



University of Wales Trinity Saint David
School of Engineering

Transient Thermal Diffusion Model Using The Explicit Finite Difference Method

Michael Anderson
P151178

12/11/2019

Advanced Computational Methods: MATLAB
M6X00408

Dr Arnould Marotin

Contents

1) Introduction	1
2) Mathematical Analysis	2
2.1. Diffusion Derivation.....	2
2.2. Taylor Series Derivations	3
2.3. Finite Difference Method	6
3) Script Construction.....	9
3.1. Aluminium Properties, User Inputs & Stability Criterion	9
3.2. Linear Heat Distribution – Script Validation	11
3.3. Gaussian Heat Distribution	13
4) Code Evaluation.....	15
4.1. Code Analysis.....	15
4.2. Code Limitations	17
4.3. Potential Improvements	17
5) Conclusion	18
6) References	19
7) Appendix.....	21
7.1. MATLAB Script	21
7.1.1. Properties of Aluminium.....	21
7.1.2. Text Box Structure Setup.....	21
7.1.3. X & Y Dimension User Inputs.....	21
7.1.4. Time Step Check	22
7.1.5. Grid Spacing & Fourier's Number Check	23
7.1.6. Plate Visualisation	25
7.1.7. Validation Experiment.....	26
7.1.8. Progression to Gaussian Question.....	28
7.1.9. Standard Deviation	29
7.1.10. Gaussian Experiment	29

Table of Figures

Figure 1 - A Basic Plate with An Applied Mesh At Time Step "t" With A Chosen Node (T_P) And Relative Surrounding Nodes	6
Figure 2 – X-Axis Dimension Input Request.....	9
Figure 3 – Time Step Input Request.....	9
Figure 4 – Time Step Input Request.....	9
Figure 5 – Time Step Input Request.....	10
Figure 6 – Visual Representation of The Plat Matrix	10
Figure 7 – Empty Plate Matrix (Zeros) With Temperatures Applied Around the Edges.....	11
Figure 8 – Defining the Old Plate for New Iteration	11
Figure 9 – Time Step Input Request.....	11
Figure 10 – Final Iteration of The Plate Matrix Once Reaching Thermal Equilibrium	11
Figure 11 – Heat Map of the Final Plate Matrix Once Reaching Thermal Equilibrium.....	12
Figure 12 - Gaussian Temperature Distribution Using Standard Deviation Equation.....	13
Figure 13 - Empty Plate Matrix with Gaussian Heat Distribution Applied	13
Figure 14 – Heat Map of the Final Plate Matrix Once Reaching Thermal Equilibrium Without 'flipud' Function	14
Figure 15 – Heat Map of the Final Plate Matrix Once Reaching Thermal Equilibrium.....	14
Figure 16 – Effect of Changing the Exit Criteria With 0.1 Second Time Step – Small (Left) – Large (Right).....	16
Figure 17 – Effect of Changing the Grid Spacing Value – 2m (Left) – 0.1m (Right)	16

Table of Tables

Table 1 – Effects of Changing the Time Step & Exit Criteria.....	15
Table 2 – Effect of Changing the Grid Spacing Value.....	16

1) Introduction

This report will detail the mathematical approach used to calculate the temperature propagation throughout a theoretical 2-dimensional (2D) aluminium plate using MATLAB and the explicit Finite Difference Method (FDM).

The mathematics behind FDM shall be discussed, the derivation for the Taylor series equation will lead into the forward, backward and central difference approximations, along with why they're used within this experiment.

To aid with solving the problem at hand, a MATLAB script will be created to implement the FDM equation. The methodology will detail the steps taken to create the script and the commands used within it.

The script will then be scrutinised for efficiency. Within this, studies were conducted to determine the effects caused by changing one variable, on both the computational time, and the time taken for the plate to reach thermal equilibrium.

The results of these studies will then be analysed, from which, both the limitations and potential improvements will be discussed to enhance the user's experience and the credibility of the script and the results generated from it.

2) Mathematical Analysis

2.1. Diffusion Derivation

Diffusion is described as “the movement of a substance from an area of high concentration, to an area of low concentration”. (BBC Bitesize, 2020)

For this experiment, it is required to calculate and then display the propagation of temperature diffusion through a theoretical 2D plate. To do this, it is necessary to derive the equation for diffusion to be able to perform the required calculation.

By using the partial differential equation that governs the diffusion and random motion of micro-particles within a material:

$$\nabla \alpha \nabla T = \frac{\delta T}{\delta t}$$

Equation 2-1

Then expanded out in 2D form:

$$\frac{\delta}{\delta x} * \alpha * \frac{\delta T}{\delta x} + \frac{\delta}{\delta y} * \alpha * \frac{\delta T}{\delta y} = \frac{\delta T}{\delta t}$$

Equation 2-2

By summing all the terms together, this can then be expressed as a differential equation:

$$\alpha * \frac{\delta^2 T}{\delta x^2} + \alpha * \frac{\delta^2 T}{\delta y^2} = \frac{\delta T}{\delta t}$$

Equation 2-3

This can be factorised:

$$\alpha * \left(\frac{\delta^2 T}{\delta x^2} + \frac{\delta^2 T}{\delta y^2} \right) = \frac{\delta T}{\delta t}$$

Equation 2-4

By dividing both sides by ‘ α ’, the final equation for diffusion is derived:

$$\frac{\delta^2 T}{\delta x^2} + \frac{\delta^2 T}{\delta y^2} = \frac{1}{\alpha} * \frac{\delta T}{\delta t}$$

Equation 2-5

2.2. Taylor Series Derivations

Originally formulated by mathematician James Gregory, then adapted formally by Brook Taylor, the 'Taylor series' uses a finite number of terms as means of an approximation on a polynomial equation. (Croft, 1977)

$$f(x) = f(x_0) + f^i(x_0) * \frac{x - x_0}{1!} + f^{ii}(x_0) * \frac{(x - x_0)^2}{2!} + f^{iii}(x_0) * \frac{(x - x_0)^3}{3!} \dots$$

Equation 2-6

The Taylor series was then adapted by Colin Maclaurin, shortening the equation when " $x_0 = 0$ ", this enables the calculation of infinitely differentiable functions summed up within one singular term, rather than writing them out one by one. (Lewis, et al., 1996)

$$\sum_{n=0}^{\infty} f^{(n)}(x_0) * \frac{(x - x_0)^n}{n!}$$

Equation 2-7

For this experiment, it is more appropriate to use the Taylor series (Equation 2-6), rather than the shortened version (Equation 2-7). By following the steps detailed below, the Taylor series can be expanded and directly related to the equation governing diffusion.

$$f(x) = f(x_0) + f^i(x_0) * \frac{x - x_0}{1!} + f^{ii}(x_0) * \frac{(x - x_0)^2}{2!} + f^{iii}(x_0) * \frac{(x - x_0)^3}{3!} \dots$$

Equation 2-8

If " $x = x_0 + h$ ":

$$f(x_0 + h) = f(x_0) + f^i(x_0) * \frac{x_0 + h - x_0}{1!} + f^{ii}(x_0) * \frac{(x_0 + h - x_0)^2}{2!} + f^{iii}(x_0) * \frac{(x_0 + h - x_0)^3}{3!} \dots$$

Equation 2-9

So by summing terms:

$$f(x_0 + h) = f(x_0) + f^i(x_0) * \frac{h}{1!} + f^{ii}(x_0) * \frac{(h)^2}{2!} + f^{iii}(x_0) * \frac{(h)^3}{3!} \dots$$

Equation 2-10

This can be transposed to make " $f^i(x_0) * \frac{h}{1!}$ " the subject:

$$f^i(x_0) * \frac{h}{1!} = f(x_0 + h) - f(x_0) - f^{ii}(x_0) * \frac{(h)^2}{2!} - f^{iii}(x_0) * \frac{(h)^3}{3!}$$

Equation 2-11

But as the factorial "1!" can cancel:

$$f^i(x_0) * h = f(x_0 + h) - f(x_0) - f^{ii}(x_0) * \frac{(h)^2}{2!} - f^{iii}(x_0) * \frac{(h)^3}{3!}$$

Equation 2-12

Divide both sides, except factorial terms, by "h":

$$\frac{f^i(x_0) * h}{h} = \frac{f(x_0 + h) - f(x_0)}{h} - f^{ii}(x_0) * \frac{(h)^2}{2!} - f^{iii}(x_0) * \frac{(h)^3}{3!}$$

Equation 2-13

This leaves a final equation called the “Backward Difference Approximation”:

$$f^i(x_o) = \underbrace{\frac{f(x_o - h) - f(x_o)}{h}}_{\text{Backwards Difference Approximation}} - \underbrace{f^{ii}(x_o) * \frac{(h)^2}{2!} - f^{iii}(x_o) * \frac{(h)^3}{3!}}_{\text{Error Term of Order 'h' = O(h)}}$$

Equation 2-14

It is possible to evaluate the extra derivative to decrease the error:

$$f(x_o + h) = f(x_o) - f^i(x_o) * \frac{h}{1!} + f^{ii}(x_o) * \frac{(h)^2}{2!} - f^{iii}(x_o) * \frac{(h)^3}{3!} \dots$$

Equation 2-15

“ $f(x_o + h)$ ” & “ $f^i(x_o)$ ” swap sides:

$$f^i(x_o) * \frac{h}{1!} = -f(x_o + h) + f(x_o) + f^{ii}(x_o) * \frac{(h)^2}{2!} - f^{iii}(x_o) * \frac{(h)^3}{3!} \dots$$

Equation 2-16

The factorial “1!” cancels, divide by ‘h’ which gives us the “Forward Difference Approximation”:

$$f^i(x_o) = \underbrace{\frac{f(x_o + h) - f(x_o)}{h}}_{\text{Forwards Difference Approximation}}$$

Equation 2-17

For a better approximation and reduce the error even further:

$$1 \text{ ----- } f(x_o + h) = f(x_o) + f^i(x_o) * \frac{h}{1!} + f^{ii}(x_o) * \frac{(h)^2}{2!} + f^{iii}(x_o) * \frac{(h)^3}{3!} \dots$$

Equation 2-18

$$2 \text{ ----- } f(x_o - h) = f(x_o) - f^i(x_o) * \frac{h}{1!} + f^{ii}(x_o) * \frac{(h)^2}{2!} - f^{iii}(x_o) * \frac{(h)^3}{3!} \dots$$

Equation 2-19

By subtracting 1 & 2 (Equation 2-18 & Equation 2-19) from each other:

$$f(x_o + h) - f(x_o - h) = 2 * f^i(x_o) * h + 2 * f^{iii}(x_o) * \frac{(h)^3}{3!} + 2 * f^5(x_o) * \frac{(h)^5}{5!} \dots$$

Equation 2-20

Transpose to make “ $2 * f^i(x_o) * h$ ” the subject:

$$2 * f^i(x_o) * h = f(x_o + h) - f(x_o - h) - 2 * f^{iii}(x_o) * \frac{(h)^3}{3!} - 2 * f^5(x_o) * \frac{(h)^5}{5!} \dots$$

Equation 2-21

Then divide by “2 * h”:

$$\frac{(2 * f^i(x_o) * h)}{2 * h} = \frac{f(x_o + h) - f(x_o - h) - 2 * f^{iii}(x_o) * \frac{(h)^3}{3!} - 2 * f^5(x_o) * \frac{(h)^5}{5!}}{2 * h}$$

Equation 2-22

By collecting and cancelling terms we are left with the “Central Difference Approximation”:

$$f^i(x_o) = \underbrace{\frac{f(x_o + h) - f(x_o - h)}{2 * h}}_{\text{Central Difference Approximation}} - \underbrace{f^{iii}(x_o) * \frac{(h)^2}{3!} - f^5(x_o) * \frac{(h)^4}{5!}}_{\text{Error Term of Order 'h' = O(h^2)}}$$

Equation 2-23

The equations derived above are purely for 1st derivative equations. In order to solve 2nd derivatives, a different approach will be required.

Where before equations 1 & 2 (Equation 2-24 & Equation 2-25) were subtracted from one another, this time they will be added together:

$$1 \text{ ----- } f(x_o + h) = f(x_o) + f^i(x_o) * \frac{h}{1!} + f^{ii}(x_o) * \frac{(h)^2}{2!} + f^{iii}(x_o) * \frac{(h)^3}{3!} \dots$$

Equation 2-24

$$2 \text{ ----- } f(x_o - h) = f(x_o) - f^i(x_o) * \frac{h}{1!} + f^{ii}(x_o) * \frac{(h)^2}{2!} - f^{iii}(x_o) * \frac{(h)^3}{3!} \dots$$

Equation 2-25

Equating to:

$$f(x_o + h) + f(x_o - h) = 2 * f(x_o) + 2 * f^{ii}(x_o) * \frac{(h)^2}{2!} + 2 * f^{iiii}(x_o) * \frac{(h)^4}{4!} + 2 * f^6(x_o) * \frac{(h)^6}{6!} \dots$$

Equation 2-26

Now “2 * fⁱⁱ(x_o) * $\frac{(h)^2}{2!}$ ” can be made the subject:

$$2 * f^{ii}(x_o) * \frac{(h)^2}{2!} = f(x_o + h) + f(x_o - h) - 2 * f(x_o) - 2 * f^{iiii}(x_o) * \frac{(h)^4}{4!} - 2 * f^6(x_o) * \frac{(h)^6}{6!} \dots$$

Equation 2-27

By cancelling terms, leaves the “Central Difference Approximation” for 2nd derivatives:

$$f^{ii}(x_o) * (h)^2 = \underbrace{f(x_o + h) + f(x_o - h) - 2 * f(x_o)}_{\text{Central Difference Approximation}} - \underbrace{2 * f^{iiii}(x_o) * \frac{(h)^4}{4!} - 2 * f^6(x_o) * \frac{(h)^6}{6!}}_{\text{Error Term of Order 'h' = O(h^2)}}$$

Equation 2-28

2.3. Finite Difference Method

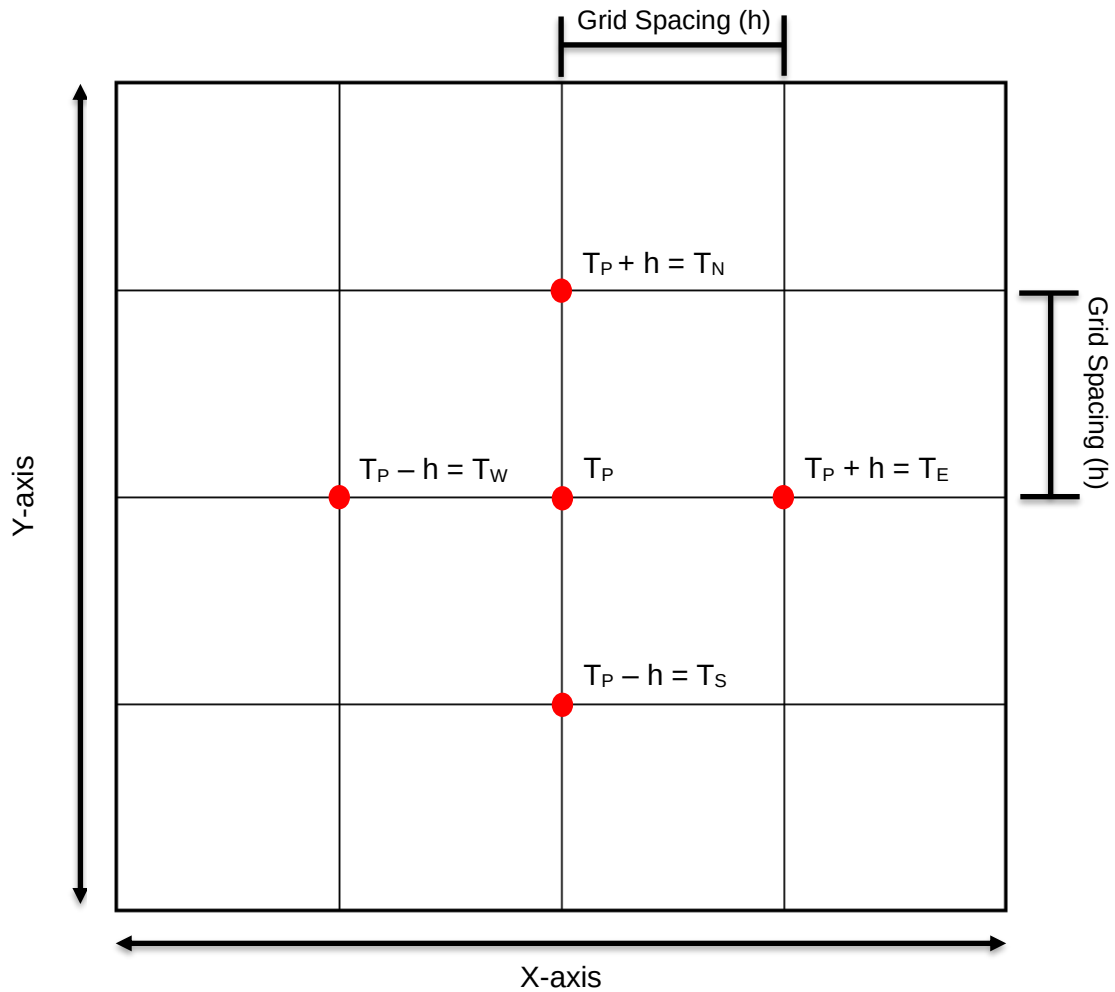


Figure 1 - A Basic Plate with An Applied Mesh At Time Step "t" With A Chosen Node (T_P) And Relative Surrounding Nodes

With association to Figure 1, the 'X' & 'Y' dimensions can be represented with the following relationships:

Along the 'X' Dimension:

$$T_E = T_P + h$$

Equation 2-29

$$T_W = T_P - h$$

Equation 2-30

Along the 'Y' Dimension:

$$T_N = T_P + h$$

Equation 2-31

$$T_S = T_P - h$$

Equation 2-32

These relationships can be used within both the Forward and Central difference approximations to create an equation that will calculate the diffusion of temperature through these nodes, relative to time.

Starting with the equation governing diffusion:

$$\frac{\delta^2 T}{\delta x^2} + \frac{\delta^2 T}{\delta y^2} = \frac{1}{\alpha} * \frac{\delta T}{\delta t}$$

Equation 2-33

Each “T” can be represented with the chosen point “T_P” from Figure 1.

$$\frac{\delta^2 T_P}{\delta x^2} + \frac{\delta^2 T_P}{\delta y^2} = \frac{1}{\alpha} * \frac{\delta T_P}{\delta t}$$

Equation 2-34

Each term within this equation can be related to either the Forward difference approximation for 1st derivative, or Central difference approximation for 2nd derivatives. (Boston University, 2015) (Dortmund University of Technology Faculty of Mathematics, n.d.)

Using the Central Difference Approximation:

$$\frac{\delta^2 T_P}{\delta x^2} = \frac{T_E^t - 2 * T_P^t + T_W^t}{h^2}$$

Equation 2-35

$$\frac{\delta^2 T_P}{\delta y^2} = \frac{T_N^t - 2 * T_P^t + T_S^t}{h^2}$$

Equation 2-36

Using the Forward Difference Approximation:

$$\frac{\delta T_P}{\delta t} = \frac{T_P^{t+\Delta T} - T_P^t}{\Delta t}$$

Equation 2-37

So when implementing the Finite Difference Method (FDM) into the equation governing diffusion:

$$\frac{T_E^t - 2 * T_P^t + T_W^t}{h^2} + \frac{T_N^t - 2 * T_P^t + T_S^t}{h^2} = \frac{1}{\alpha} * \frac{T_P^{t+\Delta T} - T_P^t}{\Delta t}$$

Equation 2-38

This equation can be rearranged to make “T_P^{t+ΔT}” the subject, then factorising terms:

$$T_P^{t+\Delta T} = \frac{\alpha \Delta T}{h^2} \left(T_E^t + T_E^t + T_E^t + T_E^t - \left(4 - \frac{h^2}{\alpha \Delta T} \right) * T_P^t \right)$$

Equation 2-39

When using ‘h’ in relation to time (Δt):

Future:

$$T_P + h = t + \Delta t$$

Equation 2-40

Past:

$$T_P - h = t - \Delta t$$

Equation 2-41

But because we are calculating what happens into the future, we treat the Past as the Present:

$$t = \text{Present}$$

Equation 2-42

The Fourier's number can be represented as:

$$Fo = \frac{\alpha \Delta T}{h^2}$$

Equation 2-43

So the Fourier's number can be substituted into Equation 2-39 to complete the FDM equation:

$$T_P^{t+\Delta T} = Fo \left(T_N^t + T_E^t + T_S^t + T_W^t - \left(4 - \frac{1}{Fo} \right) * T_P^t \right)$$

Equation 2-44

The FDM equation will be used in both experiments, this is because MATLAB is incapable of solving derivative equations, so they need to be converted into linear equations. Using the final derivation (Equation 2-44) to calculate each node within the matrix. (V.Vuorinen, 2016)

Unfortunately, this method only works with the validation plate matrix, as it is surrounded by temperatures being applied to the plate. Considering that the Gaussian distribution has three sides that are fully insulated, there are no nodes in which to reference, e.g. for the first column of the matrix there are no west nodes. This meant that for the first & last columns, the bottom row and the bottom left & right corners needed separate equations where the non-existent nodes are replaced with nodes that exist inside the plate, e.g. if there is no west node, the equation will use the east node as a replacement. Each equation used are as follows:

Central Nodes:

$$T_P^{t+\Delta T} = Fo \left(T_N^t + T_E^t + T_S^t + T_W^t - \left(4 - \frac{1}{Fo} \right) * T_P^t \right)$$

Equation 2-45

First Column:

$$T_P^{t+\Delta T} = Fo \left(T_N^t + T_E^t + T_S^t + T_E^t - \left(4 - \frac{1}{Fo} \right) * T_P^t \right)$$

Equation 2-46

End Column:

$$T_P^{t+\Delta T} = Fo \left(T_N^t + T_W^t + T_S^t + T_W^t - \left(4 - \frac{1}{Fo} \right) * T_P^t \right)$$

Equation 2-47

Bottom Row:

$$T_P^{t+\Delta T} = Fo \left(T_N^t + T_E^t + T_N^t + T_W^t - \left(4 - \frac{1}{Fo} \right) * T_P^t \right)$$

Equation 2-48

Bottom Left Corner:

$$T_P^{t+\Delta T} = Fo \left(T_N^t + T_E^t + T_N^t + T_E^t - \left(4 - \frac{1}{Fo} \right) * T_P^t \right)$$

Equation 2-49

Bottom Right Corner:

$$T_P^{t+\Delta T} = Fo \left(T_N^t + T_W^t + T_N^t + T_W^t - \left(4 - \frac{1}{Fo} \right) * T_P^t \right)$$

Equation 2-50

3) Script Construction

3.1. Aluminium Properties, User Inputs & Stability Criterion

The initial action is defining the properties of Aluminium: Density, Thermal Conductivity and Specific Heat. These properties are provided for the experiment and used within the equation for Alpha below.

$$\text{Alpha} = \frac{\text{Thermal Conductivity}}{\text{Density} * \text{Specific Heat}}$$

Equation 3-1

The task then required user input for four parameters detailing the X and Y dimensions of the plate that would be used throughout the experiments, the spacing of the nodes within the plate (grid spacing), and the time step between the calculations of the plate. When asking these, dialog boxes in the form of 'inputdlg', produced a pop-up box asking the user for their input, including a default value.

```
XAxis = inputdlg('\fontsize{10}Define X-axis dimension:','X-Axis',[1 40],{'10'},BasicStruct);
```

Figure 2 – X-Axis Dimension Input Request

The X and Y axis were pure definitions as direct inputs, but the grid spacing, and the time step were required to be checked if they were compatible within the experiment.

```
DeltaT = inputdlg('\fontsize{10}Define required time step (seconds):','Time Step',[1 50],{'5'},BasicStruct);
```

Figure 3 – Time Step Input Request

The time step was first to be analysed. It needed to be less than an hour and greater than 0.01 seconds to ensure that both accurate results were provided throughout the experiment and to reduce excessive calculation times. (Appendix - 7.1.4 - Time Step Check)

```
Grid = inputdlg('\fontsize{10}Define node grid spacing:','Grid Spacing',[1 40],{'1'},BasicStruct);
```

Figure 4 – Time Step Input Request

The grid spacing was analysed next, this needed to be checked in three ways:

1. Whether it was greater than the X and Y axis divided by 2, as the numbers wouldn't equate if this were broken.
2. Whether it was smaller than X or Y axis divided by 100, as this could lead to longer than necessary calculation times.
3. And if, when divided by the grid spacing, the X or Y axis was not a whole number, this would be the same reason as in point '1' where the matrix wouldn't be made correctly and would therefore risk the stability of the script.

If any of these criteria aren't met, an error message would appear detailing the problem, then the while loop would repeat, asking for the input for grid spacing again, ensuring all three checks were passed, triggering the exit criteria for the loop. (Appendix - 7.1.5 - Grid Spacing & Fourier's Number Check)

Once all four inputs had been received and the code had checked that their respective criteria had been met, the next action was to calculate the Fourier's number.

$$\text{Fourier's Number} = \frac{\text{Alpha} * \text{Time Step}}{\text{Grid Spacing}^2}$$

Equation 3-2

This was then analysed to see if the user inputs for the time step and grid spacing meet the stability criterion. This means that the Fourier's number for a 2D object must not be lower than or equal to zero, or greater than or equal to $\frac{1}{4}$, (Katopodes, 2018) if either of those are true, the stability and accuracy for the equation cannot be trusted to produce correct results.

```
uiwait(msgbox({'\fontsize{10} Fouriers number should make this  
equation stable!', '', 'Press OK to continue.'}, BasicStruct));
```

Figure 5 – Time Step Input Request

At each of these criteria checks (time step, grid spacing, Fourier's number) the function 'uiwait' was used to pause the script, until the user had confirmed they were happy to progress by clicking the 'ok' button.

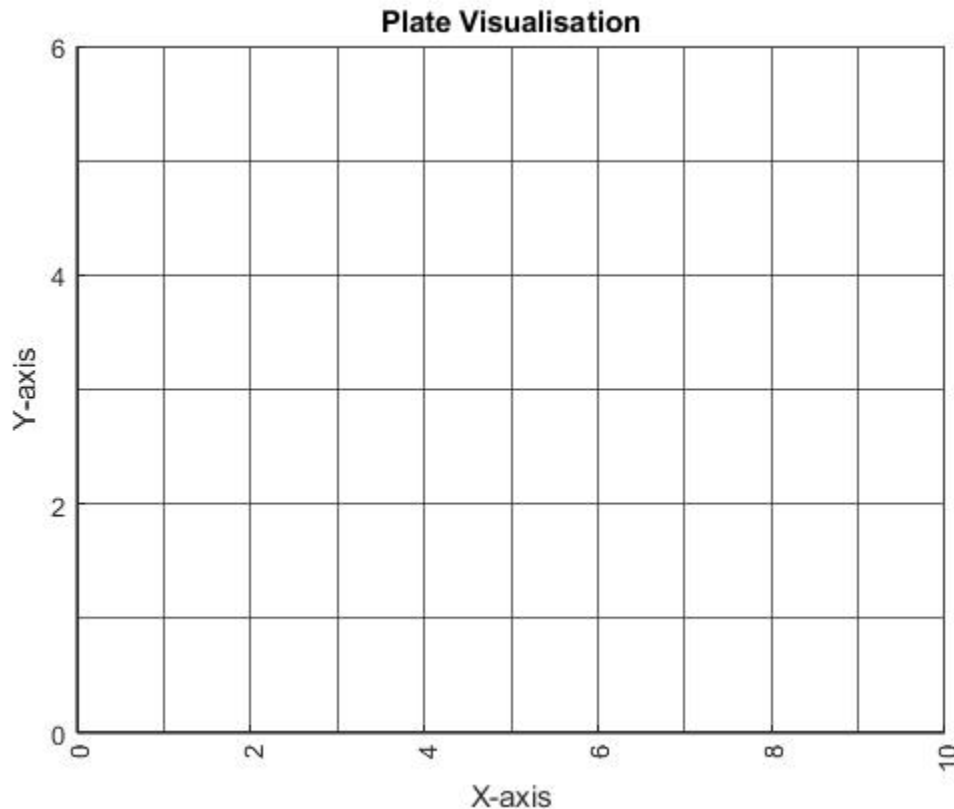


Figure 6 – Visual Representation of The Plat Matrix

After all the above steps had been processed, a graph was plotted as a visual representation of the plate matrix. The X and Y axis of the matrix were the user's inputs divided by the grid spacing, to equally space the nodes out, so that each node was represented by a cell within the matrix. The X and Y labels were spread across at twice the grid spacing value, otherwise it would become too crowded and the numbers hard to distinguish from one another, if the grid spacing was a small value. X and Y lines were then added into the plot for an accurate visualisation of where each node lay within the plate.

3.2. Linear Heat Distribution – Script Validation

To ascertain whether the script and the equations were working as required, a validation test was conducted using the same equation as for the Gaussian experiment.

Using a plate in the form of a matrix comprised of user inputted dimensions. Two extra rows and columns were added into the matrix which represent the hot, cold and linearly decreasing temperatures applied to every edge along the plate.

	1	2	3	4	5	6	7	8	9	10	11	12
1	50	45.4545	40.9091	36.3636	31.8182	27.2727	22.7273	18.1818	13.6364	9.0909	4.5455	0
2	50	0	0	0	0	0	0	0	0	0	0	0
3	50	0	0	0	0	0	0	0	0	0	0	0
4	50	0	0	0	0	0	0	0	0	0	0	0
5	50	0	0	0	0	0	0	0	0	0	0	0
6	50	0	0	0	0	0	0	0	0	0	0	0
7	50	0	0	0	0	0	0	0	0	0	0	0
8	50	45.4545	40.9091	36.3636	31.8182	27.2727	22.7273	18.1818	13.6364	9.0909	4.5455	0

Figure 7 – Empty Plate Matrix (Zeros) With Temperatures Applied Around the Edges

For this validation, a temperature of 50°C was applied to the left edge of the plate, with a temperature of 0°C applied to the right edge. This left the top and bottom edges, to which an equally spaced, linearly decreasing temperature from 50°C to 0°C in the same direction as the hot and cold temperatures applied to the left and right edges.

```
OldValidPlate = ValidPlate;
```

Figure 8 – Defining the Old Plate for New Iteration

A new plate matrix is calculated each iteration of the while loop, this is done for two reasons.

Firstly, the FDM equation uses nodes from the old plate iteration to calculate the corresponding nodes on the new plate iteration.

```
ExitCheck(Row-1,Column-1) = ((Tp/ValidPlate(Row,Column)) > 0.999999);
```

Figure 9 – Time Step Input Request

Secondly, the exit criteria for the while loop calculates when the same node from the old plate, divided by the new plate node is greater than 0.999999. At this point, in an empty matrix of zeros created beforehand, the corresponding node is replaced with a 1, once all cells within the matrix equal 1, the exit criteria is met, and therefore, exits the loop and continues onto the next experiment.

	1	2	3	4	5	6	7	8	9	10	11	12
1	50	45.4545	40.9091	36.3636	31.8182	27.2727	22.7273	18.1818	13.6364	9.0909	4.5455	0
2	50	45.4316	40.8651	36.3022	31.7445	27.1927	22.6475	18.1087	13.5758	9.0477	4.5229	0
3	50	45.4132	40.8298	36.2530	31.6854	27.1286	22.5836	18.0501	13.5272	9.0130	4.5049	0
4	50	45.4030	40.8102	36.2257	31.6525	27.0930	22.5481	18.0176	13.5003	8.9937	4.4949	0
5	50	45.4030	40.8102	36.2257	31.6525	27.0930	22.5481	18.0176	13.5003	8.9937	4.4949	0
6	50	45.4132	40.8298	36.2530	31.6854	27.1286	22.5836	18.0501	13.5272	9.0130	4.5049	0
7	50	45.4316	40.8651	36.3022	31.7445	27.1927	22.6475	18.1087	13.5758	9.0477	4.5229	0
8	50	45.4545	40.9091	36.3636	31.8182	27.2727	22.7273	18.1818	13.6364	9.0909	4.5455	0

Figure 10 – Final Iteration of The Plate Matrix Once Reaching Thermal Equilibrium

An equation was created to calculate the time taken for the plate to reach thermal equilibrium:

$$\text{Time Taken} = \frac{\text{Number of Iterations} * \text{Time Step}}{3600}$$

Equation 3-3

To aid with visualisation of the propagation of heat transferred throughout the plate, every 1,000th iteration of the while loop would update a heat map of the matrix by use of the 'contourf' function. This drastically reduced the calculation time rather than updating every iteration. This function replaced the numbers within the matrix with colours, depicting the scale of the numbers with an appropriate colour (depending on the colormap used, which for this experiment was 'Jet') so a value of zero, would return a deep blue, while a higher number showed a dark red.

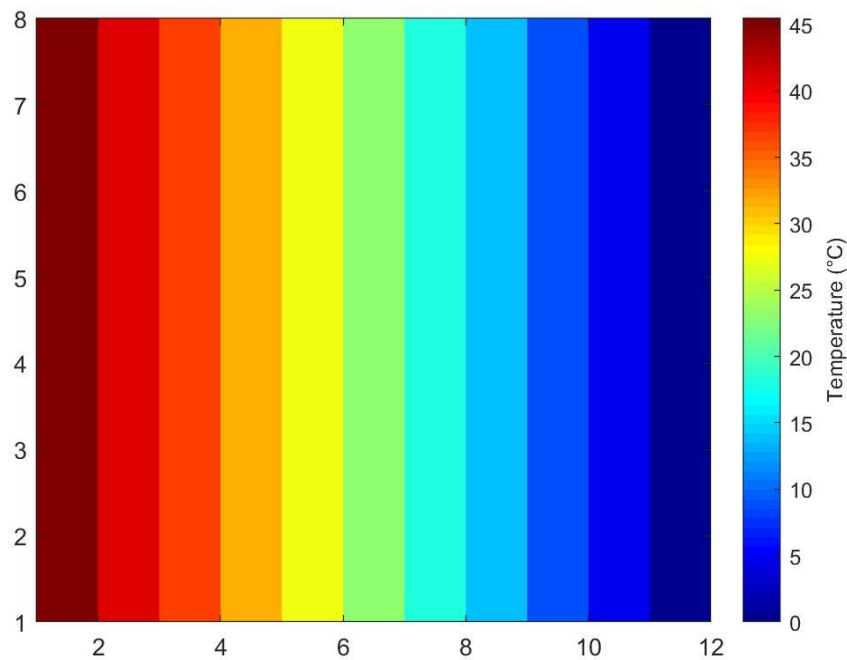


Figure 11 – Heat Map of the Final Plate Matrix Once Reaching Thermal Equilibrium

3.3. Gaussian Heat Distribution

For the major experiment being the focus of this assignment, a Gaussian distribution of temperatures would be applied to the top edge of the plate. To calculate this, an equation was needed to create a vector containing the temperatures applied to each column, ranging from 10°C on the edges, up to 60°C in the middle, called the Gaussian distribution equation:

$$Temperature = Amplitude * Exp\left(\frac{-(Column - (XPoints+1))^2}{(2 * Standard Deviation^2)}\right)$$

Equation 3-4

Where:

Amplitude = The highest temperature applied to the plate (60°C).

Exp = Exponential function.

Column = The location of the column within the matrix that the temperature will be applied.

XPoints = Is the X axis divided by the grid spacing, 1 is added to counteract odd columns.

Standard Deviation = The amount of dispersion in the equation stating the range of values.

In the equation, every variable was known except the standard deviation and the temperatures, but since the temperatures were required to be calculated, the equation would need to be rearranged to calculate the appropriate standard deviation value for the Gaussian equation to provide an adequately spaced distribution of temperatures.

$$Standard Deviation = \sqrt{\left(\frac{-(XPoints - (XPoints+1))^2}{2 * \ln\left(\frac{Boundary Temp}{Amplitude}\right)}\right)}$$

Equation 3-5

Where:

Ln = Natural logarithmic function.

Boundary Temp = The lowest temperature within the Gaussian temperature range (10°C).

	1	2	3	4	5	6	7	8	9	10
1	10.0000	20.2963	34.5129	49.1688	58.6873	58.6873	49.1688	34.5129	20.2963	10.0000

Figure 12 - Gaussian Temperature Distribution Using Standard Deviation Equation

To begin with, an empty matrix to represent the plate, was created using the user's previous inputs for X and Y axis, both divided by the grid spacing, increasing the Y axis by 1 row to account for the heat applied along the top edge of the plate. The Gaussian temperature distribution was inputted into the first row of the empty matrix, this would then change the values of every other cell, by using the FDM equation throughout every iteration.

	1	2	3	4	5	6	7	8	9	10
1	10.0000	20.2963	34.5129	49.1688	58.6873	58.6873	49.1688	34.5129	20.2963	10.0000
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0

Figure 13 - Empty Plate Matrix with Gaussian Heat Distribution Applied

An equation was created to predict the time taken for the plate to reach thermal equilibrium:

$$Time\ Taken = \frac{Number\ of\ Iterations * Time\ Step}{3600}$$

Equation 3-6

As before, the function 'contourf' plotted a heat map showing the propagation throughout the plate. This was updated every 10,000th iteration to save on excessive calculation times.

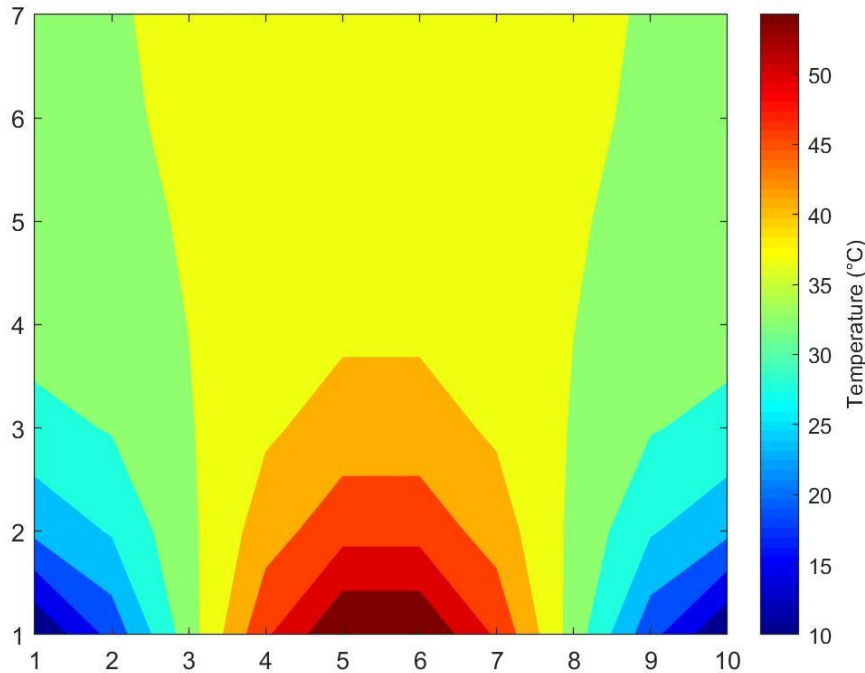


Figure 14 – Heat Map of the Final Plate Matrix Once Reaching Thermal Equilibrium Without 'flipud' Function

Unlike before with the linear temperature transfer where the side effects of 'contourf' weren't noticeable, the use of the function with the gaussian distribution, caused the plate to appear upside down (Figure 14). This made it necessary to use the function 'flipud', which flips the image upside down, causing the image to appear the correct way up (Figure 15).

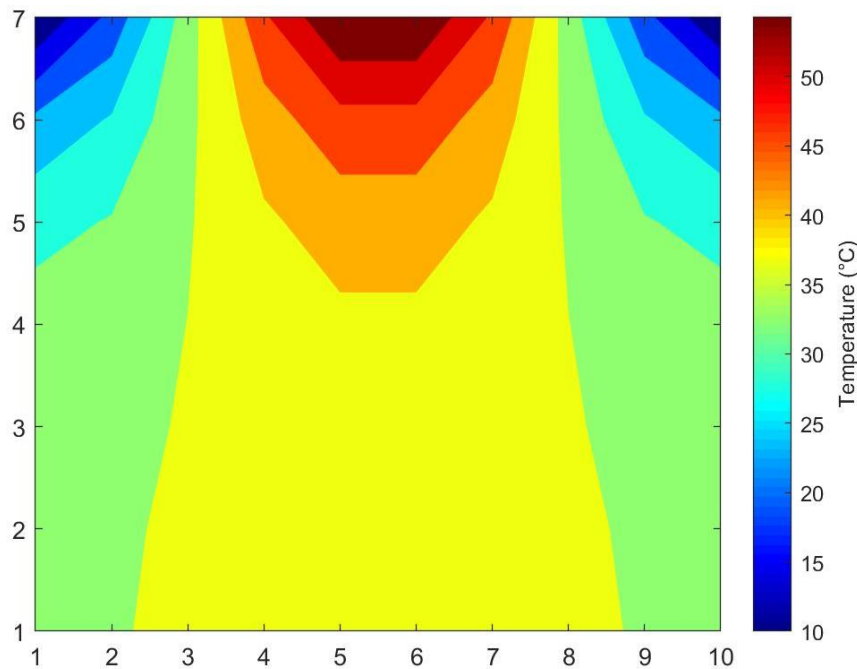


Figure 15 – Heat Map of the Final Plate Matrix Once Reaching Thermal Equilibrium

4) Code Evaluation

4.1. Code Analysis

To test the scripts durability, it will be trialled using the same inputs for the X & Y dimensions, then changing the time step and grid spacing, one at a time to see the effect on both calculation time and time taken until the plate reaches thermal equilibrium. It must be noted that all these calculation were computed on a Microsoft Surface Pro 4, so calculation times may differ depending on the hardware used.

The plate withing the previous sections used the following user inputs:

- X-axis = 10 metres
- Y-axis = 6 metres
- Time Step = 5 seconds
- Grid Spacing = 1 metre

The first to be analysed was the time step. By changing the time step between calculations, to as high as 3600 seconds or as low as 0.01 second, this would impact both the calculation time and time until thermal equilibrium is reached. E.g. if the time step is set to 3600 seconds (1 hour), there should theoretically be a reduction in calculation time, but also an inaccuracy within the calculations, as they will only be accurate to the nearest hour. For this experiment iterations between when the heat map was updated was increased to 10,000.

Changes in Time Step (s) - 10,000 Iterations for Heat Map			
Time Step (s)	Exit Criteria	Calc. Time (s)	Time Taken (Hrs)
0.01	0.99999999	771.27	57.94
0.01	0.99999999	235.33	23.03
0.01	0.9999999	69.1	5.48
5	0.9999999	6.15	83.05
3600	0.9999999	1.61	181.00
3600	0.999999	0.65	146.00
3600	0.99999	0.59	112.00
3600	0.999	0.57	77.00

Table 1 – Effects of Changing the Time Step & Exit Criteria

The effects of changing the time step were drastic, it is obvious to see that by decreasing the time step, the calculation time increased 10x but the level at which thermal equilibrium was reached also changed (Figure 16 - Left). Because of this, another study was conceived where the exit criteria was changed in order to reach the same level of thermal equilibrium as when using the 5 second time step (Figure 16 – Right). By doing this, the calculation time drastically increased by 11.61x than it originally took.

The opposite was noticed when increasing the time step. When using 3600 seconds, the calculation time only decreased by 1 second overall, but the time taken until thermal equilibrium was achieved, increased significantly by 104 hours although the same level of equilibrium was reached each time. This implies a significant amount of over calculation and shows how important the exit criteria is in relation with using different time steps.

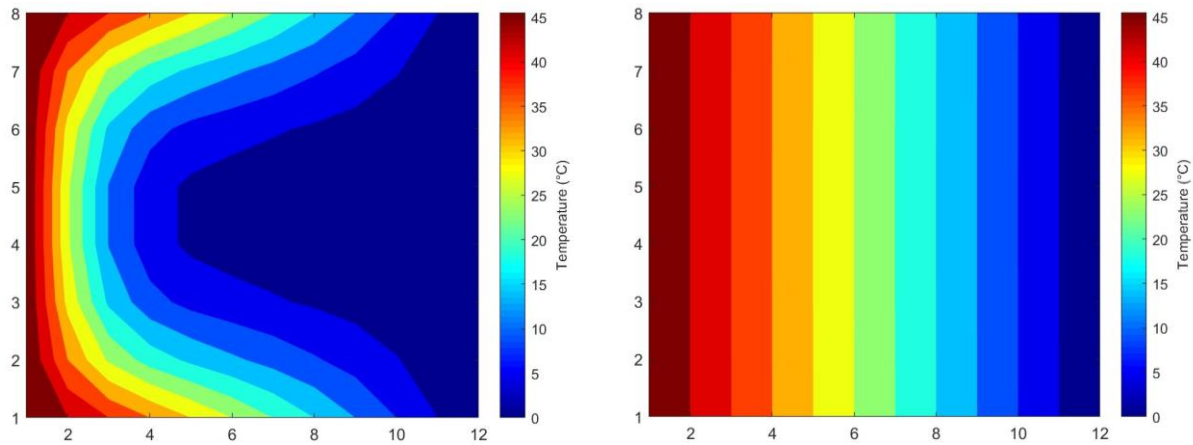


Figure 16 – Effect of Changing the Exit Criteria With 0.1 Second Time Step – Small (Left) – Large (Right)

The next variable to be analysed was the grid spacing. Defining how precise the nodes are within the plate, determines how accurate the results for each node are. E.g. if the grid spacing is a large number, the distance between the results are equally large, whereas, if the nodes are represented by a fine mesh, the node values will be smoother and more accurate values between them.

Changes in Grid Spacing (m) - 10,000 Iterations for Heat Map			
Grid Spacing (m)	Exit Criteria	Calc. Time (s)	Time Taken (Hrs)
0.1	0.999999	23.89	67.85
0.25	0.999999	15.17	70.33
0.5	0.999999	9.4	74.52
1	0.999999	6.15	83.05
2	0.999999	7.73	101.35

Table 2 – Effect of Changing the Grid Spacing Value

Unlike the previous time step alterations, changing the grid spacing didn't require modification to the exit criteria. By increasing the grid spacing, it was noticed that both the calculation time and time taken until equilibrium proportionally escalated. Whereas, by decreasing the grid spacing, the calculation time and time until thermal equilibrium changed indirectly proportionally, as the calculation time increased but the equilibrium time decreased.

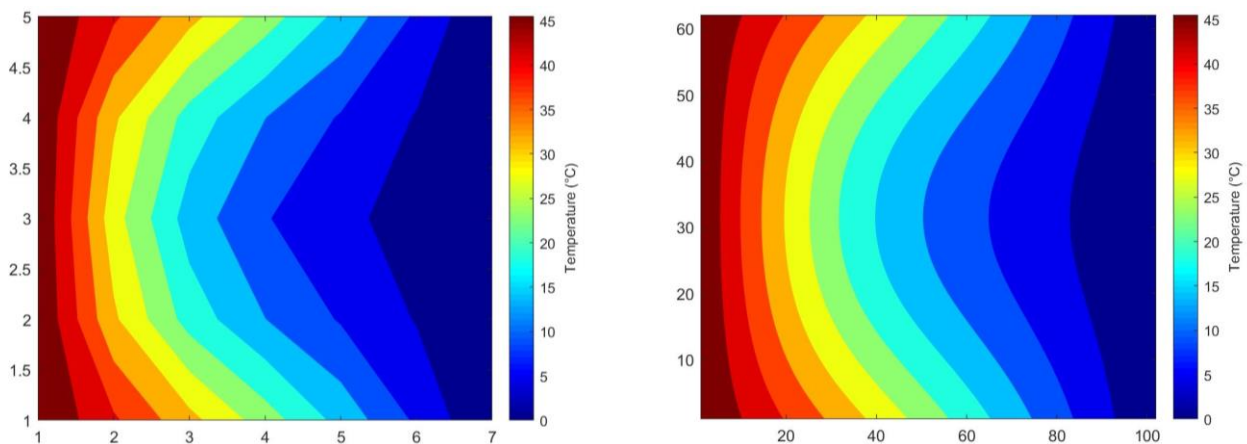


Figure 17 – Effect of Changing the Grid Spacing Value – 2m (Left) – 0.1m (Right)

Figure 17 shows a direct relation between the smoothness of the colour contour lines and the grid spacing. The smoother the lines, the less of a difference between the values held within each cell of the plate matrix. With a higher difference between the values, the potential inaccuracies increase in proportion due to a possibility of a nonlinear temperature distribution, whereas, by using a fine grid, these potential inaccuracies decrease, providing an authentic replication of thermal propagation.

4.2. Code Limitations

As discussed previously, the exit criteria of the while loop has been brought under scrutiny, depending on the time step that the user has requested. This severely affects the script's ability to determine when the plate actually reaches thermal equilibrium and the time taken for that to occur, which in turn, affects the accuracy of the end results.

It can also be noted that the time step defined by the user, entirely dictates the accuracy of each node and the final outcome. By using too great a time step, the values calculated between each iteration can vary drastically, leading to the conclusion that these results are only accurate to a certain degree, after which they cannot be trusted. The opposite can be told of having a time step at the minimum value, although this increases the accuracy of each iteration, the computational time will significantly increase, so a compromise is required.

In terms of grid spacing, with a bigger distance between each node, if the propagation of heat through the plate does not occur in a linear rate, then having a plate with only two nodes across the X-axis, for example, can only provide an estimation of the heat transfer through the plate. This, again, will lead to inaccurate results, although by having a fine mesh of nodes throughout the plate, the computational time for MATLAB to process each iteration will increase.

4.3. Potential Improvements

The biggest improvement to the script would come as a result of predicting the best exit criteria based on user inputs. If the user enters in a time step of 0.01 seconds, the user should be asked how accurate they wish the results to be. If they wished to achieve accurate results for the experiment, then an increase to the exit criteria should be made, but the user should be notified that this may lead to longer computational times and to plan accordingly, based upon the hardware they are running the MATLAB script on.

Another improvement could also be made so that the results from each experiment are saved with an external file for analysis afterwards within a '.mat' file. That way the user can specify the same X & Y dimensions but run several iterations, changing both the time step and grid spacing to determine the effects that these make on the end result. It is possible to do this, but due to time constraints this was not implemented.

The ability to also contain the script within a single while loop, enabling the user to carry out multiple experiments without having to continuously press 'run' within MATLAB each time. Although this isn't necessary, it will cancel out the effect of 'clc' and 'clear' commands at the top of the script, which may allow the user to carry over variables from a previous run.

5) Conclusion

To conclude this report, it has become apparent of both the limitations of the script and the improvements that could be made to make this experiment completely realistic.

To achieve the best results from this experiment, it must be possible for the user to understand what they want to get out of the script, in terms of accuracy. If the user knows the dimensions of the plate, they should know, therefore, exactly how accurate they want the results, although, they will generally want the smallest possible time step and grid spacing. But by doing this, the computational time will significantly increase.

The computational time also depends significantly on the hardware used for each experiment. A study could be performed on two different computers: one could be the most expensive and powerful desktop computer that money could buy, and the second could be a 15-year-old laptop. The same experiment would be performed on both hardware, using the same user inputs, and the computational time would differ exponentially.

6) References

BBC Bitesize, 2020. *What is diffusion?*. [Online]

Available at: <https://www.bbc.co.uk/bitesize/topics/z9r4jxs/articles/zngbci6>

Boston University, 2015. *Finite Difference Methods (FDMs) 1*. [Online]

Available at:

http://people.bu.edu/andasari/courses/Fall2015/LectureNotes/Lecture14_27Oct2015.pdf

[Accessed 2020].

Croft, D., 1977. *Heat Transfer Calculations Using Finite Difference Equations*. s.l.: Elsevier science & technology.

Dortmund University of Technology Faculty of Mathematics, n.d. *Finite difference method*. [Online]

Available at: <https://www.mathematik.uni-dortmund.de/~kuzmin/cfdintro/lecture4.pdf>

[Accessed 22 01 2020].

Hensen, J. L. M. & Nakhi, A. E., 1994. *FOURIER AND BIOT NUMBERS AND THE ACCURACY OF CONDUCTION MODELLING*. [Online]

Available at:

https://www3.nd.edu/~amoukasi/CBE358_Lab1/FOURIER%20AND%20BIOT%20NUMBER%20S.pdf

[Accessed 24 01 2020].

Jan L M Hensen, A. E. N., 1996. *FOURIER AND BIOT NUMBERS AND THE ACCURACY OF CONDUCTION MODELLING*. [Online]

Available at: http://www.esru.strath.ac.uk/Documents/94_bepac_fobi-effect.pdf

[Accessed 01 24 2020].

Katopodes, N. D., 2018. Stability of 2D-FTCS. In: *Free-Surface Flow: Computational Methods*. Ann Arbor: Butterworth-Heinemann, p. 355.

Lewis, R. W., Morgan, K., Thomas, H. R. & Seetharamu, K. N., 1996. *The Finite Element Method in Heat Transfer Analysis*. s.l.: John Wiley & Sons.

Moretti, R., Errera, M.-P., Couaillier, V. & Feyel, F., 2019. *Stability, Convergence and Optimization of Interface Treatments in Weak and Strong Thermal Fluid-Structure Interaction*. [Online]

Available at: <https://hal.archives-ouvertes.fr/hal-01789013/document>
[Accessed 24 01 2020].

TREFETHEN, 1994. *Accuracy, Stability & Convergence*. [Online]
Available at: <https://people.maths.ox.ac.uk/trefethen/4all.pdf>
[Accessed 24 01 2020].

University of Münster, 2016. *The Diffusion Equation*. [Online]
Available at: https://www.uni-muenster.de/imperia/md/content/physik_tp/lectures/ws2016-2017/num_methods_i/heat.pdf
[Accessed 24 01 2020].

V.Vuorinen, 2016. *Lecture 2: Finite Difference Methods in Heat Transfer*, Espoo: Aalto University School of Engineering.

7) Appendix

7.1. MATLAB Script

7.1.1. Properties of Aluminium

```
% Thermal Conductivity
K = 143;           % Watts/metre.Celcius

% Density
Rho = 2.8 *10^3;   % Kilogram/metre^3

% Specific Heat
C = 795;           % Joules/Kilogram.Celcius

% Thermal Diffusivity
Alpha = K/(Rho*C); % Metre^2/second
```

7.1.2. Text Box Structure Setup

```
BasicStruct.Interpreter = 'tex';
BasicStruct.WindowStyle = 'modal';
```

7.1.3. X & Y Dimension User Inputs

```
% Set Condition For While Loop
XCheck = 0;
YCheck = 0;

while XCheck == 0

    % User Defined X Axis Dimension
    XAxis = inputdlg('\fontsize{10}Define X-axis dimension (m): ','X-
Axis',[1 45],{'10'}, BasicStruct);

    if isempty(XAxis)

        % If X Dimension Is Empty
        errorlg('\fontsize{10}Error!! Please enter a value for the X
dimensions!','USER INPUT ERROR',BasicStruct);
        pause(2)

    else

        % Convert Input From Cell Array Into Double Precision Values
        XAxis = str2double(XAxis{1});
        XCheck = 1;

    end

end

while YCheck == 0
```

```

    % User defined X Axis Dimension
    YAxis = inputdlg('\fontsize{10}Define Y-axis dimension (m): ','Y-
Axis',[1 45],{'6'}, BasicStruct);

    if isempty(YAxis)

        % If Y Dimesion Is Empty
        errordlg('\fontsize{10}Error!! Please enter a value for the Y
dimensions!','USER INPUT ERROR',BasicStruct);
        pause(2)

    else

        % Convert Input From Cell Array Into Double Precision Values
        YAxis = str2double(YAxis{1});
        YCheck = 1;

    end

end
end

```

7.1.4. Time Step Check

```

% Set Condition For While Loop
TimeCheck = 0;

while TimeCheck == 0

    % User Defined Time Step In Seconds
    DeltaT = inputdlg('\fontsize{10}Define required time step (seconds):
','Time Step',[1 50],{'5'}, BasicStruct);

    % Convert Input From Cell Array Into Double Precision Values
    DeltaT = str2double(DeltaT{1});

    if DeltaT > 3600

        % If Time Step Is Greater Than 1 Hour, Innacurate Results Will Be
        Encountered
        errordlg('\fontsize{10}Error!! Using this large of a time step will
provide inaccurate results!','TIME STEP ERROR',BasicStruct);
        pause(2)

    elseif DeltaT < 0.01

        % If Time Step Is Smaller Than 0.01 Seconds, Innacurate Results
        Will Be Encountered
        errordlg('\fontsize{10}Error!! Using this small of a time step will
provide inaccurate results!','TIME STEP ERROR',BasicStruct);
        pause(2)

    else

        % Time Step Meets Criteria And Is Therefore Accepted
        TimeStepGood = msgbox('\fontsize{10}    Time step accepted! Press
OK to continue.','Time Step Accepted', BasicStruct);

        % Pause Script Until User Presses OK On Dialog Box
        uiwait(TimeStepGood);
    end
end

```

```

        % Change Status Of While Loop Condition To Exit Loop
        TimeCheck = 1;
    end

end

```

7.1.5. Grid Spacing & Fourier's Number Check

```

% Set Condition For While Loop
GridCheck = 0;
FouriersCheck = 0;

while GridCheck == 0 && FouriersCheck == 0

    while GridCheck == 0

        % User Defined Grid Spacing
        Grid = inputdlg('\fontsize{10}Define node grid spacing (m): ','Grid Spacing',[1 45],{'1'}, BasicStruct);

        % Convert Input From Cell Array Into Double Precision Values
        Grid = str2double(Grid{1});

        % Create X Axis Of Plate Matrix Considering Grid Spacing
        XPoints = (XAxis/Grid);

        % Create Y Axis Of Plate Matrix Considering Grid Spacing
        YPoints = (YAxis/Grid);

        % Round X Axis Of Plate Matrix To Nearest Whole Integer
        XCheck = XPoints == round(XPoints);

        % Round Y Axis Of Plate Matrix To Nearest Whole Integer
        YCheck = YPoints == round(YPoints);

        if Grid > (XAxis/2) || Grid > (YAxis/2)

            % If Grid Spacing Is Greater Than Half Of Either Of The Plate
            Dimensions,
            % Inequalities and Errors Will Be Encountered
            errordlg('\fontsize{10}Error!! Using this large of a grid will
            provide inaccurate results!','GRID ERROR',BasicStruct);
            pause(1)

            % Ensure That While Loop Criteria Is Not Met
            GridCheck = 0;

        elseif Grid < (XAxis/100) || Grid < (YAxis/100)

            % If Grid Spacing Is Less Than Either Of The Plate Dimensions
            Divided By 100,
            % Excessively Lengthy Calculation Will Be Encountered
            errordlg('\fontsize{10}Error!! Using this fine of a grid will
            provide inaccurate results and long calculation times!','GRID
            ERROR',BasicStruct);
            pause(1)

            % Ensure That While Loop Criteria Is Not Met

```

```

        GridCheck = 0;

        elseif XCheck == 0 || YCheck == 0

            % If Grid Spacing Is Not A Whole Number When Divided By Either
            Of The Plate Dimensions,
            % Inequalities and Errors Will Be Encountered
            errordlg('\fontsize{10}Error!! Using a grid that is not a whole
            number when divided by X or Y axis, will provide inaccurate results!', 'GRID
            ERROR', BasicStruct);
            pause(1)

            % Ensure That While Loop Criteria Is Not Met
            GridCheck = 0;

        else

            % Grid Spacing Meets Criteria And Is Therefore Accepted
            GridGood = msgbox('\fontsize{10} Grid spacing accepted! Press
            OK to continue.', BasicStruct);

            % Pause Script Until User Presses OK On Dialog Box To Continue
            uiwait(GridGood);

            % Change Status Of While Loop Condition To Exit Loop
            GridCheck = 1;

        end

    end

    % Stability Criterion

    while FouriersCheck == 0

        % Define Fouriers Number Variable
        Fo = (Alpha*DeltaT)/(Grid^2);

        if Fo >= 1/4

            % If The Fouriers Number Is Greater Than 1/4, The Equation Will
            Be Unstable
            errordlg('\fontsize{10}Error!! Using a Fouriers Number this big
            will make the equation unstable', 'STABILITY ERROR')
            pause(1)

        elseif Fo <= 0

            % If The Fouriers Number Is Smaller Than 0, The Equation Will
            Be Unstable
            errordlg('\fontsize{10}Error!! Using a Fouriers Number this
            small will make the equation unstable', 'STABILITY ERROR')
            pause(1)

        else

            % Stability Criterion Is Met And The Equation Should Be Stable
            % Pause Script Until User Presses OK On Dialog Box To Continue

```

```

        uiwait(msgbox({'\fontsize{10}Fouriers number should make this
equation stable!','','
continue.'},BasicStruct));

        % Change Status Of While Loop Condition To Exit Loop
        FouriersCheck = 1;

    end

end

end

```

7.1.6. Plate Visualisation

```

% A Figure Displaying The Plate Matrix For Visualisation Is Created
figure('Name','Plate Visualisation');

% Move Figure To Top Center Of Screen
movegui('north');

% Create A Rectangle To The Dimensions Of The Plate
GridPlate = rectangle('Position',[0 0 XAxis YAxis]);

if Grid < 1

    % Display Tick Markers At Twice The Value Of The Grid Spacing
    xticks(0:Grid*2:XAxis)
    yticks(0:Grid*2:YAxis)

else

    % Display Tick Markers At The Value Of The Grid Spacing
    xticks(0:Grid:XAxis)
    yticks(0:Grid:YAxis)

end

% Rotate X Axis Labels For Clear Labels
xtickangle(90)

% Display Axis For Grid Spacing Tick Markers
axis on

% Add Title And Axis Labels For Clarification
title('Plate Visualisation')
xlabel('X-axis (m)')
ylabel('Y-axis (m)')

% Create Vertical Lines To Display The Nodes Spacing
for xgrid = 0:Grid:XAxis
    xline(xgrid);
end

% Create Horizontal Lines To Display The Nodes Spacing
for ygrid = 0:Grid:YAxis
    yline(ygrid);
end

```

7.1.7. Validation Experiment

```
% Define The Temperatures Applied To Either Side Of The Plate
HeatApplied = 50;    % Degrees Celcius
ColdApplied = 0;     % Degrees Celcius

% Redefine X And Y Axis Of The Plate Accounting For The Heat Sources Around
The Outside
XPoints = (XAxis/Grid)+2;
YPoints = (YAxis/Grid)+2;

% Linear Decrease In Temperature From Left To Right Side Of The Plate Along
The Stop And Bottom Row
LinearTempTransfer = linspace(HeatApplied,ColdApplied,XPoints);

% Create Pre-Allocated Zeros Matrix For Plate Matrix To Be Used In The
Validation Test
ValidPlate = zeros(YPoints,XPoints);

% Replace The First Column With The Heat Applied To The Plate
ValidPlate(:,1)=HeatApplied;

% Replace The Last Column With The Cold Applied To The Plate
ValidPlate(:,end)=ColdApplied;

% Replace The First Row With The Linear Heat Applied To The Plate
ValidPlate(1,:)=LinearTempTransfer;

% Replace The Last Row With The Linear Heat Applied To The Plate
ValidPlate(YPoints,:)=LinearTempTransfer;

% Initial Count Of Iterations, To Be Counted Throughout The Loop
Iteration = 1;

% Initial Count Of Time Taken, To Be Counted Throughout The Loop
TimeTaken = 0;

% Create Zeros Matrix For Exit Criteria Check
ExitCheck = zeros(YPoints-2,XPoints-2);

% Define Variable For While Loop Criteria
ExitLoop = 0;

tic

while ExitLoop == 0

    % Create Old Plate For Exit Criteria
    OldValidPlate = ValidPlate;

    % Define Starting Column For While Loop
    Column = 2;

    while Column<XPoints

        % Define Starting Row For While Loop
        Row = 2;

        while Row<YPoints
```

```

        % Define Point On Old Plate
        Tp = OldValidPlate(Row,Column);

        % Central Nodes =      Fouriers      *      East Node      +
West Node      +      North Node      +      South Node      -
4-(1/Fouriers) * TPresent
        ValidPlate(Row,Column) = Fo * (OldValidPlate(Row,Column+1) +
OldValidPlate(Row,Column-1) + OldValidPlate(Row-1,Column) +
OldValidPlate(Row+1,Column) - (4-(1/Fo)) * Tp) ;

        % If Old Point Divided By New Point Is Greater Than 0.999999,
Replace With 1 in Exit Criteria Check
        ExitCheck(Row-1,Column-1) = ((Tp/ValidPlate(Row,Column)) >
0.999999);

        % Increase Row Increment By 1
        Row = Row + 1;

    end

    % Increase Column Increment By 1
    Column = Column + 1;

end

% Increase Time Taken By 1
TimeTaken = TimeTaken + 1;

% Increase Iteration Number By 1
Iteration = Iteration + 1;

% Once Iterations Equals 10000, The Heat Map Will Update With New
Values
if Iteration == 10000

    % Create Figure For Heat Map
    figure(10)

    % Move Figure To Bottom Left Corner Of Screen
    movegui('southwest');

    % Define Method Of Heat Mapping
    contourf(ValidPlate,10,'edgecolor','none');

    % Add Colour Bar For Colour Scale
    c = colorbar;
    c.Label.String = ['Temperature (' char(176) 'C)'];

    % Define Colours Used In Heat Map
    colormap(jet)

    % Enable Axis For Location Along Plate
    axis on

    % Reset Iterations to 1
    Iteration = 1;

end

```

```

    % Once All Row And Columns Of Exit Criteria Equal 1, Exit While Loop
    if ExitCheck(:, :) == 1

        % Change Status Of While Loop Condition
        ExitLoop = 1;

    end

end

% Calculate Time Taken For Validation Plate Experiment
TimeTakenValidation = (TimeTaken*DeltaT)/3600;

% Close Heat Map Figure So It Can Be Labelled
close(figure(10))

% Name Figure And Axis Labels For Differentiation Between Validation And
Gaussian Experiments
figure('Name','Linear Heat Transfer - Validation Test End Result')
xlabel('X-axis (m)')
ylabel('Y-axis (m)')

% Move Figure To Bottom Left Corner Of Screen
movegui('southwest');

% Apply Heat Map To Validation Test
contourf(ValidPlate,10,'edgecolor','none');

% Add Colour Bar For Colour Scale
c = colorbar;
c.Label.String = ['Temperature (' char(176) 'C)'];

% Define Colours Used In Heat Map
colormap(jet)

% Enable Axis Along Plate
axis on

toc

% Create Dialog Box With Definition Of Text And Value Of Time Taken
% & Pause Script Until User Presses OK On Dialog Box To Continue
uiwait(msgbox(sprintf(' Validation test has completed in %4.2f Hours.\n\n
Press OK to confirm.',TimeTakenValidation)));

```

7.1.8. Progression to Gaussian Question

```

% Pose Question To User If They Wanted To Continue To Gaussian Experiment
Continue = questdlg(' Would you like to proceed to the Gaussian
experiment?', 'Continue To Gaussian?', 'Yes', 'No', 'Yes');

switch Continue
case 'Yes'
    % If Yes, Script Will Continue
    Gaussian = 1;

case 'No'
    % If No, Script Will exit

```

```

        return
end

```

7.1.9. Standard Deviation

```

% If Yes Was Selected, Then Scirpt Will Progress
if Gaussian == 1

    % Define The Highest Temperature Applied
    Amplitude = 60;

    % Define The Lowest Temperature Applied
    BoundaryTemp = 10;

    % Redefine X Axis Of The Plate To Dimension Of Plate
    XPoints = (XAxis/Grid);

    % Rearrange The Gaussian Applied Temperature Equation To Determine
    Standard Deviation
    StandardDeviation = sqrt((- (XPoints - ((XPoints + 1) / 2))^2) / (log(
    BoundaryTemp / Amplitude)*2));

    % Create Pre-Allocated Zeros Matrix For Temperature During While Loop
    Temp = zeros(1,XPoints);

    % Define Starting Column For While Loop Condition
    Column = 1;

    while Column <= (XPoints)

        % Using Gaussian Equation To Determine Applied Temperature For Each
        Column
        Temp(1,Column) = Amplitude * exp((- (Column - (XPoints + 1) / 2)^2)
        / (2 * (StandardDeviation)^2));

        % Increase Column Increment By 1
        Column = Column + 1;

    end

end

```

7.1.10. Gaussian Experiment

```

% Redefine Y Axis Of The Plate Accounting For The Heat Source Applied Along
The Top
YPoints = (YAxis/Grid)+1;

% Create Pre-Allocated Zeros Matrix For Plate Matrix To Be Used In The
Gaussian Test
Plate = zeros(YPoints,XPoints);

% Replace The First Row With The Heat Applied To The Plate
Plate(1,:) = Temp;

% Initial Count Of Iterations, To Be Incremented Throughout The Loop

```

```

Iteration = 1;

% Initial Count Of Time Taken, To Be Incremented Throughout The Loop
TimeTaken = 0;

% Create Zeros Matrix For Exit Criteria Check
ExitCheck = zeros(YPoints,XPoints);

% Define Variable For While Loop Criteria
ExitLoop = 0;

while ExitLoop == 0

    % Create Old Plate For Exit Criteria
    OldPlate = Plate;

    % Define Starting Column For While Loop
    Column = 2;

    while Column < XPoints

        % Define Starting Row For While Loop
        Row = 2;

        while Row < YPoints

            % Define Point In Central Nodes From Old Plate
            Tp = OldPlate(Row,Column);

            % Central Nodes = Fouriers * East Node + West Node
            + North Node + South Node - 4-(1/Fouriers) * TPoint
            Plate(Row,Column) = Fo * (Plate(Row,Column+1) +
            Plate(Row,Column-1) + Plate(Row+1,Column) + Plate(Row-1,Column) - (4-
            (1/Fo)) * Tp) ;

            % If Old Point Divided By New Point Is Greater Than 0.999999,
            Replace With 1 in Exit Criteria Check
            ExitCheck(Row,Column) = ((Tp/Plate(Row,Column)) > 0.999999);

            % Define Point On First Column From Old Plate
            Tp = OldPlate(Row,1);

            % First Column = Fouriers * East Node + East Node + North
            Node + South Node - 4-(1/Fouriers) * TPoint
            Plate(Row,1) = Fo * (Plate(Row,1+1) + Plate(Row,1+1) +
            Plate(Row+1,1) + Plate(Row-1,1) - (4-(1/Fo)) * Tp);

            % If Old Point Divided By New Point Is Greater Than 0.999999,
            Replace With 1 in Exit Criteria Check
            ExitCheck(Row,1) = ((Tp/Plate(Row,1)) > 0.999999);

            % Define Point On End Column From Old Plate
            Tp = OldPlate(Row,XPoints);

            % End Column = Fouriers * West Node + West
            Node + North Node + South Node - 4-(1/Fouriers) *
            TPoint
            Plate(Row,XPoints) = Fo * (Plate(Row,XPoints-1) +
            Plate(Row,XPoints-1) + Plate(Row+1,XPoints) + Plate(Row-1,XPoints) - (4-
            (1/Fo)) * Tp);

```

```

        % If Old Point Divided By New Point Is Greater Than 0.999999,
        Replace With 1 in Exit Criteria Check
        ExitCheck(Row,XPoints) = ((Tp/Plate(Row,XPoints)) > 0.999999);

        % Increase Row Increment By 1

        Row = Row + 1;
    end

    % Define Point On End Column From Old Plate
    Tp = OldPlate(YPoints,Column);

    % Bottom Row =      Fouriers *      East Node      +      West
Node      +      North Node      +      North Node      -      4-
(1/Fouriers) * TPoint
    Plate(YPoints,Column) = Fo * (Plate(YPoints,Column+1) +
Plate(YPoints,Column-1) + Plate(YPoints-1,Column)+ Plate(YPoints-1,Column)
- (4-(1/Fo)) * Tp);

    % If Old Point Divided By New Point Is Greater Than 0.999999,
    Replace With 1 in Exit Criteria Check
    ExitCheck(YPoints,Column) = ((Tp/Plate(YPoints,Column)) >
0.999999);

    % Increase Column Increment By 1
    Column = Column + 1;

end

% Define Point On End Column From Old Plate
Tp = OldPlate(YPoints,1);

% Bottom Left Corner = Fouriers * East Node +      East Node      +
North Node      +      North Node      -      4-(1/Fouriers) * TPoint
    Plate(YPoints,1) = Fo * (Plate(YPoints,1+1) + Plate(YPoints,1+1) +
Plate(YPoints-1,1) + Plate(YPoints-1,1) - (4-(1/Fo)) * Tp);

% If Old Point Divided By New Point Is Greater Than 0.999999, Replace
With 1 in Exit Criteria Check
ExitCheck(YPoints,1) = ((Tp/Plate(YPoints,1)) > 0.999999);

% Define Point On End Column From Old Plate
Tp = OldPlate(YPoints,XPoints);

% Bottom Right Corner = Fouriers *      West Node      +      West
Node      +      North Node      +      North Node      -      4-
(1/Fouriers) * TPoint
    Plate(YPoints,XPoints) = Fo * (Plate(YPoints,XPoints-1) +
Plate(YPoints,XPoints-1) + Plate(YPoints-1,XPoints) + Plate(YPoints-
1,XPoints)- (4-(1/Fo)) * Tp);

% If Old Point Divided By New Point Is Greater Than 0.999999, Replace
With 1 in Exit Criteria Check
ExitCheck(YPoints,XPoints) = ((Tp/Plate(YPoints,XPoints)) > 0.999999);

% Increase Time Taken By 1
TimeTaken = TimeTaken + 1;

```

```

    % Increase Iteration Number By 1
    Iteration = Iteration + 1;

    % Once Iterations equals 10000, The Heat Map Will Update With New
    Values
    if Iteration == 10000

        % Create New Variable To Plot As Contourf Flips Variable Upside
        Down, This Variable Is Also Flipped To Counteract This
        PlotPlate = flipud(Plate);

        % Create Figure For Heat Map
        figure(20)

        % Move Figure To Bottom Left Corner Of Screen
        movegui('southeast');

        % Define Method Of Heat Mapping
        contourf(PlotPlate,10,'edgecolor','none');

        % Add Colour Bar For Colour Scale
        c = colorbar;
        c.Label.String = ['Temperature (' char(176) 'C)'];

        % Define Colours Used In Heat Map
        colormap(jet)

        % Enable Axis For Location Along Plate
        axis on

        % Reset Iterations to 1
        Iteration = 1;

    end

    % Once All Row And Columns Of Exit Criteria Equal 1, Exit While Loop
    if ExitCheck(2:end,:) == 1

        % Change Status Of While Loop Condition
        ExitLoop = 1;

    end

end

% Calculate Time Taken For Validation Plate Experiment
TimeTakenGaussian = (TimeTaken*DeltaT)/3600;

% Close Heat Map Figure So It Can Be Labelled
close(figure(20))

% Name Figure For Differentiation Between Validation And Gaussian
Experiments
figure('Name','Gaussian Temperature Transfer Experiment End Result')
xlabel('X-axis (m)')
ylabel('Y-axis (m)')

% Move Figure To Bottom Left Corner Of Screen
movegui('southeast');

```

```

% Apply Heat Map To Gaussian Test
contourf(PlotPlate,10,'edgecolor','none');

% Add Colour Bar For Colour Scale
c = colorbar;
c.Label.String = ['Temperature (' char(176) 'C)'];

% Define Colours Used In Heat Map
colormap(jet)

% Enable Axis Along Plate
axis on

% Define Text For Dialog Box
formatspec = (' Gaussian Temperature Test Has Completed in %4.2f
Hours.\n\n Press OK to finish Experiment. ');

% Create Dialog Box With Text And Value Of Time Taken For Dialog Box
msgbox(sprintf(formatspec,TimeTakenGaussian));

```