^{\infty} \left(\frac{1}{(100 \cdot k \cdot \pi)^2} \cdot [1 - (-1)^k] \cdot \frac{\sin\left(\frac{k \cdot \pi \cdot y}{Y_{dim}}\right)}{\sinh\left(\frac{k \cdot \pi \cdot X_{dim}}{Y_{dim}}\right)} \cdot \left[\cosh\left(\frac{k \cdot \pi \cdot x}{Y_{dim}}\right) - \cosh\left(\frac{k \cdot \pi \cdot (x - X_{dim})}{Y_{dim}}\right) \right] \right) \quad (Eq. 18)$$

The analytical solution represents the steady-state temperature distribution $T(x, y)$ in a rectangular domain governed by the 2D heat equation. The boundary conditions are crucial, with fixed temperatures T_0 and $-T_0$ on the north and south boundaries, creating a strong gradient along the y-direction. The east and west boundaries are insulated, with the Neumann condition enforcing zero heat flux.

The analytical solution is used to test the code results by calculating the maximum difference between the numerical solution at thermal equilibrium and the analytical solution.

6 Code Details and Explanations

6.1 Section 1: Workspace Management Commands

Section 1 of the code aims to ensure a clean and reset workplace, as shown in Figure 5, including the following commands:

- **clc**: Clears the command window, provides a clean view for new inputs and outputs.
- **clear**: Clears all variable from the workspace, ensures no leftover data from previous run.
- **close all**: Close all open figure windows.

```
2 %% 1. Workplace Management Commands
3 clc % Clears all text from the command window
4 clear % Removal of all variables in current workspace
5 pause(0.5)
6 close all % Close all figure windows
```

Figure 5: Image of Section 1 Workspace Management Commands.

6.2 Section 2: Defining the Material Properties

Table 1 provides the material properties for aluminium:

Table 1: Provided Aluminium Properties.

Material Property	Value
Conductivity (k)	143 W/m.C
Density (ρ)	2.8 x 10 ³ kg/m ³
Specific Heat (c)	795 J/kg.C

Thermal diffusivity was calculated using Equation (19).

$$a = \frac{k}{\rho c} \quad (\text{Eq. 19})$$

Figure 6 illustrates the defined material properties and the thermal diffusivity equation.

```
8 %% 2. Define Material Properties
9 k=143; % Conductivity of Aluminium (W/m.C)
10 rho=2.8e3; % Density Value (kg/m^3)
11 cp=795; % Specific Heat Value (J/kg.C)
12 alpha=k/(rho*cp); % Thermal diffusivity (m^2/s)
```

Figure 6: Image of Section 2 Defining the Material Properties.

6.3 Spatial Consideration

6.3.1 Defining plate dimensions

The code allows users to input the width (Xdim) and length (Ydim) of an aluminium plate, ensuring the entries are numeric and positive, as show in Figure 7. There were no maximum limits applied.

```
14 %% 3. User inputs for Spatial Consideration
15 Xdim=input('Please enter the width (Xdim) of the plate in metres:'); % Prompt for user to enter material width value Xdim
16 while ~isnumeric(Xdim) || Xdim <=0 % Input condition, ensuring entry is recognised as a number and is positive
17     fprintf('Xdim must be a positive number.\n'); % Automated response if conditions are not met
18     Xdim=input('Please enter the width (Xdim) of the plate in metres:'); % Re-prompt for user to enter material width value Xdim
19 end
20
21 Ydim=input('Please enter the width (Ydim) of the plate in metres:'); % Prompt for user to enter material width value Ydim
22 while ~isnumeric(Ydim) || Ydim <=0 % Input condition checking entry is recognised as a number and is positive
23     fprintf('Ydim must be a positive number.\n'); % Automated response if conditions are not met
24     Ydim=input('Please enter the width (Ydim) of the plate in metres:'); % Re-prompt for user to enter material width value Ydim
25 end
```

Figure 7: Image of Section 3 showing Xdim and Ydim inputs including the set conditions.

The code includes automated responses notifying users when input conditions are invalid and prompts them to provide a correct entry. Execution will not proceed until a suitable input is submitted.

6.3.2 Defining Node Distance

The code prompts for the user to define the grid spacing (h) and applied several conditions to validate the input. As illustrated in Figure 8, the following criteria must be satisfied:

- The entry must be numerical.
- The entry must evenly divide both the previously inputted Xdim and Ydim.
- The entry must be a positive number.
- The entry must not be equal to the previously inputted Xdim or Ydim.

```
26 % Grid spacing validation
27 while ~isnumeric(h) || mod(Xdim, h) ~= 0 || mod(Ydim, h) ~= 0 || h<=0 || h==Xdim || h==Ydim % Input conditions
28     if h<=0
29         fprintf('Grid spacing must be a positive number.\n'); % Automated response if a negative number is entered
30     elseif mod(Xdim,h)~=0 || mod(Ydim,h)~=0
31         fprintf('Grid spacing must divide evenly into both Xdim and Ydim. It can also not be equal to Xdim or Ydim.\n'); % Automated response if conditions are not met
32     elseif h==Xdim || h==Ydim
33         fprintf('Grid spacing cannot be equal to Xdim or Ydim. It can also not be equal to Xdim or Ydim.\n'); % Automated response if conditions are not met
34     end
35     h=input('Please enter your grid spacing in metres:'); %Re-prompt for grid spacing value
36 end
```

Figure 8: Image of Section 3 showing grid spacing inputs including the set conditions.

6.4 Compute Number of Rows and Columns

The code utilises the previously entered values for Xdim, Ydim and h to calculate the required number of matrix rows and columns. As show in Figure 9, these calculations are subsequently used in the next stage to initialise the T1 temperature matrix.

```
41 %% 4. Compute the number of rows and columns
42 Rows=(Ydim/h)+1; % Matrix rows
43 Columns=(Xdim/h)+1; %Matrix Columns
```

Figure 9: Image of Section 4 showing calculation for rows and columns.

6.5 Temperature Matrix T1

The temperature matrix T1 is initialised using the row and column values computed in Section 4. The code then prompts the user to enter a fixed north boundary temperature T_0 . As show in Figure 10, several conditions are applied to validate the input, ensuring that the entry is both numerical and positive.

```

45 %% 5. Temperature Matrix T1
46 T1=zeros(Rows,Columns); % Initialise matrix "T1"
47 T_North=input('Please define the fixed temperature for the North boundary in °C:'); % Prompt for user to define boundary temperature for the North boundary
48 while ~isnumeric(T_North) || T_North<=0 % Input condition checking entry is a number and is a positive number
49     fprintf('The fixed temperature must be a positive number.\n'); % Automated response if conditions are not met
50     T_North=input('Please define the fixed temperature for the North boundary in °C:'); % Re-prompt for user to define boundary temperature for the North boundary
51 end
52 T_South=-T_North; % T_South to be the negative value of T_North
53 T1(1,:)=T_North; % Applying dirichlet boundary conditions to the north boundary (Top boundary row 1)
54 T1(end,:)=T_South; % Applying dirichlet boundary conditions to the south boundary (Bottom boundary last row)
55 disp('Temperature matrix after applying boundary conditions:'); % Text to be displayed in command window
56 disp(T1); % Update T1 to be displayed with applied boundary conditions

```

Figure 10: Image of Section 5 showing Temperature Matrix T1 Code.

As stated in the assignment brief the south boundary of the plate is equal to $-T_0$. Therefore, the code automatically determines the south boundary temperature (T_{South}) based off the user input for the North boundary (T_{North}).

The code applies Dirichlet boundary conditions to the north and south edges of the temperature matrix T1, imposing fixed temperatures. The north boundary receives a specific heat input, while the south boundary maintains an opposing temperature, enabling heat transfer modelling. T1 is initialised based on user inputs, as shown in Figure 11.

```

Command Window
Please enter the width (Xdim) of the plate in metres:
12
Please enter the width (Ydim) of the plate in metres:
12
Please enter your grid spacing in metres:
2
Please define the fixed temperature for the North boundary in °C:
50
Temperature matrix after applying boundary conditions:
 50   50   50   50   50   50   50
  0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  0    0    0    0    0    0    0
  0    0    0    0    0    0    0
-50  -50  -50  -50  -50  -50  -50

```

Figure 11: Temperature matrix T1 with applied inputs.

6.6 Temporal Consideration

The code enables the user to select a timestep (ΔT) for the model. To ensure valid input, the code incorporates conditions that prevent the selection of non-numeric or negative values, as show in Figure 12.

```

58 %% 6. User input for temporal consideration
59 dT=input('Please enter timestep (dT):'); % Prompt for user to enter time step dT
60 while ~isnumeric(dT) || dT<=0 % Input condition checking entry is a number and is a positive number
61     fprintf('The timestep must be a positive number.\n'); % Automated response if conditions are not met
62     dT=input('Please enter timestep (dT):'); % Re-prompt for user to enter time step dT
63 end

```

Figure 12: Image of Section 6 User ΔT input with conditions

The assignment brief specifies a stability criterion requiring a Fourier number (F_0) less than 0.5 before proceeding further. To enforce this, the code as shown in Figure 13 ensures that the selected ΔT value does not exceed the maximum permissible threshold.

```
Fo=alpha*dT/h^2; % Fourier calculation
dT_max=0.25*h^2/alpha;
while Fo>0.25 % Stability criteria
    fprintf('Time step %.5f exceeds the maximum allowed value of %.5f.\n',dT,dT_max);
    dT=input('Please enter a time step below the maximum value:'); %Prompt for user to enter suitable time step
    Fo=alpha*dT/h^2; % Recalculation of Fo with alternative dT
end
```

Figure 13: Image of Section 6 Stability Criterion Check.

The code is designed to be user friendly, with prompts intended to guide the user throughout the process. As shown in Figure 14, in the event that an unsuitable ΔT is selected, the re-prompt message provide information on the maximum allowable value to assist the user.

[illegible]

Figure 14: Image of re-prompt message for an unsuitable ΔT entry.

6.7 Initialisation of Temperature Matrix

6.7.1 Implementation of the Neumann Boundary Conditions

The temperature matrix T2 initialisation maintains constant Dirichlet boundary conditions on the north and south edges, reusing the T1 setup. Neumann boundary conditions are applied on the east and west edges by modifying the EFDM equation. The terms “j=2:Columns-1” and “i=2:Rows-1” update only the interior points, simulating a zero-flux condition for insulated edges. The EFDM equation, shown in Figure 16 and represented by Equation 20, updates the temperature matrix T2.

```
78 T2=T1; % T2 will keep the same boundary conditions as T1 matrix
79 for i=2:Rows-1 % Update internal points using the explicit finite difference formula
80     for j=2:Columns-1
81         T2(i,j)=T1(i,j)+Fo*(T1(i+1,j)+T1(i-1,j)+T1(i,j+1)+T1(i,j-1)-4*T1(i,j));
82     end
```

Figure 15: Image of the modified EFDM equation to apply Neumann Boundary Conditions.

```

        j=1;
        T2(i,j)=T1(i,j)+Fo*(T1(i+1,j)+T1(i-1,j)+T1(i,j+1)+T1(i,j-1)-4*T1(i,j));
        j=Columns;
        T2(i,j)=T1(i,j)+Fo*(T1(i+1,j)+T1(i-1,j)+T1(i,j-1)+T1(i,j+1)-4*T1(i,j));
    end
end

```

Figure 16: Final EFDM equation implemented in the code.

6.8 Convergence Criterion Threshold

The assignment didn't specify a convergence criterion, but an example value of 0.001 was provided. To ensure consistency, the code uses this value directly without user input. The convergence criterion determines when the simulation stops, based on the maximum acceptable temperature change. The loop ends once the largest temperature difference between matrices T1 and T2 is below the threshold, indicating thermal equilibrium. The code starts with "max_diff=inf" to ensure the condition "max_diff>=Thermal_Equil" is true, and updates "max_diff" as "max(max(abs(T2-T1)))". When "max_diff" is below the threshold, the simulation stops as shown in Figure 17.

```
Thermal_Equil=0.001; % Set threshold for thermal equilibrium
max_diff=inf; % Initialised to "inf" guarantees loop will run at least once
time_step=0; % Will track the number of time steps the simulation has gone through
figure(1)
while max_diff>=Thermal_Equil % Time stepping loop to run until thermal equilibrium
    T2=T1; % T2 will keep the same boundary conditions as T1 matrix
    for i=2:Rows-1 % Update internal points using the explicit finite difference formula
        for j=2:Columns-1
            T2(i,j)=T1(i,j)+Fo*(T1(i+1,j)+T1(i-1,j)+T1(i,j+1)+T1(i,j-1)-4*T1(i,j));
        end
    end
    T1=T2;
    T2(i,j)=T1(i,j)+Fo*(T1(i+1,j)+T1(i-1,j)+T1(i,j+1)+T1(i,j-1)-4*T1(i,j));
    j=Columns;
    T2(i,j)=T1(i,j)+Fo*(T1(i+1,j)+T1(i-1,j)+T1(i,j-1)+T1(i,j-1)-4*T1(i,j));
end
max_diff=max(max(abs(T2-T1))); % Calculate the maximum difference between T1 and T2
T1=T2; % Update temperature matrix with the new values
time_step=time_step+1; % Time step counter
Thermal_Equil_Seconds=time_step*dt; % Calculate seconds taken to achieve thermal equilibrium
```

Figure 17: Image of relevant code lines to the Convergence Criterion Threshold.

6.9 Calculation of T2

The code calculates T2 using the user input and the EFDM equation. Once executed, the T2 calculation generates a visual representation of the temperature matrix, provided in Appendix B. The code also outputs the updated T2 matrix, showing changes to the interior points, as shown in Figure 18.

```
12
Please enter your grid spacing in metres:
2
Please define the fixed temperature for the North boundary in °C:
50
Temperature matrix after applying boundary conditions:
50 50 50 50 50 50 50
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
-50 -50 -50 -50 -50 -50 -50

Please enter timestep (dT):
3600
Numerical Solution Temperature Distribution at Thermal Equilibrium (T2 Matrix):
50.0000 50.0000 50.0000 50.0000 50.0000 50.0000 50.0000
33.3178 33.3178 33.3178 33.3178 33.3178 33.3178 33.3178
16.6512 16.6512 16.6512 16.6512 16.6512 16.6512 16.6512
0 0 0 0 0 0 0
-16.6512 -16.6512 -16.6512 -16.6512 -16.6512 -16.6512 -16.6512
-33.3178 -33.3178 -33.3178 -33.3178 -33.3178 -33.3178 -33.3178
-50.0000 -50.0000 -50.0000 -50.0000 -50.0000 -50.0000 -50.0000
```

Figure 18: Image of completed T2 matrix after code execution.

The code also presents the time required to achieve thermal equilibrium in a user friendly format, as shown in Figure 19.

Time to reach thermal equilibrium: 5 days, 4 hours, 0 minutes, 0 seconds

Figure 19: Time to reach thermal equilibrium output.

6.10 Analytical Solution

The code sets up a third temperature matrix for the analytical solution TA, based on Equation (18) as shown in Figure 20. The assignment brief did not specify the number of terms to sum, 100 terms were chosen to refine the temperature profile and enhance accuracy.

```

106 %% 10. Analytical Solution
107 TA=zeros(Rows,Columns); % Initialise Matrix "TA"
108 for m = 1:Rows
109     for n = 1:Columns
110         x = (n-1) * h; % Convert column index to x-coordinate
111         y = (m-1) * h; % Convert row index to y-coordinate
112         T_analytical = -T_North + (2 * T_North / Ydim) * y; % Base linear profile
113
114         for k = 1:100 % Sum up to 100 terms for better accuracy
115             term = (1 / ((100 * k * pi)^2)) * (1 - (-1)^k) * ...
116                 sin(k * pi * y / Ydim) / sinh(k * pi * Xdim / Ydim) * ...
117                 (cosh(k * pi * x / Ydim) - cosh(k * pi * (x - Xdim) / Ydim));
118             T_analytical = T_analytical + term;
119         end
120
121         TA(m, n) = T_analytical; % Store result in the matrix
122     end
123 end

```

Figure 20: Analytical solution implemented in the code.

The outputs of the updated TA matrix, as show in Figure 21.

```

Analytical Solution Temperature Distribution at Thermal Equilibrium (TA Matrix):
50.0000  50.0000  50.0000  50.0000  50.0000  50.0000  50.0000
33.3333  33.3333  33.3333  33.3333  33.3333  33.3333  33.3333
16.6667  16.6667  16.6667  16.6667  16.6667  16.6667  16.6667
-0.0000  -0.0000  -0.0000  0  0.0000  0.0000  0.0000
-16.6667 -16.6667 -16.6667 -16.6667 -16.6667 -16.6667 -16.6667
-33.3333 -33.3333 -33.3333 -33.3333 -33.3333 -33.3333 -33.3333
-50.0000 -50.0000 -50.0000 -50.0000 -50.0000 -50.0000 -50.0000

```

Figure 21: Update TA matrix output in the code.

Additionally, the code calculates the maximum temperature difference between T2 and TA matrices to validate the accuracy of the T2 matrix, as shown in Figure 22 and output shown in Figure 23.

```

173 % Calculate and display the maximum error
174 max_error = max(Difference(:));
175 fprintf('Maximum Error between Numerical and Analytical solutions: %.5f\n', max_error);

```

Figure 22: Maximum temperature difference calculation T2 and TA temperature distribution.

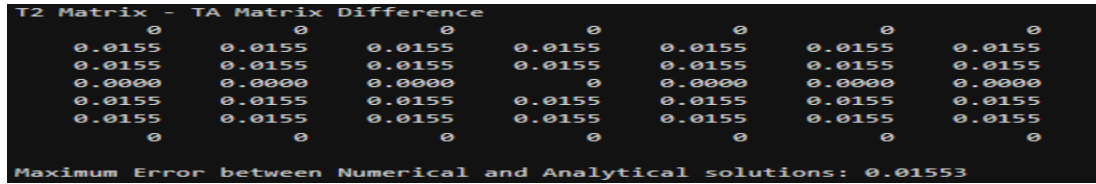


Figure 23: Maximum difference matrix code output.

7 Code Validation

The code was validated against the analytical solution by calculating the maximum temperature difference between the numerical solution (T2) and the analytical solution (TA). Two code executions were performed: Example 1 used the inputs from Table 1, and Example 2 used a different set of inputs from Table 2. A convergence criterion of 0.002 was applied for both executions.

Table 2: User inputs for Example 1 code validation execution.

User Input Requirements	Inputted Values
Width (Xdim) - m	12
Length (Ydim) - m	12
North Boundary Temperature - °C	50
Grid Spacing (h) - m	2
Time Step (dt) - Seconds	3600

Table 3: User inputs for Example 2 code validation execution.

User Input Requirements	Inputted Values
Width (Xdim) - m	15
Length (Ydim) - m	15
North Boundary Temperature - °C	50
Grid Spacing (h) - m	3
Time Step (ΔT) - Seconds	2000

The complete code output for Example 1 validation is provided in Appendix C, and the output for Example 2 is in Appendix D. These appendices include the updated matrices for T2, TA, and the T2-TA difference matrix. In summary, the examples concluded the following:

Example 1

- Time to reach thermal equilibrium: 5 days, 4 hours, 0 minutes, 0 seconds.
- Maximum temperature difference between T2 and TA: 0.01553 °C.

Example 2

- Time to reach thermal equilibrium: 7 days, 7 hours, 0 minutes, 0 seconds.
- Maximum temperature difference between T2 and TA: 0.04923 °C.

The validation of the numerical solution T2 against the analytical solution TA shows good model performance with small maximum differences. In Example 1, the maximum difference was 0.01553°C, indicating high accuracy and reliable convergence. This suggests the EFDM is well implemented and the grid spacing and time step are sufficient.

In Example 2, the maximum difference increased to 0.04923°C, due to grid spacing and time step. Reducing the grid spacing to 1m lowered the maximum difference to 0.04319°C, and increasing the time step to 3600 seconds reduced it further to 0.02371°C. This highlights the sensitivity to user inputs, particularly grid spacing and time step.

Further analysis and discussion on code limitations are in Sections 8 and 9. In conclusion, T2 closely matches TA, confirming the model's accuracy and reliability, with varying inputs demonstrating the impact of user choices.

8 Sensitivity Analysis of the Code

A sensitivity analysis was conducted to evaluate the impact of grid spacing (h), time step (Δt) and the convergence criterion. The primary focus was evaluating the time required for the numerical solution (T2) to reach thermal equilibrium. The maximum temperature difference was recorded during the sensitivity analysis to further validate the code.

Dimensions were given as 12 m (Xdim) by 12 m (Ydim). The north boundary temperature was not specified in the brief and was fixed at 50°C for all sensitivity analysis cases to maintain consistency.

8.1 Influence of the Grid Spacing

The first sensitivity analysis focused on grid spacing (h), which defines the spatial resolution of the model. Grid spacing was varied as the input, while all other parameters remained fixed. The code was executed 13 times, reducing h incrementally from 4 m to 0.2 m. The results of the analysis are summarized in Table 3.

Table 4: Sensitivity Analysis Results for Varying Grid Spacing

Grid Spacing (h) Metres	Time Step (Δt) Seconds	Convergence Criterion (°C) °C	Thermal Equilibrium Time (Seconds)	Max temp diff T2 – TA (°C)
--------------------------------	-------------------------------------	-------------------------------------	--	-------------------------------

4	150	0.001	282600	0.55234
3	150	0.001	278700	0.46580
2	150	0.001	255150	0.41325
1.5	150	0.001	258600	0.39717
1.2	150	0.001	253200	0.38927
1	150	0.001	254700	0.38572
0.75	150	0.001	253350	0.38167
0.6	150	0.001	252600	0.38064
0.5	150	0.001	252300	0.37937
0.4	150	0.001	251700	0.37861
0.3	150	0.001	251850	0.37761
0.25	150	0.001	251700	0.37779
0.2	150	0.001	251700	0.37713

Figure 24 presents the results graphically, with the thermal equilibrium time plotted on the Y-axis and the h variable along the X-axis.

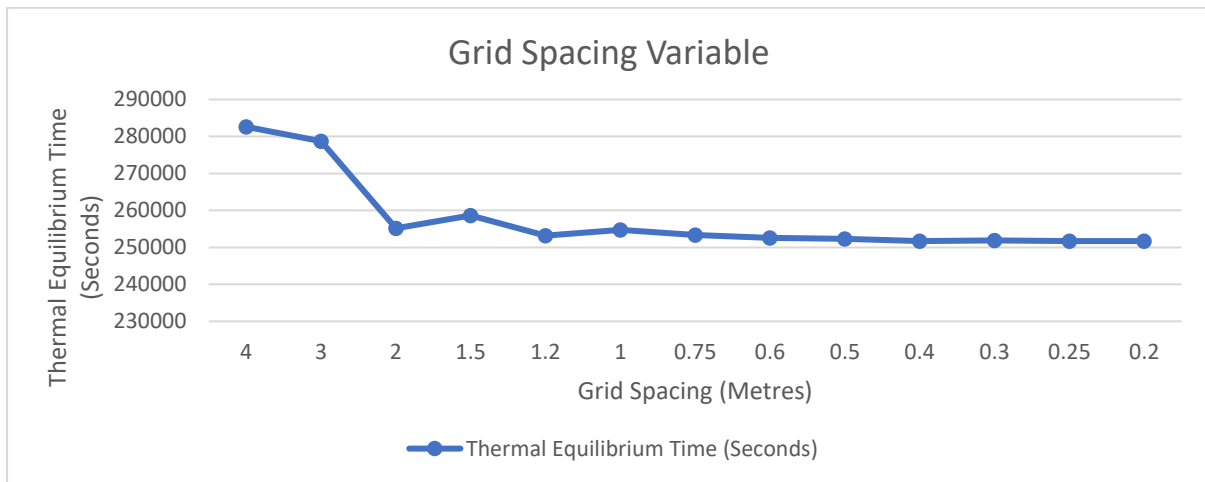


Figure 24: Sensitivity Analysis Results for Varying Grid Spacing Graph.

The results and graphs demonstrate that as h decreases, the thermal equilibrium time initially drops significantly before stabilising at smaller grid spacings. At larger grid spacings like 4m and 3m, the thermal equilibrium time is higher, reflecting increased truncation error due to the higher h value.

As h is reduced below 2m, the thermal equilibrium time begins to converge resulting in minimal variation between 1m to 0.2m. The stabilisation suggests that the grid resolution is sufficient to capture the heat diffusion accurately without further significant gains reducing h further.

8.2 Influence of the Time Step

Sensitivity analysis was performed focusing on the alteration of the time step (Δt) reducing it from 2000 to 25 seconds. The results of the analysis are summarised in Table 5.

Table 5: Sensitivity Analysis Results for Varying Time Step

Grid Spacing (h) Metres	Time Step (Δt) Seconds	Convergence Criterion ($^{\circ}\text{C}$) $^{\circ}\text{C}$	Thermal Equilibrium Time (Seconds)	Max temp diff $T_2 - T_A$ ($^{\circ}\text{C}$)
0.75	2000	0.0001	528000	0.02746
0.75	1500	0.0001	513000	0.03722
0.75	1000	0.0001	492000	0.05590
0.75	800	0.0001	480000	0.06999
0.75	600	0.0001	463800	0.09422
0.75	400	0.0001	441200	0.14267
0.75	200	0.0001	402000	0.28604
0.75	100	0.0001	362400	0.57410
0.75	50	0.0001	322600	1.14921
0.75	25	0.0001	282775	2.29876

Figure 24 presents the results graphically, with the thermal equilibrium time plotted on the Y-axis and the time step variable along the X-axis.

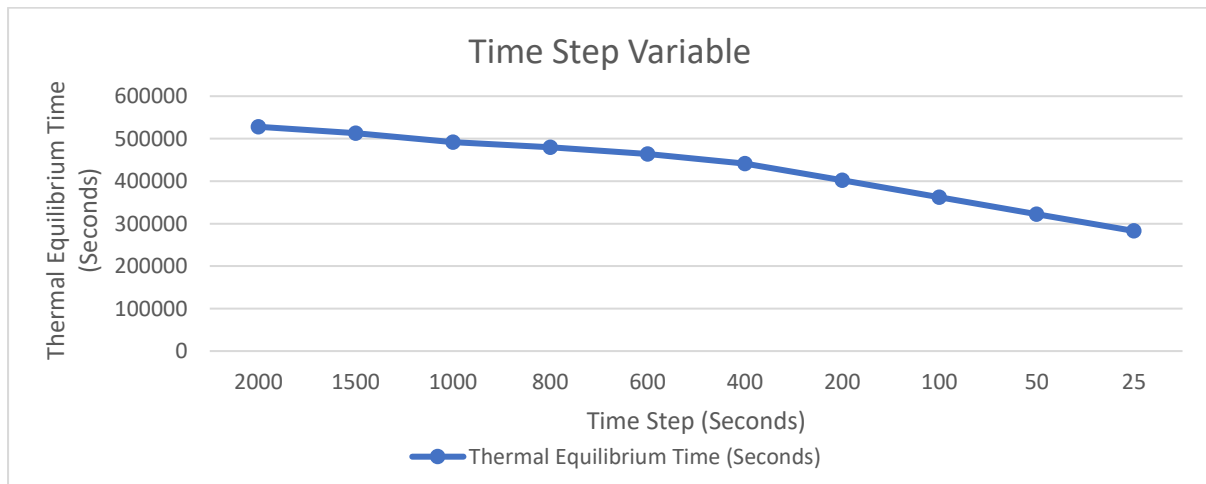


Figure 25: Sensitivity Analysis Results for Varying Time Step Graph.

The results indicate that reducing the time step (Δt) decreases the time to reach thermal equilibrium, allowing the simulation to converge faster. However, smaller time steps also increase the maximum temperature difference between the numerical solution (T_2) and the analytical solution (T_A).

Smaller time steps require more iterations, leading to an increased number of EFDM approximations. These approximations introduce truncation errors, which can accumulate over time, as seen in the results. Additionally, rounding errors can contribute, as more computations provide more opportunities for such errors (Burden & Faires, 2016). Therefore, while a reduced time step accelerates convergence, it comes at the cost of reduced accuracy due to the greater number of iterations.

8.3 Influence of the Convergence Criterion

Final sensitivity analysis was performed on the impact of the convergence criterion. The convergence criterion was reduced from 0.1 to 0.000001°C. The results of the analysis are summarised in Table 6.

Table 6: Sensitivity Analysis Results for Varying Convergence Criterion

Grid Spacing (<i>h</i>) Metres	Time Step (Δt) Seconds	Convergence Criterion (°C) °C	Thermal Equilibrium Time (Seconds)	Max temp diff T2 – TA (°C)
1	500	0.1	57500	11.50551
1	500	0.01	191000	1.14485
1	500	0.001	324000	0.11488
1	500	0.0001	457500	0.01144
1	500	0.00001	590500	0.00116
1	500	0.000001	723500	0.00013

Figure 25 presents the results graphically, with the thermal equilibrium time plotted on the Y-axis and the convergence criterion variable along the X-axis.

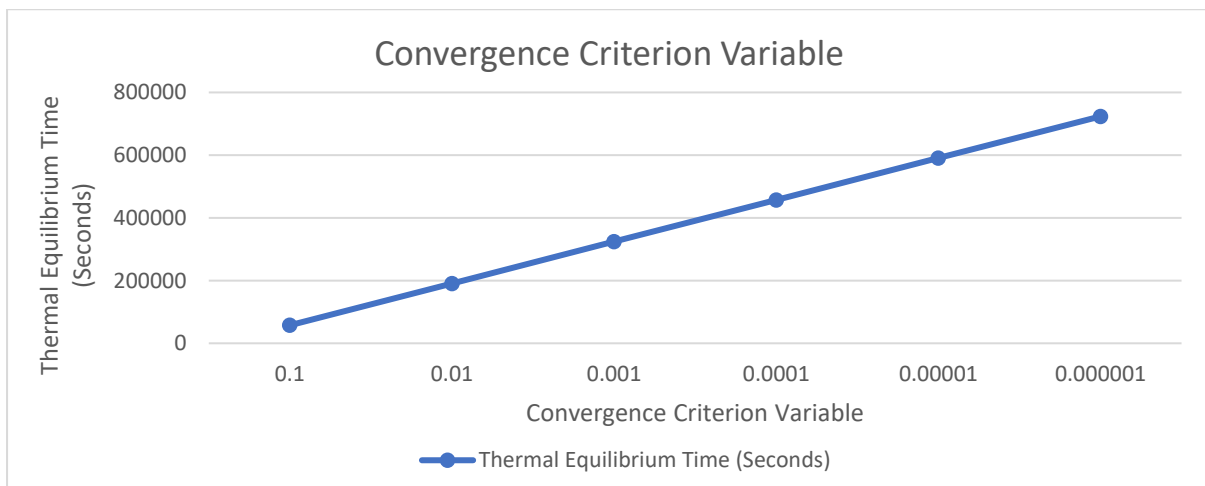


Figure 26: Sensitivity Analysis Results for Varying Convergence Criterion Graph.

The analysis shows that as the convergence criterion is reduced, the time required to reach thermal equilibrium increases. At the same time, the maximum temperature difference between the numerical solution (T2) and the analytical solution (TA) decreases significantly, suggesting improved accuracy in the numerical solution with a smaller, stricter convergence criterion.

Reducing the convergence criterion enhances the precision of the numerical solution by allowing smaller temperature changes before concluding the simulation, improving result accuracy (Chapra & Canale, 2015). Therefore, a balance between computational time and precision must be considered when setting the convergence criterion.

9 Limitations of the Mode

9.1 Round-off Error

Round off error, occurs when numerical values are approximated to fit a finite number of digits. This can cause a loss of precision within the model, especially when using iterative numeric methods such as EFDM. These round-off errors can accumulate over time and often the discrepancies can be difficult to detect (Chapra, 2017). As a result the calculation can continue to progress which can lead to a reduction in the accuracy of the final result (Ascher & Greif, 2011).

9.2 Truncation Error

Truncation errors are inherent to EFDM due to the truncation of Taylor series expansions. The forward difference approximation has an error of $O(h)$, while the central difference approximation has an error of $O(h^2)$. As the truncation error depends on grid spacing (h), reducing h decreases the error as shown in the sensitivity analysis performed in Section 8.1. The forward difference error decreases linearly, while the central difference error decrease quadratically, which makes the central difference approximate more accurate.

9.3 Stability

A stable numerical scheme ensures that errors do not grow uncontrollably as the computation progresses (Tannehill et al., 1997). The EFDM is conditionally stable, meaning its stability depends on the proper selection of grid spacing and time step. The Courant-Friedrichs-Lewy (CFL) condition defines the relationship between the time step and grid spacing (Kubrusly & de Moura, 2010). In this model, the Fourier number, with a limit of 0.25, is used to ensure stability, as it simulates heat transfer rather than fluid dynamics, where the CFL condition is typically applied.

9.4 Complex Geometries

The EFDM dependency on the use of structured grids reduces the ability to apply this code to curved boundaries or complex geometries. Within complex geometries the boundary is not aligned with the grid, this misalignment can cause approximations that misrepresent the actual geometry leading to discrepancies in the numerical solution (Smith, 1985).

9.5 Coding Limitations

The approach of defining thermal equilibrium by a minimum temperature difference between the matrices is a simplified version. Thermal equilibrium involves the stabilization of energy within the system, where heat transfer ceases and no net energy flow occurs (Finn, 2017). This method fails to account for energy conservation or the spatial distribution of

temperature which are key components of true equilibrium. Additionally, relying solely on small temperature differences can lead to premature conclusions of equilibrium, overlooking ongoing heat transfer or slow convergence.

10 Suggested Improvements and Future Developments

10.1 Use of Alternative Finite Difference Methods (FDM)

Alternating Direction Implicit (ADI) and Implicit Finite Difference Method (FDM) are alternative FDM for solving the partial differential heat equation. Table 7 compares these methods to EFDM, based on information from “*Numerical Differential Equations: Theory and Technique, ODE Methods, Finite Differences, Finite Elements and Collocation*” (Loustau, 2016).

Table 7: Comparison of ADI, Implicit FDM and EFDM

Feature	ADI	Implicit FDM	EFDM
Stability	Unconditionally stable	Unconditionally stable	Conditionally stable
Applicability	Multi-dimensional and time dependent problems	Suitable for wide range of problem types	Simple geometries and low dimensions
Accuracy	Good accuracy for large time steps	High accuracy	High accuracy for small time steps
Complexity	More complex to implement (alternating directions)	Easier to implement for 1D, more challenging for multi-dimensional	Simple for structured grid, complex for irregular grids
Grid Requirements	Works best with uniform grids	Works with both uniform and non-uniform grids	Primarily for uniform grids
Boundary Conditions	Works well for regular boundaries	Handles various boundary types	Works well for simple boundaries

10.2 Use of Numerical Alternatives

The use of numerical alternative to FDM could be an improvement to enable the handling of complex geometries. Table 8 provides a comparison of a selection of numerical alternatives, Finite Element Method (FEM), Finite Volume Method (FVM) and Boundary Element Method (BEM). The comparison has been performed based on information gathered from “*Advanced Numerical and Semi-Analytical Methods for Differential Equations*” (Chakraverty, et al., 2019).

Table 8: Comparison of FEM, FVM and BEM

Feature	FEM	FVM	BEM
Stability	Conditioned stability	Condition stability	Conditioned stability
Applicability	Complex geometries	Flow, conservation and material movement	Complex boundaries
Accuracy	High for complex geometries	High for flow conservation and material movement	High for complex boundaries
Complexity	High requires meshing	Moderate	High – Boundary discretisation and solving of integral equations
Grid Requirements	Unstructured grids	Structured and unstructured grids	Boundary discretisation only
Boundary Conditions	Handle complex conditions well	Handle complex conditions well	Specialises in complex boundaries

10.3 Alternatives to Calculating Thermal Equilibrium

A limitation to the code is defining thermal equilibrium by a minimum temperature difference between the matrices. Alternative approaches to this could potentially improve the accuracy of the model if implemented. Table 9 provides potential improvement opportunities to more accurately determine in the model when thermal equilibrium has been achieved.

Table 9: Alternatives to Calculating Thermal Equilibrium

Alternative Approach	Approach Description
Time-Weighted Average of Temperature Changes	Calculate the average temperature change over multiple steps to smooth fluctuations and provide a more stable indication of equilibrium (Patankar, 1980).
Temperature Gradient Assessment	Monitor the rate of change over distance, a small gradient indicates the system is close to equilibrium (Incropera & DeWitt, 2007).
Experimental Validation	Compare numerical results with experimental data to ensure model accuracy and capture real world thermal behaviour (Taylor & Hughes, 1998).
Use of AI	Use machine learning to identify patterns in temperature data, improving the predictions of when equilibrium is reach (Zhang & Xie, 2020).

11 Conclusion

This report successfully developed and evaluated a 2D heat diffusion model using EFDM. The model effectively simulated heat diffusion within an aluminium plate, applying both Dirichlet and Neumann boundary conditions to achieve thermal equilibrium. Validation of the numerical solution against the analytical solution confirmed the accuracy and reliability. Sensitivity analysis highlighted the influence of key user inputs such as grid spacing, time step and convergence criteria on the numerical solution performance. While the EFDM demonstrated effectiveness in comparison to the analytical solution, its limitations were also acknowledged. Particularly, its conditional stability compared to the unconditionally stable ADI and implicit FDM methods.

The report's objectives were addressed throughout. First, the heat equation was introduced and its importance was explained. Next, the discretisation process was explained, demonstrating how the Taylor series expressions were transposed and truncated for use in EFDM. Resulting in the development of approximation expressions that were able to be used to replace the derivatives. The MATLAB implementation was outlined step by step, with emphasis placed on the built in condition to ensure input stability and accuracy. Additionally, sensitivity analysis demonstrated the impact of varying inputs on model performance, while the validation through comparison with the analytical solution revealed a relatively small maximum difference, indicating the model's reliability.

In conclusion, the report achieved its aim of developing and evaluating a heat diffusion model based on EFDM. The report also addressed the model's limitation and briefly reviewing potential future improvements for future development of the code to allow the code to become more accurate, adaptable and reliable.

References

- Ascher, U. M. & Greif, C., 2011. *A First Course in Numerical Methods*. Vancouver: Society for Industrial and Applied Mathematics.
- Burden, R. L. & Faires, J. D., 2016. *Numerical Analysis*. 10th Hrsg. Boston: Cengage.
- Causon, D. M. & Mingham, C. G., 1997. *Introductory Finite Difference Methods for PDEs*. 1st Hrsg. Dorrecht: Springer .
- Chakraverty, S., Mahato, N., Karunakar, P. & Rao, T. D., 2019. *Advanced Numerical and Semi-Analytical Methods for Differential Equations*. 1st Hrsg. New Jersey: Wiley.
- Chang, C. Y. & Ma, C. C., 2001. Transient thermal conduction analysis of a rectangular plate with multiple insulated cracks by the alternating method. *International Journal of Heat and Mass Transfer*, Band 44, pp. 2423-2437.
- Chapra, S. C., 2017. *Applied Numerical Methods with MATLAB for Engineers and Scientists*. 4th Hrsg. New York: McGraw-Hill Education .
- Chapra, S. C. & Canale, R. P., 2015. *Numerical methods for engineers*. 7th Hrsg. New York: McGraw-Hill Education.
- Croft, D. R. & Lilley, D. G., 1977. *Heat Transfer Calculations using the Finite Difference Equations*. 1 Hrsg. London: Applied Science Publishers Ltd.
- Evans, L. C., 2022. *Partial Differential Equations*. 2nd Hrsg. Rhode Island : American Mathematical Society.
- Ferziger, J. H. & Peric, M., 2002. *Computational Methods for Fluid Dynamics*. 3rd Hrsg. Berlin: Springer-Verlag.
- Finn, C. B., 2017. *Thermal Physics*. 3rd Hrsg. Boca Raton: CRC Press.
- Guerrero, J. P., 2021. *From the Classical to the Stochastic Heat Equation*, Barcelona: University of Barcelona.
- Haftka, R. T. & Gurdal, Z., 2012. *Elements of Structural Optimization*. 3rd Hrsg. Alphen aan den Rijn: Kluwer.
- Incropera, F. P. & DeWitt, D. P., 2007. *Introduction to Heat Transfer*. 5th Hrsg. New Jersey: Wiley.
- Koshizuka, S., Shibata, K., Kondo, M. & Matsunaga, T., 2018. *Moving Particle Semi-implicit Method*. 1 Hrsg. Tokyo: Elsevier Inc. .
- Kubrusly, C. S. & de Moura, C. A., 2010. *The Courant–Friedrichs–Lewy (CFL) Condition: 80 Years After Its Discovery*. 1st Hrsg. Basel: Birkhäuser.
- Loustau, J., 2016. *Numerical Differential Equations: Theory and Technique, ODE Methods, Finite Differences, Finite Elements and Collocation*. 1st Hrsg. Singapore: World Scientific Publishing Company .
- Manglik, R., 2024. *Fundamentals of Numerical Analysis*. 3rd Hrsg. Lucknow: EduGorilla Publication.
- Özişik, M. N., 2017. *Finite Difference Methods in Heat Transfer*. Florida : CRC Press.

- Patankar, S. V., 1980. *Numerical Heat Transfer and Fluid Flow*. 1st Hrsg. New York: Hemisphere Publishing Corporation.
- Recktenwald, G. W., 2004. *Finite-Difference Approximations to the Heat Equation*, Portland: Portland State University.
- Smith, G. D., 1985. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. 3rd Hrsg. Oxford: Oxford University Press.
- Tannehill, J. C., Anderson, D. & Pletcher, R. H., 1997. *Computational Fluid Mechanics and Heat Transfer*. 2nd Hrsg. Iowa: Taylor & Francis.
- Taylor, C. & Hughes, T. J., 1998. *Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. 1st Hrsg. Mineola: Dover Publications.
- Webster, C. G., 2025. *Numerical Methods for Partial Differential Equations*. 41 Hrsg. Austin: Wiley.
- Zhang, L. & Xie, L., 2020. Artificial intelligence and machine learning in thermal management: A review.. *Applied Thermal Engineering*, Band 168, p. 114805.

Appendices

Appendix A: MATLAB Code for 2D Diffusion Equation Model

```
%% 2D Diffusion Equation Model using the Explicit Finite Difference Method (Code)
%% 1. Workplace Management Commands
clc                % Clears all text from the command window
clear              % Removal of all variables in current workspace
pause(0.5)
close all          % Close all figure windows

%% 2. Define Material Properties
k=143;             % Conductivity of Aluminium (W/m.C)
rho=2.8e3;         % Density Value (kg/m^3)
cp=795;           % Specific Heat Value (J/kg.C)
alpha=k/(rho*cp); % Thermal diffusivity (m^2/s)

%% 3. User inputs for Spatial Consideration
Xdim=input('Please enter the width (Xdim) of the plate in metres:'); % Prompt for
user to enter material width value Xdim
while ~isnumeric(Xdim) || Xdim <=0 % Input condition, ensuring entry is recognised
as a number and is positive
    fprintf('Xdim must be a positive number.\n'); % Automated response if
conditions are not met
    Xdim=input('Please enter the width (Xdim) of the plate in metres:'); % Re-
prompt for user to enter material width value Xdim
end

Ydim=input('Please enter the height (Ydim) of the plate in metres:'); % Prompt for
user to enter material width value Ydim
while ~isnumeric(Ydim) || Ydim <=0 % Input condition checking entry is recognised
as a number and is positive
    fprintf('Ydim must be a positive number.\n'); % Automated response if
conditions are not met
    Ydim=input('Please enter the width (Ydim) of the plate in metres:'); % Re-
prompt for user to enter material width value Ydim
end

h=input('Please enter your grid spacing in metres:'); % Prompt for user to enter
grid spacing

% Grid spacing validation
while ~isnumeric(h) || mod(Xdim, h) ~= 0 || mod(Ydim, h) ~= 0 || h<=0 || h==Xdim
|| h==Ydim % Input conditions
    if h<=0
        fprintf("Grid spacing must be a positive number.\n"); % Automated response
if a negative number is entered
    elseif mod(Xdim,h)~=0 || mod(Ydim,h)~=0
        fprintf("Grid spacing must divide evenly into both Xdim and Ydim. It can
also not be equal to Xdim or Ydim.\n"); % Automated response if conditions are not
met
    elseif h==Xdim || h==Ydim
        fprintf("Grid spacing cannot be equal to Xdim or Ydim. It can also not be
equal to Xdim or Ydim\n"); % Automated response if conditions are not met
    end
    h=input('Please enter your grid spacing in metres:'); %Re-prompt for grid
spacing value
end
```

```

%% 4. Compute the number of rows and columns
Rows=(Ydim/h)+1; % Matrix rows
Columns=(Xdim/h)+1; %Matrix Columns

%% 5. Temperature Matrix T1
T1=zeros(Rows,Columns); % Initialise matrix "T1"
T_North=input ('Please define the fixed temperature for the North boundary
in °C:'); % Prompt for user to define boundary temperature for the North boundary
while ~isnumeric(T_North) || T_North<=0 % Input condition checking entry is a
number and is a positive number
    fprintf('The fixed temperature must be a positive number.\n'); % Automated
response if conditions are not met
    T_North=input ('Please define the fixed temperature for the North boundary
in °C:'); % Re-prompt for user to define boundary temperature for the North
boundary
end
T_South=-T_North; % T_South to be the negative value of T_North
T1(1, :)=T_North; % Applying dirichlet boundary conditions to the north boundary
(Top boundary row 1)
T1(end, :)=T_South; % Applying dirichlet boundary conditions to the south boundary
(Bottom boundary last row)
disp('Temperature matrix after applying boundary conditions:'); % Text to be
displayed in command window
disp(T1); % Update T1 to be displayed with applied boundary conditions

%% 6. User input for temporal consideration
dT=input('Please enter timestep (dT):'); % Prompt for user to enter time step dT
while ~isnumeric(dT) || dT<=0 % Input condition checking entry is a number and is
a positive number
    fprintf('The timestep must be a positive number.\n') % Automated response if
conditions are not met
    dT=input('Please enter timestep (dT):'); % Re-prompt for user to enter time
step dT
end
Fo=alpha*dT/h^2; % Fourier calculation
dT_max=0.25*h^2/alpha;
while Fo>0.25 % Stability criteria
    fprintf('Time step %.5f exceeds the maximum allowed value
of %.5f.\n',dT,dT_max);
    dT=input('Please enter a time step below the maximum value:'); %Prompt for
user to enter suitable time step
    Fo=alpha*dT/h^2; % Recalculation of Fo with alternative dT
end

%% 7/8/9. Initialise temperature matrix T2 & Convergence Criteria Threshold &
Calculation of T2
Thermal_Equil=0.001; % Set threshold for thermal equilibrium
max_diff=inf; % Initialised to "inf" guarantees loop will run at least once
time_step=0; % Will track the number of time steps the simulation has gone through
figure(1)
while max_diff>=Thermal_Equil % Time stepping loop to run until thermal
equilibrium
    T2=T1; % T2 will keep the same boundary conditions as T1 matrix
    for i=2:Rows-1 % Update internal points using the explicit finite difference
formula
        for j=2:Columns-1
            T2(i,j)=T1(i,j)+Fo*(T1(i+1,j)+T1(i-1,j)+T1(i,j+1)+T1(i,j-1)-
4*T1(i,j));
        end
    end
end

```

```

    j=1;
    T2(i,j)=T1(i,j)+Fo*(T1(i+1,j)+T1(i-1,j)+T1(i,j+1)+T1(i,j-1)-4*T1(i,j));
    j=Columns;
    T2(i,j)=T1(i,j)+Fo*(T1(i+1,j)+T1(i-1,j)+T1(i,j+1)+T1(i,j-1)-4*T1(i,j));
  end
  max_diff=max(max(abs(T2-T1))); % Calculate the maximum difference between T1
  and T2
  T1=T2; % Update temperature matrix with the new values
  time_step=time_step+1; % Time step counter
  Thermal_Equil_Seconds=time_step*dT; % Calculate seconds taken to achieve
  thermal equilibrium
% end
  contourf(flip(T2)); % Display the temperature matrix
  colorbar; % Add a color scale
  c=colorbar;
  ylabel(c,'Temperature (°C)');
  colormap(jet); %Apply the "Jet" colour map
  title('Numerical Solution Temperature Distribution at Thermal Equilibrium (T2
  Matrix)');
  xlabel('Matrix Columns');
  ylabel('Matrix Rows');
  axis equal tight; % Ensure square grid cells
  pause(0.2)

end
disp('Numerical Solution Temperature Distribution at Thermal Equilibrium (T2
  Matrix):');
disp(T2);

%% 10. Analytical Solution
TA=zeros(Rows,Columns); % Initialise Matrix "TA"
for m = 1:Rows
  for n = 1:Columns
    x = (n-1) * h; % Convert column index to x-coordinate
    y = (m-1) * h; % Convert row index to y-coordinate
    T_analytical = -T_North + (2 * T_North / Ydim) * y; % Base linear profile

    for k = 1:100 % Sum up to 100 terms for better accuracy
      term = (1 / ((100 * k * pi)^2)) * (1 - (-1)^k) * ...
        sin(k * pi * y / Ydim) / sinh(k * pi * Xdim / Ydim) * ...
        (cosh(k * pi * x / Ydim) - cosh(k * pi * (x - Xdim) / Ydim));
      T_analytical = T_analytical + term;
    end

    TA(m, n) = T_analytical; % Store result in the matrix
  end
end
% Display Analytical Solution
disp('Analytical Solution Temperature Distribution at Thermal Equilibrium (TA
  Matrix):');
disp(flip(TA));
figure(2);
contourf(TA); % Display analytical temperature matrix
colorbar;
c=colorbar;
ylabel(c,'Temperature (°C)');
colormap(jet); %Apply the "Jet" colour map
title('Analytical Solution Temperature Distribution at Thermal Equilibrium (TA
  Matrix)');

```

```

xlabel('Matrix Columns');
ylabel('Matrix Rows');
axis equal tight;
%% Comparison of Numerical and Analytical Solutions

% Calculate the absolute difference between numerical and analytical solutions
Difference = abs(T2 - flip(TA));
disp('T2 Matrix - TA Matrix Difference');
disp(Difference);

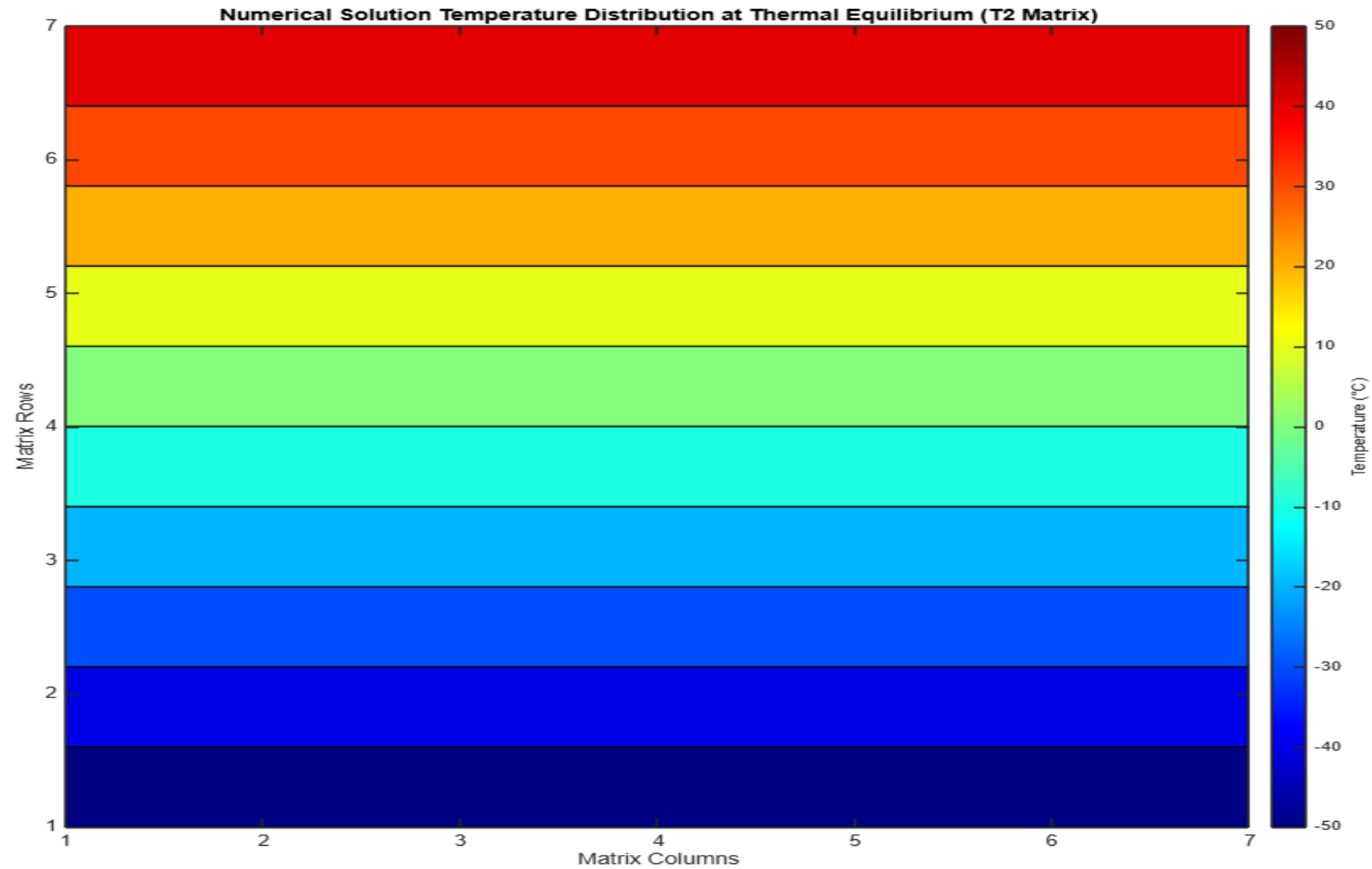
% Display the absolute difference
figure(3);
contourf(Difference); % Display difference matrix
clim([min(Difference(:)),max(Difference(:))]);
colorbar;
c=colorbar;
ylabel(c,'Temperature Difference (°C)');
colormap(jet); %Apply the "Jet" colour map
title('Difference (|T2 - TA|)');
xlabel('Matrix Columns');
ylabel('Matrix Rows');
axis equal tight;

% Calculate and display the maximum error
max_error = max(Difference(:));
fprintf('Maximum temperature difference between Numerical and Analytical
solutions: %.5f\n', max_error);

% Display thermal equilibrium time in the correct format
if Thermal_Equil_Seconds<60
    % Display in seconds
    fprintf('%d seconds\n', Thermal_Equil_Seconds);
elseif Thermal_Equil_Seconds < 3600
    % Convert to minutes and seconds
    minutes = floor(Thermal_Equil_Seconds / 60);
    seconds = mod(Thermal_Equil_Seconds, 60);
    fprintf('%d minutes, %d seconds\n', minutes, seconds);
elseif Thermal_Equil_Seconds < 86400
    % Convert to hours, minutes, and seconds
    hours = floor(Thermal_Equil_Seconds / 3600);
    minutes = floor(mod(Thermal_Equil_Seconds, 3600) / 60);
    seconds = mod(Thermal_Equil_Seconds, 60);
    fprintf('%d hours, %d minutes, %d seconds\n', hours, minutes, seconds);
else
    % Convert to days, hours, minutes, and seconds
    days = floor(Thermal_Equil_Seconds / 86400);
    hours = floor(mod(Thermal_Equil_Seconds, 86400) / 3600);
    minutes = floor(mod(Thermal_Equil_Seconds, 3600) / 60);
    seconds = mod(Thermal_Equil_Seconds, 60);
    fprintf('Time to reach thermal equilibrium: %d days, %d hours, %d minutes, %d
seconds\n', days, hours, minutes, seconds);
end

```

Appendix B: Numerical Solution Figure Output Example



Appendix C: Example 1 Validation Code Output

```

Please enter the width (Xdim) of the plate in metres:
12
Please enter the height (Ydim) of the plate in metres:
12
Please enter your grid spacing in metres:
50
Grid spacing must divide evenly into both Xdim and Ydim. It can also not be equal to Xdim or Ydim.
Please enter your grid spacing in metres:
2
Please define the fixed temperature for the North boundary in °C:
50
Temperature matrix after applying boundary conditions:
  50   50   50   50   50   50   50
   0    0    0    0    0    0    0
   0    0    0    0    0    0    0
   0    0    0    0    0    0    0
   0    0    0    0    0    0    0
   0    0    0    0    0    0    0
  -50  -50  -50  -50  -50  -50  -50

Please enter timestep (dT):
3600
Numerical Solution Temperature Distribution at Thermal Equilibrium (T2 Matrix):
  50.0000  50.0000  50.0000  50.0000  50.0000  50.0000  50.0000
  33.3178  33.3178  33.3178  33.3178  33.3178  33.3178  33.3178
  16.6512  16.6512  16.6512  16.6512  16.6512  16.6512  16.6512
   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
 -16.6512 -16.6512 -16.6512 -16.6512 -16.6512 -16.6512 -16.6512
 -33.3178 -33.3178 -33.3178 -33.3178 -33.3178 -33.3178 -33.3178
 -50.0000 -50.0000 -50.0000 -50.0000 -50.0000 -50.0000 -50.0000

Analytical Solution Temperature Distribution at Thermal Equilibrium (TA Matrix):
  50.0000  50.0000  50.0000  50.0000  50.0000  50.0000  50.0000
  33.3333  33.3333  33.3333  33.3333  33.3333  33.3333  33.3333
  16.6667  16.6667  16.6667  16.6667  16.6667  16.6667  16.6667
  -0.0000  -0.0000  -0.0000   0.0000   0.0000   0.0000   0.0000
 -16.6667 -16.6667 -16.6667 -16.6667 -16.6667 -16.6667 -16.6667
 -33.3333 -33.3333 -33.3333 -33.3333 -33.3333 -33.3333 -33.3333
 -50.0000 -50.0000 -50.0000 -50.0000 -50.0000 -50.0000 -50.0000

T2 Matrix - TA Matrix Difference
   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
  0.0155  0.0155  0.0155  0.0155  0.0155  0.0155  0.0155
  0.0155  0.0155  0.0155  0.0155  0.0155  0.0155  0.0155
  0.0000  0.0000  0.0000   0.0000   0.0000   0.0000   0.0000
  0.0155  0.0155  0.0155  0.0155  0.0155  0.0155  0.0155
  0.0155  0.0155  0.0155  0.0155  0.0155  0.0155  0.0155
   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000

Maximum temperature difference between Numerical and Analytical solutions: 0.01553
Time to reach thermal equilibrium: 5 days, 4 hours, 0 minutes, 0 seconds
  
```

Appendix D: Example 2 Validation Code Output

```

Please enter the width (Xdim) of the plate in metres:
15
Please enter the height (Ydim) of the plate in metres:
15
Please enter your grid spacing in metres:
3
Please define the fixed temperature for the North boundary in °C:
50
Temperature matrix after applying boundary conditions:
  50   50   50   50   50   50
   0    0    0    0    0    0
   0    0    0    0    0    0
   0    0    0    0    0    0
   0    0    0    0    0    0
  -50  -50  -50  -50  -50  -50

Please enter timestep (dT):
2000
Numerical Solution Temperature Distribution at Thermal Equilibrium (T2 Matrix):
  50.0000  50.0000  50.0000  50.0000  50.0000  50.0000
  29.9508  29.9508  29.9508  29.9508  29.9508  29.9508
   9.9696   9.9696   9.9696   9.9696   9.9696   9.9696
  -9.9696  -9.9696  -9.9696  -9.9696  -9.9696  -9.9696
 -29.9508 -29.9508 -29.9508 -29.9508 -29.9508 -29.9508
 -50.0000 -50.0000 -50.0000 -50.0000 -50.0000 -50.0000

Analytical Solution Temperature Distribution at Thermal Equilibrium (TA Matrix):
  50.0000  50.0000  50.0000  50.0000  50.0000  50.0000
  30.0000  30.0000  30.0000  30.0000  30.0000  30.0000
  10.0000  10.0000  10.0000  10.0000  10.0000  10.0000
 -10.0000 -10.0000 -10.0000 -10.0000 -10.0000 -10.0000
 -30.0000 -30.0000 -30.0000 -30.0000 -30.0000 -30.0000
 -50.0000 -50.0000 -50.0000 -50.0000 -50.0000 -50.0000

T2 Matrix - TA Matrix Difference
   0      0      0      0      0      0
  0.0492  0.0492  0.0492  0.0492  0.0492  0.0492
  0.0304  0.0304  0.0304  0.0304  0.0304  0.0304
  0.0304  0.0304  0.0304  0.0304  0.0304  0.0304
  0.0492  0.0492  0.0492  0.0492  0.0492  0.0492
   0      0      0      0      0      0

Maximum temperature difference between Numerical and Analytical solutions: 0.04923
Time to reach thermal equilibrium: 7 days, 7 hours, 0 minutes, 0 seconds
  
```