



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

University of Wales Trinity Saint David

Engineering Academic Discipline

2D Diffusion Equation Model Using the Explicit Finite Difference

Method (Report)

Kyle Edward Evans

2029909

Mechanical and Manufacturing Engineering

26/01/2025

Computational Methods (ACMA6012)

Word Count: 3208 words

Table of Contents

Table of Contents.....	ii
List of Figures	iii
List of Tables	iii
List of Equations	iii
1. Introduction	1
1.1. Aims, Objectives, and Structure.....	1
2. Governing Equations.....	2
3. Numerical Methodology.....	3
3.1. Taylor Series and Finite Differences	3
3.2. Explicit Scheme and Discretisation	4
3.3. Stability and Fourier Number	5
4. Initial and Boundary Conditions	6
4.1. Analytical Solution	6
5. Code Details and Explanation	7
5.1. User Interface and Fallback Mechanism	7
5.2. Local Functions for Readability	9
5.3. Export Functionality for Heatmaps	9
5.4. Simulation Loop and Convergence	10
6. Validation Against Analytical Solution.....	12
7. Sensitivity Analysis.....	13
8. Model Limitations and Future Work	15
9. Conclusion	16
10. References	17
11. Appendices	20
11.1. Appendix A: MATLAB Code.....	20
11.2. Appendix B: Analytical Solution Code	25
11.3. Appendix C: Sensitivity Analysis Tables	26

List of Figures

Figure 1 Grid Discretisation Diagram	5
Figure 2 User Interface Awaiting Initial User Input.....	7
Figure 3 Interface Initialisation and UI Check	8
Figure 4 Edit Field Generation Loop.....	8
Figure 5 List of Local Functions	9
Figure 6 Export Heatmap Button	9
Figure 7 Temperature Matrix Convergence Loop	10
Figure 8 Final Temperature Distribution	11
Figure 9 Command Window Output for Matrix Display	11
Figure 10 Analytical Solution Results	12
Figure 11 Influence of Grid Spacing on Thermal Equilibrium Time	13
Figure 12 Influence of Time Step on Thermal Equilibrium Time	14
Figure 13 Influence of Tolerance on Thermal Equilibrium Time.....	14

List of Tables

Table 1 Influence of the Grid Spacing	26
Table 2 Influence of the Time Step.....	27
Table 3 Influence of the Tolerance Condition	27

List of Equations

(Equation 1 Governing Heat Equation).....	2
(Equation 2 Forward Time Difference)	3
(Equation 3 Backward Time Difference).....	3
(Equation 4 Central Space: Difference East & West).....	4
(Equation 5 Central Space: Difference North & South).....	4
(Equation 6 Heat Equation: Approximation Substitution).....	4
(Equation 7 Heat Equation: Temperature Subject)	4
(Equation 8 Fourier Number Equation).....	5
(Equation 9 Stability Criterion for Fourier Number).....	5
(Equation 10 Explicit Finite Difference Equation).....	5
(Equation 11 Neumann Boundary Conditions)	6
(Equation 12 Analytical Solution for Thermal Equilibrium)	6

1. Introduction

This report presents a two-dimensional heat diffusion model to determine how temperature changes in a solid aluminium plate over time. The model uses the explicit finite difference method (EFDM) to achieve this. The process begins by dividing the plate into a uniform grid and calculating the temperature at each grid point at each step until the system reaches thermal equilibrium (Croft and Lilley, 1977; Patankar, 1980; Cadence CFD Solutions, 2022a). This method is popular for solving engineering problems involving time-dependent heat equations because it is simple and straightforward to implement in code (Smolski, 2023). This report uses MATLAB to implement the model and produce a graphical representation of the results based on the dimensions, temperature, and time steps.

1.1. Aims, Objectives, and Structure

This report aims to create a computational model in MATLAB to simulate the temperature distribution in a two-dimensional aluminium plate using the explicit finite difference method. The objectives to achieve this goal include the following:

- Divide the plate into a grid in MATLAB and calculate the temperature at each point.
- Update each point with the new temperature until it reaches thermal equilibrium.
- Conduct a sensitivity analysis, identify the limitations, and suggest improvements.

The report is organised to provide a thorough analysis and is structured as follows:

- **Governing Equation:** Introduces and explains the heat diffusion equation.
- **Numerical Method:** Describes the explicit finite difference method and procedure.
- **Boundary Conditions:** Defines the initial and boundary conditions for the code.
- **Code Implementation:** Includes snapshots of the code with detailed explanations.
- **Analytical Solution Validation:** Compares the numerical results with the analytical.
- **Sensitivity Analysis:** Demonstrates how code responds to changes in parameters.
- **Model Limitations:** Discuss the model's limitations due to the method and code.
- **Improvements and Future Work:** Suggest ways to improve the model.

2. Governing Equations

Heat conduction and thermal diffusion refer to the process by which heat moves through a material due to a temperature gradient (Bergman et al., 2011; Çengel, 2008). The heat diffusion equation, shown in Equation 1, governs the process of conduction in solids, which explains how temperature changes over time within a two-dimensional plate with no internal heat sources (Croft and Lilley, 1977). Understanding how heat behaves can help engineers improve the heat management of components, electronics, and machinery.

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = \frac{\rho \times c_p}{k} \times \frac{\partial T}{\partial t} \quad (\text{Equation 1 Governing Heat Equation})$$

In this equation for an aluminium plate:

- **T** represents the temperature
- **x** and **y** are the coordinates
- **ρ** is the density of aluminium in kg/m³
- **c_p** is the specific heat capacity in J/kg·°C
- **k** is the thermal conductivity W/m·°C
- **t** is the time in seconds

The equation expresses thermal diffusivity, which is found by dividing a material's thermal conductivity by the product of its density and specific heat capacity. A high thermal diffusivity indicates that heat will move through the material more quickly than in a material with a low diffusivity. The use of this equation relies on a few key assumptions. First, heat transfer occurs through conduction, without convection or radiation contribution. Second, the material has uniform properties. Third, no sources are causing internal heat generation.

These assumptions are appropriate for modelling a two-dimensional aluminium plate, as the material typically has uniform and well-defined properties. Additionally, heat is applied through conduction and from boundary conditions without considering internal heat sources. Therefore, these approximations are acceptable and work well enough for the analysis.

3. Numerical Methodology

The abovementioned governing equation explains how temperature changes over time in a material. However, it is not easily implementable in its current form as a partial differential equation. Instead, an approximate solution is achievable by rewriting the differential heat equation in a finite-difference form (Iserles, 2009; Patankar, 1980). By dividing the material into a grid, the computer can solve the temperature at each point using the resulting linear equation. This process of continuous data into discrete values is known as discretisation and is commonplace in computational fluid dynamics (Cadence CFD Solutions, 2022b). The solution follows the following steps so that MATLAB can solve the problem:

3.1. Taylor Series and Finite Differences

Taylor's theorem states that any smooth, continuous function, such as those in the heat equation, can be approximated using polynomials (Chapra, 2023; Cadence CFD Solutions, 2022b). The finite difference method uses this concept to approximate the derivatives, enabling computers to solve the temperature at specific points numerically. The difference approximations include three main types: forward difference, backward difference, and central difference (Croft and Lilley, 1977). In this case, the first-order derivative, representing the rate of temperature change over time, can be approximated using either the forward or backward difference methods. The forward difference approximation is used for the time derivative, as shown in Equation 2.

$$\frac{\partial T}{\partial t} = \frac{T_p^{t+\Delta t} - T_p^t}{\Delta t} \quad (\text{Equation 2 Forward Time Difference})$$

In this application, the forward difference approximation is more appropriate because it is simple and easy to implement in MATLAB. This method calculates the next temperature based on the current value, resulting in only one unknown variable per equation. While the backward difference approximation, as seen in Equation 3, is another option, it involves solving simultaneous equations at each step. As a result, the computational power necessary to be involved will increase, and implementation will be more challenging.

$$\frac{\partial T}{\partial t} = \frac{T_p^t - T_p^{t-\Delta t}}{\Delta t} \quad (\text{Equation 3 Backward Time Difference})$$

The second-order derivative describes how the temperature changes across different points in space according to the heat equation. The central difference approximation is most suitable since it provides better accuracy than the forward and backward difference methods by considering both sides of the point (Iserles, 2009; Chapra and Canale, 2015). Rewriting the governing heat equation to include Equations 4 and 5 will result in a linear equation that can be solved numerically using a computer.

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_E^t - 2 \times T_p^t + T_W^t}{h^2} \quad (\text{Equation 4 Central Space: Difference East \& West})$$

$$\frac{\partial^2 T}{\partial y^2} = \frac{T_N^t - 2 \times T_p^t + T_S^t}{h^2} \quad (\text{Equation 5 Central Space: Difference North \& South})$$

3.2. Explicit Scheme and Discretisation

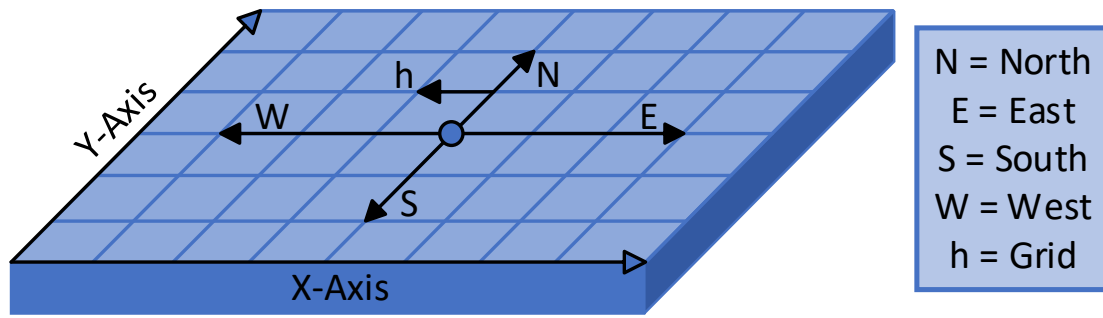
The explicit finite method requires substituting the forward difference approximation for time and the central difference approximation for space into the governing heat equation, which leads to Equation 6 (Patankar, 1980; Cadence CFD Solutions, 2022b). Rearranging it to find the unknown temperature produces Equation 7. This explicit scheme is more straightforward than implicit methods, allowing for direct calculations at each grid point. However, certain limitations are necessary to ensure stability, as discussed in the next section (Chapra and Canale, 2015).

$$\frac{T_E^t - 2 \times T_p^t + T_W^t}{h^2} + \frac{T_N^t - 2 \times T_p^t + T_S^t}{h^2} = \frac{\rho \times c_p}{k} \times \frac{T_p^t - T_p^{t-\Delta t}}{\Delta t} \quad (\text{Equation 6 Heat Equation: Approximation Substitution})$$

$$T_p^{t+\Delta t} = T_p^t + \frac{k}{\rho \times c_p} \times \left(\frac{T_E^t - 2 \times T_p^t + T_W^t}{h^2} + \frac{T_N^t - 2 \times T_p^t + T_S^t}{h^2} \right) \quad (\text{Equation 7 Heat Equation: Temperature Subject})$$

This study discretises the face of an aluminium plate into a uniform grid, where each section represents a specific location on the plate. Figure 1 shows a graphical representation of this grid below. In the graphic, h represents the grid spacing, which is the same in both axes. It also shows the cardinal directions, influencing the result as the heat equation calls for the north, east, south, and west temperatures. Similarly, the time domain also requires discretisation into equal time steps, which must be carefully balanced with the grid resolution to ensure a stable operation.

Figure 1 Grid Discretisation Diagram



(Source: Author's Figure)

3.3. Stability and Fourier Number

The stability of the explicit finite difference method depends on the Fourier number. This dimensionless number represents heat conduction and storage ratio (Wolfram, 2018).

Equation 8 presents the expression for Fourier's number, where α represents the thermal diffusivity of the material. The thermal diffusivity is approximately $6.42 \times 10^{-5} \text{ m}^2/\text{s}$, based on the following properties: a conductivity of $143 \text{ W/m}\cdot^\circ\text{C}$, a density of $2.8 \times 10^3 \text{ kg/m}^3$, and a specific heat of $795 \text{ J/kg}\cdot^\circ\text{C}$. This value resembles the thermal data of aluminium alloys published by Engineers Edge (2025).

$$F_0 = \frac{\alpha \Delta t}{h^2}$$

(Equation 8 Fourier Number Equation)

The stability criterion states that the Fourier number must be greater than zero and less than or equal to $1/4$ (0.25), as seen in Equation 9. The solution may be unstable and unusable if the result does not meet these conditions. Transposing and substituting the Fourier number into heat equations results in Equation 10, the final formula for accurately predicting the temperature distribution of an aluminium plate while also maintaining stability. Maintaining an appropriate Fourier number helps to ensure a stable prediction. It is also important to consider how the initial temperature and east and west boundaries influence the results.

$$0 < F_0 \leq \frac{1}{4}$$

(Equation 9 Stability Criterion for Fourier Number)

$$T_p^{t+\Delta t} = F_0 \left(T_E^t + T_W^t + T_N^t + T_S^t - \left(4 - \frac{1}{F_0} \right) \times T_p^t \right)$$

(Equation 10 Explicit Finite Difference Equation)

4. Initial and Boundary Conditions

The initial boundary condition defines the starting temperature across the plate. Meanwhile, the boundary condition sets the temperature at the outer edges, identifiable by their cardinal directions (North, South, East, and West). Since there is no internal heat generation, the plate temperature is zero degrees Celsius. Initially, the user applies the Dirichlet boundary conditions to the north and south edges by setting fixed temperature values (Bergman et al., 2011; Nuclear Power, 2025). Similarly, according to Neumann conditions, the east and west boundaries assume zero heat flux. In other words, there is perfect insulation along these boundaries with no heat transfer. The numerical representation in Equation 11 shows that the temperature in the East is the same as in the West.

$$T_E = T_W \quad \text{(Equation 11 Neumann Boundary Conditions)}$$

4.1. Analytical Solution

The forthcoming numerical solutions require validation to ensure the accuracy of the approximations. The analytical solution, Equation 12, provides the exact temperature distribution at thermal equilibrium using the initial and boundary conditions, including the Dirichlet boundary conditions at the North and South edges and Neumann conditions at the east and west edges (Croft and Lilley, 1977; Chang and Ma, 2001). However, it is important to note that this is specific to these initial and boundary conditions. Its sole purpose is to validate the written code and assess the accuracy of the numerical method.

$$T(x, y) = -T_0 + \frac{2 \times T_0}{y_{dim}} \times y + \sum_{k=1}^{\infty} \left(\frac{1}{(100 \times k \times \pi)} \times [1 - (-1)^k] \times \frac{\sin\left(\frac{k \times \pi \times x}{Y_{dim}}\right)}{\sinh\left(\frac{k \times \pi \times x_{dim}}{Y_{dim}}\right)} \times \left[\cosh\left(\frac{k \times \pi \times x}{Y_{dim}}\right) - \cosh\left(\frac{k \times \pi \times (x - x_{dim})}{Y_{dim}}\right) \right] \right)$$

(Equation 12 Analytical Solution for Thermal Equilibrium)

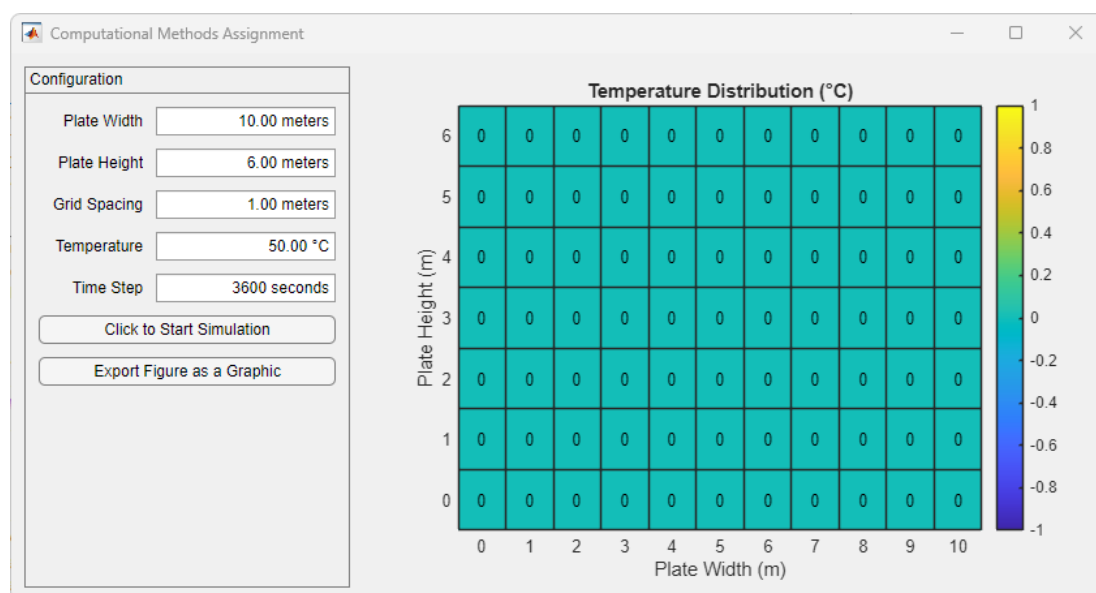
5. Code Details and Explanation

The following MATLAB code calculates the temperature distribution of an aluminium plate and creates a graphical representation of the results. The complete script is available in Appendix A for reference. The program will attempt to run in a user-friendly window but also includes a command-line mode for systems lacking a graphical user interface (GUI). It allows users to input their parameters, including plate dimensions, grid spacing, temperature, and time step and returns an exportable heat map.

5.1. User Interface and Fallback Mechanism

Some functions were incompatible during the initial tests using MATLAB on desktop, online, and mobile platforms. In response, the script starts by attempting to initialise the 'figure' function using the 'try' function (MathWorks, 2024a). If MATLAB encounters an error, it defaults to the 'catch' function, which runs an alternative version. MATLAB includes built-in hardware detection functions (MathWorks, 2024b). However, they cannot distinguish between the online and mobile applications, making the 'try-catch' workaround the only option. Figure 2 greets the user on start-up. The heatmap is initialised with zeros and will be updated based on data the user inputs into the configuration window on the left.

Figure 2 User Interface Awaiting Initial User Input



(Source: Author's Figure)

Figure 3 contains a snapshot of the interface initialisation code. The following actions depend on the logical variable that returns either true or false depending on the user's hardware. The code snippet also creates the UI window, sets up the panels, and defines the dimensions for the heatmap and parameter entry fields. Building a loop to generate the input fields in Figure 4 has several benefits. They make it more straightforward to modify the parameters to suit future needs and updates. It also saves having to rewrite the same code five times, which can be time-consuming in new revisions.

Figure 3 Interface Initialisation and UI Check

```
uiCompatible = true;
try
    % Create main UI figure and layout
    figMain = uifigure('Name', 'Computational Methods Assignment', ...
        'WindowStyle', 'alwaysontop', 'Resize','on');
    figMain.Position(3) = 2 * figMain.Position(4);
    mainGrid = uigridlayout(figMain, [1 3]);
    mainGrid.ColumnWidth = {250,'1x'};
    % Adds and layout the configuration panel for user input
    panelConfig = uipanel(mainGrid,'Title','Configuration');
    configGrid = uigridlayout(panelConfig,[3 2]);
    configGrid.RowHeight = {22, 22, 22, 22, 22, 22, 22};
    configGrid.ColumnWidth = {80,'1x'};
catch
    % Switch to the command window mode if UI creation fails
    uiCompatible = false;
    fprintf(['UI features are unavailable on your device.\n'...
        'Switching to command window mode.\n']);
end
```

(Source: Author's Figure)

Figure 4 Edit Field Generation Loop

```
% Parameters for numerical input fields
paramNames = {'Plate_Width', 'Plate_Height', 'Grid_Spacing', ...
    'Temperature', 'Time_Step'};
paramUnits = {'%.2f meters', '%.2f meters', '%.2f meters', ...
    '%.2f °C', '%.0f seconds'};
paramLimits = [0, Inf; 0, Inf; 0, Inf; 0, Inf; 0, Inf];
paramDefaults = [10, 6, 1, 50, 3600];
% Loops to create labels and input fields
editFields = struct();
for i = 1:length(paramNames)
    uilabel(configGrid, 'Text', strrep(paramNames{i}, '_', ' '), ...
        'HorizontalAlignment', 'right');
    editFields.(paramNames{i}) = uieditfield(configGrid, 'numeric', ...
        'ValueDisplayFormat', paramUnits{i}, ...
        'Limits', paramLimits(i, :), ...
        'LowerLimitInclusive','off', 'UpperLimitInclusive','on', ...
        'Value', paramDefaults(i));
end
```

(Source: Author's Figure)

5.2. Local Functions for Readability

The code uses local functions for several reasons. They make the script easier to read and are necessary to implement the buttons. This approach also eliminates the need to rewrite large sections of code. For example, the function simulating the temperature distribution, which is the script's core, is utilised by both the UI and the command window mode, as shown in Figure 5.

Figure 5 List of Local Functions

```
%% Local Functions (UI Mode)
% Validates user input and returns the user inputs in UI mode
function [plateWidth, plateHeight, gridSpacing, ...

% Runs the simulation and updates the plot with the results in UI mode
function runSimulationUI(editFields, figMain, plateHeatmap, aluDiffusivity)
% Exports the heatmap as a graphic in the current folder
function exportHeatmap(plateHeatmap)

%% Local Functions (Command Window Mode)
% Validates user input and returns the user inputs in command window mode
function [plateWidth, plateHeight, gridSpacing, ...
% Runs the simulation and posts the plot in command window mode
function runSimulationCW(aluDiffusivity)
```

(Source: Author's Figure)

5.3. Export Functionality for Heatmaps

The built-in heatmap function in MATLAB lacks an export or save images feature when used in a UI figure. In response to this issue, the code includes an export button that allows users to save the heatmap as an image file. As shown in Figure 6, the export function generates the filename based on the current date and time and saves the image in the script folder.

Figure 6 Export Heatmap Button

```
% Exports the heatmap as a graphic in the current folder
function exportHeatmap(plateHeatmap)
uniqueTime = datetime;
uniqueTime.Format = 'yyyyMMddHHmmss';
graphicName = sprintf('heatmap_output_%s.png', uniqueTime);
exportgraphics(plateHeatmap, graphicName, 'Resolution', 300);
fprintf('Heatmap successfully saved as %s in current folder\n', graphicName);
end
```

(Source: Author's Figure)

5.4. Simulation Loop and Convergence

The local function seen in Figure 7 serves as the core of this script. It creates the initial matrix using the 'zeros' function, with the number of rows and columns provided by the user's input influencing the size. Next, it loops and updates the temperature at each point in the matrix. The loop continues due to the while function until the convergence flag becomes true. The flag will only become true once the changes in the new temperature matrix become less than the specified convergence threshold.

Figure 7 Temperature Matrix Convergence Loop

```
% Simulate temperature distribution until convergence is achieved
function matrixT2 = simulateTemperature(numRows, numCols, northTemp, southTemp,
numFourier)
% Initialise the temperature matrices
matrixT1 = zeros(numRows, numCols);
matrixT1(1, :) = northTemp;
matrixT1(end, :) = southTemp;
matrixT2 = matrixT1;
convergence = false;
valueCC = 0.01;
fourierConstant = 4 - 1/numFourier;
% Iterate and update the matrix until convergence is true
while convergence == false
    % Starts with the interior point of the matrix
    for i = 2:numRows-1
        for j = 2:numCols-1
            calcNorth = matrixT1(i-1, j);
            calcSouth = matrixT1(i+1, j);
            calcEast = matrixT1(i, j+1);
            calcWest = matrixT1(i, j-1);
            matrixT2(i, j) = numFourier * (calcNorth + calcSouth + ...
            calcEast + calcWest - (fourierConstant) * matrixT1(i,j));
        end
    end
    % Finishes with the boundary columns
    for k = 2:numRows-1
        calcNorth = matrixT1(k-1, 1);
        calcSouth = matrixT1(k+1, 1);
        calcEast = matrixT1(k, 1+1);
        matrixT2(k, 1) = numFourier * (calcNorth + calcSouth + calcEast + ...
        calcEast - (fourierConstant) * matrixT1(k,1));

        calcNorth = matrixT1(k-1, numCols);
        calcSouth = matrixT1(k+1, numCols);
        calcWest = matrixT1(k, numCols-1);
        matrixT2(k, numCols) = numFourier * (calcNorth + calcSouth + ...
        calcWest + calcWest - (fourierConstant) * matrixT1(k,numCols));
    end
    % Check for convergence
    if max(abs(matrixT2 - matrixT1)) < valueCC
        convergence = true;
    else
        matrixT1 = matrixT2;
    end
end
end
```

(Source: Author's Figure)

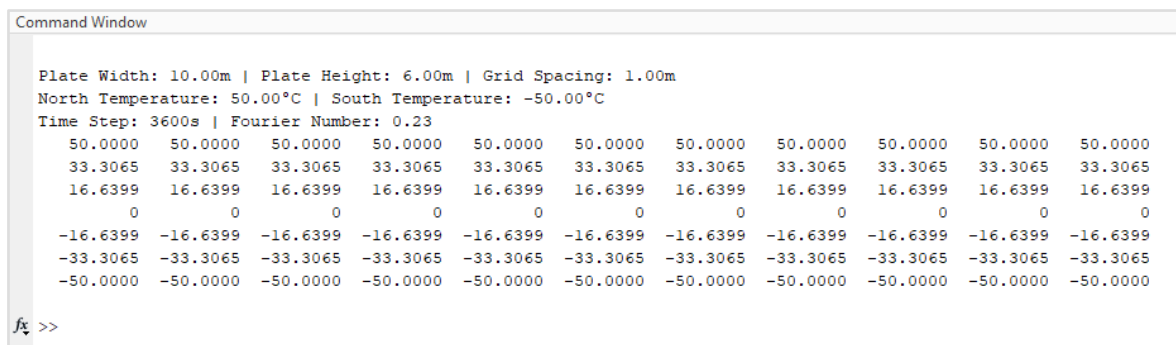
Once the function finishes its execution, the script updates the heatmap in the figure window, as shown in Figure 8. This user interface allows users to modify parameters without restarting the script. Additionally, the script displays a copy of the temperature matrix with all parameters in the MATLAB command window so users can copy the text. Figure 9 shows a copy of the temperature matrix. If the user runs another set of parameters, the new matrix will appear below it, stacking the results and allowing users to scroll and compare them. The command window mode produces a heatmap and a text copy of the matrix.

Figure 8 Final Temperature Distribution



(Source: Author's Figure)

Figure 9 Command Window Output for Matrix Display

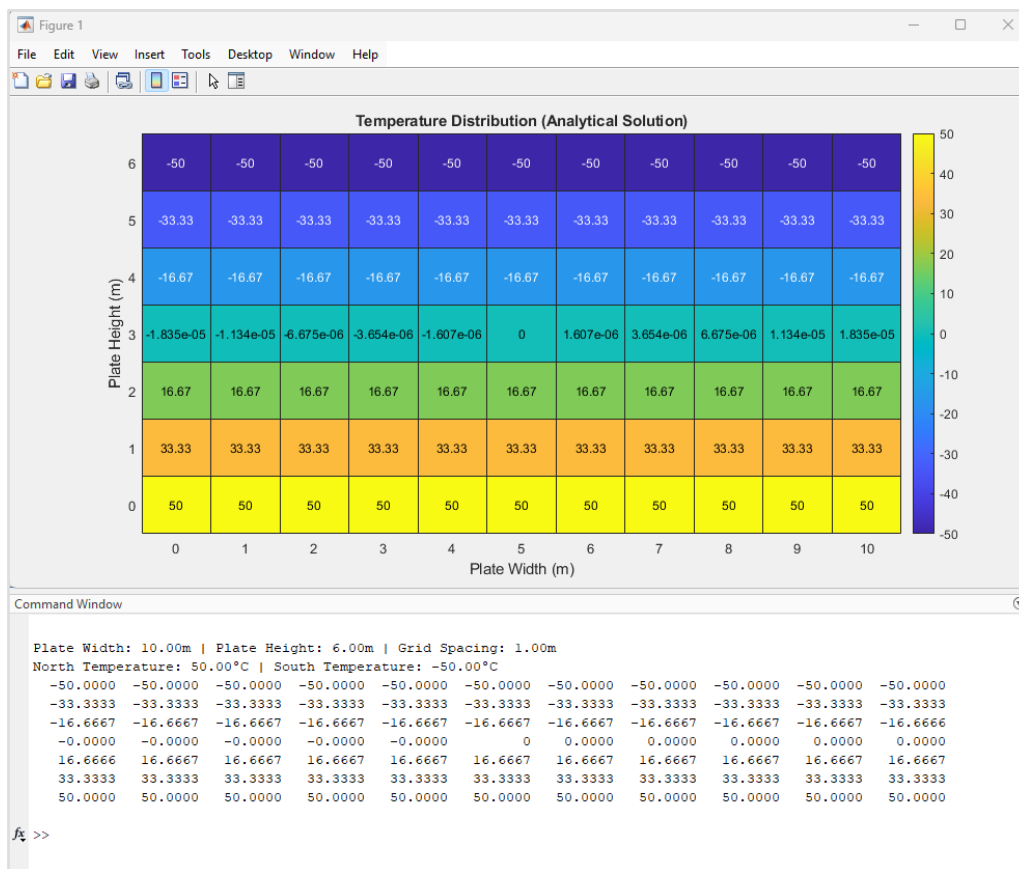


(Source: Author's Figure)

6. Validation Against Analytical Solution

As the initial and boundary condition section explains, the analytical solution gives the precise temperature distribution at the thermal equilibrium under these boundary conditions. This exercise aims to validate the accuracy of the numerical model using the explicit finite difference method. Appendix B contains the MATLAB implementation of the analytical solution. Figure 10 contains the results, and the parameters used are the same as those in Figures 8 and 9 for easy comparison.

Figure 10 Analytical Solution Results



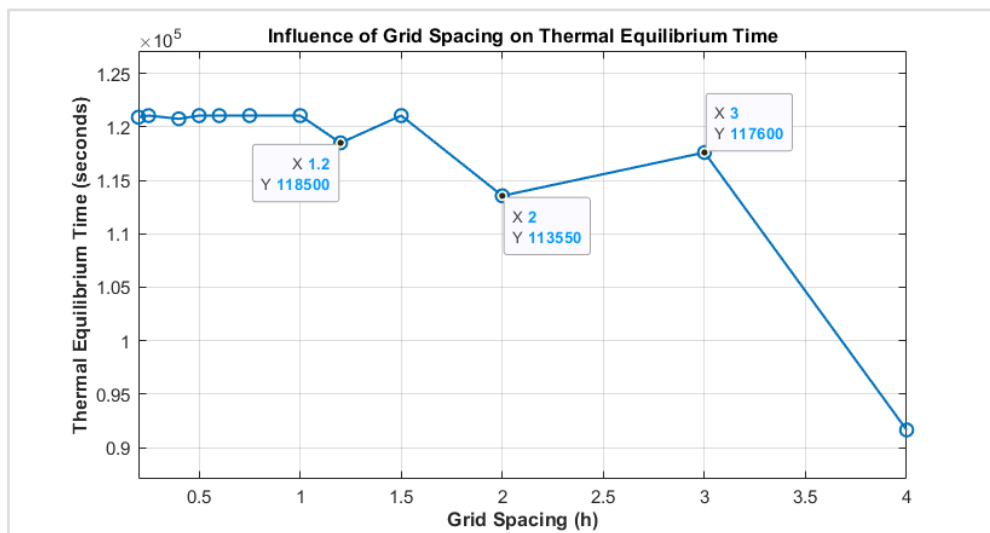
(Source: Author's Figure)

Comparing the two temperature matrices reveals a maximum difference of $\pm 0.0268^\circ\text{C}$, translating into an error percentage of approximately 0.08%. The acceptability of this error value is industry and application-dependent, but as an initial base implementation, anything below 5% is acceptable (Slater, 2021; Mazumder, 2024).

7. Sensitivity Analysis

This sensitivity analysis aims to understand how different parameters, such as grid space, time step, and convergence tolerance, impact the numerical model's performance. All tests use a plate size of 12 by 12 meters with a temperature of 50°C. The raw data is also available under Appendix C. Only one parameter varied per test, while the others remained fixed. Grid spacing determines the grid's resolution, affecting the matrix's size. As the resolution increases, so does the matrix. As shown in Figure 11, increasing the grid spacing reduces the resolution and leads to shorter run times. The difference between three and four meters is the most noticeable. The highest resolutions of one meter or less produce the most accurate temperature distribution. However, this comes with significantly longer computation times, typical in similar projects (Shen, Shih, Xiao, 2011; Comsol, 2025).

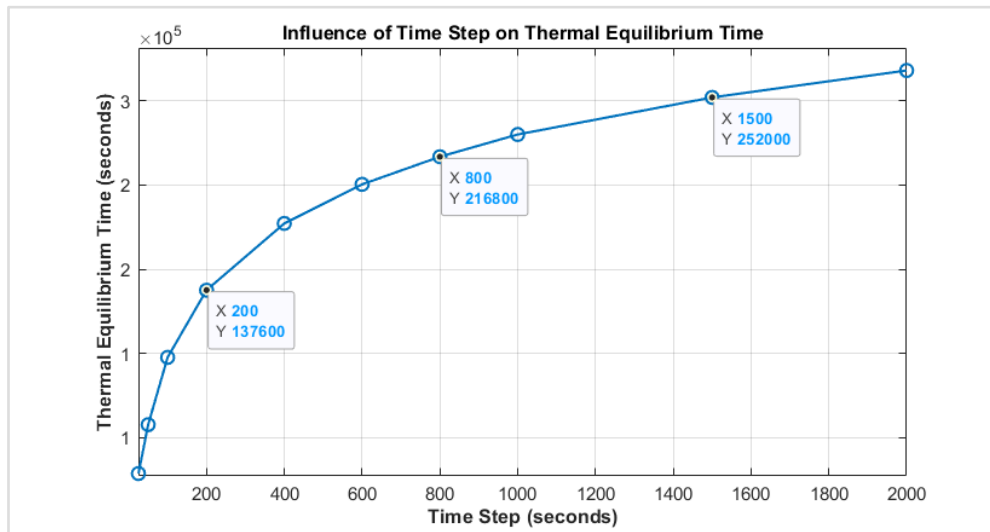
Figure 11 Influence of Grid Spacing on Thermal Equilibrium Time



(Source: Author's Figure)

The time step significantly impacts the overall computation time. Smaller time steps require more iterations to reach thermal equilibrium, as shown in Figure 12. Meanwhile, large-time steps require less computational power and reach equilibrium more quickly. However, as previously discussed, large-time steps risk numerical instability, violating the Courant-Friedrichs-Lewy (CFL) conditions, which is why the Fourier number must be less than or equal to 1/4 for the script to function (Chapra and Canale, 2015; Caminha, 2024)

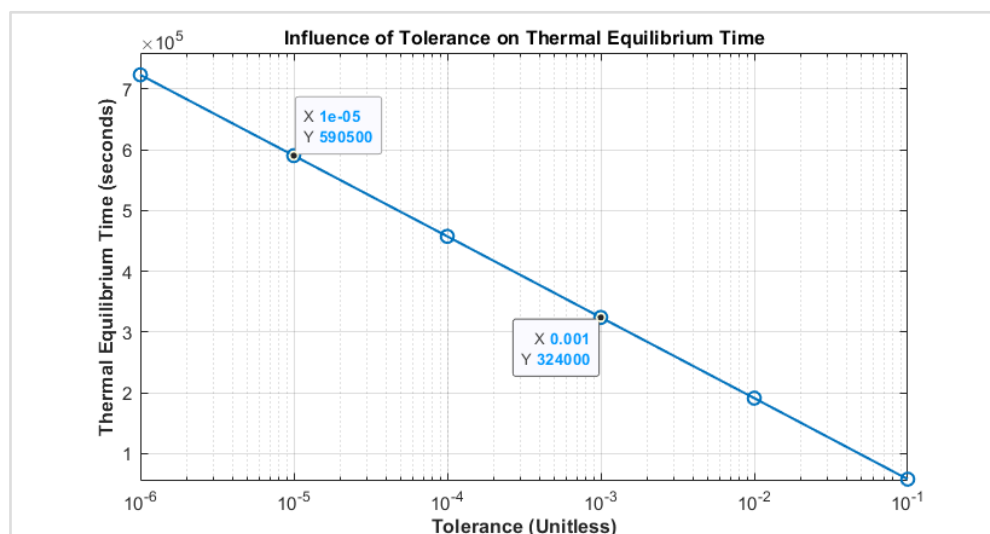
Figure 12 Influence of Time Step on Thermal Equilibrium Time



(Source: Author's Figure)

The tolerance condition dictates the script's convergence point. A tighter tolerance or smaller value increases the time to reach equilibrium, requiring further iterations to meet the conditions. Figure 13 features a continuous downward trend as the tolerance value increases, resulting in lower run time. However, higher tolerance values may lead to less accurate results, as they cause the loop to terminate too early (Liu, Yeoh, and Tu, 2018).

Figure 13 Influence of Tolerance on Thermal Equilibrium Time



(Source: Author's Figure)

8. Model Limitations and Future Work

The explicit finite difference method suffers from several inherent limitations. Firstly, the time step restriction makes solving a large complex time-consuming and requires significant computational power. As a result, the current numerical model is only suitable for simple geometries, like the aluminium plate, with small time steps. In addition, the numerical method introduces two more types of errors: round-off and truncation. Round-off errors happen due to computers being unable to represent all real numbers. Instead of using exact values, they work with approximations, which can lead to minor errors. In applications like this, numerical models can accumulate and produce more significant errors over multiple iterations (Chapra, 2023; Wolfram Mathworld, 2025). Similarly, truncation errors occur from approximating mathematical operations. In this project, several assumptions were necessary while using the explicit finite difference method to approximate the derivatives, likely resulting in some truncation error (Croft and Lilley, 1977; Liu, Yeoh, and Tu, 2018).

The current numerical model fails to accurately reflect real-world conditions by applying steady-state boundary conditions using the Dirichlet boundary conditions to the north and south edges, which fix the values in place. Similarly, the east and west edges have no bearing on the distribution due to the insulation. Future work must improve the script's functionality by considering real-world conditions. In addition, the numerical model might benefit from switching to an alternative numerical method that does not suffer from some of the limitations. For example, the implicit finite difference method (IFDM) is known for its increased stability and accuracy compared to the current model, allowing for increased time steps while remaining stable (Liu and Sen, 2009; Djordjevich, Savović, and Janićijević, 2017). The goal must be to produce a more stable, accurate script capable of handling complex geometries beyond a simple two-dimensional plate.

9. Conclusion

This report details a script and analysis for a two-dimensional heat diffusion model using the explicit finite difference method to simulate temperature distribution on an aluminium plate in MATLAB. The model has undergone validation against the analytical solution for the specific boundary conditions, demonstrating a maximum error of $\pm 0.0268^{\circ}\text{C}$, or 0.08%, within an acceptable range for this application.

The model also underwent a sensitivity analysis to examine how the various parameters (grid spacing, time step, and tolerance) affect the time it takes to reach thermal equilibrium. The trade-off is between accuracy and run time. While the current settings are adequate for simple problems and geometry, the model would be challenging to scale, especially with complex geometry. Future work involves exploring alternate numerical methods which feature better scalability. The application would also benefit by better reflecting real-world conditions. Overall, the model serves as a decent foundation for future improvements.

10. References

- Bergman, T.L. et al. (2011) *Introduction to Heat Transfer*. 6th edn. Jefferson City: John Wiley & Sons.
- Cadence CFD Solutions (2022a) 'Using the 2D Finite Difference Method for Heat Transfer Analysis', *Cadence Design Systems* [blog]. Available at: <https://resources.system-analysis.cadence.com/blog/msa2022-using-the-2d-finite-difference-method-for-heat-transfer-analysis> (Accessed: 8 January 2025).
- Caminha, G. (2024) *The CFL Condition and How to Choose Your Timestep Size*. Available at: <https://www.simscale.com/blog/cfl-condition/> (Accessed: 24 January 2025).
- Çengel, Y.A. (2008) *Introduction to Thermodynamics and Heat Transfer*. 2nd edn. New York City: McGraw-Hill.
- Chang, C.Y. and 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*, 44, pp. 2423-2437.
- Chapra, S.C. (2023) *Applied Numerical Methods with MATLAB for Engineers and Scientists*. 5th edn. New York City: McGraw-Hill.
- Chapra, S.C. and Canale, R.P. (2015) *Numerical Methods for Engineers*. 7th edn. New York City: McGraw-Hill.
- Comsol (2025) *The Finite Element Method (FEM)*. Available at: <https://www.comsol.com/multiphysics/finite-element-method> (Accessed: 24 January 2025).
- Croft, D.R. and Lilley, D.G. (1977) *Heat Transfer Calculations Using Finite Difference Equations*. London: Applied Science Publishers LTD.
- Djordjević, A., Savović, S. and Janićijević, A. (2017) 'Explicit finite-difference solution of two-dimensional solute transport with periodic flow in homogenous porous media', *Journal of Hydrology and Hydromechanics*, 65 (4), pp. 426-432.
- Engineers Edge (2025) *Thermal Diffusivity Table*. Available at: https://www.engineersedge.com/heat_transfer/thermal_diffusivity_table_13953.htm (Accessed: 14 January 2025).
- Iserles, A. (2009) *A First Course in the Numerical Analysis of Differential Equations*. 2nd edn. Cambridge: Cambridge University Press.

Liu, C., Yeoh, G.H. and Tu, J. (2018) *Computational Fluid Dynamics: A Practical Approach*. 3rd edn. Oxford: Butterworth-Heinemann.

Liu, Y. and Sen, M.K. (2009) 'A practical implicit finite-difference method: examples from seismic modelling', *Journal of Geophysics and Engineering*, 6 (3), pp. 231-249.

MathWorks (2024a) *Use try/catch to Handle Errors*. Available at: https://uk.mathworks.com/help/matlab/matlab_prog/use-trycatch-to-handle-errors.html (Accessed: 20 January 2025).

MathWorks (2024b) *Information about computer on which MATLAB is running*. Available at: <https://uk.mathworks.com/help/matlab/ref/computer.html> (Accessed: 20 January 2025).

MathWorks (2024c) *Recognize and Avoid Round-Off Errors*. Available at: <https://uk.mathworks.com/help/symbolic/recognize-and-avoid-roundoff-errors.html> (Accessed: 26 January 2025).

Mazumder, H.M. (2024) .Comparative study of turbulence models in CFD for transonic flow over the ONERA M6 wing', *Discover Mechanical Engineering*, 3 (46).

Nuclear Power (2025) *Dirichlet Boundary Condition – Type I Boundary Condition*. Available at: <https://www.nuclear-power.com/nuclear-engineering/heat-transfer/thermal-conduction/heat-conduction-equation/dirichlet-boundary-condition-type-i-boundary-condition/> (Accessed: 18 January 2025).

Patankar, S.V. (1980) *Numerical Heat Transfer and Fluid Flow*. London: Hemisphere Publishing Corporation.

Shen, B., Shih, A.J. and Xiao, G. (2011) 'A Heat Transfer Model Based on Finite Difference Method for Grinding', *Journal of Manufacturing Science and Engineering*, 133 (1), pp. 031001-1 – 031001-10.

Slater, J.W. (2021) *Overview of CFD Verification and Validation*. Available at: <https://www.grc.nasa.gov/www/wind/valid/tutorial/overview.html> (Accessed: 25 January 2025).

Smolski, A. (2023) 'Explicit Finite Difference with Black-Scholes Formula (with code)', *Medium* [blog]. Available at: <https://antonismolski.medium.com/implementation-of-black-scholes-formula-using-finite-difference-method-with-code-965fd0539808> (Accessed: 8 January 2025).

Wolfram (2018) *Fourier Number Using Thermal Diffusivity*. Available at:
<https://resources.wolframcloud.com/FormulaRepository/resources/Fourier-Number-Using-Thermal-Diffusivity> (Accessed: 14 January 2025).

Wolfram Mathworld (2025) *Roundoff Error*. Available at:
<https://mathworld.wolfram.com/RoundoffError.html> (Accessed: 26 January 2025).

Intentionally left blank; see next page.

11. Appendices

11.1. Appendix A: MATLAB Code

```
%% Workspace Management Commands
clc
clear variables
close all

%% Material Properties
aluConductivity = 143;
aluDensity = 2.8e3;
aluSpecHeat = 795;
aluDiffusivity = aluConductivity / (aluDensity * aluSpecHeat);

%% User Interface (Figure Properties)
% Check for UI compatibility (MATLAB mobile does not support it)
uiCompatible = true;

try
    % Create main UI figure and layout
    figMain = uifigure('Name', 'Computational Methods Assignment', ...
        'WindowStyle', 'alwaysontop', 'Resize', 'on');
    figMain.Position(3) = 2 * figMain.Position(4);
    mainGrid = uigridlayout(figMain, [1 3]);
    mainGrid.ColumnWidth = {250, '1x'};

    % Adds and layout the configuration panel for user input
    panelConfig = uipanel(mainGrid, 'Title', 'Configuration');
    configGrid = uigridlayout(panelConfig, [3 2]);
    configGrid.RowHeight = {22, 22, 22, 22, 22, 22, 22, 22}; % One for each input variable
    configGrid.ColumnWidth = {80, '1x'};

catch
    % Switch to the command window mode if UI creation fails
    uiCompatible = false;
    fprintf(['UI features are unavailable on your device.\n'...
        'Switching to command window mode.\n']);

end

%% Define User Inputs
if uiCompatible
    % Parameters for numerical input fields
    paramNames = {'Plate_Width', 'Plate_Height', 'Grid_Spacing', ...
        'Temperature', 'Time_Step'};
    paramUnits = {'%.2f meters', '%.2f meters', '%.2f meters', ...
        '%.2f °C', '%.0f seconds'};
    paramLimits = [0, Inf; 0, Inf; 0, Inf; 0, Inf; 0, Inf];
    paramDefaults = [10, 6, 1, 50, 3600];

    % Loops to create labels and input fields
    editFields = struct();
    for i = 1:length(paramNames)
```

```

        ylabel(configGrid, 'Text', strrep(paramNames{i}, '_', ' '), ...
            'HorizontalAlignment', 'right');
        editFields.(paramNames{i}) = uieditfield(configGrid, 'numeric', ...
            'ValueDisplayFormat', paramUnits{i}, ...
            'Limits', paramLimits(i, :), ...
            'LowerLimitInclusive','off', 'UpperLimitInclusive','on', ...
            'Value', paramDefaults(i));
    end

    % Create a placeholder plot for initial display
    defaultMatrix = zeros(paramDefaults(2) + 1, paramDefaults(1) + 1);
    plateHeatmap = heatmap(mainGrid, defaultMatrix, ...
        'Title', 'Temperature Distribution (°C)', ...
        'XLabel', 'Plate Width (m)', 'YLabel', 'Plate Height (m)', ...
        'Colormap', parula, 'ColorbarVisible', 'on');
    plateHeatmap.Layout.Column = 2;
    plateHeatmap.XDisplayLabels = string(0:paramDefaults(3):paramDefaults(1));
    plateHeatmap.YDisplayLabels = string(flip(0:paramDefaults(3):paramDefaults(2)));

    % Creates a button to run the simulation
    runButton = uibutton(configGrid, 'Text', 'Click to Start Simulation', 'ButtonPushedFcn', ...
        @(btn,event) runSimulationUI(editFields, figMain, plateHeatmap, aluDiffusivity));
    runButton.Layout.Column = [1 2];

    % Creates a button to export the figure as a graphic
    exportButton = uibutton(configGrid, 'Text', 'Export Figure as a Graphic', ...
        'ButtonPushedFcn', @(btn,event) exportHeatmap(plateHeatmap));
    exportButton.Layout.Column = [1 2];

else
    % Run the simulation in command window mode
    runSimulationCW(aluDiffusivity);
end

%% Local Functions (All)
% Simulate temperature distribution until convergence is achieved
function matrixT2 = simulateTemperature(numRows, numCols, northTemp, southTemp, numFourier)
% Initialise the temperature matrices
matrixT1 = zeros(numRows, numCols);
matrixT1(1, :) = northTemp;
matrixT1(end, :) = southTemp;
matrixT2 = matrixT1;

% Set convergence criteria
convergence = false;
valueCC = 0.01;
fourierConstant = 4 - 1/numFourier;

% Iterate and update the matrix until convergence is true
while convergence == false
    % Starts with the interior point of the matrix
    for i = 2:numRows-1
        for j = 2:numCols-1
            calcNorth = matrixT1 (i-1, j);

```

```

        calcSouth = matrixT1 (i+1, j);
        calcEast = matrixT1 (i, j+1);
        calcWest = matrixT1 (i, j-1);
        matrixT2(i, j) = numFourier * (calcNorth + calcSouth + calcEast + calcWest ...
            - (fourierConstant) * matrixT1(i,j));
    end
end

% Finishes with the boundary columns
for k = 2:numRows-1
    calcNorth = matrixT1 (k-1, 1);
    calcSouth = matrixT1 (k+1, 1);
    calcEast = matrixT1 (k, 1+1);
    matrixT2(k, 1) = numFourier * (calcNorth + calcSouth + calcEast + calcEast ...
        - (fourierConstant) * matrixT1(k,1));

    calcNorth = matrixT1 (k-1, numCols);
    calcSouth = matrixT1 (k+1, numCols);
    calcWest = matrixT1 (k, numCols-1);
    matrixT2(k, numCols) = numFourier * (calcNorth + calcSouth + calcWest + calcWest ...
        - (fourierConstant) * matrixT1(k,numCols));
end

% Check for convergence
if max(abs(matrixT2 - matrixT1)) < valueCC
    convergence = true;
else
    matrixT1 = matrixT2;
end
end
end

%% Local Functions (UI Mode)
% Validates user input and returns the user inputs in UI mode
function [plateWidth, plateHeight, gridSpacing, ...
    timeStep, numFourier] = inputValidation(editFields, figMain, aluDiffusivity)

% Extracts values from the input fields in the figure
plateWidth = editFields.Plate_Width.Value;
plateHeight = editFields.Plate_Height.Value;
gridSpacing = editFields.Grid_Spacing.Value;
timeStep = editFields.Time_Step.Value;

% Checks if grid spacing divides the plate dimensions evenly
if rem(plateWidth, gridSpacing) ~= 0 || rem(plateHeight, gridSpacing) ~= 0
    uialert(figMain, 'Grid spacing must evenly divide both plate dimensions.', 'Input Error');
    numFourier = NaN; % This is necessary to prevent errors
    return
end

% Calculate maximum time step and the fourier number
maxTimeStep = gridSpacing / (4 * aluDiffusivity);
numFourier = (aluDiffusivity * timeStep) / gridSpacing ^ 2;

% Checks stability of the time step

```

```

if numFourier > 0.25 || numFourier < 0
    uialert(figMain, sprintf(['This time step fails to meet the ' ...
        'stability criterion.\n\nThe maximum allowable time step is ' ...
        '%.2f seconds.\n\nThe edit field has been updated, ' ...
        'please continue.'], maxTimeStep), 'Input Error');

    editFields.Time_Step.Value = floor(maxTimeStep);
    return
end
end

% Runs the simulation and updates the plot with the results in UI mode
function runSimulationUI(editFields, figMain, plateHeatmap, aluDiffusivity)

% Validates user input and ends if validation fails
[plateWidth, plateHeight, gridSpacing, ...
    timeStep, numFourier] = inputValidation(editFields, figMain, aluDiffusivity);
if isnan(numFourier)
    return
end

% Extract and assign the input values
northTemp = editFields.Temperature.Value;
southTemp = -northTemp;

% Initialise the temperature matrices
numRows = plateHeight / gridSpacing + 1;
numCols = plateWidth / gridSpacing + 1;
matrixT2 = simulateTemperature(numRows, numCols, northTemp, southTemp, numFourier);

% Update the heatmap
plateHeatmap.ColorData = matrixT2;
plateHeatmap.XDisplayLabels = string(0 : gridSpacing : plateWidth);
plateHeatmap.YDisplayLabels = string(flip(0 : gridSpacing : plateHeight));

% Print details in command window
fprintf(['\nPlate Width: %.2fm | Plate Height: %.2fm | Grid Spacing: %.2fm\n' ...
    'North Temperature: %.2f°C | South Temperature: %.2f°C\n' ...
    'Time Step: %.0fs | Fourier Number: %.2f\n'], plateWidth, plateHeight, ...
    gridSpacing, northTemp, southTemp, timeStep, numFourier)
disp(matrixT2)
end

% Exports the heatmap as a graphic in the current folder
function exportHeatmap(plateHeatmap)
uniqueTime = datetime;
uniqueTime.Format = 'yyyyMMddHHmmss';

graphicName = sprintf('heatmap_output_%.s.png', uniqueTime);
exportgraphics(plateHeatmap, graphicName, 'Resolution', 300);
fprintf('Heatmap successfully saved as %s in current folder\n', graphicName);
end

%% Local Functions (Command Window Mode)
% Validates user input and returns the user inputs in command window mode

```

```

function [plateWidth, plateHeight, gridSpacing, ...
    timeStep, northTemp, numFourier] = commandWindowInput(aluDiffusivity)

% Initialise a while loop flag for grid spacing divisibility
divisibilityCheck = false;
fprintf('Please ensure that all values are positive numerals.\n\n')

% Loops until the user provides valid dimensions and grid spacing
while divisibilityCheck == false
    plateWidth = input('Enter Plate Width (metres): ');
    plateHeight = input('Enter Plate Height (metres): ');
    gridSpacing = input('Enter Grid Spacing (metres): ');

    % Checks if the grid spacing evenly divides both dimensions
    if rem(plateWidth, gridSpacing) ~= 0 || rem(plateHeight, gridSpacing) ~= 0
        fprintf('Grid spacing must evenly divide both plate dimensions.\nTry again.')
    else
        divisibilityCheck = true;
    end
end

% Prompts the user for temperature input and creates a stability flag
northTemp = input('Enter Temperature (celcius): ');
maxTimeStep = gridSpacing / (4 * aluDiffusivity);
stabilityCriterion = false;

% Loops until the user provides a valid time step
while stabilityCriterion == false
    timeStep = input('Enter Time Step (seconds): ');
    numFourier = (aluDiffusivity * timeStep) / gridSpacing ^ 2;

    % Checks stability and provides the maximum allowable time step
    if numFourier > 0.25 || numFourier < 0
        fprintf(['The time step fails to meet the stability criterion. \n' ...
            'The maximum allowable value is %.2f seconds please try again \n'], maxTimeStep)
    else
        % Exits the loop if user input is valid
        stabilityCriterion = true;
    end
end

% Runs the simulation and posts the plot in command window mode
function runSimulationCW(aluDiffusivity)

% Extracts user input from the command window input
[plateWidth, plateHeight, gridSpacing, ...
    timeStep, northTemp, numFourier] = commandWindowInput(aluDiffusivity);
southTemp = -northTemp;

% Initialise the temperature matrices
numRows = plateHeight / gridSpacing + 1;
numCols = plateWidth / gridSpacing + 1;
matrixT2 = simulateTemperature(numRows, numCols, northTemp, southTemp, numFourier);

```

```

% Plot the result as a heat map
plateHeatmap = heatmap(matrixT2, ...
    'Title', 'Temperature Distribution', ...
    'XLabel', 'Plate Width (m)', 'YLabel', 'Plate Height (m)', ...
    'Colormap', parula, ...
    'ColorbarVisible', 'on');
plateHeatmap.XDisplayLabels = string(0 : gridSpacing : plateWidth);
plateHeatmap.YDisplayLabels = string(flip(0 : gridSpacing : plateHeight));

% Print details in command window
fprintf(['\nPlate Width: %.2fm | Plate Height: %.2fm | Grid Spacing: %.2fm\n' ...
    'North Temperature: %.2f°C | South Temperature: %.2f°C\n' ...
    'Time Step: %.0fs | Fourier Number: %.2f\n'], plateWidth, plateHeight, ...
    gridSpacing, northTemp, southTemp, timeStep, numFourier)
disp(matrixT2)
end

```

11.2. Appendix B: Analytical Solution Code

```

%% Workspace Management Commands
clc;
clear variables;
close all;

%% Material Parameters
nsTemp = 50;          % Boundary temperature (°C)
xDim = 10;            % Plate width (m)
yDim = 6;             % Plate height (m)
gridSpacing = 1;      % Grid spacing (m)

%% Matrix Initialisation
numRows = yDim / gridSpacing + 1;
numCols = xDim / gridSpacing + 1;
matrixTA = zeros(numRows, numCols);
matrixTA(1, :) = nsTemp;
matrixTA(end, :) = -nsTemp;

xPos = linspace(0, xDim, numCols);
yPos = linspace(0, yDim, numRows);

for i = 1:numRows
    for j = 1:numCols
        sumResult = 0;
        for k = 1:100
            term1 = 1 / (100 * k * pi)^2;
            term2 = (1 - (-1)^k);
            term3 = sin((k * pi * yPos(i)) / yDim) / sinh((k * pi * xDim) / yDim);
            term4 = cosh((k * pi * xPos(j)) / yDim) - cosh((k * pi * (xPos(j) - xDim)) / yDim);
            sumResult = sumResult + (term1 * term2 * term3 * term4);
        end
        matrixTA(i, j) = -nsTemp + (2 * nsTemp / yDim) * yPos(i) + sumResult;
    end
end
end

```

```

%% Heatmap Plot & Command Window Print
% Plot the result as a heat map
plateHeatmap = heatmap(matrixTA, ...
    'Title', 'Temperature Distribution (Analytical Solution)', ...
    'XLabel', 'Plate Width (m)', 'YLabel', 'Plate Height (m)', ...
    'Colormap', parula, ...
    'ColorbarVisible', 'on');
plateHeatmap.XDisplayLabels = string(0 : gridSpacing : xDim);
plateHeatmap.YDisplayLabels = string(flip(0 : gridSpacing : yDim));

% Print details in command window
fprintf(['\nPlate Width: %.2fm | Plate Height: %.2fm | Grid Spacing: %.2fm\n' ...
    'North Temperature: %.2f°C | South Temperature: %.2f°C\n'], xDim, yDim, ...
    gridSpacing, nsTemp, -nsTemp)
disp(matrixTA)

```

11.3. Appendix C: Sensitivity Analysis Tables

Table 1 Influence of the Grid Spacing

Grid Spacing h	Time step (fixed)	Tolerance (fixed)	Thermal Equilibrium Time (in seconds)
4	150	0.001	91650
3	150	0.001	117600
2	150	0.001	113550
1.5	150	0.001	121050
1.2	150	0.001	118500
1	150	0.001	121050
0.75	150	0.001	121050
0.6	150	0.001	121050
0.5	150	0.001	121050
0.4	150	0.001	120750
0.3	150	0.001	121050
0.25	150	0.001	121050
0.2	150	0.001	120900

(Source: Author's Figure)

Table 2 Influence of the Time Step

Grid Spacing h (fixed)	Time step	Tolerance (fixed)	Thermal Equilibrium Time (in seconds)
0.75	2000	0.001	268000
0.75	1500	0.001	252000
0.75	1000	0.001	230000
0.75	800	0.001	216800
0.75	600	0.001	200400
0.75	400	0.001	177200
0.75	200	0.001	137600
0.75	100	0.001	97700
0.75	50	0.001	57750
0.75	25	0.001	28750

(Source: Author's Figure)

Table 3 Influence of the Tolerance Condition

Grid Spacing h (fixed)	Time step (fixed)	Tolerance	Thermal Equilibrium Time (in seconds)
1	500	0.1	57500
1	500	0.01	191000
1	500	0.001	324000
1	500	0.0001	457500
1	500	0.00001	590500
1	500	0.000001	723500

(Source: Author's Figure)