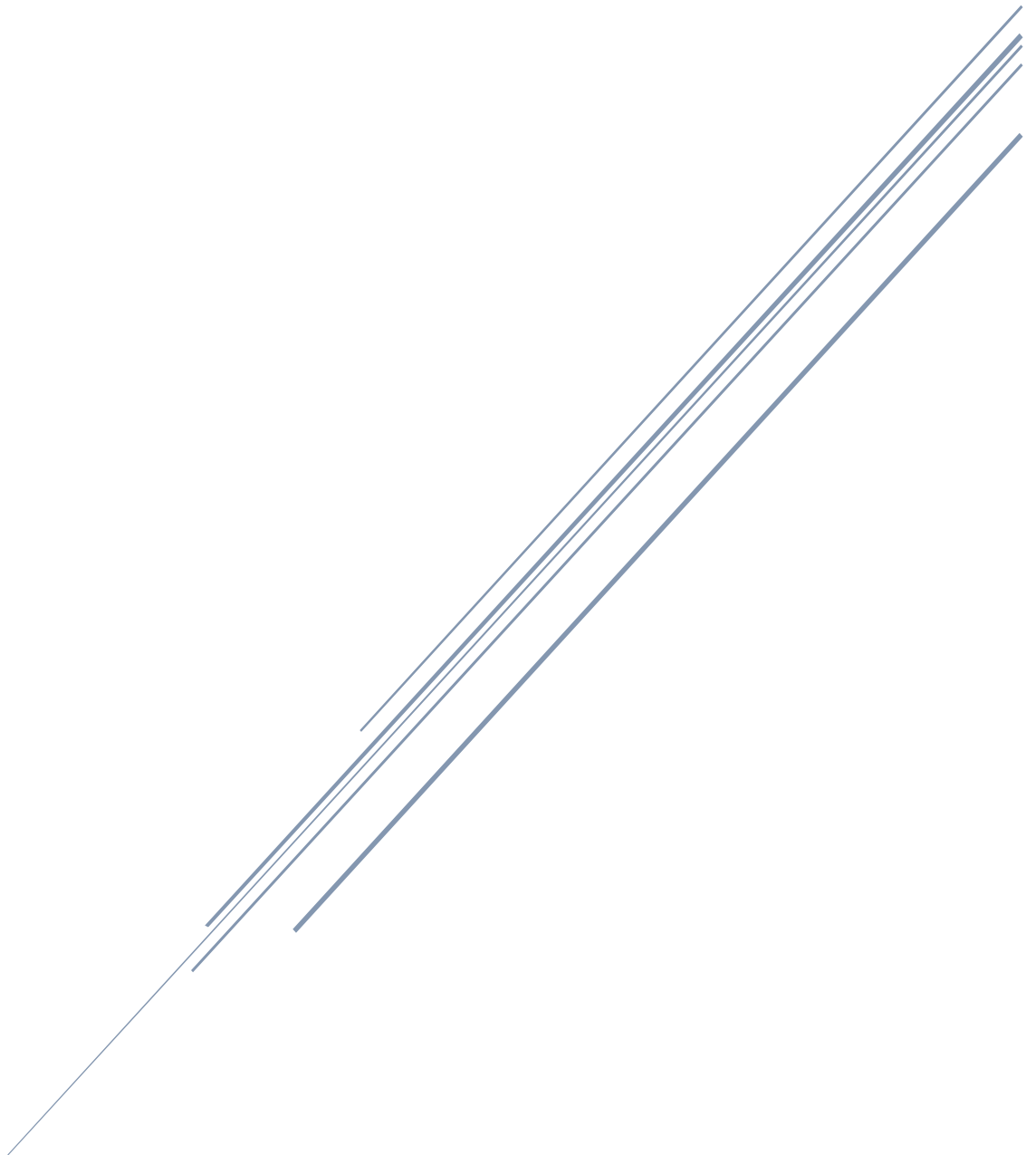


2D DIFFUSION EQUATION MODEL USING THE EXPLICIT FINITE DIFFERENCE METHOD

ACMA6012 – Computational Methods



Name: Joseph McBride
Student ID: 2113455

Table of Contents

Table of Contents.....	1
Table of Figures	2
1 Introduction.....	3
2 Governing Equation.....	4
3 Numerical Method to Discretise Governing Equation.....	5
3.1 Taylor Series.....	5
3.1.1 Taylor Series Expansion.....	5
3.1.2 Forward Difference Approximation.....	5
3.1.3 Backward Difference Approximation.....	6
3.1.4 Central Difference Approximation.....	6
3.1.5 Central (Second Derivative) Difference Approximation.....	7
3.2 Explicit Finite Difference Scheme.....	8
3.2.1 Replace Temporal First Derivative by Forward Difference Approximations.....	8
3.2.2 Replace Spatial Second Derivative by Forward Difference Approximations.....	8
3.2.3 Derive the Explicit Finite Difference Form Introducing the Fourier Number.....	8
3.2.4 Fourier Number.....	9
3.2.5 Define the Stability Criterion.....	9
3.2.6 Discretise the Spatial Domain.....	10
3.2.7 Discretise the Temporal Domain.....	10
4 Initial Domain, Boundary Conditions and Analytical Solution.....	11
4.1 Initial Domain Conditions.....	11
4.2 Boundary Conditions.....	11
4.3 Analytical Solution.....	11
5 Code Details and Explanations.....	12
6 Analysis of Code Results.....	16
6.1 Code Validation.....	16
6.1.1 Grid Spacing.....	16
6.1.2 Time Step.....	18
6.1.3 Thermal Equilibrium Tolerance.....	19
6.2 Sensitivity Analysis.....	20
6.2.1 Grid Spacing Against Time to Reach Thermal Equilibrium.....	20
6.2.2 Time Step Against Time to Reach Thermal Equilibrium.....	21
6.2.3 Thermal Equilibrium Tolerance Against Time to Reach Thermal Equilibrium.....	22
7 Model Limitations.....	23
7.1 Limitations due to the Coding.....	23
7.2 Limitations due to Round Off Errors.....	23
7.3 Limitations due to Truncation Errors.....	23
8 Improvements for Future Developments.....	24
9 Conclusion.....	25
10 References.....	26
11 Appendix.....	27

Table of Figures

Figure 1 - Parabolic Heat Equation (Croft D.R, Lilley D.G, 1977)	4
Figure 2 - 2D Finite Difference Method Grid.....	4
Figure 3 - Analytical Solution Equation of the Temperature at Thermal Equilibrium (Daileda, 2012)..	11
Figure 4 - MATLAB Code Section 1	12
Figure 5 - MATLAB Code Section 2	12
Figure 6 - MATLAB Code Section 3	13
Figure 7 - MATLAB Code Section 4	13
Figure 8 - MATLAB Code Section 5	14
Figure 9 - MATLAB Code Section 6	14
Figure 10 - MATLAB Code Section 7	15
Figure 11 - MATLAB Code Section 8	15
Figure 12 - MATLAB Code Section 9	15
Figure 13 - Output Response Grid Spacing of 1 (Matrix)	16
Figure 14 - Output Response Grid Spacing of 1 (Graph)	16
Figure 15 - Output Response Grid Spacing of 1.5 (Matrix)	16
Figure 16 - Output Response Grid Spacing of 1.5 (Graph)	17
Figure 17 - Output Response Grid Spacing of 0.5 (Matrix)	17
Figure 18 - Output Response Grid Spacing of 0.5 (Graph)	17
Figure 19 - Output Matrix (Time Step - 3600).....	18
Figure 20 - Output Matrix (Time Step - 60).....	18
Figure 21 - Output Matrix (Tight Thermal Equilibrium Tolerance).....	19
Figure 22 - Output Matrix (Open Thermal Equilibrium Tolerance)	19
Figure 23 - Grid Spacing Versus Time Taken to Reach Thermal Equilibrium Plot.....	20
Figure 24 - Time Step Versus Time Taken to Reach Thermal Equilibrium Plot.....	21
Figure 25 - Thermal Equilibrium Tolerance Versus Time Taken to Reach Thermal Equilibrium Plot.....	22
Figure 26 - History of Numerical Methods (Zienkiewicz, 2005)	24
Figure 27 - Code Page 1.....	27
Figure 28 - Code Page 2.....	28
Figure 29 - Code Page 3.....	29

1 Introduction

This report will provide an overview of the 2D Diffusion Equation Model using the EFDM (Explicit Finite Difference Method). More specifically, the temperature distribution within an aluminium plate through time. The first section of this report will cover the governing equation for two-dimensional parabolic heat transfer. Following this the numerical method for discretising the governing equation will be covered in depth. Also covered in this report will be the initial boundary conditions that have been setup and implementation of the analytical solution within the code. After reviewing the previously mentioned points, sections of the code will be provided along with explanations regarding the functionality. A sensitivity analysis will then be conducted on the code to illustrate how the output differs in response to a change in grid spacing, time step and thermal equilibrium tolerance.

The limitations of the model will also be covered within this report. Whether these be limitations with the numerical method applied or limitations due in relation to coding. Based on the previous section, suggested improvements for the model can be made for future developments. Finally, the conclusion will provide a brief overview of the report's findings. Overall, the aim of this report is to provide a detailed explanation of the 2D Diffusion Equation Model using the EFDM while critically evaluating the solution.

2 Governing Equation

The Explicit Finite Difference Method can be applied when creating a mathematical model with the purpose of analysing heat transfer between objects. Mathematical modelling is a technique often deployed within industry to understand how a real-world application will perform through the means of equations. There are multiple benefits of creating a mathematical model for a real-world problem which include enabling Engineers to make predictions of how the application will perform and allow them to optimise aspects before physically building a design. It also allows for the simulation of different scenarios so the response of the application under varying conditions can be understood. One of the drawbacks of mathematical modelling is that it is only as good as the parameter input, which means it may not exactly simulate the real-world scenario.

For the mathematical model covered in this report will utilise the 2D Explicit Finite Difference Method to simulate temperature distribution within an aluminium plate through time. The EFDM model outputs an approximate solution for a given heat transfer scenario. Figure 1 below shows the heat equation.

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = \frac{\rho \cdot c_p}{k} \cdot \frac{\partial T}{\partial t}$$

Figure 1 - Parabolic Heat Equation (Croft D.R, Lilley D.G, 1977)

In the equation shown in Figure 1 the temperature (T) is a function of position (x, y) and time (t). Other terms also include thermal conductivity (k), density (p) and the specific heat (c). Also taken into consideration is thermal diffusivity (k/c.p). The heat equation is often used when analysing the conduction properties of a material. It can also be applied to diffusion applications of solids and fluids. Both reasons are contributing factors to what makes the heat equation important.

At the beginning of 2D EFDM mathematical model, a grid needs to be defined with nodes at each intersection. An example of this can be found in Figure 2. The nodes at each intersection signify the temperature at each point on the aluminium plate surface.

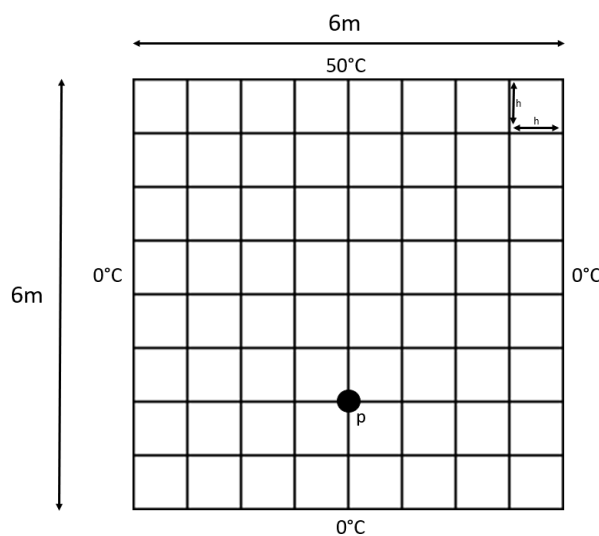


Figure 2 - 2D Finite Difference Method Grid

3 Numerical Method to Discretise Governing Equation

Now that the governing equation has been introduced the discretisation can take place to generate parameters that will eventually be utilised within the mathematical model for heat transfer using the 2D Explicit Finite Difference Method. The first stage involves using the Taylor Series to create difference approximations of the first and second derivatives.

3.1 Taylor Series

To generate the difference approximations for first and second derivatives Taylor Series expansions is used. This is required for the forward, backward, central, and central to the second-order derivative approximations.

3.1.1 Taylor Series Expansion

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!} \cdot (x - x_0)^1 + \frac{f''(x_0)}{2!} (x - x_0)^2 + \dots \sum_{n=0}^{\infty} \frac{f^n(x_0)}{n!} (x - x_0)^n$$

Employing the Taylor Series expression above, the equation can be modified to calculate the heat transfer through the aluminium plate. This can be found below, for both the forward and backward directions.

$$f(x_0 + h) - f(x_0) = f(x_0) + f'(x_0) \frac{h}{1!} + f''(x_0) \frac{h^2}{2!} + \dots \sum_{n=0}^{\infty} f^n(x_0) \frac{h^n}{n!}$$

$$f(x_0 - h) - f(x_0) = f(x_0) + f'(x_0) \frac{h}{1!} + f''(x_0) \frac{h^2}{2!} + \dots \sum_{n=0}^{\infty} f^n(x_0) \frac{h^n}{n!}$$

3.1.2 Forward Difference Approximation

Following the introduction of the Taylor Series expression, a numerical technique must be adopted that leverages the Finite Difference Method derived from the Taylor Series to construct a linear equation. This is achieved by firstly making f' the subject of the equation, as shown below.

$$f(x_0 + h) - f(x_0) = f'(x_0)h + f''(x_0) \frac{h^2}{2!} + \dots \sum_{n=0}^{\infty} f^n(x_0) \frac{h^n}{n!}$$

Both sides of the equation must then be divided by h , as shown in the next equation.

$$\frac{f(x_0 + h) - f(x_0)}{h} = f(x_0) + f'(x_0) \frac{h}{1!} + f''(x_0) \frac{h^2}{2!} + f'''(x_0) \frac{h^3}{3!} + \dots$$

Finally, the equation for the forward difference approximation to the first derivative is generated. This can be seen below. Oh in this equation represents the error value.

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} + (Oh)$$

3.1.3 Backward Difference Approximation

The same process must be completed for calculating the backward difference approximation for the first derivative. The difference is that $f(x_0 - h)$ due to the opposing direction being calculated. As done before, f^1 is made the subject of the equation. This can be found below.

$$f(x_0 - h) - f(x_0) = f'(x_0)h + f''(x_0)\frac{h^2}{2!} + \dots \sum_{n=0}^{\infty} f^n(x_0)\frac{h^n}{n!}$$

Both sides of the equation must then be divided by h , as shown in the next equation.

$$\frac{f(x_0 - h) - f(x_0)}{h} = f'(x_0) + f''(x_0)\frac{h}{2!} + f'''(x_0)\frac{h^2}{3!} + \dots$$

A linear equation, shown below, can then be derived. This represents the backwards difference approximation for the first derivative.

$$f'(x_0) = \frac{f(x_0) - f(x_0 - h)}{h} + (0h)$$

3.1.4 Central Difference Approximation

Now that the forward and backward difference approximations have been defined, the process can be completed for the central difference approximation. This approximation considers both sides of a chosen node, hence $f(x_0 + h)$ and $f(x_0 - h)$ both being incorporated into the equation.

$$f(x_0 + h) - f(x_0 - h) = 2f'(x_0)h + 2f'''(x_0)\frac{h^3}{3!} + 2f^{(5)}(x_0)\frac{h^5}{5!} + \dots$$

Both sides must then be divided by $2h$, as shown below.

$$\frac{f(x_0 + h) - f(x_0 - h)}{2h} = f'(x_0) + f'''(x_0)\frac{h^2}{3!} + f^{(5)}(x_0)\frac{h^4}{5!} + \dots$$

After dividing both sides by $2h$ the central difference approximation for the first derivative can be equated. This can be found below.

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} + (0h)^2$$

Now that the three approximations for the forward, backward and central differences have been determined for the first derivative, the central difference approximation can now be utilised further. This is due to the error value $(0h^2)$ being smaller, making it a more accurate estimation. The next stage is to apply the central difference approximation for the second derivative of the heat transfer equation. Doing so will provide greater accuracy of the temperature changing through the aluminium plate, due to the method considering the temperature of nodes either side of the selected point.

3.1.5 Central (Second Derivative) Difference Approximation

As mentioned previously, the central difference approximation provides greater accuracy when calculating the heat transfer through the aluminium plate. To do so, the forward difference approximation must be added to the backwards difference approximation. This can be seen below.

$$f(x_0 + h) = f(x_0) + f'(x_0)h + f''(x_0)\frac{h^2}{2!} + \dots \sum_{n=0}^{\infty} f^n(x_0)\frac{h^n}{n!}$$

$$f(x_0 - h) = f(x_0) - f'(x_0)h + f''(x_0)\frac{h^2}{2!} + \dots \sum_{n=0}^{\infty} f^n(x_0)\frac{h^n}{n!}$$

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

To equate the central difference approximation for the second derivative, $2f''(x_0)$ must be taken to the other side of the equation, as shown below.

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

The above equation must then be divided by h^2 , which is displayed in the following equation.

$$\frac{f(x_0 + h) - 2.f(x_0) + f(x_0 - h)}{h^2} = f''(x_0) + 0h^2$$

Finally, the central difference approximation for the second derivative can be generated. This equation can be found below.

$$f''(x_0) = \frac{f(x_0 + h) - 2.f(x_0) + f(x_0 - h)}{h^2} + (0h)^2$$

3.2 Explicit Finite Difference Scheme

The following section breaks down the discretisation for the spatial and temporal domains. Below is the discretised form of the heat equation, which will be employed within the heat transfer mathematical model.

$$\frac{\partial^2 T_p}{\partial x^2} + \frac{\partial^2 T_p}{\partial y^2} = \frac{1}{\alpha} \times \frac{\partial T_p}{\partial t}$$

3.2.1 Replace Temporal First Derivative by Forward Difference Approximations

Shown below is the equation that will be utilised to replace the temporal first derivative with forward difference approximations.

$$\frac{\partial T}{\partial t} = \frac{T_p^{t+\Delta t} - T_p^t}{\Delta t}$$

Overall, the above equation expresses how the temperature at a specific point change over time. The change in temperature at a chosen node is based on the difference in temperature at different times.

3.2.2 Replace Spatial Second Derivative by Forward Difference Approximations

The below equations replace the spatial second derivative with forward difference approximations, for the X and Y axis respectively.

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_E - 2T_p + T_W}{h^2}$$

$$\frac{\partial^2 T}{\partial y^2} = \frac{T_N - 2T_p + T_S}{h^2}$$

Temperature distribution is essentially calculated utilising the above equations, by computing the difference in temperature between two points. Once the differences are calculated between the two points, the value is divided by h^2 , which is the spatial step size.

3.2.3 Derive the Explicit Finite Difference Form Introducing the Fourier Number

The equations derived above can now be substituted back into the heat equation, which is displayed at the beginning of this chapter.

$$\frac{T_E^t - 2T_p^t + T_W^t}{h^2} + \frac{T_N^t - 2T_p^t + T_S^t}{h^2} = \frac{1}{\alpha} \cdot \frac{T_p^{t+\Delta t} - T_p^t}{\Delta t}$$

Factorising the above equation by h^2 introduces the spatial domain into the equation. This can be seen below.

$$\frac{1}{h^2} \cdot (T_E + T_W + T_N + T_S - 4T_p) = T_p^{t+\Delta t} - T_p^t$$

3.2.4 Fourier Number

The Fourier number indicates the relationship between the rate of heat transfer and storage within the aluminium plate, making it an essential aspect of the mathematical simulation. This is defined in the below equation.

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

Following the definition of the Fourier number, it can subsequently be integrated into the heat equation, which is displayed below.

$$fo \cdot (T_E + T_W + T_N + T_S - 4T_p + \frac{1}{fo} T_p = T_p^{t+\Delta t}$$

After the integration of the Fourier number into the heat transfer equation, it must be made normal by multiplying and dividing by T_p^t . This can be found in the equation below.

$$fo \cdot (T_E + T_W + T_N + T_S - T_p(4 - \frac{1}{fo}) = T_p^{t+\Delta t}$$

The heat equation derived above, that utilises the Explicit Finite Difference Method, can now be employed in the simulation model to equate the heat transfer through the specified material, until thermal equilibrium is reached.

3.2.5 Define the Stability Criterion

For this mathematical model, the stability criterion ensures that the result of the simulation remains accurate. If the conditions of the stability criterion are not met, the simulation will become unstable, making it a critical element of the model.

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

3.2.6 Discretise the Spatial Domain

The spatial domain is discretised as follows:

$$x = Nx$$

$$y = Ny$$

The grid spacing for the spatial domains is defined as follows:

$$\Delta x = Lx$$

$$\Delta y = Ly$$

The grid points for the spatial domain are discretised as follows:

$$\text{Where, } x = i$$

$$\text{Then, } Nx - 1$$

$$\text{Where, } y = j$$

$$\text{Then, } Ny - 1$$

3.2.7 Discretise the Temporal Domain

The temporal domain is discretised as follows:

$$t = Nt$$

The time step for the temporal domain is defined as follows:

$$\Delta t = dt$$

4 Initial Domain, Boundary Conditions and Analytical Solution

4.1 Initial Domain Conditions

This section of the report defines the initial boundary conditions to be considered within the mathematical model. The initial domain condition can be found below:

$$T = (x, y, 0) = 0 \text{ for } 0 < x < 6 \text{ \& } 0 < y < 6$$

The above condition details the initial temperature parameters of the 6x6 metre aluminium plate. This is achieved by defining the temperature as 0°C at every point within the plate. As this is the initial domain condition, no heating temperatures have been applied to the object at this point.

4.2 Boundary Conditions

Now that the initial domain conditions have been explained, it is time to look at the boundary conditions that need to be applied throughout the simulation. The conditions for the y axis boundaries can be found below:

$$\begin{aligned} T(0, y, t) &= 0 \\ T(6, y, t) &= 0 \end{aligned} \text{ for } 0 \leq y \leq 6 \text{ and } 0 \leq t \leq \text{Thermal Equilibrium}$$

The conditions above provide the parameters for the left and right boundaries. Based on this condition the boundaries conditions remain the same over time and does not fluctuate. The conditions for the x axis can be found below:

$$\begin{aligned} T(x, 0, t) &= f(x) \\ T(x, 6, t) &= 0 \end{aligned} \text{ for } 0 \leq x \leq 6 \text{ and } 0 \leq t \leq \text{Thermal Equilibrium}$$

Based on the boundary conditions provided above, it can be determined that a varying temperature condition is allowable for the top boundary due to the function included. This is not the case for the bottom boundary, which remains a constant temperature as time progresses. The function provided for the top boundary is as follows:

$$f(x) = \begin{cases} 12.5x & \text{if } 0 \leq x \leq (\frac{2}{3} \times 6) \\ -25x + 150 & \text{if } (\frac{2}{3} \times 6) < x \leq 6 \end{cases}$$

As mentioned previously, the above function provides the top boundary condition. If x is within the first range specified, a linear increase in temperature will be seen across the top boundary. If x is within the second range specified, a linear decrease in temperature will be seen across the top boundary. The rate in which the temperature increases or decreases is determined by 12.5 and -25 respectively.

4.3 Analytical Solution

$$T(x, y) = \frac{450}{\pi^2} \cdot \sum_{n=1}^{\infty} \left(\frac{\sin\left(\frac{2 \cdot n \cdot \pi}{3}\right)}{n^2 \cdot \sinh(n \cdot \pi)} \cdot \sin\left(\frac{n \cdot \pi \cdot x}{6}\right) \cdot \sinh\left(\frac{n \cdot \pi \cdot y}{6}\right) \right)$$

Figure 3 - Analytical Solution Equation of the Temperature at Thermal Equilibrium (Daileida, 2012)

Figure 3 shows the analytical solution equation which the code will be tested against. This equation utilises sine functions while taking into consideration the variation in direction between the x and y axis. This allows for computing of the temperature distribution throughout the aluminium plate.

5 Code Details and Explanations

The code details and explanations will be covered in this section of the report. Aspects previously covered within this report have been incorporated into the code solution to achieve the appropriate outcome.

```
% Material properties for aluminum
k = 143; % Thermal conductivity (W/m.C)
rho = 2.8e3; % Density (kg/m^3)
cp = 795; % Specific heat (J/kg.C)

% Calculate thermal diffusivity
alpha = k / (rho * cp);

% Plate dimensions
Lx = 6; % Length of the plate in the x-direction
Ly = 6; % Length of the plate in the y-direction
```

Figure 4 - MATLAB Code Section 1

The first section of the code, displayed above in Figure 4, defines a few of the initial parameters for the mathematical model. Firstly, the material properties for the aluminium plate. These properties being the thermal conductivity, density, and specific heat. All these properties are then subsequently used to calculate the thermal diffusivity of the material. This is defined as alpha within the code. Finally, this section of the code defines the plate dimensions of the aluminium plate. As per the geometry requirements, the plate dimensions are defined as 6x6 metres.

```
% Prompt user for grid spacing
validGrid = false;
while ~validGrid
    gridSpacing = input('Enter the grid spacing (must comply with plate dimensions): ');
    if gridSpacing <= 0 || gridSpacing > min(Lx, Ly)
        disp('Invalid grid spacing. Please choose a value within the plate dimensions.');
```

```
    else
        validGrid = true;
    end
end

% Grid parameters
Nx = (Lx / gridSpacing) + 1; % Number of grid points in the x-direction
Ny = (Ly / gridSpacing) + 1; % Number of grid points in the y-direction
dx = Lx / (Nx - 1); % Grid spacing in the x-direction
dy = Ly / (Ny - 1); % Grid spacing in the y-direction
```

Figure 5 - MATLAB Code Section 2

The next section of this report covers the grid parameters. A valid grid flag is set to false before a while loop is initialised. During this loop the user is asked to enter a grid spacing value, if the value entered is less than or equal to zero, and if the grid spacing value is greater than the plate dimensions then an error message will be displayed. The user will then be asked to enter another value. If a valid grid spacing value is entered the flag will be set to true and the code will progress. Following this the number of grid points in the x and y direction are setup and the grid spacing is defined.

```

% Time parameters
dt = input('Enter Time Step Value: '); % Time step

% Calculate stability criterion
stabilityCriterion = alpha * dt / (gridSpacing^2);

% Check if the stability criterion is met
if stabilityCriterion > 0.25
    disp('Warning: Stability criterion may not be fulfilled. Consider choosing a smaller time step. ');
    % You may choose to prompt the user to enter a different time step or handle it accordingly.
end

```

Figure 6 - MATLAB Code Section 3

In the code displayed in Figure 6, the user is asked to enter a time step value. This value is then used within the stability criterion calculation which also incorporates the grid spacing value. If the outcome of the stability criterion calculation is greater than 0.25 a warning will be displayed to the user, and they will be asked to enter a smaller time step.

```

% Initialize temperature field with boundary conditions
T = zeros(Nx, Ny);

% Set left, right, and bottom boundaries to zero
T(:, [1, end]) = 0; % Left and right boundaries
T(1, :) = 0; % Bottom boundary

% Set interior of the matrix to zero
T(2:Nx-1, 2:Ny-1) = 0;

% Function for the top boundary condition T(x, 0, t) = f(x)
dx = Lx / (Nx - 1); % Grid spacing in the x-direction
x_values = linspace(0, Lx, Nx); % Use linspace for x values
f_top = @(x) 12.5 * x .* (0 <= x) .* (x <= 2/3 * Lx) + (-25 * x + 150) .* ((2/3 * Lx) < x) .* (x <= Lx);
T(1, 2:Ny-1) = f_top(x_values(2:Nx-1));

```

Figure 7 - MATLAB Code Section 4

Figure 7 initialises the boundary conditions and sets up a function for the temperature response at the top boundary. Firstly, a matrix of zeros is initialised where the grid points in the x and y direction are represented respectively. The left, right and bottom boundary conditions are set by indexing the value zero to the appropriate point in the matrix. Next, the interior domain of the matrix is set to by specifying the point grid points in both the x and y direction. Finally, a function for the top boundary is defined by determining the grid spacing in the x direction, then using the 'linspace' function in MATLAB to specify values between to linear points. This will subsequently calculate the difference between the start and stop points and generate the points required. The following lines define the provided function and where it should be applied to in the matrix.

```

% Display the initial temperature matrix
disp('Initial Temperature Matrix:');
disp(T);

% Initialize the change in temperature
delta_T = Inf;

% Initialize iteration counter
iterations = 0;

% Run the simulation until thermal equilibrium is reached
while delta_T > 1e-4
    T_new = T;
    for i = 2:Nx-1
        for j = 2:Ny-1
            T_new(i, j) = T(i, j) + alpha * dt * ((T(i+1, j) - 2*T(i, j) + T(i-1, j)) / dx^2 + (T(i, j+1) - 2*T(i, j) + T(i, j-1)) / dy^2);
        end
    end
end

```

Figure 8 - MATLAB Code Section 5

In Figure 8, the code displays the initial temperature matrix to the user to ensure it has been setup correctly. The next step initialises a variable that is equal to infinity. This is used later to establish if thermal equilibrium has been reached. The iterations counter is set to zero at the beginning of the while loop, this counter tracks how many times the loop is run. The while loop will then run until thermal equilibrium is reached. Using the finite difference method, the matrix is iterated repeatedly. The 2D Explicit Finite Difference Method allows for the temperature at each grid point to be updated.

```

% Calculate the maximum change in temperature
delta_T = max(max(abs(T_new - T)));

% Update the temperature matrix
T = T_new;

% Increment iteration counter
iterations = iterations + 1;
end

% Calculate the time taken for the simulation
total_time = iterations * dt;

```

Figure 9 - MATLAB Code Section 6

When the while loop is running the maximum change in temperature is calculated, using the 'max' and 'abs' functions within MATLAB. These functions are used to take the maximum absolute values from the two different matrixes and the values are subtracted to work out the difference. After this, the temperature matrix is updated, and the iterations counter is incremented. Finally, after leaving the while loop, the total time is calculated by multiplying the number of iterations by the time step.

```

% Initialize temperature field with boundary conditions
analytical_solution_matrix = zeros(Nx, Ny);

% Set left, right, and top boundaries to zero
analytical_solution_matrix(:, [1, end]) = 0; % Left and right boundaries
analytical_solution_matrix(1, :) = 0; % Top boundary

% Set interior of the matrix to zero
analytical_solution_matrix(2:Nx-1, 2:Ny-1) = 0;

% Function for the top boundary condition T(x, 0, t) = f(x)
dx = Lx / (Nx - 1); % Grid spacing in the x-direction
x_values = linspace(0, Lx, Nx); % Use linspace for x values
f_top = @(x) 12.5 * x .* (0 <= x) .* (x <= 2/3 * Lx) + (-25 * x + 150) .* ((2/3 * Lx) < x) .* (x <= Lx);
analytical_solution_matrix(1, 2:Ny-1) = f_top(x_values(2:Nx-1));

% Display the initial temperature matrix
disp('Initial Analytical Temperature Matrix:');
disp(analytical_solution_matrix);

```

Figure 10 - MATLAB Code Section 7

In the section of code displayed in Figure 10, a new matrix is initialised to process the analytical solution. The method of doing so is the same as for the Explicit Finite Difference Method section of the code. After the matrix is initialised, it is then displayed to the user to clarify that it has been done so correctly.

```

% Iterate through the interior elements of the matrix
for i = 2:Nx-1
    for j = 2:Ny-1
        % Calculate the value using the given formula
        for n = 1:200 % You can adjust the upper limit of n based on your requirements
            analytical_solution_matrix(i, j) = analytical_solution_matrix(i, j) + (sin((2 * n * pi) / 3) / (n^2 * sinh(n * pi))) * (sin((n * pi * i) / 6)) * (sinh((n * pi * j) / 6));
        end
        analytical_solution_matrix(i, j) = analytical_solution_matrix(i, j) * 450 / pi^2;
    end
end

```

Figure 11 - MATLAB Code Section 8

This section of the code processes the analytical solution by iterating over the interior elements of the analytical solution matrix. This iteration process will be carried out two hundred times before it is completed. The final matrix is then scaled after the two hundred iterations are completed.

```

% Display the time it takes to reach thermal equilibrium
disp(['Time to reach thermal equilibrium: ' num2str(total_time) ' seconds']);

% Display the temperature matrix at equilibrium
disp('Temperature Matrix at Thermal Equilibrium: ');
disp(T);

% Display the temperature matrix at equilibrium
disp('Analytical Temperature Matrix: ');
disp(analytical_solution_matrix);

% Plot the final temperature distribution
figure;
imagesc(linspace(0, Ly, Ny), linspace(0, Lx, Nx), T);
colorbar;
title('Temperature Distribution in the Aluminum Plate (Thermal Equilibrium)');
xlabel('x-axis');
ylabel('y-axis');

% Plot the final temperature distribution (analytical)
figure;
imagesc(linspace(0, Ly, Ny), linspace(0, Lx, Nx), analytical_solution_matrix);
colorbar;
title('Temperature Distribution in the Aluminum Plate (Analytical)');
xlabel('x-axis');
ylabel('y-axis');

```

Figure 12 - MATLAB Code Section 9

Figure 12 shows the final section of the code, in which results of both the numerical and analytical solutions are displayed to the user. Firstly, the total time to reach thermal equilibrium is converted from a number to a string before being displayed. Then both the final numerical and analytical temperature matrixes are displayed to the user. Finally, a 2D graph is created for both the numerical and analytical solutions.

6 Analysis of Code Results

Now that the code has been outlined and explained, analysis and validation can be completed to examine the behaviour of the code. Included in the validation will be a sensitivity analysis, to determine how the code responds to changing the input parameters. The input parameters that will be focused on are the grid spacing, time step and thermal equilibrium tolerance.

6.1 Code Validation

6.1.1 Grid Spacing

The first input parameter to cover in the validation is the grid spacing. To determine the effect that an increase and decrease in the grid spacing has on the simulation, the value will be increased and decreased by 0.5. After doing so the following outcomes were achieved:

```
Temperature Matrix at Thermal Equilibrium:
0  12.5000  25.0000  37.5000  50.0000  25.0000  0
0  8.8574  17.0857  23.5718  25.5516  14.7144  0
0  5.8503  10.9250  14.1627  13.9316  8.3124  0
0  3.6301  6.6205  8.2450  7.7190  4.6149  0
0  2.0624  3.7044  4.5037  4.1068  2.4412  0
0  0.9265  1.6502  1.9809  1.7827  1.0544  0
0  0  0  0  0  0  0
```

Figure 13 - Output Response Grid Spacing of 1 (Matrix)

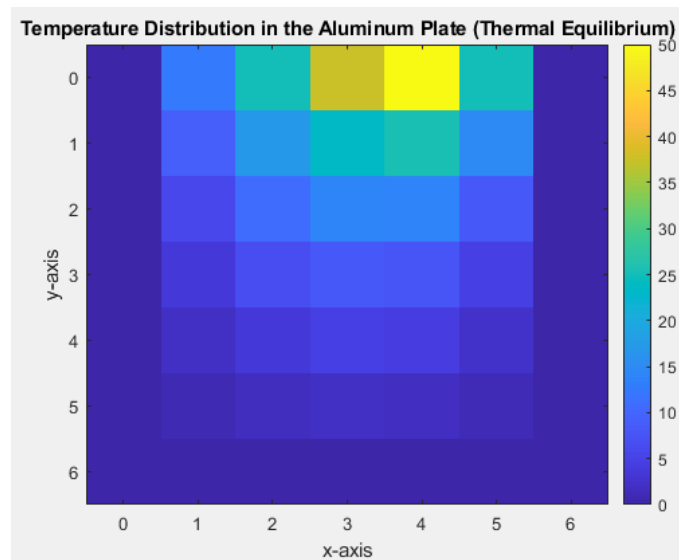


Figure 14 - Output Response Grid Spacing of 1 (Graph)

```
Temperature Matrix at Thermal Equilibrium:
0  18.7500  37.5000  37.5000  0
0  10.4383  17.8778  15.4606  0
0  5.1546  8.1535  6.4939  0
0  2.0678  3.1457  2.4026  0
0  0  0  0  0
```

Figure 15 - Output Response Grid Spacing of 1.5 (Matrix)

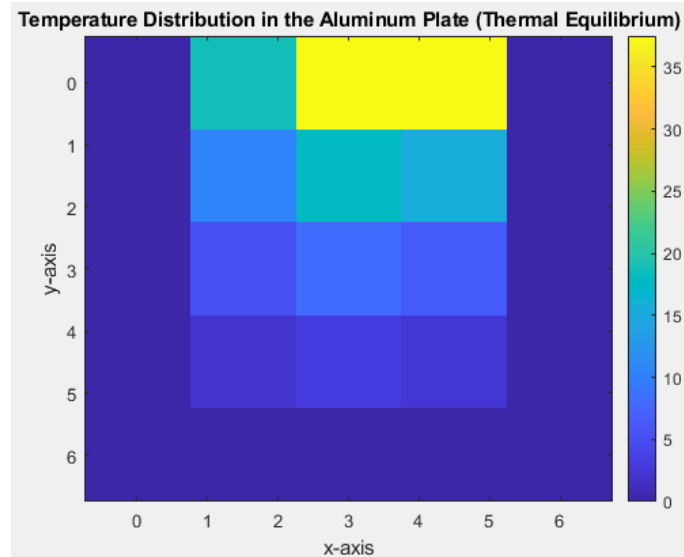


Figure 16 - Output Response Grid Spacing of 1.5 (Graph)

Temperature Matrix at Thermal Equilibrium:

0	6.2500	12.5000	18.7500	25.0000	31.2500	37.5000	43.7500	50.0000	56.2500	62.5000	68.7500	75.0000	81.2500	87.5000	93.7500	100.0000
0	5.8945	10.6292	15.3639	20.0986	24.8333	29.5680	34.3027	39.0374	43.7721	48.5068	53.2415	57.9762	62.7109	67.4456	72.1803	76.9150
0	4.4603	8.9403	13.4203	17.9003	22.3803	26.8603	31.3403	35.8203	40.3003	44.7803	49.2603	53.7403	58.2203	62.7003	67.1803	71.6603
0	3.0261	6.0521	9.0781	12.1041	15.1301	18.1561	21.1821	24.2081	27.2341	30.2601	33.2861	36.3121	39.3381	42.3641	45.3901	48.4161
0	1.5919	3.1838	4.7757	6.3676	7.9595	9.5514	11.1433	12.7352	14.3271	15.9190	17.5109	19.1028	20.6947	22.2866	23.8785	25.4704
0	0.1577	0.3154	0.4731	0.6308	0.7885	0.9462	1.1039	1.2616	1.4193	1.5770	1.7347	1.8924	2.0501	2.2078	2.3655	2.5232
0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Figure 17 - Output Response Grid Spacing of 0.5 (Matrix)

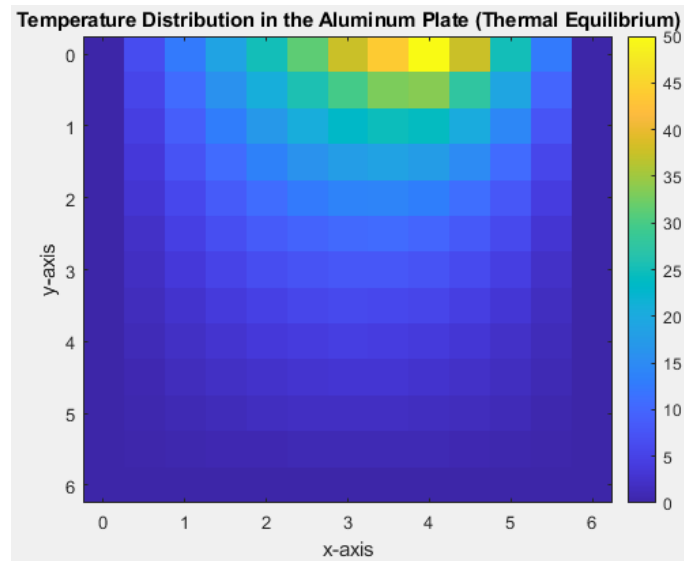


Figure 18 - Output Response Grid Spacing of 0.5 (Graph)

After increasing and decreasing the grid spacing, it can be determined that a smaller grid spacing lead to an increased resolution whereas an increased grid spacing lead to a lower resolution. To run the simulation with a smaller grid spacing, a decreased time step was also required.

It is important to note that the grid spacing has a critical affect on the accuracy of the solution. Truncation errors can be introduced, depending on the value of grid spacing entered (Smith, 2003). These errors can be reduced by decreasing the grid spacing value, hence the error being more prevalent in coarser grids. Truncation errors occur during the discretisation of equations that are applied to a grid, making them applicable to the Explicit Finite Difference Method.

6.1.2 Time Step

It can be determined from the mathematical model that smaller time steps resulted in the simulation being of a higher accuracy. The “time step size has a significant effect on accuracy and performance” (SIMION, 2021). Generally, “smaller steps generally improve accuracy but take longer to compute because there are more steps” (SIMION, 2021). This is also supported by the output of the simulation. See below for the results:

Temperature Matrix at Thermal Equilibrium:							
0	12.5000	25.0000	37.5000	50.0000	25.0000		0
0	8.8693	17.1063	23.5956	25.5722	14.7263		0
0	5.8709	10.9606	14.2039	13.9673	8.3330		0
0	3.6539	6.6617	8.2926	7.7602	4.6387		0
0	2.0830	3.7400	4.5448	4.1425	2.4618		0
0	0.9384	1.6707	2.0047	1.8033	1.0663		0
0	0	0	0	0	0		0

Figure 19 - Output Matrix (Time Step - 3600)

Temperature Matrix at Thermal Equilibrium:							
0	12.5000	25.0000	37.5000	50.0000	25.0000		0
0	8.8574	17.0857	23.5718	25.5516	14.7144		0
0	5.8503	10.9250	14.1627	13.9316	8.3124		0
0	3.6301	6.6205	8.2450	7.7190	4.6149		0
0	2.0624	3.7044	4.5037	4.1068	2.4412		0
0	0.9265	1.6502	1.9809	1.7827	1.0544		0
0	0	0	0	0	0		0

Figure 20 - Output Matrix (Time Step - 60)

Even though the accuracy is improved with the decrease in the time step, it can have a negative effect on the time it takes for the simulation to compute. Ultimately the simulation will take longer to complete with a smaller time step. It is important to consider the trade off between the computational efficiency and the accuracy of the simulation. An increased computing duration could be a small price to pay, if it means that the solution output by the simulation is of a much higher accuracy. Computing efficiency can also be increased by changing the various hardware elements.

When evaluating the use of the time step within the model, it is essential to consider the potential errors that may arise. One of these is round off errors, which occur due to the computing via finite precision arithmetic. When a computer processes calculations multiple times, round off errors will likely accumulate, due to the results being rounded multiple times over. Steps could have potentially been taken to mitigate the effects of round off errors within the simulation, such as specifying the number of decimal places to compute the result to.

6.1.3 Thermal Equilibrium Tolerance

When making the tolerance for thermal equilibrium tighter, the results output from the simulation were of a higher accuracy. After opening the tolerance up, the simulation was less accurate and a greater fluctuation in temperature could be seen. Both examples are shown below:

```
Temperature Matrix at Thermal Equilibrium:
0  12.5000  25.0000  37.5000  50.0000  25.0000  0
0  8.6969  16.8075  23.2502  25.2728  14.5533  0
0  5.5737  10.4455  13.6084  13.4511  8.0348  0
0  3.3129  6.0706  7.6095  7.1679  4.2966  0
0  1.7896  3.2315  3.9570  3.6330  2.1674  0
0  0.7699  1.3785  1.6670  1.5105  0.8971  0
0  0  0  0  0  0  0
```

Figure 21 - Output Matrix (Tight Thermal Equilibrium Tolerance)

```
Temperature Matrix at Thermal Equilibrium:
0  12.5000  25.0000  37.5000  50.0000  25.0000  0
0  8.8693  17.1063  23.5956  25.5722  14.7263  0
0  5.8709  10.9606  14.2039  13.9673  8.3330  0
0  3.6539  6.6617  8.2926  7.7602  4.6387  0
0  2.0830  3.7400  4.5448  4.1425  2.4618  0
0  0.9384  1.6707  2.0047  1.8033  1.0663  0
0  0  0  0  0  0  0
```

Figure 22 - Output Matrix (Open Thermal Equilibrium Tolerance)

Even though a higher accuracy was observed in this case, this outcome may not always be the outcome. There are various factors which contribute to this, one of which being the lack of physical realism within the simulation. For example, real world applications may not always achieve perfect thermal equilibrium, whereas the model will do so. This results in unrealistic behaviours being encountered.

6.2 Sensitivity Analysis

Conducting a sensitivity analysis is a crucial step when analysing a mathematical simulation model. It assists with understanding how the model output performs when the various input parameters, and variables, are modified. The sensitivity analysis will also help with understanding the relationship between the models input and output parameters, to determine if there are any trends in the data. During the sensitivity analysis, the focus will be specifically on the grid spacing, time step and thermal equilibrium tolerance, against the time taken to reach thermal equilibrium.

6.2.1 Grid Spacing Against Time to Reach Thermal Equilibrium

Grid Spacing (h)	Time Step (Seconds)	Time to Reach Thermal Equilibrium (Seconds)	Tolerance
3	150	209700	0.0001
1.5	150	190200	0.0001
1	150	187800	0.0001
0.75	150	185700	0.0001
0.5	150	185100	0.0001
0.25	150	184500	0.0001
0.1	150	58350	0.0001

Table 1 - Grid Spacing Versus Time Taken to Reach Thermal Equilibrium

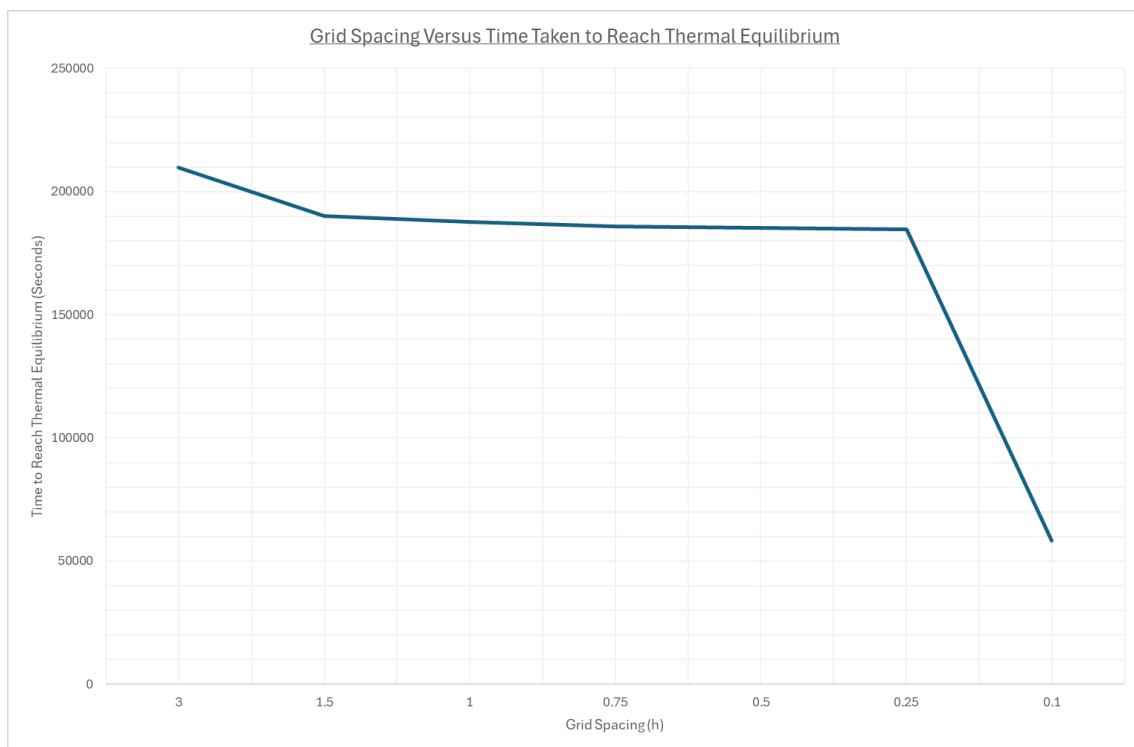


Figure 23 - Grid Spacing Versus Time Taken to Reach Thermal Equilibrium Plot

The above sensitivity analysis, performed on the grid spacing, shows that the time taken to reach thermal equilibrium reduces as the grid spacing is decreased. This supports the conclusions made in the previous section.

6.2.2 Time Step Against Time to Reach Thermal Equilibrium

Grid Spacing (h)	Time Step (Seconds)	Time to Reach Thermal Equilibrium (Seconds)	Tolerance
1	2000	258000	0.0001
1	1000	240000	0.0001
1	500	222000	0.0001
1	250	202500	0.0001
1	100	176200	0.0001
1	50	156100	0.0001
1	25	136025	0.0001

Table 2 - Time Step Versus Time Taken to Reach Thermal Equilibrium

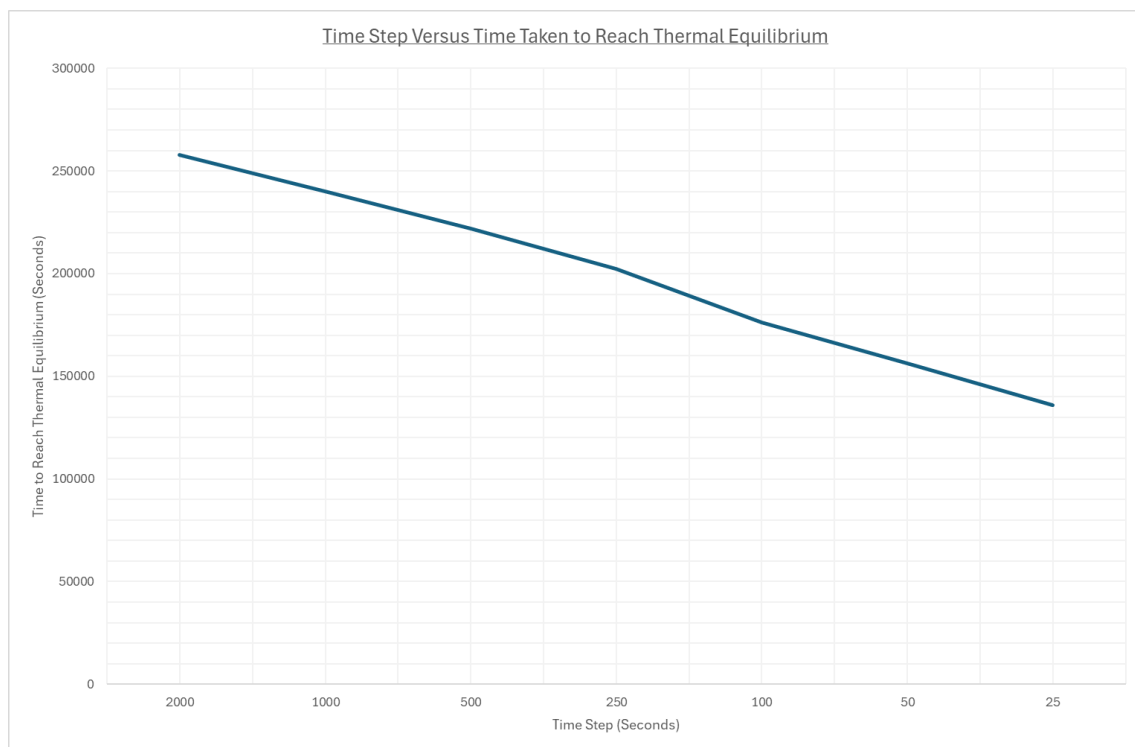


Figure 24 - Time Step Versus Time Taken to Reach Thermal Equilibrium Plot

Based on the above sensitivity analysis, of the time step, it can be determined that a decrease in time taken to reach thermal equilibrium is observed when the time step is lowered. As mentioned in the previous section, it is important to consider the introduction of truncation errors when the time step is decreased. Also, the trade offs associated with the computing times, which are discussed in the previous chapter.

6.2.3 Thermal Equilibrium Tolerance Against Time to Reach Thermal Equilibrium

Grid Spacing (h)	Time Step (Seconds)	Time to Reach Thermal Equilibrium (Seconds)	Tolerance
1	150	321300	0.000001
1	150	254550	0.00001
1	150	187800	0.0001
1	150	121050	0.001
1	150	54300	0.01
1	150	11250	0.1

Table 3 - Thermal Equilibrium Tolerance Versus Time Taken to Reach Thermal Equilibrium

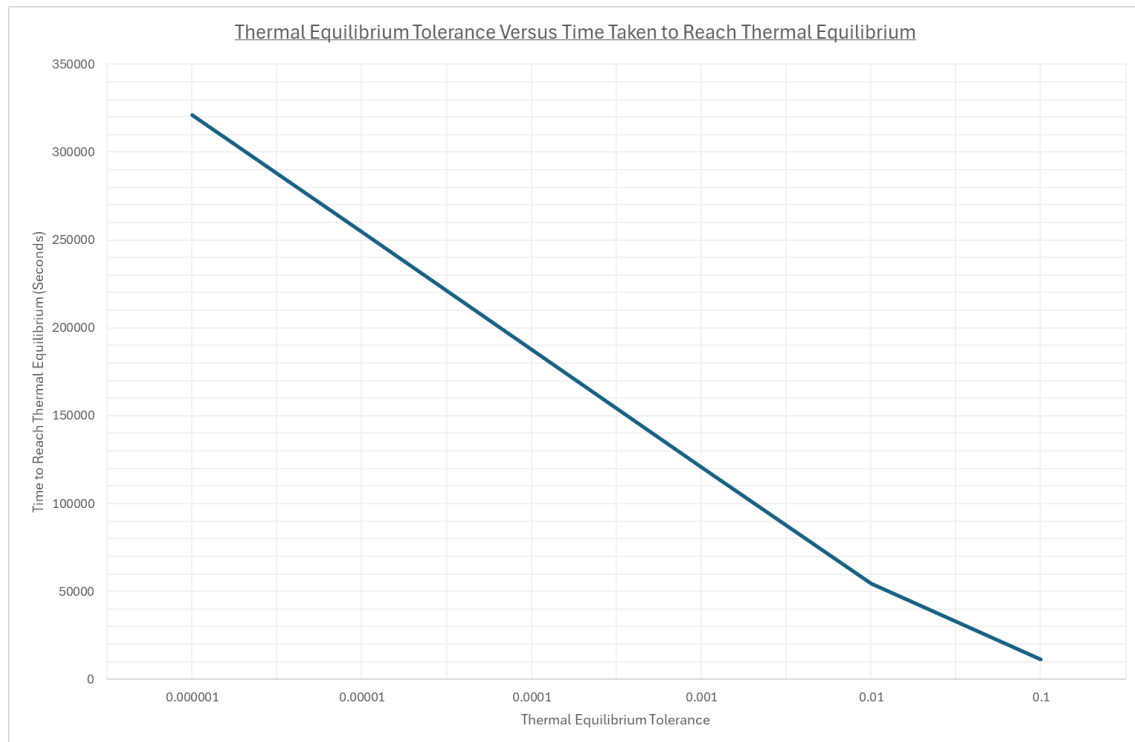


Figure 25 - Thermal Equilibrium Tolerance Versus Time Taken to Reach Thermal Equilibrium Plot

Analysing the above information, a larger thermal equilibrium tolerance results in a decreased time to reach thermal equilibrium. Even though the computational time is reduced with a larger tolerance, the overall accuracy of the simulation will be lower. The trade-offs between the various input parameters must be taken into consideration, to select the most appropriate values which optimise the accuracy of the model, along with the computational efficiency.

7 Model Limitations

Now that the results of the code have been analysed, it is time to identify areas of limitations with the mathematical model. These limitations could be caused by various factors, which will be discussed in the following section.

7.1 Limitations due to the Coding

One of the limitations with the coding that can be drawn is its inability to handle more complex geometries. This is due to the fixed boundary conditions that are programmed. Another limitation associated with the coding is that it does not fully capture the complexity of real-world physics of a material reaching thermal equilibrium. As a result, the material behaviour during the heat transfer process may not be entirely accurate.

7.2 Limitations due to Round Off Errors

Due to the employment of a computational model, round off errors will likely occur. This is due to the inability to compute results to an infinite level of precision. When the simulation computes the results over time, the errors will accumulate leading to inaccuracy in the final solution.

Unfortunately, these types of errors are not just specific to this application and are often present in computational simulations.

7.3 Limitations due to Truncation Errors

Truncation errors occur during the discretisation of equations that are applied to a grid. As mentioned in the analysis and validation section, decreasing the grid spacing value can reduce these types of errors. Based on this fact, truncation errors are more prevalent in coarser grids compared to ones of a finer resolution. Overall, this will achieve a higher accuracy simulation, which will also increase the computational time.

8 Improvements for Future Developments

There are multiple improvements that could be made for future developments of the code. One of the ways in which an improvement could be made is by choosing a more complex numerical method for the model to compute. More specifically, the Finite Element Method could be adopted instead. The key differences between the EFDM and the FEM include the FEM being able to compute more complex geometries and provide more accurate modelling. Figure 26 below displays a breakdown of the evolution of numerical methods over the years.

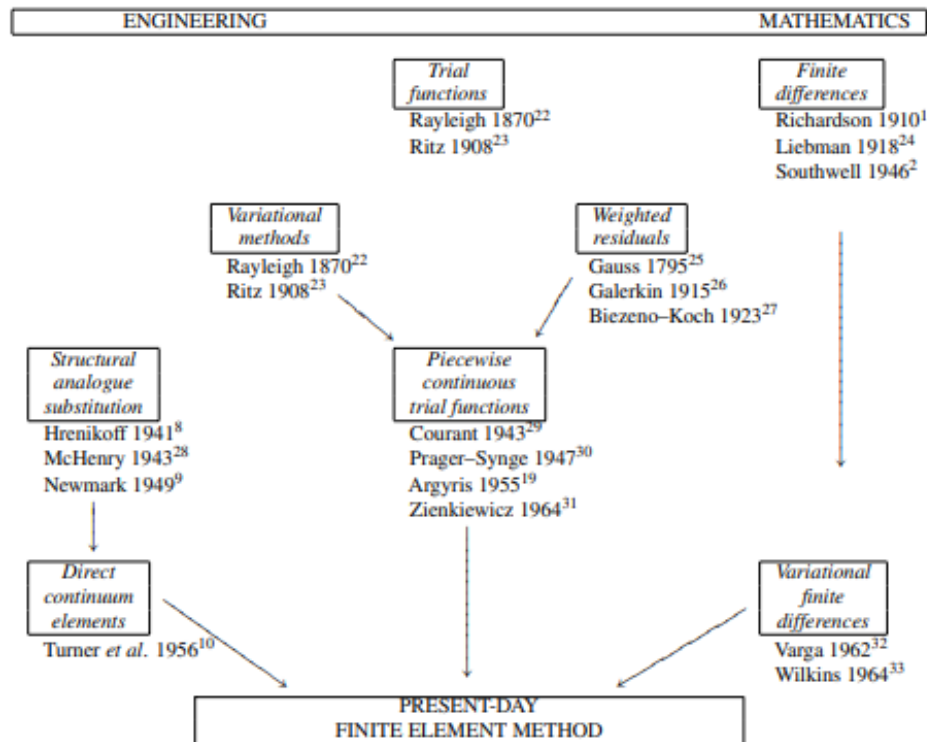


Figure 26 - History of Numerical Methods (Zienkiewicz, 2005)

Other advantages of FEM include being able to compute solutions for an unstructured mesh, whereas this is a limitation with EFDM. It would also allow for analysis of non-linear scenarios unlike the numerical method used within this scenario. Some of the simpler improvements that could be made to the current solution include implementing a higher definition visualisation means. This is due to the current animation being basic compared to what is available. Tools can also be utilised for analysing the temperature post the visualisation being generated.

The Implicit Finite Difference Method (IFDM) could also be utilised in the future, as it is generally considered to be the more stable and accurate numerical method. One of the reasons that the IFDM is considered more stable than the EFDM, is that it does not require a strict stability criterion. It can also handle larger time steps which provides a higher degree of accuracy. The code could also be enhanced more complex boundary conditions, which would make the model more suitable for real world scenarios. For example, the introduction of dynamic boundary conditions would further improve the simulation.

Implementing the improvements suggested above would make the numerical model more robust, along with increasing accuracy.

9 Conclusion

To conclude this report, the Explicit Finite Difference Method has been successfully utilised to model the temperature distribution within an aluminium plate over time. The governing equation, outlined at the beginning of the report, was discretised employing Taylor Series expansions. This enabled the implementation of forward, backward, and central difference approximations, which were subsequently deployed in the computational model.

The analysis and validation section highlighted the impact of the input parameters, including grid spacing, time step and thermal equilibrium tolerance, on the simulation's accuracy. Smaller grid spacing and time steps resulted in increased simulation accuracy but required greater computational resource. Also included was the importance of the stability criterion, to achieve reliable results.

Even with the robustness of the EFDM, the limitations of the model were considered, such as handling complex geometries and the potential for accumulating round off errors. Also, the potential of truncation errors based on the grid spacing selected by the user. These limitations presented opportunities for future improvements, such as the introduction of the Implicit Finite Difference Method.

As a final point, the EFDM provided a useful tool for computationally simulating heat transfer in two dimensional domains. Future developments could further improve this simulation, achieving a more precise and higher accuracy mathematical simulation model.

10 References

Croft, D. L. D., 1977. *Heat Transfer Calculations Using Finite Difference Equations*. London: Applied Science Publishers.

Daileda, R., 2012. *Analytical Solution of Temperature at Thermal Equilibrium*. San Antonio: Trinity University.

SIMION, 2021. *SIMION*. [Online]

Available at:

[https://simion.com/info/time_step_size.html#:~:text=Time%20step%20size%20has%20a,e.g.%20PA%20grid%20unit%20size\).](https://simion.com/info/time_step_size.html#:~:text=Time%20step%20size%20has%20a,e.g.%20PA%20grid%20unit%20size).)

Smith, G., 2003. *Numerical Solution of Partial Difference Methods*. 3rd ed. Oxford: Clarendon Press.

Zienkiewicz, O. T. R. Z. J., 2005. *The Finite Element Method: Its Basis and Fundamentals*.

11 Appendix

```

% Material properties for aluminum
k = 143; % Thermal conductivity (W/m.C)
rho = 2.8e3; % Density (kg/m^3)
cp = 795; % Specific heat (J/kg.C)

% Calculate thermal diffusivity
alpha = k / (rho * cp);

% Plate dimensions
Lx = 6; % Length of the plate in the x-direction
Ly = 6; % Length of the plate in the y-direction

% Prompt user for grid spacing
validGrid = false;
while ~validGrid
    gridSpacing = input('Enter the grid spacing (must comply with plate dimensions):');
    if gridSpacing <= 0 || gridSpacing > min(Lx, Ly)
        disp('Invalid grid spacing. Please choose a value within the plate dimensions.');
```

↙

```

    else
        validGrid = true;
    end
end

% Grid parameters
Nx = (Lx / gridSpacing) + 1; % Number of grid points in the x-direction
Ny = (Ly / gridSpacing) + 1; % Number of grid points in the y-direction
dx = Lx / (Nx - 1); % Grid spacing in the x-direction
dy = Ly / (Ny - 1); % Grid spacing in the y-direction

% Time parameters
dt = input('Enter Time Step Value: '); % Time step

% Calculate stability criterion
stabilityCriterion = alpha * dt / (gridSpacing^2);

% Check if the stability criterion is met
if stabilityCriterion > 0.25
    disp('Warning: Stability criterion may not be fulfilled. Consider choosing a smaller time step.');
```

↙

```

    % You may choose to prompt the user to enter a different time step or handle it accordingly.
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initialize temperature field with boundary conditions
T = zeros(Nx, Ny);

% Set left, right, and bottom boundaries to zero
T(:, [1, end]) = 0; % Left and right boundaries
T(1, :) = 0; % Bottom boundary
```

Figure 27 - Code Page 1

```

% Set interior of the matrix to zero
T(2:Nx-1, 2:Ny-1) = 0;

% Function for the top boundary condition T(x, 0, t) = f(x)
dx = Lx / (Nx - 1); % Grid spacing in the x-direction
x_values = linspace(0, Lx, Nx); % Use linspace for x values
f_top = @(x) 12.5 * x .* (0 <= x) .* (x <= 2/3 * Lx) + (-25 * x + 150) .* ((2/3 * Lx) <
< x) .* (x <= Lx);
T(1, 2:Ny-1) = f_top(x_values(2:Nx-1));

% Display the initial temperature matrix
disp('Initial Temperature Matrix:');
disp(T);

% Initialize the change in temperature
delta_T = Inf;

% Initialize iteration counter
iterations = 0;

% Run the simulation until thermal equilibrium is reached
while delta_T > 1e-4
    T_new = T;
    for i = 2:Nx-1
        for j = 2:Ny-1
            T_new(i, j) = T(i, j) + alpha * dt * ((T(i+1, j) - 2*T(i, j) + T(i-1, j))) /
/ dx^2 + (T(i, j+1) - 2*T(i, j) + T(i, j-1)) / dy^2);
        end
    end

    % Calculate the maximum change in temperature
    delta_T = max(max(abs(T_new - T)));

    % Update the temperature matrix
    T = T_new;

    % Increment iteration counter
    iterations = iterations + 1;
end

% Calculate the time taken for the simulation
total_time = iterations * dt;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initialize temperature field with boundary conditions
analytical_solution_matrix = zeros(Nx, Ny);

% Set left, right, and top boundaries to zero
analytical_solution_matrix(:, [1, end]) = 0; % Left and right boundaries
analytical_solution_matrix(1, :) = 0; % Top boundary

% Set interior of the matrix to zero
analytical_solution_matrix(2:Nx-1, 2:Ny-1) = 0;

```

Figure 28 - Code Page 2

```

% Function for the top boundary condition  $T(x, 0, t) = f(x)$ 
dx = Lx / (Nx - 1); % Grid spacing in the x-direction
x_values = linspace(0, Lx, Nx); % Use linspace for x values
f_top = @(x) 12.5 * x .* (0 <= x) .* (x <= 2/3 * Lx) + (-25 * x + 150) .* ((2/3 * Lx) <
< x) .* (x <= Lx);
analytical_solution_matrix(1, 2:Ny-1) = f_top(x_values(2:Nx-1));

% Display the initial temperature matrix
disp('Initial Analytical Temperature Matrix:');
disp(analytical_solution_matrix);

% Iterate through the interior elements of the matrix
for i = 2:Nx-1
    for j = 2:Ny-1
        % Calculate the value using the given formula
        for n = 1:200 % You can adjust the upper limit of n based on your requirements
            analytical_solution_matrix(i, j) = analytical_solution_matrix(i, j) + (sin(
((2 * n * pi) / 3) / (n^2 * sinh(n * pi))) * (sin((n * pi * i) / 6)) * (sinh((n * pi * j) / 6)));
        end
        analytical_solution_matrix(i, j) = analytical_solution_matrix(i, j) * 450 / pi^2;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Display the time it takes to reach thermal equilibrium
disp(['Time to reach thermal equilibrium: ' num2str(total_time) ' seconds']);

% Display the temperature matrix at equilibrium
disp('Temperature Matrix at Thermal Equilibrium: ');
disp(T);

% Display the temperature matrix at equilibrium
disp('Analytical Temperature Matrix: ');
disp(analytical_solution_matrix);

% Plot the final temperature distribution
figure;
imagesc(linspace(0, Ly, Ny), linspace(0, Lx, Nx), T);
colorbar;
title('Temperature Distribution in the Aluminum Plate (Thermal Equilibrium)');
xlabel('x-axis');
ylabel('y-axis');

% Plot the final temperature distribution (analytical)
figure;
imagesc(linspace(0, Ly, Ny), linspace(0, Lx, Nx), analytical_solution_matrix);
colorbar;
title('Temperature Distribution in the Aluminum Plate (Analytical)');
xlabel('x-axis');
ylabel('y-axis');

```

Figure 29 - Code Page 3