

University of Wales Trinity Saint David
School of Engineering

Advanced Computational Methods
M6X00408

Samuel Treleaven
1702453

24/01/20

Transient Thermal Diffusion Model
using the Finite Difference Method

Contents

1	Introduction	3
2	The Constitutive Equation.....	3
3	Numerical Methods	4
3.1	Taylor Series	4
3.1.1	Forwards Difference Approximation	4
3.1.2	Backwards Difference Approximation.....	5
3.1.3	Central Difference Approximation.....	6
3.2	Finite Difference Method.....	8
4	Boundary Conditions	11
5	The Model	16
5.1	User Input and Validation.....	16
5.2	Constants and Fourier Number Validation	17
5.3	Setting up Initial Matrix.....	18
5.4	Stop Criteria	19
5.5	Calculating Plate Temperature until Thermal Equilibrium	20
5.6	Graphical Representation of Plate.....	21
5.7	Results.....	22
5.8	Linear Script Validation.....	24
6	Results.....	25
7	Limitations and Improvements.....	28
8	Conclusion.....	28
	Bibliography.....	29
	Appendices.....	30

Appendices

Appendix A – Gaussian Script

Appendix B – Linear Script

1 Introduction

For this assignment MATLAB was used to create a model to solve a time-dependent thermal diffusion equation. The model uses the finite difference method, FDM, to calculate the temperature distribution on a created rectangular plate of aluminium. For the purpose of the calculations, the plate will be calculated as if it is two-dimensional. The model will allow the user to decide the height, width and grid spacings as well as the time intervals of which the temperature will be recorded.

2 The Constitutive Equation

The equation that is used to calculate the new temperature at each node after the set time step is as follows in *equation 2.1*. This is referred to a lot throughout the document.

$$T_{i,j} = Fo \left(T_{i,j-1} + T_{i,j+1} + T_{i+1,j} + T_{i-1,j} - \left(4 - \frac{1}{Fo} \right) T_{ij} \right)$$

Equ. 2.1 – The Constitutive Equation

The coordinates (i, j) define the location of the point that is being calculated and Fo is Fouriers number which is a dimensionless expression. This comes from the following *equation 2.2*.

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

Equ. 2.2 Fourier's Number (Price, 2004)

The Fouriers number is calculated using α , a fixed constant for the material, which is the thermal diffusivity, the grid spacing, h , and the time step, Δt . However, there is a maximum permissible time step once α and the grid spacing have been set, this is called the stability criterion. For a two dimensional shape, this is calculated using Equation 2-3.

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

*Equ. 2.3
(Ferziger & Perić, 2002)*

3 Numerical Methods

Two numerical methods aided in the derivation of the constitutive equation, *Equation 2.1*, these were the Taylor Series and the Explicit Finite Difference Method.

3.1 Taylor Series

To be able to calculate the temperature at each node on the plate, an approximation for the first and second derivative of a function have to be calculated. These approximations come from the Taylor series which is defined in *equation 3.1*.

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

Equ. 3.1
(Harman, et al., 1997)

This series written out in its expanded form can be seen in *equation 3.2*. This is the equation that is used to calculate the forward, backward and central difference approximations.

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

Equ. 3.2

3.1.1 Forwards Difference Approximation

Equation 3.3 shows how the forward's difference approximation was calculated by substituting $x = x_0 + h$ into the Taylor series, *equation 3.2*.

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

Equ. 3.3

This equation can be rearranged for the first order derivative:

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

Equ. 3.4

To make this equation an approximation, an assumption has to be made. That since h is very small, and tends towards zero, the values from $f''(x_0)$ until $f^{(n)}$ are assumed to be negligible. The error in this approximation is of order h . This means as h gets smaller, the error in the approximation also gets smaller in a linear fashion.

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

Equ. 3.5

3.1.2 Backwards Difference Approximation

To calculate the backwards difference approximation, the same steps are followed that were carried out in the forward's difference method but using the substitution of $x = x_0 - h$. This can be seen in *equation 3.6* and the rearranged equation can be seen in *equation 3.7*

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

Equ. 3.6

$$f'(x_0) = \frac{-f(x_0 - h) + f(x_0)}{h} + \left(\frac{f''(x_0)h}{2!} - \dots \pm \frac{f^{(n)}h^{(n-1)}}{n!} \right)$$

Equ. 3.7

Similarly to the forward's difference approximation, the assumption is made that after $f''(x_0)$ the values are deemed negligible. The error in this approximation is also of order h , therefore, as h gets smaller so does the error in the approximation, in a linear fashion.

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

Equ. 3.8

3.1.3 Central Difference Approximation

To try and get an improved approximation the two *equations* 3.4 and 3.8 are subtracted from each other to give *equation* 3.9.

$$\begin{aligned} f(x_0 + h) - f(x_0 - h) &= \left(f(x_0) + f'(x_0) \cdot \frac{(h)^1}{1!} + f''(x_0) \cdot \frac{(h)^2}{2!} + \dots + f^{(n)}(x_0) \cdot \frac{(h)^n}{n!} \right) \\ &\quad - \left(f(x_0) - f'(x_0) \cdot \frac{(h)^1}{1!} + f''(x_0) \cdot \frac{(h)^2}{2!} - \dots \pm f^{(n)}(x_0) \cdot \frac{(h)^n}{n!} \right) \end{aligned}$$

Equ. 3.9

This equation, like previously, can be rearranged for the first-order derivative producing *equation* 3.10.

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

Equ. 3.10

The values in the brackets are assumed to be negligible, as h gets very small, which leaves the first-order central difference approximation shown in *equation* 3.11. The error of this approximation is of order h^2 , therefore for small h values, the error will be much less than in the previous approximations which means this approximation is more accurate, so will be better to use this approximation over the forward and backwards difference approximation.

A drawback of this is that both values either side of the point will need to be known whereas the forward and backwards only need to know one point next to the value.

$$f'(x) = \frac{f(x + h) - f(x - h)}{2h}$$

Equ. 3.11

To approximate the second-order derivative, the central difference approximation is used but this time *equation* 3.4 and 3.8 are added together, as shown in *equation* 3.12.

$$\begin{aligned}
 & f(x_0 + h) + f(x_0 - h) \\
 &= \left(f(x_0) + f'(x_0) \cdot \frac{(h)^1}{1!} + f''(x_0) \cdot \frac{(h)^2}{2!} + \dots + f^{(n)}(x_0) \cdot \frac{(h)^n}{n!} \right) \\
 &+ \left(f(x_0) - f'(x_0) \cdot \frac{(h)^1}{1!} + f''(x_0) \cdot \frac{(h)^2}{2!} - \dots \pm f^{(n)}(x_0) \cdot \frac{(h)^n}{n!} \right)
 \end{aligned}$$

Equ. 3.12

Equation 3.12 can be rearranged for the second derivative which can be seen in equation 3.13.

$$f''(x_0) = \frac{f(x_0 + h) - 2f(x_0) + f(x_0 - h)}{h^2} - \left(\frac{2f^{(4)}(x_0)h^2}{4!} + 2f^{(6)}(x_0) \frac{h^4}{6!} + \dots \right)$$

Equ. 3.13

Like previously, the part in the bracket is assumed to be negligible so the approximation can be seen in equation 3.14. The error in this approximation is also order h^2 so another more accurate approximation, however, it is the only approximation for the second-order derivative

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

Equ. 3.14

A visual representation of the different types of approximation is shown in figure 3.1.

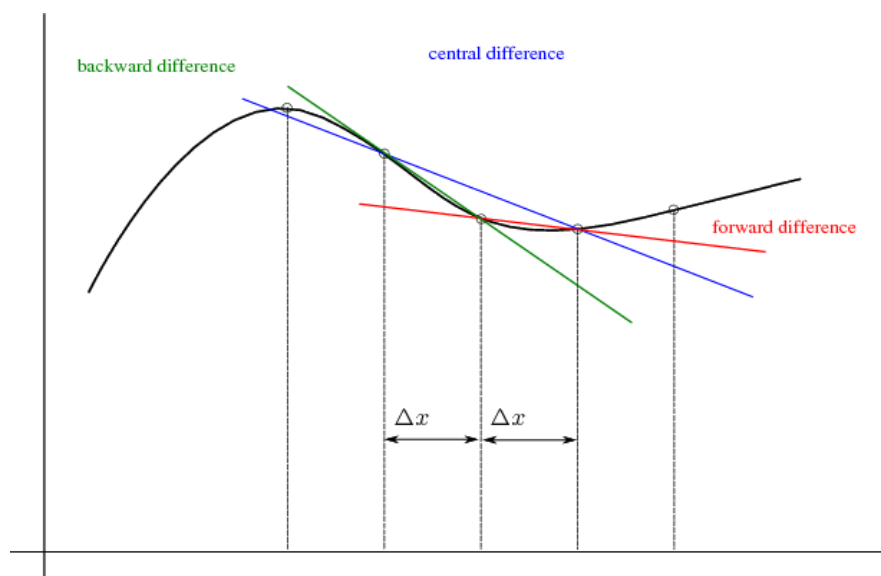


Figure 3.1 – Numerical Equation Approximations
(Henizl, 2007)

3.2 Finite Difference Method

To calculate the change in temperature by conduction over a certain time, a thermal diffusion equation is used which is as seen in *equation 3.15*

$$\nabla^2 T = \frac{1}{\alpha} \cdot \frac{dT}{dt}$$

Equ. 3.15 Thermal Diffusion
(Harman, et al., 1997, p. 607)

Where T is temperature, t is time, ∇ is the Laplacian operator and α is the thermal diffusivity.

For plates which are two dimensional, the Laplacian operator is defined by *Equation 3.16*.

$$\nabla^2 = \frac{d^2 T}{dx^2} + \frac{d^2 T}{dy^2}$$

Equ. 3.16 The Laplacian Operator
(Harman, et al., 1997, p. 607)

Substituting the Laplacian operation, as seen above in *equation 3.16*, into the thermal diffusion equation seen in 3.15, results in the following, 3.17.

$$\frac{dT}{dt} = \alpha \left(\frac{d^2 T}{dx^2} + \frac{d^2 T}{dy^2} \right)$$

Equ. 3.17

For each material alpha, the thermal diffusivity is defined by

$$\alpha = \frac{k}{\rho \cdot c}$$

Equ. 3.18
(Liu & F, 2014)

Where k is the thermal conductivity, ρ is the density and c is the specific heat.

To find the first and second-order derivatives in *equation 2.10*, a specific region of interest needs to be used, this can be seen in *figure 2.1* below.

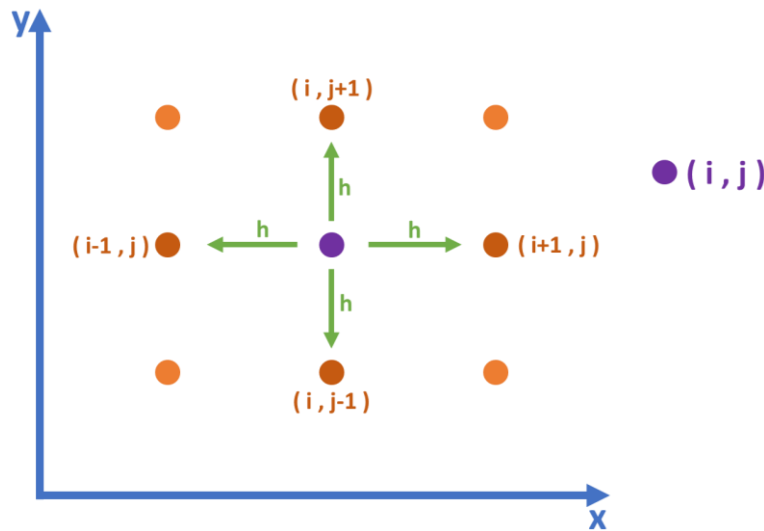


Figure 2.1 Region of Interest on a 2D Plate

Equations 3.19 and *3.20* were created from using the central difference equation calculated before, see *Equation 3.14*. *Equation 3.19* demonstrates the equation needed for the x-axis, and *Equation 3.20* demonstrates the equations needed for the y-axis. Where i and j represent the coordinates displayed in *Figure 2.1*.

$$\frac{d^2 T_{ij}}{dx^2} = \frac{T_{i-1,j}^t - 2T_{ij}^t + T_{i+1,j}^t}{\Delta x^2}$$

Equ. 3.19

$$\frac{d^2 T_{ij}}{dy^2} = \frac{T_{i,j-1}^t - 2T_{ij}^t + T_{i,j+1}^t}{\Delta y^2}$$

Equ. 3.20

Using the forward difference method equation, see *equation 3.5*, the derivative for temperature with respect to time could be created, *equation 3.21*. The reason the forward difference method was used over the other two, first-order derivative approximation, was due to the fact that the current temperature was known and the future temperature was required to be found. Whereas the other two use the previous temperature which will not work to calculate the next temperature as it is also an unknown. The only negative to the forward's difference approximation is that it has an error of order h whereas if the central difference method was used it would be more accurate with an error of order h^2 .

$$\frac{dT_{ij}}{dt} = \frac{T_{ij}^{t+\Delta t} - T_{ij}^t}{\Delta t}$$

Equ. 3.21

The grid spacing is equal in the x-axis and y-axis so $\Delta x = \Delta y = h$.

The simplified *equation 3.22* below is formed from substituting *equations 3.19*, *3.20* and *3.21* into the earlier *equation 3.17*.

$$\frac{T_{ij}^{t+\Delta t} - T_{ij}^t}{\Delta t} = \alpha \left(\frac{T_{ij-1}^t + T_{i-1,j}^t - 4T_{ij}^t + T_{i+1,j}^t}{h^2} \right)$$

Equ. 3.22

The only unknown in *equation 3.22* is $T_{ij}^{t+\Delta t}$. Therefore, if this is made the subject of the equation the outcome will be *equation 3.23*.

$$T_{ij}^{t+\Delta t} = \alpha \Delta t \left(\frac{T_{ij-1}^t + T_{i-1,j}^t - 4T_{ij}^t + T_{i+1,j}^t}{h^2} \right) + T_{ij}^t$$

Equ. 3.23

Simplifying this equation creates the following, *equation 3.24*. When substituting Fouriers Number, as seen in *equation 2.2*, the result is our final equation, *equation 2.1*.

$$T_{ij}^{t+\Delta t} = \frac{\alpha \Delta t}{h^2} \left(T_{i,j-1} + T_{i,j+1} + T_{i+1,j} + T_{i-1,j} - \left(\frac{4 - h^2}{\alpha \Delta t} \right) T_{ij} \right) \quad Fo = \frac{\alpha \Delta t}{h^2}$$

Equ. 3.24

Equ. 2.2 – Fourier's Number

$$T_{i,j} = Fo \left(T_{i,j-1} + T_{i,j+1} + T_{i+1,j} + T_{i-1,j} - \left(4 - \frac{1}{Fo} \right) T_{ij} \right)$$

Equ. 2.1 – The Constitutive Equation

4 Boundary Conditions

For the linear method the boundary conditions where known, this is called a Dirichlet boundary (Cheng & Cheng, 2005). So at all points, the temperature is known so no calculation is needed to be carried out.

The left edge is 50 degrees unless another value is chosen, and the right edge is zero degrees. The temperature on the top and bottom edges go from 50 to 0 degrees in a linear fashion. Like shown in a couple of examples below. The initial internal plate temperatures were 0 degrees.

	1	2	3	4	5	6	7	8	9	10	11	
1	50	45	40	35	30	25	20	15	10	5	0	
2	50	0	0	0	0	0	0	0	0	0	0	
3	50	0	0	0	0	0	0	0	0	0	0	
4	50	0	0	0	0	0	0	0	0	0	0	
5	50	0	0	0	0	0	0	0	0	0	0	
6	50	45	40	35	30	25	20	15	10	5	0	
7												

Figure 4.1

	1	2	3	4	5	6	7	8	9	10	11	
1	50	45	40	35	30	25	20	15	10	5	0	
2	50	0	0	0	0	0	0	0	0	0	0	
3	50	0	0	0	0	0	0	0	0	0	0	
4	50	0	0	0	0	0	0	0	0	0	0	
5	50	0	0	0	0	0	0	0	0	0	0	
6	50	0	0	0	0	0	0	0	0	0	0	
7	50	0	0	0	0	0	0	0	0	0	0	
8	50	0	0	0	0	0	0	0	0	0	0	
9	50	0	0	0	0	0	0	0	0	0	0	
10	50	0	0	0	0	0	0	0	0	0	0	
11	50	0	0	0	0	0	0	0	0	0	0	
12	50	0	0	0	0	0	0	0	0	0	0	
13	50	0	0	0	0	0	0	0	0	0	0	
14	50	0	0	0	0	0	0	0	0	0	0	
15	50	0	0	0	0	0	0	0	0	0	0	
16	50	0	0	0	0	0	0	0	0	0	0	
17	50	0	0	0	0	0	0	0	0	0	0	
18	50	0	0	0	0	0	0	0	0	0	0	
19	50	0	0	0	0	0	0	0	0	0	0	
20	50	0	0	0	0	0	0	0	0	0	0	
21	50	45	40	35	30	25	20	15	10	5	0	
22												

Figure 4.2

	1	2	3	4	5	
1	50	37.5000	25	12.5000	0	
2	50	0	0	0	0	
3	50	37.5000	25	12.5000	0	

Figure 4.3

For the Gaussian method, the boundary temperature along the top edge was known, Dirichlet Boundary, however the left, right and bottom edges were not known but the derivatives where so this is called a Neumann boundary condition (Cheng & Cheng, 2005). Since the derivative was known, the new equation for each side could be created using the first-order central difference method, as seen previously in *equation 3.5*.

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

Equ. 3.5

Since there is no change in temperature at the boundaries, the following equation is formed.

$$f'(x) = \frac{dT}{dx} = \frac{dT}{dy} = 0$$

Equ. 4.1

Therefore, using the central difference method, at point $T_{i,j}$ and the derivative in *equation 4.1*, *equation 4.2* is created.

$$0 = \frac{T_{i+1,j} - T_{i-1,j}}{2h}$$

Equ. 4.2

Through rearranging, the following can be proven

$$0 = T_{i+1,j} - T_{i-1,j}$$

$$T_{i+1,j} = T_{i-1,j}$$

Using our Constitutive Equation *equation 2.1*, repeated below, the values $T_{i+1,j}$ can be substituted for $T_{i-1,j}$ for the left boundary and the opposite by substituting $T_{i-1,j}$ for $T_{i+1,j}$ for the right boundary.

$$T_{i,j} = Fo \left(T_{i,j-1} + T_{i,j+1} + T_{i+1,j} + T_{i-1,j} - \left(4 - \frac{1}{Fo} \right) T_{i,j} \right)$$

Equ. 2.1 – The Constitutive Equation

The same can be repeated for the bottom boundary and the following can be proven.

$$0 = \frac{T_{i,j+1} - T_{i,j-1}}{2h}$$

Equ. 4.3

$$0 = T_{i,j+1} - T_{i,j-1}$$

$$\therefore T_{i,j+1} = T_{i,j-1}$$

So for the bottom edge, substituting $T_{i,j+1}$ for $T_{i,j-1}$ into our final equation, *equation 2.1*, will give the bottom boundary equation.

However, for the corner nodes, these methods need to be combined. For the bottom-left the equation is

$$T_{i,j} = Fo \left(T_{i,j+1} + T_{i,j+1} + T_{i+1,j} + T_{i+1,j} - \left(4 - \frac{1}{Fo} \right) T_{ij} \right)$$

Equ. 4.4

For the bottom-right node, the equation is

$$T_{i,j} = Fo \left(T_{i,j+1} + T_{i,j+1} + T_{i-1,j} + T_{i-1,j} - \left(4 - \frac{1}{Fo} \right) T_{ij} \right)$$

Equ. 4.5

The Dirichlet Boundary, top edge, is calculated using the normal distribution formula:

$$T(x) = Ae^{\frac{-(x-x_0)^2}{2\sigma^2}}$$

Equ. 4.6
(Abramowitz & Stegun, 1965)

Where $T(x)$ is the function of temperature along the boundary in the x-axis, A is the peak Temperature, x is the variable of the node position along the top boundary, x_0 is the middle node on the boundary and σ is the standard deviation. The distribution can be seen in *figure 4.4*.

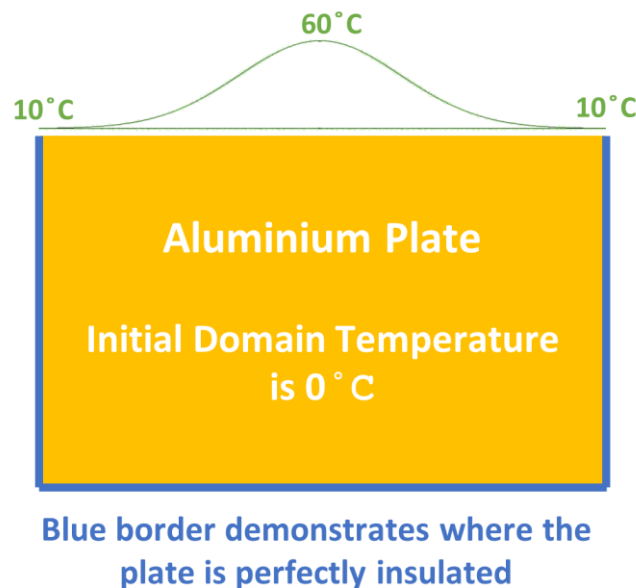


Figure 4.4 Initial Plate

The values in the formula are calculated and shown in the following equations.

$$x_0 = \frac{\text{No. of } x \text{ nodes} + 1}{2}$$

Equ. 4.7

$$A = 60$$

To calculate the standard deviation, the equation is rearranged.

$$\frac{T(x)}{A} = e^{\frac{-(x-x_0)^2}{2\sigma^2}}$$

Equ. 4.8

$$\sigma^2 = \frac{-(x-x_0)^2}{2 \log\left(\frac{T(x)}{A}\right)}$$

Equ. 4.10

$$\log\left(\frac{T(x)}{A}\right) = \frac{-(x-x_0)^2}{2\sigma^2}$$

Equ. 4.9

$$\sigma = \sqrt{\frac{-(x-x_0)^2}{2 \log\left(\frac{T(x)}{A}\right)}}$$

Equ. 4.11

T(x) is taken to be the lowest temperature of 10 °C. So the following equation can be used to calculate the standard deviation for the top boundary on any plate where x is the number of nodes and x_0 is the middle node.

$$\sigma = \sqrt{\frac{-(x-x_0)^2}{2 \log\left(\frac{10}{60}\right)}}$$

Equ. 4.12

The result of this is then substituted back into the original distribution equation to produce the temperature values along the top edge.

The initial internal temperatures were all 0 °C, as can be seen in the examples below.

	1	2	3	4	5	6	7	8	9
1	10.0000	21.8999	38.3366	53.6435	60	53.6435	38.3366	21.8999	10.0000
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0

Figure 4.5

	1	2	3	4	5	6	7	8	9	10	11
1	10.0000	19.0605	31.4787	45.0450	55.8503	60	55.8503	45.0450	31.4787	19.0605	10.0000
2	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0

Figure 4.6

	1	2	3	4	5
1	10.0000	38.3366	60	38.3366	10.0000
2	0	0	0	0	0
3	0	0	0	0	0

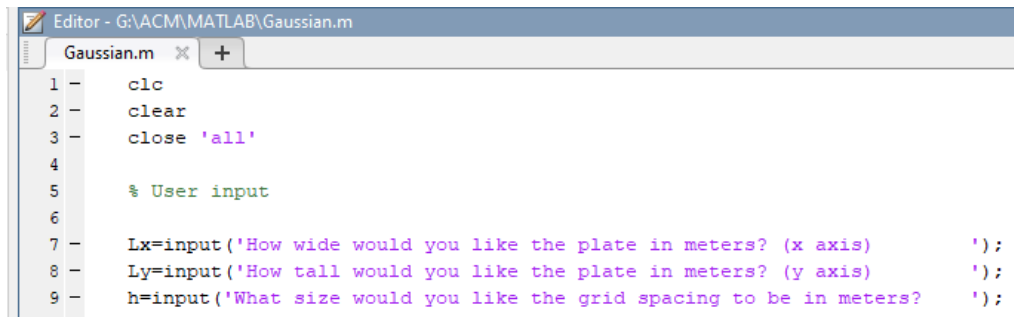
Figure 4.7

5 The Model

The full script can be seen in appendix A, the individual sections will be talked about below.

5.1 User Input and Validation

The first part of the script clears the command window, workspace and closes any figures. The script then asks the user to define the width, height and grid spacing of the plate. As can be seen in *figure 5.2*.

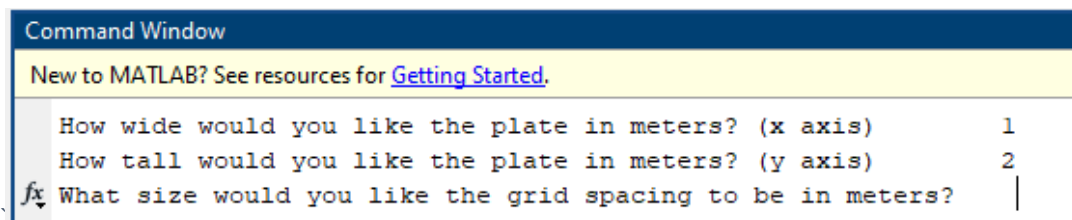


```

1 -   clc
2 -   clear
3 -   close 'all'
4
5 -   % User input
6
7 -   Lx=input('How wide would you like the plate in meters? (x axis) ');
8 -   Ly=input('How tall would you like the plate in meters? (y axis) ');
9 -   h=input('What size would you like the grid spacing to be in meters? ');

```

Figure 5.1 script



```

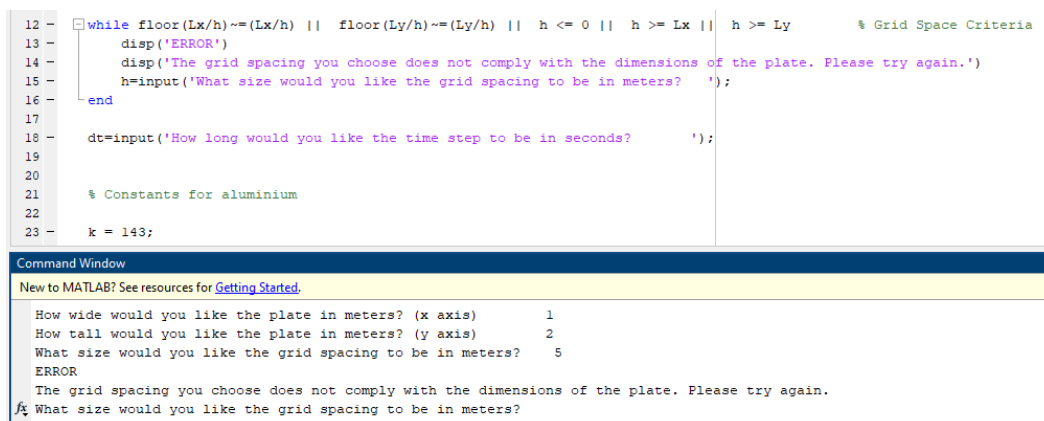
Command Window
New to MATLAB? See resources for Getting Started.

How wide would you like the plate in meters? (x axis)      1
How tall would you like the plate in meters? (y axis)      2
fx What size would you like the grid spacing to be in meters? |

```

Figure 5.2 command window

The script then checks to see if the grid spacing will fit uniformly on the defined plate. If it doesn't the user has to re-enter their grid spacing. This will keep happening until the user input a valid grid spacing. See *figure 5.3*.



```

12 - while floor(Lx/h)~=Lx/h || floor(Ly/h)~=Ly/h || h <= 0 || h >= Lx || h >= Ly % Grid Space Criteria
13 -     disp('ERROR')
14 -     disp('The grid spacing you choose does not comply with the dimensions of the plate. Please try again.')
15 -     h=input('What size would you like the grid spacing to be in meters? ');
16 - end
17
18 - dt=input('How long would you like the time step to be in seconds? ');
19
20
21 - % Constants for aluminium
22
23 - k = 143;

```

```

Command Window
New to MATLAB? See resources for Getting Started.

How wide would you like the plate in meters? (x axis)      1
How tall would you like the plate in meters? (y axis)      2
What size would you like the grid spacing to be in meters?  5
ERROR
The grid spacing you choose does not comply with the dimensions of the plate. Please try again.
fx What size would you like the grid spacing to be in meters?

```

Figure 5.3 grid spacing

The user is then prompted to enter a time step for the calculation, see *figure 5.4*. This will be checked that it meets the criteria in section 5.2 and if it doesn't they'll be prompted to re-attempt a time step.

```

18 - dt=input('How long would you like the time step to be in seconds? ');
19
20
21 % Constants for aluminium
22
23 - k = 143;

```

Command Window

New to MATLAB? See resources for [Getting Started](#).

How wide would you like the plate in meters? (x axis)	2
How tall would you like the plate in meters? (y axis)	1
What size would you like the grid spacing to be in meters?	0.1
fx How long would you like the time step to be in seconds?	

Figure 5.4 time step

5.2 Constants and Fourier Number Validation

The script then calculates the thermal diffusivity using *equation 3.18*, using the constants which have been assigned for aluminium. Which are $k = 143\text{W/mC}$, $\rho = 2.8 \times 10^3\text{kg/m}^3$ and $c = 795\text{J/kg.C}$ (Marotin, 2019).

Fouriers Number is then calculated using *equation 2.2*.

```

Gaussian.m  x  +
19
20
21 % Constants for aluminium
22
23 - k = 143;
24 - p = 2.8*10^3;
25 - c = 795;
26
27 - alpha=k/(p*c);
28
29 - Fo=alpha*dt/(h^2);
30
31 - while Fo < 0 || Fo >= 0.25 % Time Step Criteria
32 -     disp('ERROR')
33 -     disp('The time step you choose is invalid. Please try again.')
34 -     dt=input('How long would you like the time step to be in seconds? ');
35 -
36 -     Fo=alpha*dt/(h^2);
37 - end

```

Figure 5.5 constants

The stability criterion is then checked for validity using *equation 2.3*. If the stability criterion is broken, the user is prompted to enter a new time step until they select a value which doesn't break the stability criterion.

Command Window

New to MATLAB? See resources for [Getting Started](#).

How wide would you like the plate in meters? (x axis)	2
How tall would you like the plate in meters? (y axis)	1
What size would you like the grid spacing to be in meters?	0.1
How long would you like the time step to be in seconds?	1000000000000000
ERROR	
The time step you choose is invalid. Please try again.	
fx How long would you like the time step to be in seconds?	

Figure 5.6

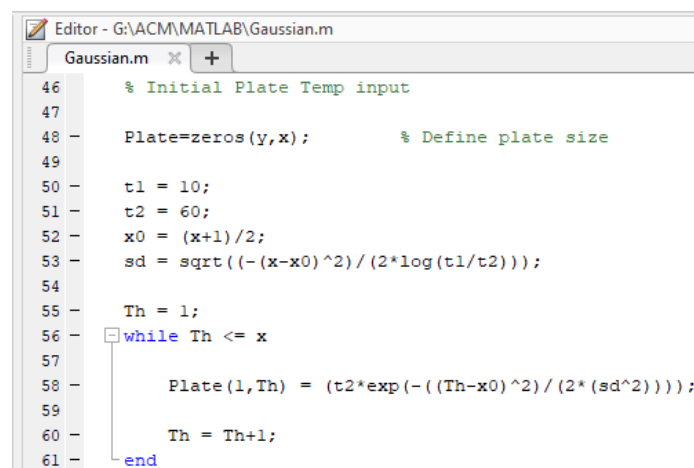
The number of nodes in both axes are then calculated by dividing the plate dimensions by the grid spacing and adding one, values x and y in *figure 5.7*.

```
41 - x=Lx/h;  
42 - y=Ly/h;  
43 - x=x+1;           % No. of x axis nodes  
44 - y=y+1;           % No. of y axis nodes  
45
```

Figure 5.7 node calculation

5.3 Setting up Initial Matrix

A matrix is then created to represent the plate with all values of zero, this is because the initial plate as explained in section 4 is zero apart from the top edge. The standard deviation and Gaussian distribution are then calculated for the specific plate dimensions. This assigns the temperature value at each node on the top row, see *figure 5.8*.



```
Editor - G:\ACM\MATLAB\Gaussian.m  
Gaussian.m x +  
46 % Initial Plate Temp input  
47  
48 Plate=zeros(y,x); % Define plate size  
49  
50 t1 = 10;  
51 t2 = 60;  
52 x0 = (x+1)/2;  
53 sd = sqrt(-(x-x0)^2/(2*log(t1/t2)));  
54  
55 Th = 1;  
56 while Th <= x  
57  
58 Plate(1,Th) = (t2*exp(-(Th-x0)^2/(2*(sd^2))));  
59  
60 Th = Th+1;  
61 end
```

Figure 5.8 initial matrix

5.4 Stop Criteria

This code must run until Thermal Equilibrium, or 'TE', is met. This means a criterion must be made to assess when Thermal Equilibrium is met. This has been done by setting up two matrices, TE_monitor and TE_final where TE_monitor contains all zeros, apart from on the top row which is all ones, and TE_final contains all ones. The first matrix will be changed from a value of zero to a value of one at each node when thermal equilibrium has been reached. Once this matrix is the same as the second matrix thermal equilibrium will have been reached, See *figure 5.9*.

```

62
63 % Calculat
64
65 - TE_monitor=ones(y,x); % Set up dimensions of matrix to monitor thermal equilibrium
66 - TE_monitor(2:y,l:x)=0; % Input values of matrix to monitor thermal equilibrium
67 - TE_final=ones(y,x); % Matrix that will be displayed by TE_monitor when thermal equilibrium is reached
68 - TE_tolerance=0.999999; % The tolerance in which the previous temp divided by the current temp has to be 1
69 - TE_checker=0;

```

Figure 5.9 stop criteria

To decide at each node whether thermal equilibrium has been reached a calculation, TE_checker, is carried out. This is where the temperature at the old node is divided by the temperature at the new node. This value is then checked to see if it is greater than the assigned tolerance, TE_tolerance, of 0.999999. This checks to see if the old temperature is greater than 99.9999% of the new temperature. If it is then that node in the TE_monitor matrix is turned from a zero to a one. See *figure 5.10*.

```

86 - TE_checker=Plate_old(r,c)/Plate(r,c);
87 - if TE_checker > TE_tolerance; % Checks to see if thermal equilibrium has been reached
88 -     TE_monitor(r,c) = 1;
89 - end

```

Figure 5.10 thermal equilibrium criteria

Whilst the sum of all the values in the TE_monitor matrix are less than the sum of all the values in the TE_final matrix, the plate values are calculated. Once it is equal the calculation will stop as thermal equilibrium will have been reached.

```

72
73 - while sum(TE_monitor,'all') < sum(TE_final,'all')
74
75 -     Iteration=Iteration+1;
76 -     Plate_old=Plate;

```

Figure 5.11 calculating criteria

5.5 Calculating Plate Temperature until Thermal Equilibrium

Before each calculation takes place the number of iterations is calculated by taking the current iterations and adding one.

```

71 - Iteration=0; % Set up number
72
73 - while sum(TE_monitor,'all') < sum(TE_final,'all')
74
75 -     Iteration=Iteration+1;
76 -     Plate_old=Plate;

```

Figure 5.12

The equations derived in section 4 are used to calculate the internal and boundary temperatures of the plate.

```

72
73 - while sum(TE_monitor,'all') < sum(TE_final,'all') % Runs whilst all the points in monitor matrix haven'
74
75 -     Iteration=Iteration+1;
76 -     Plate_old=Plate;
77
78 -     x=2;
79 -     c=2;
80
81 -     while c < x
82 -         for r=2:y-1
83 -             Tp=Plate(r,c);
84 -             Plate(r,c)=Fo*(Plate_old(r,c+1)+Plate_old(r,c-1)+Plate_old(r-1,c)+Plate_old(r+1,c)-(4-(1/Fo))*Tp);
85
86 -             TE_checker=Plate_old(r,c)/Plate(r,c);
87 -             if TE_checker > TE_tolerance % Checks to see if thermal equilibrium has been reached
88 -                 TE_monitor(r,c) = 1;
89 -             end
90
91 -         end
92
93 -         c=c+1;
94 -     end
95

```

Figure 5.13 The internal node

```

96 % Left Edge
97
98 - c = 1;
99 - r = 2;
100
101 - while r < y
102
103 -     Tp=Plate(r,c);
104 -     Plate(r,c)=Fo*(Plate_old(r,c+1)+Plate_old(r,c-1)+Plate_old(r-1,c)+Plate_old(r+1,c)-(4-(1/Fo))*Tp);
105
106 -     TE_checker=Plate_old(r,c)/Plate(r,c);
107 -     if TE_checker > TE_tolerance % Checks to see if thermal equilibrium has been reached
108 -         TE_monitor(r,c) = 1;
109 -     end
110
111 -     r = r+1;
112 - end
113
114 % Right Edge
115
116 - c = x;
117 - r = 2;
118
119 - while r < y
120
121 -     Tp=Plate(r,c);
122 -     Plate(r,c)=Fo*(Plate_old(r,c-1)+Plate_old(r,c+1)+Plate_old(r-1,c)+Plate_old(r+1,c)-(4-(1/Fo))*Tp);
123
124 -     TE_checker=Plate_old(r,c)/Plate(r,c);
125 -     if TE_checker > TE_tolerance % Checks to see if thermal equilibrium has been reached
126 -         TE_monitor(r,c) = 1;
127 -     end
128
129 -     r = r+1;
130
131 - end

```

Figure 5.14 The Left and right edge

```

132 % Bottom Edge
133
134 c = 2;
135
136 while c < x
137     Tp=Plate(y,c);
138     Plate(y,c)=Fo*(Plate_old(y,c+1)+Plate_old(y,c-1)+Plate_old(y-1,c)+Plate_old(y-1,c)-(4-(1/Fo))*Tp);
139
140     TE_checker=Plate_old(y,c)/Plate(y,c);
141     if TE_checker > TE_tolerance % Checks to see if thermal equilibrium has been reached
142         TE_monitor(y,c) = 1;
143     end
144     c = c+1;
145 end
146
147 % Bottom Left Corner
148
149 Tp=Plate(y,1);
150 Plate(y,1)=Fo*(Plate_old(y,2)+Plate_old(y,2)+Plate_old(y-1,1)+Plate_old(y-1,1)-(4-(1/Fo))*Tp); % Ca
151
152 TE_checker=Plate_old(y,1)/Plate(y,1);
153 if TE_checker > TE_tolerance % Checks to see if thermal equilibrium has been reached
154     TE_monitor(y,1) = 1;
155 end
156
157 % Bottom Right Corner
158
159 Tp=Plate(y,x);
160 Plate(y,x)=Fo*(Plate_old(y,x-1)+Plate_old(y,x-1)+Plate_old(y-1,x)+Plate_old(y-1,x)-(4-(1/Fo))*Tp);
161
162 TE_checker=Plate_old(y,x)/Plate(y,x);
163 if TE_checker > TE_tolerance % Checks to see if thermal equilibrium has been reached
164     TE_monitor(y,x) = 1;
165 end
166

```

Figure 5.15 The bottom edge and corner node

5.6 Graphical Representation of Plate

A live plot is displayed to the user whilst all these calculations are taken place using a colour map to give the user a visual representation of what is happening. This colour map is set up displaying the matrix as a figure and using standardised colours within MATLAB. Can be seen in *figure 5.16*.

```

175 % Setting up live plot
176
177 plotted_plate = flipud(Plate);
178 f = figure(1);
179 fl.WindowState = 'maximized';
180 set(gcf, 'color', 'w');
181
182 colormap(jet)
183 contourf(ploted_plate)
184 key=colorbar;
185 key.Label.String='Temperature (Degrees Celsius)';
186
187 title('Temperature Distrubution across the 2D Plate')
188 xlabel('Number of nodes on the x axis (width)')
189 ylabel('Number of nodes on the y axis (height)')
190 xticks(1:x)
191 yticks(1:y)

```

Figure 5.16 live plot

A title is also given as well as axis labels and tick values, the colour map, depending on user input, will start off looking like *figure 5.17*

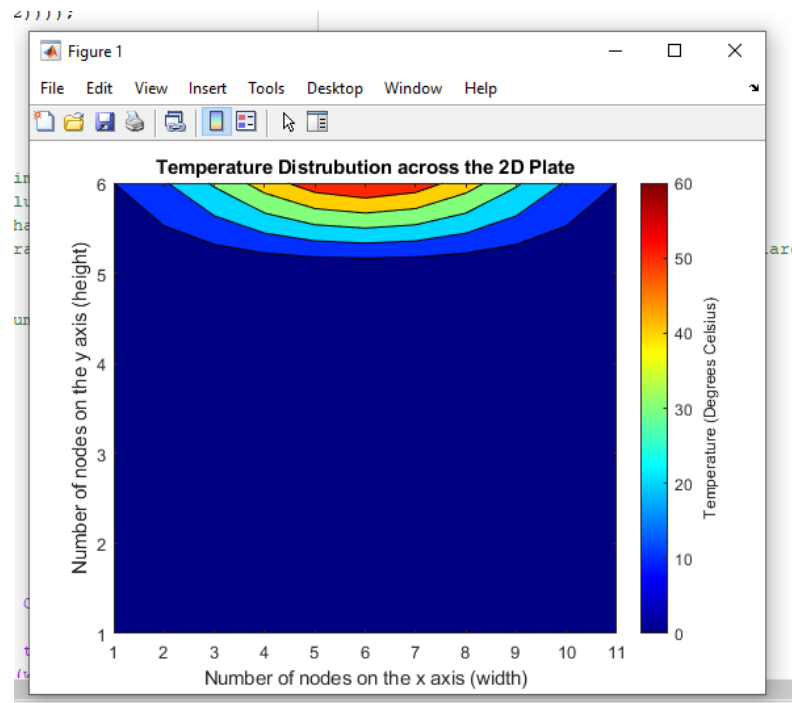


Figure 5.17 titles

5.7 Results

Once thermal equilibrium has been reached the while loop is exited, leaving the final colour map of the matrix on the user's computer. See *figure 5.18* for an example plate.

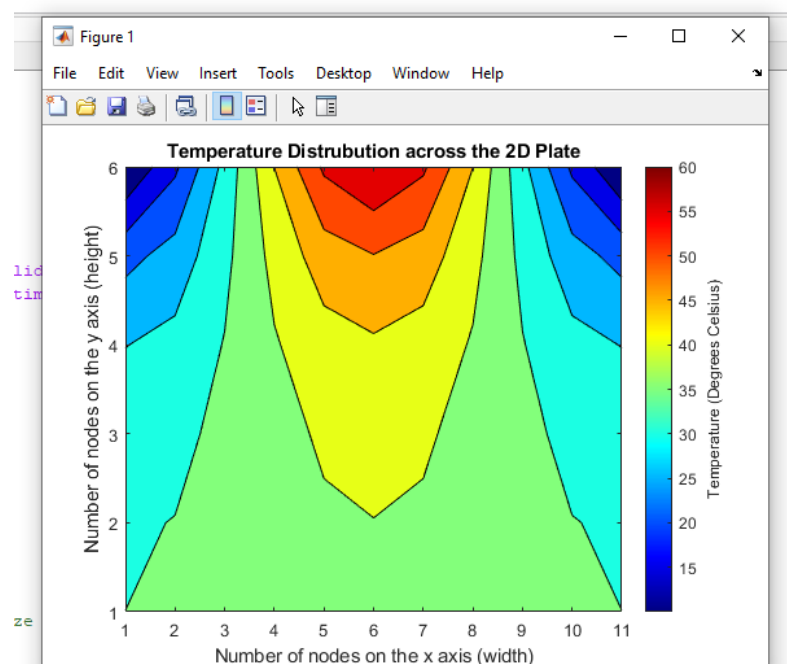


Figure 5.18

The total time is then calculated by multiplying the time step by the number of iterations, this is also calculated in hours by dividing by 3600, see *figure 5.19*. The time in seconds and hours is displayed to

the user in a pop-up message box, as well as a second message box displaying the final matrix as its raw values. These message boxes have titles and can be seen in *figures 5.20 and 5.21*.

```

196 % Results
197
198 - Iteration;
199 - Total_Time = Iteration*dt; % Total time for the plate to reach thermal equilibriu
200 - Total_Hour = Total_Time/3600; % Total time for the plate to reach thermal equilibriu
201 - Plate
202
203 - msgbox(num2str(Plate), 'Final Plate Values at each Node') % Matrix displayed as Values
204
205
206 - total_time=sprintf('The total time was %d seconds or %f hours', Total_Time, Total_Hour);
207 - f = msgbox(total_time, 'Thermal Equilibrium Time');
208
209

```

Figure 5.19

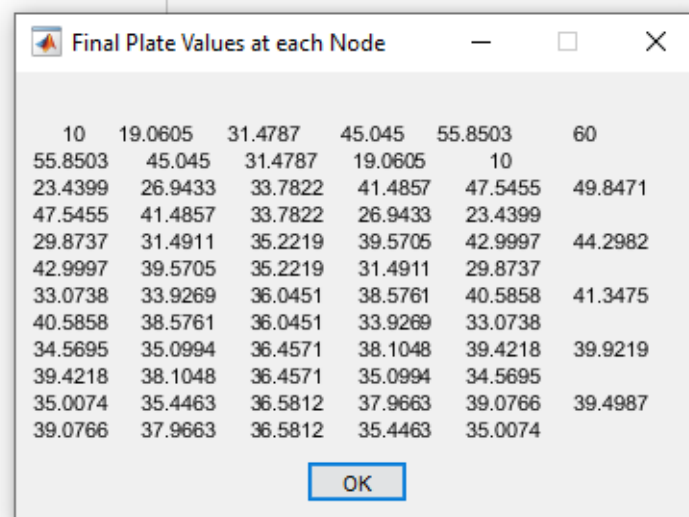


Figure 5.20

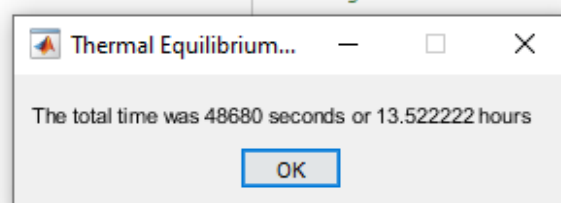


Figure 5.21

5.8 Linear Script Validation

To ensure this script is valid a simpler script was written using known boundary conditions as the internal temperature would be known at thermal equilibrium. This script can be seen in appendix B. It uses the same calculations as the Gaussian script but doesn't calculate the boundary temps as they are constant. Since this script could be checked by hand and confirmed to be correct, the code internally was deemed to be valid. Also, since the boundary value in the Gaussian look correct next to the assumed correct internal calculation, a deduction can be made that the script is assumed to be correct. The script was also compared to other students and similar results were given which is another form of validation. The reason the same results would not be seen amongst students is due to everyone's criteria on when thermal equilibrium is reached being different.

6 Results

For a constant plate size of 2 meters by 1 meter, in the x and y axis respectively, two tables of results were produced. The first, *table 6.1*, was to see how the grid spacing effects the time to reach equilibrium and the second, *table 6.2*, is to see how the time step effects how long it takes to reach equilibrium.

Grid Spacing, h (m)	Timestep, dt (s)	Time to Equilibrium, t (s)
0.05	5	11625
0.1	5	11665
0.2	5	11760
0.25	5	11950
0.5	5	12980

Table 6.1

From *table 6.1*, a correlation can be made that the bigger the grid spacing the longer the model takes to reach thermal equilibrium. This is as expected for larger grid spacings as the nodes are further apart. Therefore, two adjacent nodes would have a node between it for half the spacing, so the method would transfer heat faster through more nodes.

From these results, a deduction can be made that the smaller the grid spacing the more accurate the results. However, this would not always be the case, if computer limitations were ignored, and an extremely small grid spacing was used, the result would be incorrect. This is due to the grid having lots more nodes and the accumulated error at all these nodes would cause a much bigger error than a larger grid spacing.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	50	47.5000	45	42.5000	40	37.5000	35	32.5000	30	27.5000	25	22.5000	20	17.5000	15	12.5000	10	7.5000	5	2.5000	0
2	50	47.4998	44.9996	42.4994	39.9993	37.4991	34.9990	32.4989	29.9988	27.4988	24.9987	22.4988	19.9988	17.4989	14.9990	12.4991	9.9993	7.4994	4.9996	2.4998	0
3	50	47.4996	44.9993	42.4989	39.9986	37.4983	34.9981	32.4979	29.9977	27.4976	24.9976	22.4976	19.9977	17.4979	14.9981	12.4983	9.9986	7.4989	4.9993	2.4996	0
4	50	47.4995	44.9990	42.4985	39.9981	37.4977	34.9973	32.4971	29.9969	27.4968	24.9967	22.4968	19.9969	17.4971	14.9973	12.4977	9.9981	7.4985	4.9990	2.4995	0
5	50	47.4994	44.9988	42.4982	39.9977	37.4973	34.9969	32.4966	29.9963	27.4962	24.9961	22.4962	19.9963	17.4966	14.9969	12.4973	9.9977	7.4983	4.9988	2.4994	0
6	50	47.4994	44.9987	42.4982	39.9976	37.4971	34.9967	32.4964	29.9961	27.4960	24.9959	22.4960	19.9961	17.4964	14.9967	12.4971	9.9976	7.4982	4.9988	2.4994	0
7	50	47.4994	44.9988	42.4982	39.9977	37.4973	34.9969	32.4966	29.9963	27.4962	24.9961	22.4962	19.9963	17.4966	14.9969	12.4973	9.9977	7.4983	4.9988	2.4994	0
8	50	47.4995	44.9990	42.4985	39.9981	37.4977	34.9973	32.4971	29.9969	27.4968	24.9967	22.4968	19.9969	17.4971	14.9973	12.4977	9.9981	7.4985	4.9990	2.4995	0
9	50	47.4996	44.9993	42.4989	39.9986	37.4983	34.9981	32.4979	29.9977	27.4976	24.9976	22.4976	19.9977	17.4979	14.9981	12.4983	9.9986	7.4989	4.9993	2.4996	0
10	50	47.4998	44.9996	42.4994	39.9993	37.4991	34.9990	32.4989	29.9988	27.4988	24.9987	22.4988	19.9988	17.4989	14.9990	12.4991	9.9993	7.4994	4.9996	2.4998	0
11	50	47.5000	45	42.5000	40	37.5000	35	32.5000	30	27.5000	25	22.5000	20	17.5000	15	12.5000	10	7.5000	5	2.5000	0

Figure 6.1 – grid spacing is 0.1m

	1	2	3	4	5	6	7	8	9	10	11
1	50	45	40	35	30	25	20	15	10	5	0
2	50	44.9992	39.9985	34.9979	29.9975	24.9974	19.9975	14.9979	9.9985	4.9992	0
3	50	44.9987	39.9975	34.9966	29.9960	24.9958	19.9960	14.9966	9.9975	4.9987	0
4	50	44.9987	39.9975	34.9966	29.9960	24.9958	19.9960	14.9966	9.9975	4.9987	0
5	50	44.9992	39.9985	34.9979	29.9975	24.9974	19.9975	14.9979	9.9985	4.9992	0
6	50	45	40	35	30	25	20	15	10	5	0

Figure 6.2 - grid spacing is 0.2m

	1	2	3	4	5
1	50	37.5000	25	12.5000	0
2	50	37.4962	24.9947	12.4962	0
3	50	37.5000	25	12.5000	0

Figure 6.3 - grid spacing is 0.5m

In figures 6.1, 6.2 and 6.3 the accuracy of the results can be analysed. The value in the middle of each plate is 24.995, 24.9958 and 24.9947. The smaller the grid spacing, the closer the value is to 25, which is the actual value the node should reach to be in thermal equilibrium. Therefore, this also agrees to the earlier statement that smaller grid spacing, within reason, will give a more accurate result.

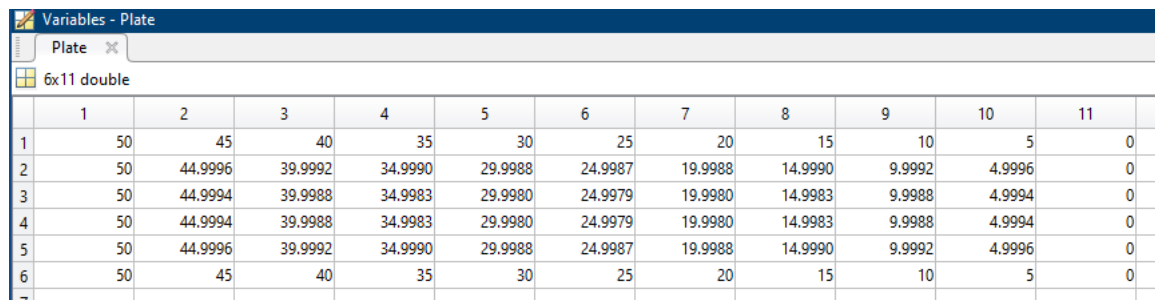
Grid Spacing, h (m)	Timestep, dt (s)	Time to Equilibrium, t (s)
0.2	2	10578
0.2	5	11760
0.2	10	12650
0.2	20	13500
0.2	50	14550

Table 6.2

Table 6.2 shows that as the time step of the code is increased, the time to thermal equilibrium will also increase. This is caused due to the time being calculated at larger steps making the approximation less accurate. This is because the time step is what carries the error of first-order dt . Here, the time step is the error order in the forward's difference equation so the approximation is only accurate for a small value of dt .

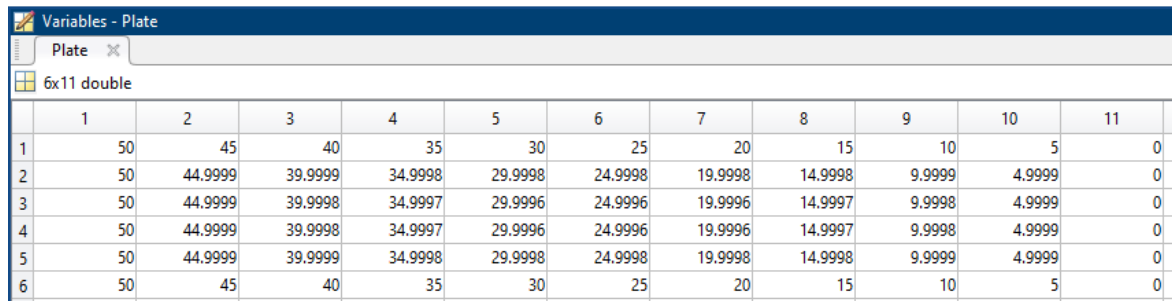
	1	2	3	4	5	6	7	8	9	10	11
1	50	45	40	35	30	25	20	15	10	5	0
2	50	44.9980	39.9961	34.9947	29.9938	24.9935	19.9938	14.9947	9.9962	4.9980	0
3	50	44.9967	39.9937	34.9914	29.9899	24.9894	19.9900	14.9915	9.9938	4.9968	0
4	50	44.9967	39.9937	34.9914	29.9899	24.9894	19.9900	14.9915	9.9938	4.9968	0
5	50	44.9980	39.9961	34.9947	29.9938	24.9935	19.9938	14.9947	9.9962	4.9980	0
6	50	45	40	35	30	25	20	15	10	5	0

Figure 6.4 – time step is 2s



	1	2	3	4	5	6	7	8	9	10	11
1	50	45	40	35	30	25	20	15	10	5	0
2	50	44.9996	39.9992	34.9990	29.9988	24.9987	19.9988	14.9990	9.9992	4.9996	0
3	50	44.9994	39.9988	34.9983	29.9980	24.9979	19.9980	14.9983	9.9988	4.9994	0
4	50	44.9994	39.9988	34.9983	29.9980	24.9979	19.9980	14.9983	9.9988	4.9994	0
5	50	44.9996	39.9992	34.9990	29.9988	24.9987	19.9988	14.9990	9.9992	4.9996	0
6	50	45	40	35	30	25	20	15	10	5	0

Figure 6.5– time step is 10s



	1	2	3	4	5	6	7	8	9	10	11
1	50	45	40	35	30	25	20	15	10	5	0
2	50	44.9999	39.9999	34.9998	29.9998	24.9998	19.9998	14.9998	9.9999	4.9999	0
3	50	44.9999	39.9998	34.9997	29.9996	24.9996	19.9996	14.9997	9.9998	4.9999	0
4	50	44.9999	39.9998	34.9997	29.9996	24.9996	19.9996	14.9997	9.9998	4.9999	0
5	50	44.9999	39.9999	34.9998	29.9998	24.9998	19.9998	14.9998	9.9999	4.9999	0
6	50	45	40	35	30	25	20	15	10	5	0

Figure 6.6– time step is 50s

If the values for the node (6,2) are evaluated, the accuracy can be analysed visually. The values 24.9935, 24.9987 and 24.9998 are for *figure 6.4*, 6.5 and 6.6 respectively. The reason it appears that the larger time step is more accurate is only because when the calculations took place the larger time step took longer to satisfy the tolerance to reach thermal equilibrium. If the temperatures were all compared at the same moment in time it would be seen that the smaller time step would be closer to thermal equilibrium.

7 Limitations and Improvements

The limitation of this model is that it is only an approximation for small grid spacing. However, as discussed previously if h goes really small then there will be lots of nodes which will result in a higher accumulation of errors, therefore making the results invalid. Also, the code doesn't consider the real world where radiation and the environment it is in will affect the results.

To improve the model a more accurate constitutive equation could be created so that a better approximation could be made. Also, to make the model act like real life the script could request the user to enter environment parameters, such as room temperatures. The model could also be improved by altering the temperature calculation to allow for the fact that some radiation would take place as nothing can be perfectly insulated. For this, the user could specify the insulation materials and enter the constants for that material or the user could select a pre-given insulation.

8 Conclusion

To conclude, the model is useful to understand the trends of what would happen in an approximated and ideal world. The script used was a simple way to calculate multiple results at such a quick speed for the user. However, if the user needs real-life values rather than trends, an improved model would have to be made or a specific physical investigation would need to be carried out for that user.

Bibliography

Abramowitz, M. & Stegun, I. A., 1965. *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables*. Dover: Dover Publications.

Cheng, A. H. D. & Cheng, D. T., 2005. Heritage and early history of the boundary element method. *Engineering Analysis with Boundary Elements*, March, 29(3), pp. 268-302.

Croft, D. R., Stone, J. A. R. & Lilley, D. G., 1986. *Heat Transfer Calculation Using Finite Difference Equations*. Sheffield: Pavic Publications.

Faires, J. D. & Burden, R., 2013. Boundary-Value Problems for Ordinary Differential Equations. In: P. Boardman, ed. *Numerical Methods*. 4 ed. Boston: Cengage, pp. 441-451.

Ferziger, J. H. & Perić, M., 2002. Finite Difference Methods. In: *Computational Methods for Fluid Dynamics*. New York: Springer, pp. 39-62.

Harman, T. L., Dabney, J. & Richert, N., 1997. *Advanced Engineering Mathematics Using MATLAB*. Boston: PWS Publishing Company.

Henizl, R., 2007. *Concepts for Scientific Computing*. na: na.

Herrmann, W. & Bertholf, L. D., 1983. Explicit Finite Difference Method. In: T. Belytschko & T. J. R. Hughes, eds. *Computational Methods in Mechanics*. New York: Elsevier, pp. 361-400.

Liu, P. S. & F, C. G., 2014. Characterization Methods: Physical Properties. In: *Porous Materials*. Oxford: Elsevier Inc, pp. 493-532.

Marotin, A., 2019. *Assignment Brief*. Swansea: UWTSD.

Price, R. M., 2004. *Unsteady (Transient) Conduction: Lectures: Facstaff*. [Online]
Available at: <http://facstaff.cbu.edu/rprice/lectures/unsteady.html>
[Accessed November 2019].

Appendices

Appendix A – Gaussian Script

```
clc
clear
close 'all'

% User input

Lx=input('How wide would you like the plate in meters? (x axis) ');
Ly=input('How tall would you like the plate in meters? (y axis) ');
h=input('What size would you like the grid spacing to be in meters? ');

while floor(Lx/h)~= (Lx/h) || floor(Ly/h)~= (Ly/h) || h <= 0 || h >= Lx ||
h >= Ly % Grid Space Criteria
    disp('ERROR')
    disp('The grid spacing you choose does not comply with the dimensions
of the plate. Please try again.')
    h=input('What size would you like the grid spacing to be in meters?
');
end

dt=input('How long would you like the time step to be in seconds? ');

% Constants for aluminium

k = 143;
p = 2.8*10^3;
c = 795;

alpha=k/(p*c);

Fo=alpha*dt/(h^2);

while Fo < 0 || Fo >= 0.25 % Time Step Criteria
    disp('ERROR')
    disp('The time step you choose is invalid. Please try again.')
    dt=input('How long would you like the time step to be in seconds? ');
    Fo=alpha*dt/(h^2);
end

x=Lx/h;
y=Ly/h;
x=x+1; % No. of x axis nodes
y=y+1; % No. of y axis nodes

% Initial Plate Temp input

Plate=zeros(y,x); % Define plate size
```

```
t1 = 10;
t2 = 60;
x0 = (x+1)/2;
sd = sqrt((- (x-x0)^2)/(2*log(t1/t2)));

Th = 1;
while Th <= x

    Plate(1,Th) = (t2*exp(-(Th-x0)^2)/(2*(sd^2)));

    Th = Th+1;
end

% Calcualte

TE_monitor=ones(y,x); % Set up dimensions of matrix to
monitor thermal equilibrium
TE_monitor(2:y,1:x)=0; % Input values of matrix to monitor
thermal equilibrium
TE_final=ones(y,x); % Matrix that will be displayed by
TE_monitor when thermal equilibrium is reached
TE_tolerance=0.999999; % The tolerance in which the previous
temp divided by the current temp has to be larger than to say that thermal
equilibrium has been reached
TE_checker=0;

Iteration=0; % Set up number of iterations

while sum(TE_monitor,'all') < sum(TE_final,'all') % Runs whilst all
the points in monitor matrix haven't reached thermal equilibrium (i.e not
all one)

    Iteration=Iteration+1;
    Plate_old=Plate;

    r=2;
    c=2;

    while c < x;
        for r=2:y-1;
            Tp=Plate(r,c);
            Plate(r,c)=Fo*(Plate_old(r,c+1)+Plate_old(r,c-1)+Plate_old(r-
1,c)+Plate_old(r+1,c)-(4-(1/Fo))*Tp); % Calculate new temp for each
node (not inc. the edges)

            TE_checker=Plate_old(r,c)/Plate(r,c);
            if TE_checker > TE_tolerance; % Checks to see if
thermal equilibrium has been reached
                TE_monitor(r,c) = 1;
            end
        end

        c=c+1;
    end

    % Left Edge
```

```
c = 1;
r = 2;

while r < y

    Tp=Plate(r,c);
    Plate(r,c)=Fo*(Plate_old(r,c+1)+Plate_old(r,c+1)+Plate_old(r-
1,c)+Plate_old(r+1,c)-(4-(1/Fo))*Tp);           % Calculate new temp for each
node on the left edge

    TE_checker=Plate_old(r,c)/Plate(r,c);
    if TE_checker > TE_tolerance;                % Checks to see if
thermal equilibrium has been reached
        TE_monitor(r,c) = 1;
    end

    r = r+1;
end

% Right Edge

c = x;
r = 2;

while r < y

    Tp=Plate(r,c);
    Plate(r,c)=Fo*(Plate_old(r,c-1)+Plate_old(r,c-1)+Plate_old(r-
1,c)+Plate_old(r+1,c)-(4-(1/Fo))*Tp);           % Calculate new temp for each
node on the right edge

    TE_checker=Plate_old(r,c)/Plate(r,c);
    if TE_checker > TE_tolerance;                % Checks to see if
thermal equilibrium has been reached
        TE_monitor(r,c) = 1;
    end

    r = r+1;
end

% Bottom Edge

c = 2;

while c < x

    Tp=Plate(y,c);
    Plate(y,c)=Fo*(Plate_old(y,c+1)+Plate_old(y,c-1)+Plate_old(y-
1,c)+Plate_old(y-1,c)-(4-(1/Fo))*Tp);           % Calculate new temp for each
node on the bottom edge not including the corners

    TE_checker=Plate_old(y,c)/Plate(y,c);
    if TE_checker > TE_tolerance;                % Checks to see if
thermal equilibrium has been reached
        TE_monitor(y,c) = 1;
    end
```

```

        c = c+1;
    end

    % Bottom Left Corner

    Tp=Plate(y,1);
    Plate(y,1)=Fo*(Plate_old(y,2)+Plate_old(y,2)+Plate_old(y-
    1,1)+Plate_old(y-1,1)-(4-(1/Fo))*Tp); % Calculate new temp for
    bottom left node

    TE_checker=Plate_old(y,1)/Plate(y,1);
    if TE_checker > TE_tolerance; % Checks to see if thermal
    equilibrium has been reached
        TE_monitor(y,1) = 1;
    end

    % Bottom Right Corner

    Tp=Plate(y,x);
    Plate(y,x)=Fo*(Plate_old(y,x-1)+Plate_old(y,x-1)+Plate_old(y-
    1,x)+Plate_old(y-1,x)-(4-(1/Fo))*Tp); % Calculate new temp for
    bottom right node

    TE_checker=Plate_old(y,x)/Plate(y,x);
    if TE_checker > TE_tolerance; % Checks to see if thermal
    equilibrium has been reached
        TE_monitor(y,x) = 1;
    end

    Time_step = Iteration * dt; % Cumulative time after each
    iteration for newly calculated plate

    % Setting up live plot

    plotted_plate = flipud(Plate);
    f = figure(1);
    f1.WindowState = 'maximized';
    set(gcf, 'color', 'w');

    colormap(jet)
    contourf(plotted_plate)
    key=colorbar; %
    Inputting a Key for the image %
    key.Label.String='Temperature (Degrees Celsius)'; %
    Giving the key a label

    title('Temperature Distrubution across the 2D Plate') %
    Giving the graph a title
    xlabel('Number of nodes on the x axis (width)') %
    Labeling the x axis
    ylabel('Number of nodes on the y axis (height)') %
    Labeling the y axis
    xticks(1:x) %
    inputting the interval between ticks on the x axis

```

```
        yticks(1:y)                                     %  
inputting the interval between ticks on the y axis  
  
end  
  
% Results  
  
Iteration;  
Total_Time = Iteration*dt;           % Total time for the plate to reach  
thermal equilibrium in seconds  
Total_Hour = Total_Time/3600;        % Total time for the plate to reach  
thermal equilibrium in hours  
Plate  
  
msgbox(num2str(Plate), 'Final Plate Values at each Node') % Matrix  
displayed as Values  
  
total_time=sprintf('The total time was %d seconds or %f hours', Total_Time,  
Total_Hour);           % Putting total times in to text  
f = msgbox(total_time, 'Thermal Equilibrium Time');  
% Displaying Time to equilibrium in a message box
```

END OF APPENDIX A

Appendix B – Linear Script

```
clc
clear
close 'all'

% User input

Lx=input('How wide would you like the plate in meters? (x axis) ');
Ly=input('How tall would you like the plate in meters? (y axis) ');
h=input('What size would you like the grid spacing to be in meters? ');

while floor(Lx/h)~= (Lx/h) || floor(Ly/h)~= (Ly/h) || h <= 0 || h >= Lx ||
h >= Ly % Grid Space Criteria
    disp('ERROR')
    disp('The grid spacing you choose does not comply with the dimensions
of the plate. Please try again.')
    h=input('What size would you like the grid spacing to be in meters?
');
end

dt=input('How long would you like the time step to be in seconds? ');

% Constants

k = 143;
p = 2.8*10^3;
c = 795;

alpha=k/(p*c);

Fo=alpha*dt/(h^2);

while Fo < 0 || Fo >= 0.25 % Time Step Criteria
    disp('ERROR')
    disp('The time step you choose is invalid. Please try again.')
    dt=input('How long would you like the time step to be in seconds? ');
    Fo=alpha*dt/(h^2);
end

x=Lx/h;
y=Ly/h;
x=x+1; % No. of x axis nodes
y=y+1; % No. of y axis nodes

% Initial Plate Temp input

Ti=input('What would you like the initial heating temperature to be in
degrees celsius (i.e 50)? '); %Temp of heated side of the plate
```

```
Plate=zeros(y,x);           % Define plate size
Plate(1:y,1)=Ti;           % Define heating temp on first collumn
Plate(1:y,x)=0;           % Define boundary temp on last collumn

T_side=linspace(Ti,0,x);    % Define boundary vector for temp from Ti
to 0

Plate(1,1:x)=T_side;        % Define boundary temp on the top row
Plate(y,1:x)=T_side;        % Define boundary temp on the bottom row

% Calcualte

TE_monitor=ones(y,x);       % Set up dimensions of matrix to
monitor thermal equilibrium
TE_monitor(2:y-1,2:x-1)=0;  % Input values of matrix to monitor
thermal equilibrium
TE_final=ones(y,x);         % Matrix that will be displayed by
TE_monitor when thermal equilibrium is reached
TE_tolerance=0.999999;      % The tolerance in which the previous
temp divided by the current temp has to be larger than to say that thermal
equilibrium has been reached
TE_checker=0;

Iteration=0;                % Set up number of iterations

while sum(TE_monitor,'all') < sum(TE_final,'all') % Runs whilst all
the points in monitor matrix haven't reached thermal equilibrium (i.e all
ones)

    Iteration=Iteration+1;
    Plate_old=Plate;

    r=2;
    c=2;

    while c < x;
        for r=2:y-1;
            Tp=Plate(r,c);
            Plate(r,c)=Fo*(Plate_old(r,c+1)+Plate_old(r,c-1)+Plate_old(r-
1,c)+Plate_old(r+1,c)-(4-(1/Fo))*Tp); %calculate new temp for each
node

            TE_checker=Plate_old(r,c)/Plate(r,c);
            if TE_checker > TE_tolerance; % Checks to see if
thermal equilibrium at node (r,c) has been reached
                TE_monitor(r,c) = 1;
            end
        end

        c=c+1;
    end

    Time_step = Iteration * dt; % Cumulative time after each
iteration for newly calculated plate

    % Setting up live plot
```

```
    plotted_plate = flipud(Plate);
    f = figure(1);
    f1.WindowState = 'maximized';
    set(gcf, 'color', 'w');

    colormap(jet)
    contourf(plotted_plate)
    key=colorbar; %
    Inputting a Key for the image
    key.Label.String='Temperature (Degrees Celsius)'; %
    Giving the key a label

    title('Temperature Distrubution across the 2D Plate') %
    Giving the graph a title
    xlabel('Number of nodes on the x axis (width)') %
    Labeling the x axis
    ylabel('Number of nodes on the y axis (height)') %
    Labeling the y axis
    xticks(1:x) %
    inputting the interval between ticks on the x axis
    yticks(1:y) %
    inputting the interval between ticks on the y axis

end

% Results

Iteration;
Total_Time = Iteration*dt; % Total time for the plate to reach
thermal equilibrium in seconds
Total_Hour = Total_Time/3600; % Total time for the plate to reach
thermal equilibrium in hours
Plate

msgbox(num2str(Plate), 'Final Plate Values at each Node') % Matrix
displayed as Values

total_time=sprintf('The total time was %d seconds or %f hours', Total_Time,
Total_Hour); % Putting total times in to text
f = msgbox(total_time, 'Thermal Equilibrium Time');
% Displaying Time to equilibrium in a message box
```

END OF APPENDIX B
