



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

UNIVERSITY OF WALES TRINITY SAINT
DAVID

SCHOOL OF ENGINEERING

ADVANCED COMPUTATIONAL METHODS

TRANSIENT THERMAL DIFFUSION
MODEL USING THE FINITE DIFFERENCE
METHOD.

M6X00408

MAXIMILIAN HART
1700625

27-01-2019

Table of Contents

1	Introduction	4
2	Mathematics Analysis.....	4
2.1	Thermal Diffusion Equation	4
2.2	Finite Difference Method	5
2.2.1	Background Information	5
2.2.2	Taylor Series	5
2.2.3	Central Difference Approximation	5
2.2.4	Forward Difference Approximation	6
2.2.5	Forming a Final Equation - Explicit Method	6
3	Linear and Gaussian Boundary Conditions	7
3.1	Validation Boundary Conditions - Linear Decrease	7
3.2	Assignment Boundary Conditions - Gaussian Curve	7
4	Script Details	8
4.1	User Script Selection.....	8
4.2	Defining Constants.....	8
4.3	User Input and Validation.....	9
4.4	Fourier Number Calculation and Validation.....	10
4.5	Heat Transfer Modelling Matrix	10
4.6	Stop Criterion.....	11
4.7	Heat Transfer Matrix Graphical Display	11
5	Evaluation - Initial Linear Code Results and Analysis	13
5.1	Varying Timestep - Linear Boundary Conditions	13
5.2	Varying Grid Spacing - Linear Boundary Conditions.....	14
5.3	Constant Fourier Number - Gaussian Boundary Conditions	15
5.4	Increasing Fourier Number - Gaussian Boundary Conditions	16
5.5	Varying Stop Criterion Accuracy	17
6	Extra Work - Creating a MATLAB App.....	18
7	Code Limitations, Analysis and Improvements	20
8	Referenced Work	21
9	Appendices	23
9.1	Appendix 1a - User Selection Script	23
9.2	Appendix 1b - Linear Plate Script	24
9.3	Appendix 1c - Gaussian Plate Script	29

Table of Figures

Figure 1 - Plate with nodes identified	4
Figure 2 - Plate split into grid	4
Figure 3 - Grid demonstrating N,E,S,W nodes	5
Figure 4 - Validated linear decrease heat transfer matrix	7
Figure 5 - Initial Boundary Conditions - (Marotin, 2019)	7
Figure 6 - User script selection box	8
Figure 7 - Constant Fourier Number Graphical Results	15
Figure 8 - Increasing Fourier Number Graphical Results	16
Figure 9 - Varying Stop Criterion Accuracy Graphical Results.....	17
Figure 10 - Thermal Diffusion Model MATLAB App Interface.....	18

Table of Tables

Table 1 - Plot iteration calculations 1.....	12
Table 2 - Plot iteration calculations 2.....	12
Table 3 - Results: Varying Timestep	13
Table 4 - Results: Varying Grid Spacing	14
Table 5 - Results: Constant Fourier Number	15
Table 6 - Results: Increasing Grid Spacing.....	16
Table 7 - Results: Varying Stop Criterion Accuracy	17

1 Introduction

For this report, the author used MathWorks® MATLAB to develop a thermal diffusion model that takes user inputs and set boundary conditions and calculates the heat transfer throughout a plate of aluminium. Within the model, the boundary conditions and material properties are pre-set, and the user chooses plate dimensions and configures the Fourier Number. The model shall use the explicit finite difference method to calculate the heat at set nodes throughout the plate, along a uniform grid. The calculations are time-dependant.

As an extra task, the author used the MATLAB Apps programme to create a standalone application to complete this task. This was completed once a fully functioning script was finished and tested. The script includes many user-friendly features to make it as easy to use as possible, the app takes this a step further and achieves the most user-friendly experience available within MATLAB. This opportunity was taken to learn the MATLAB Apps software as it will be a useful tool within the author's major project dissertation.

2 Mathematics Analysis

2.1 Thermal Diffusion Equation

This model is fundamentally based on the thermal diffusion equation, which governs the properties of heat dispersal throughout a 2D material:

$$\nabla^2 T = \frac{1}{\alpha} \cdot \frac{\partial T}{\partial t} \quad \therefore \quad \nabla^2 T \cdot \alpha = \frac{\partial T}{\partial t}$$

Equation 1 - Thermal Diffusion Equation

The change in temperature (∇T) is calculated from the inverse of constant alpha (α) multiplied by the differential of temperature (T) with respect to time (t) (Berciu, 2007).

Within the equation, ∇^2 is the Laplace operator, defined by the divergence of the second order differential equation (Tan and Li, 2018). Therefore, the first equation can be re-written:

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \quad \therefore \quad \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} = \frac{1}{\alpha} \cdot \frac{\partial T}{\partial t}$$

Equation 2 - Thermal Diffusion Equation Rearranged

Within these equations the constant alpha, thermal diffusivity ($\frac{m^2}{s}$), is always defined by $\alpha = \frac{k}{\rho \cdot c}$ where k is thermal conductivity ($\frac{W}{m.K}$), ρ is density ($\frac{kg}{m^3}$), and c is the specific heat capacity ($\frac{J}{kg.K}$) (Salazar, 2003).

This equation shall be used to calculate the temperature of each node throughout the plate at a given timestep. The plate will be split into a uniform grid, with both x and y dimensions divided by an equal grid spacing value. A node is formed at each intersection of a gridline, and here a temperature shall be calculated.

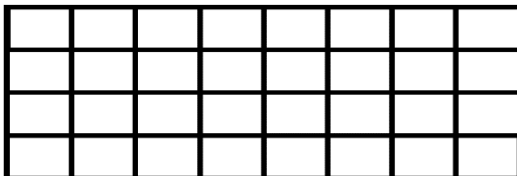


Figure 2 - Plate split into grid

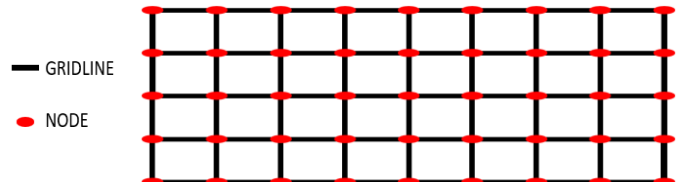


Figure 1 - Plate with nodes identified

2.2 Finite Difference Method

2.2.1 Background Information

Solving multiple iterations of second order differential equations is complex and time consuming, moreover, MATLAB is much more efficient at solving linear equations than differential equations, therefore, an approximative approach can be used; the finite difference method.

With the grid evenly divided, all nodes will have adjacent node locations that can be used to calculate the temperature from a previous timestep. The derivatives in the above equation can be substituted by the difference of adjacent nodes, known as the forward, backward or central differences, according to a Taylor series expansion.

2.2.2 Taylor Series

A Taylor series expansion serves the purpose of equating a complex expression into an infinite number of simplified terms. These terms are calculated from the derivative's functions and continually get smaller, therefore, after a point can be taken as negligible and discarded. This method creates a polynomial expression, as the derivatives, and therefore powers of x , increase with each term (Magyar, 2003).

$$f(x) = f(x_0) + f'(x_0) \cdot \left(\frac{(x - x_0)^1}{1!} \right) + f''(x_0) \cdot \left(\frac{(x - x_0)^2}{2!} \right) + f'''(x_0) \cdot \left(\frac{(x - x_0)^3}{3!} \right) + \dots$$

Equation 3 - Taylor Series Expansion

$$\therefore \text{This can be more simply stated as: } f(x) = \sum_{n=0}^{\infty} \left(f(x_0)^n \cdot \frac{(x-x_0)^n}{n!} \right)$$

Equation 4 - Taylor Series - Sigma Notation

2.2.3 Central Difference Approximation

Using the first three terms of the Taylor series, and taking $f(x)$ as T , h as grid spacing and X_0 as T_p (the node of current calculation), the second derivative of the thermal diffusion equation can be approximated as:

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

Equation 5 - Central Difference Approximation

The first numerator adds one grid spacing in x , hence is with respect to the node to the east of T_p ; the third similarly is with respect to the node west of T_p . If this equation is written in terms of y , the nodes with respect to north and south are used. Therefore, this equation can be re-written in two forms and combined back into the original (Croft and Stone, 1989; Lewis *et al.*, 1996):

$$\therefore \frac{d^2 T_p}{dx^2} = \frac{T_E^t - 2 \cdot T_P^t + T_W^t}{h^2} \quad \text{and} \quad \frac{d^2 T_p}{dy^2} = \frac{T_N^t - 2 \cdot T_P^t + T_S^t}{h^2}$$

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

Equation 6 - Central Difference Approximations in X and Y

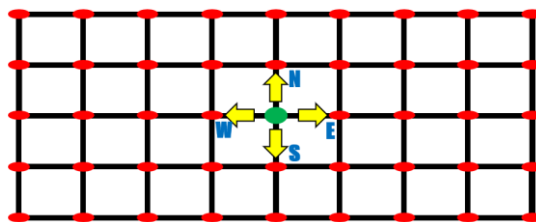


Figure 3 - Grid demonstrating N,E,S,W nodes

2.2.4 Forward Difference Approximation

Using the first two terms of the Taylor series, the first derivative within the thermal diffusion equation can also be approximated, with respect to the timestep Δt , by finding node temperatures at two adjacent times (Pei-baiZhou, 2013).

$$\frac{\partial T_P}{\partial t} = \frac{f(x_0 + h) - f(x_0)}{\Delta t} \quad \therefore \quad \frac{\partial T_P}{\partial t} = \frac{T_P^{t+\Delta t} - T_P^t}{\Delta t}$$

Equation 7 - Forward Difference Approximation

2.2.5 Forming a Final Equation - Explicit Method

From the above approximations, a final equation can be formed to find the temperature of a chosen node within the plate, using data from surrounding nodes at a previous timestep:

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

Equation 8 - Explicit Method Temperature Equation

This can then be transposed to make T_P at the next timestep the subject (Croft and Stone, 1989):

$$T_P^{t+\Delta t} = \frac{\alpha \cdot \Delta t}{h^2} \cdot (T_N^t + T_E^t + T_S^t + T_W^t - \left(4 - \frac{h^2}{\alpha \cdot \Delta t}\right) \cdot T_P^t)$$

Equation 9 - Transposed Temperature Equation

The term $\left(\frac{\alpha \cdot \Delta t}{h^2}\right)$ is referred to as the Fourier Number (F_O). It is dimensionless as it is a ratio, comparing the rate at which heat is diffused against the rate at which heat is stored. This number ensures this thermal diffusion approximation will work, as to get a result, a stability criterion must be within set conditions (Pupeikis, et al., 2012):

$$0 < F_O \leq \frac{1}{4} \text{ in a 2D system}$$

Therefore, the equation can be written more simply:

$$T_P^{t+\Delta t} = F_O \cdot (T_N^t + T_E^t + T_S^t + T_W^t - (4 - (F_O^{-1})) \cdot T_P^t)$$

Equation 10 - Final Temperature Equation

This is the equation that forms the explicit finite difference approximation method. It is the explicit method as the temperatures at a previous timestep are used to explicitly calculate an unknown temperature value using the forward difference approximation. This is an accurate, reliable and relatively simple method however alpha and grid spacing values are fixed by the user, this means there will always be a maximum useable timestep to satisfy the stability criterion (Davis, 2010).

The implicit method uses the backward difference approximation. This means more equations are required to be solved simultaneously to calculate the temperature diffusion gradient. Moreover, the implicit method allows for much larger timesteps to be used as the stability criterion does not have to be satisfied (Ran, 2012).

However, due to the explicit method only using one equation, it significantly reduces computing time and makes for a more coherent, neat code. It is considered to be sufficiently accurate to give a trustworthy solution (Davis, 2010).

3 Linear and Gaussian Boundary Conditions

3.1 Validation Boundary Conditions - Linear Decrease

Initially the author implemented boundary conditions starting at 50°C on the left hand side of the plate, linearly decreasing to zero on the right hand side along the top edge. The rest of the left and right edges, and bottom edges were set to zero. This was a way of validating the mathematics used within the script, as it is easy to predict how this heat transfer should look at equilibrium. Fundamentally the author expects the linear decrease to be mirrored along the bottom edge, with a line of symmetry running along the middle of the plate horizontally. If this is seen, the mathematics of the model is validated and can be trusted on more complex distributions.

50.0000	43.7500	37.5000	31.2500	25.0000	18.7500	12.5000	6.2500	0.0000
50.0000	43.7480	37.4964	31.2453	24.9949	18.7453	12.4964	6.2481	0.0000
50.0000	43.7472	37.4949	31.2433	24.9928	18.7433	12.4949	6.2473	0.0000
50.0000	43.7480	37.4964	31.2453	24.9949	18.7453	12.4964	6.2481	0.0000
50.0000	43.7500	37.5000	31.2500	25.0000	18.7500	12.5000	6.2500	0.0000

Figure 4 - Validated linear decrease heat transfer matrix

3.2 Assignment Boundary Conditions - Gaussian Curve

For this report, the given boundary conditions were as follows:

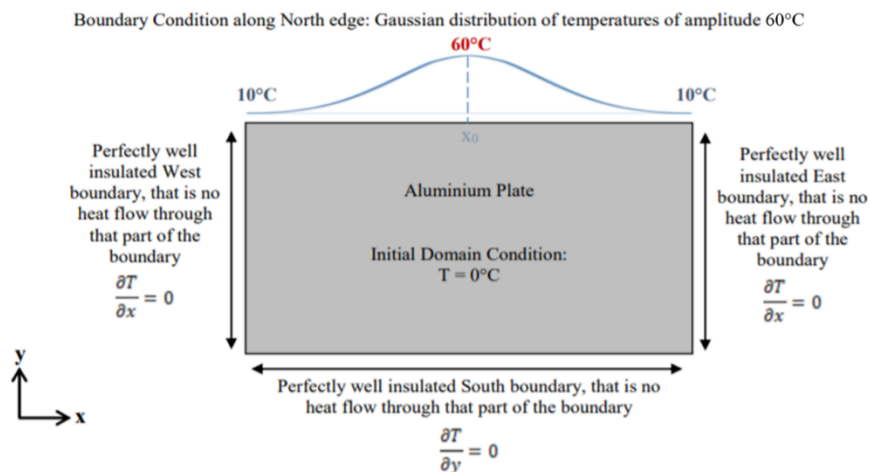


Figure 5 - Initial Boundary Conditions - (Marotin, 2019)

As figure five shows, the boundary conditions are the same as in section 3.1, apart from the top edge which ranges between 10°C and 60°C, its shape characterised by a Gaussian function. This shape follows the trends of a symmetrical, normal distribution bell curve defined by:

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

Equation 11 - Gaussian Distribution Equation

Where T = temperature, x = current column, x_0 = amplitude location, A = amplitude (60°C),
 σ = standard deviation

The author rearranged this equation to find the standard deviation required to find the correct distribution:

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

Equation 12 - Standard Deviation Equation (Penn State, 2006)

The Gaussian curve simulates a single heat source in the centre of the plate, with heat propagating to the far edges and down to the bottom. After the user has defined the plate dimensions (explained below in section 4.3), the author could implement the boundary conditions to the model:

```
plate = zeros([y_points x_points]);
x_0 = ((x_points+1)/2);
Sd = sqrt((- (x_points-x_0)^2) / (2*(log(t1/t2))));

temp_count = 1;
while temp_count <= x_points

    plate(1,temp_count) = (t2*exp(-(temp_count-x_0)^2) / (2*(Sd^2))));

    temp_count = temp_count + 1;

end
```

Initially, a correctly sized matrix of zeros is produced. Next, the x-axis mid-point is found (x_0), then the standard deviation equation is implemented. From here a while loop can be formed to calculate each value in the top row of the matrix using the original Gaussian equation.

4 Script Details

The following script details apply to the Gaussian distribution and unless stated otherwise are the same for the linear distribution.

4.1 User Script Selection

This model contains three MATLAB scripts. User selection, linear distribution and Gaussian distribution. This makes for a more user-friendly experience as the mathematics behind the model do not have to be accessed. The user selection script is run, and a message box appears to ask the user to select which model to use. Details of this can be found in appendix 1. Details of how this is implemented within the app can be found in section 6.

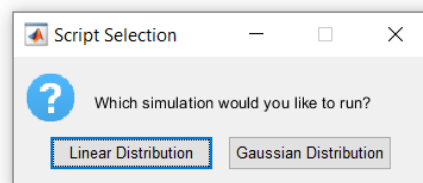


Figure 6 - User script selection box

4.2 Defining Constants

The script initially defines the required constants for the aluminium plate:

```
k = 143;           % Aluminium Conductivity (W/m.C)
p = 2.8e3;         % Aluminium Density (kg/m^3)
c = 795;           % Aluminium Specific heat (J/kg.C)

t1 = 10;           % Start Temperature (GAUSSIAN) (C)
t2 = 60;           % Highest Temperature (GAUSSIAN) (C)

alpha = k / (p*c); % Thermal diffusivity (m^2/s)
```

The start and end temperatures are defined in the brief, thus are taken as constants.

4.3 User Input and Validation

To run the script, the user must first input the X and Y dimensions of the aluminium plate:

```
x_dim = input(' X-axis dimension (m) = ');
while isempty(x_dim) == 1
    warning '* X-axis dimension undefined. Please re-enter *'
    clear x_dim
    x_dim = input(' X-axis dimension (m) = ');
end
```

The above script allows the user to define the x dimensions. As with all user inputs in this script, a while loop is used to ensure the user does not press enter without inputting a value, as this would stop the model from working. A while loop is used as this means the script cannot progress until a suitable value is entered. More input validation can be seen within the script, in appendix 1.

The model then asks the user to input grid spacing and timestep. The script checks that the grid spacing is not greater than or equal to the plate dimensions, as this would either not work, or output just boundary conditions. It also ensures grid spacing is greater than zero.

```
while grid_space >= y_dim || grid_space >= x_dim || grid_space <= 0
    warning '* Grid Spacing must be less than X and Y dimensions *'
    clear grid_space
    grid_space = input(' Please re-enter grid spacing (m) = ');
end
```

The script then ensures the grid spacing chosen is a factor of both x and y dimensions, as if not the model will not work. X and Y points calculate the rows and columns (+1 to the value to include starting at zero). Next, the script rounds the X and Y points ratio is an integer, if so, the grid spacing is a factor and can be used. The while loop prompts the user to re-enter if incorrect.

```
x_points = (x_dim/grid_space + 1);
y_points = (y_dim/grid_space + 1);
x_check = x_points == round(x_points);
y_check = y_points == round(y_points);

while x_check == 0 || y_check == 0
    warning '* Incompatible. Please choose a factor of both X & Y dimensions *'
    clear grid_space & x_points & y_points & x_check & y_check
    grid_space = input(' Pleasere-enter grid spacing (m) = ');
    x_points = (x_dim/grid_space + 1);
    y_points = (y_dim/grid_space + 1);
    x_check = x_points == round(x_points);
    y_check = y_points == round(y_points);
end
```

With the correct dimensions and grid spacing defined, the user can input a timestep. The script calculates the maximum permissible timestep by rearranging the Fourier number inequality and displays this to the user.

```
max_time_step = ((grid_space^2)/(4*alpha));
fprintf('Time step must be between 0 and %7.5f seconds',max_time_step)
```

The 'fprintf' function allows a variable to be displayed to the user, imbedded in text. '%7.5f' displays seven significant figures, to five decimal places.

4.4 Fourier Number Calculation and Validation

Initially, the script calculates the Fourier number value for the user's inputs using the equation in section 2.2.5. Following this, the value is checked against the stability criterion:

```
while Fo >= (1/4) || Fo < 0
    warning '* Stability criterion broken. Please try different timestep value *'
    fprintf('Time step must be between 0 and %7.5f seconds',max_time_step)
    clear time_step & Fo
    time_step = input(' Please re-enter time step (s) = ');
    Fo = (alpha*time_step) / (grid_space^2);
end
```

The while loop checks the value, displays a warning if not satisfied, reminds the user of the maximum permissible timestep, clears incorrect values then allows them to be correctly re-defined.

4.5 Heat Transfer Modelling Matrix

After an initial boundary condition matrix is created under variable 'plate', as explained in section 3.2, this is assigned to 'plate_old', from which the plate of the next timestep's values can be calculated. By implementing the explicit finite difference approximation (equation 10), two while loops can calculate the interior values of the matrix:

```
while column < x_points
    row = 2;
    while row < y_points
        Tp = plate(row,column);
        plate(row,column) = Fo * (plate_old(row - 1,column) + ...
            plate_old(row,column + 1) + plate_old(row + 1,column) + ...
            plate_old(row,column - 1) - (4 - (1/Fo)) * Tp);

        row = row + 1;
    end

    column = column + 1;
end
```

where the colours represent north - east - south - west.

The boundary conditions of the Gaussian plate include undefined edge temperatures; this means the equation cannot be implemented as one of the four directions required will be missing. To solve this, the boundary conditions state that the outside edges are perfectly well insulated, therefore, no heat transfer occurs. Because of this, the equation can be adapted so that for the left and right edge west = east, for the bottom edge south = north, and for the bottom left and bottom right nodes west = east and south = north (Széchenyi István University, 2009). For example, to calculate the bottom edge values:

```
row = y_points;
column = 2;
while column < x_points
    Tp = plate(row,column);
    plate(row,column) = Fo * (plate_old(row - 1,column) + ...
        plate_old(row,column + 1) + plate_old(row - 1,column) + ...
        plate_old(row,column - 1) - (4 - (1/Fo)) * Tp);

    column = column + 1;
end
```

4.6 Stop Criterion

The script has to stop the heat transfer simulation when thermal equilibrium is reached, or as close to it as possible. To establish this point, the difference between each node within the matrix at adjacent timesteps is compared. At thermal equilibrium, no change will be seen. To implement this the script creates a second correctly sized matrix of zeros and sets the top, boundary condition nodes to 1's. Another matrix of equal dimensions is also created containing all 1's. Finally a variable 'stop_criterion_comparison' is defined at 0.9999999. After some trial and error testing, the author found this to give a good balance between an accurate value and sufficiently small computing time. This is changeable by the user within the app, as detailed in section 6.

```
stop_checker = zeros([y_points x_points]);
stop_checker(1,:) = 1;
exit_condition = ones([y_points x_points]);

stop_criterion_comparison = 0.9999999;
```

At the end of each node calculation, the current 'plate', value is compared to the previous, 'plate_old' value to see if they are within tolerance of the stop criterion comparison.

```
stop_criterion = plate_old(row,column) / plate(row,column);

if stop_criterion > stop_criterion_comparison
    stop_checker(row,column) = 1;
end
```

When the if statement is satisfied for the current node, a 1 is placed in the stop_checker matrix at the corresponding location. Once the sum of stop_checker and exit_condition are equal, the script stops calculating. This is controlled by every temperature calculation placed within one while loop:

```
while sum(stop_checker, 'all') < sum(exit_condition, 'all')

    * temperature calculations - see appendix 1 *

end
```

4.7 Heat Transfer Matrix Graphical Display

As well as viewing the matrix of nodal temperatures, the user can also evaluate the results through a contour plot. Before the calculations begin, the user is asked whether to view a live plot throughout the calculation, or a plot of just the final plate:

```
plot_display = questdlg(...
    'Would you like to see a live plot of the simulation? ', 'Yes','No');
switch plot_display
    case 'Yes'
        plot_display = 1;
    case 'No'
        plot_display = 0;
    case 'Cancel'
        plot_display = 0;
end
```

This displays the question to the user with the three possible answers. The code defines the plot_display variable according to each answer, to be implemented into an if statement later in the code. If 'no' is chosen, the user only sees a plot of the final plate.

At the start of the temperature calculation equations, the variable 'iteration_counter_plot' is defined as 1. This counts how many plates are being calculated throughout the calculation. Once the current plate's interior and exterior values have been calculated, 1 is added to this value, this is used for the live plot. If the matrix was plotted every iteration, when a small timestep was chosen the plot would take an extremely long time to complete. To shorten this time, the author's code analyses the size of the timestep chosen by the user by comparing it to the maximum permissible timestep. From here, the code plots the matrix every given amount of calculations, resetting the counter after each plot.

A trial and error experiment was carried out to find a good value for each category:

Table 1 - Plot iteration calculations 1

X = 2m Y = 1m h = 0.1m max Δt = 38.92s		
Δt (s)	(Δt / max Δt)	Total number of plates
1	0.0257	48447
10	0.2570	6298
20	0.5139	3366
30	0.7709	2328
38	0.9765	1877

Table 2 - Plot iteration calculations 2

X = 8m Y = 6m h = 0.4m max Δt = 622.66s		
Δt (s)	(Δt / max Δt)	Total number of plates
10	0.0161	144152
100	0.1606	19643
250	0.4015	8689
500	0.8030	4658
600	0.9636	3950

From this, a suitable figure for each range could be tested and implemented. It was found that the following work best:

```
if(time_step/max_time_step) > 0 && (time_step/max_time_step) <= 0.1
    plot_iteration = 1000;

elseif(time_step/max_time_step) > 0.1 && (time_step/max_time_step) <= 0.5
    plot_iteration = 100;

elseif(time_step/max_time_step) > 0.5 && (time_step/max_time_step) <= 1
    plot_iteration = 10;

end
```

From here, an if statement checks if the user selected to see a live plot, and if the iteration is correct to plot:

```
if plot_display == 1 && iteration_counter_plot == plot_iteration

* plotting code - see appendix 1 *

End
```

At the end of the calculations and plots, the script displays a dialogue box prompting the user to view the final plate matrix and displaying real time taken until thermal equilibrium. This is calculated by multiplying the amount of plates (variable 'iteration_counter_time' multiplied by timestep):

```
total_time = ((iteration_counter_time)*(time_step))/(3600);

end_msg = sprintf...
    ('Simulation complete. Time until thermal equilibrium: %4.2f hours.'...
,total_time);
end_msg = [end_msg newline 'Open variable "plate" to view matrix.'];
end_msgbox = msgbox(end_msg);
```

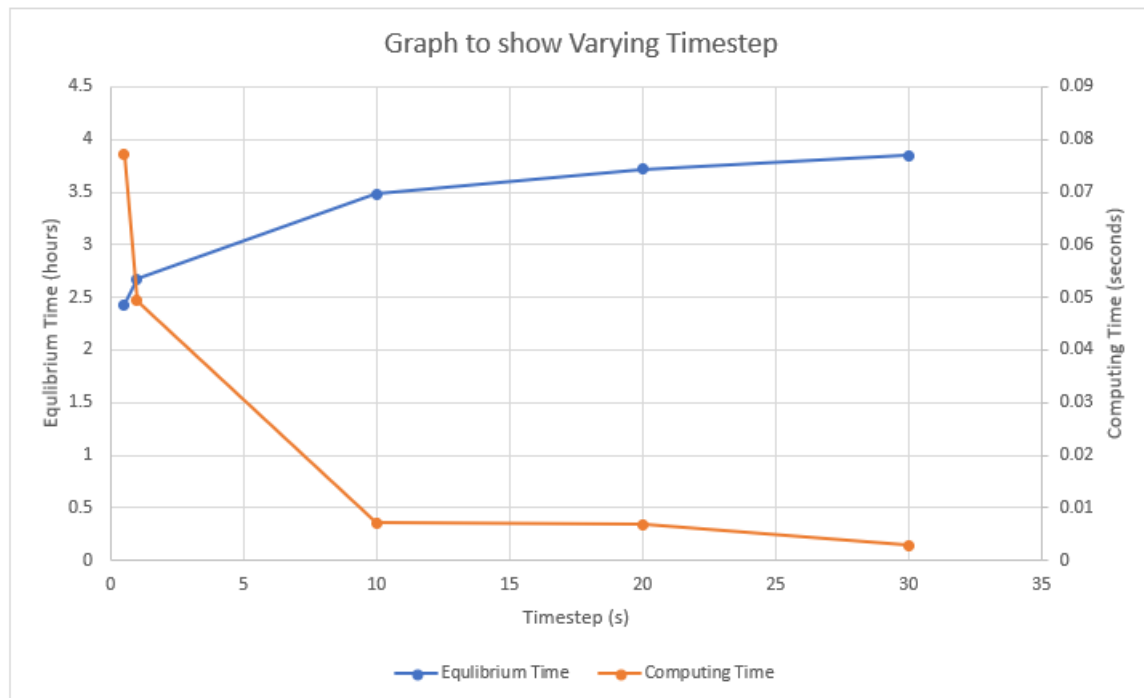
5 Evaluation - Initial Linear Code Results and Analysis

Because the linear boundary conditions were only for validation, results shall be analysed from the Gaussian boundary conditions set in the assignment specification. For each test, to calculate computing time, the MATLAB 'tic - toc' feature was used, with 'tic' to start counting just before the heat calculations and 'toc' finishing the counting just after the final calculation. Each test was run twice, and the average computing time calculated.

5.1 Varying Timestep - Linear Boundary Conditions

Table 3 - Results: Varying Timestep

X = 2m Y = 1m h = 0.1m				
Test	Δt (s)	Fo	Time to equilibrium (h)	Computing time (s)
1	0.5	$3.21e^{-3}$	2.43	0.0772
2	1	$6.42e^{-3}$	2.67	0.0493
3	10	0.06	3.48	0.0071
4	20	0.13	3.72	0.0068
5	30	0.19	3.85	0.0028



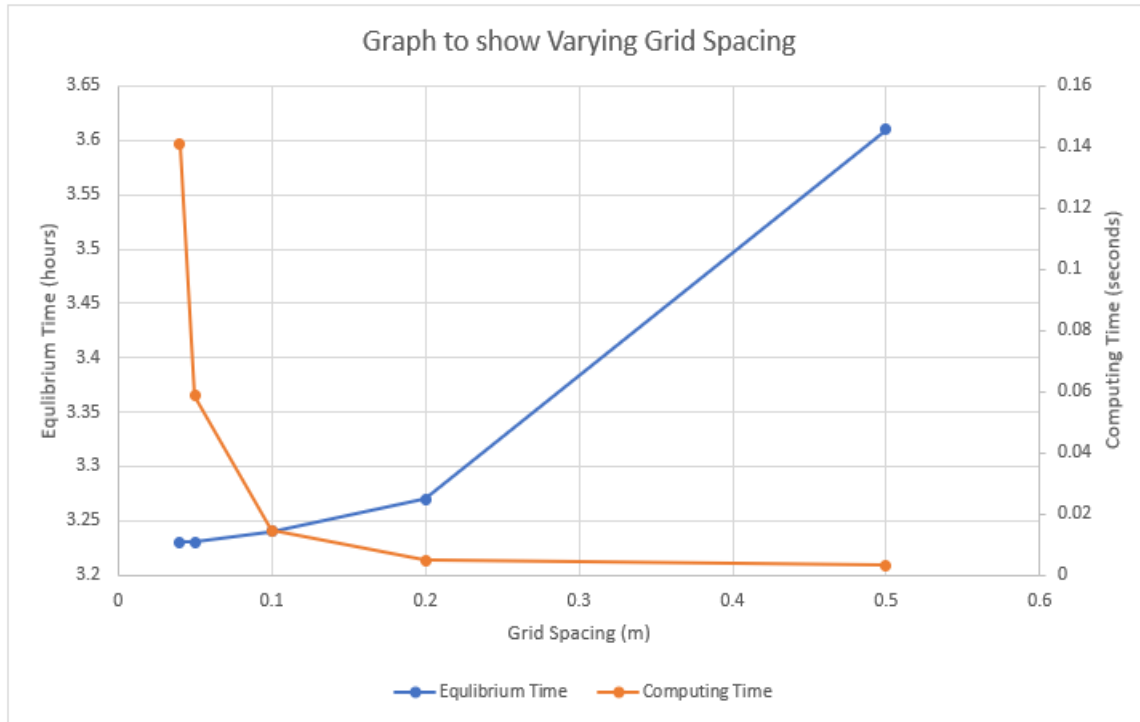
Equation 13 - Graph of Varying Timestep

- As the timestep increases, so does time to equilibrium, computing time decreases.
- Significant change in time to reach equilibrium - shows timestep is a key factor in accuracy of model
- Increasing Fourier number shows higher time to equilibrium.
- It is presumed that the lower equilibrium times are more accurate, as a lower timestep is a more comprehensive analysis of the problem with closer, thus more accurate calculations. Therefore showing an increased timestep decreases accuracy.

5.2 Varying Grid Spacing - Linear Boundary Conditions

Table 4 - Results: Varying Grid Spacing

X = 2m Y = 1m Δt = 5s				
Test	h (m)	Fo	Time to equilibrium (h)	Computing time (s)
1	0.04	0.200	3.23	0.1413
2	0.05	0.128	3.23	0.0586
3	0.1	0.032	3.24	0.0145
4	0.2	0.008	3.27	0.0049
5	0.5	0.002	3.61	0.0031



Equation 14 - Graph of Varying Grid Spacing

- As grid spacing increases, the time to equilibrium increases, computing time decreases.
- Smaller change in time to equilibrium - showing grid spacing is significant but not as key as timestep in accuracy of model.
- Here a lower Fourier number shows higher time to equilibrium, this disagrees with section 5.1 thus showing Fourier number and accuracy cannot be directly correlated, rather it influences model accuracy as a result of timestep and grid spacing change.
- Drawing same conclusions as section 5.1, lower equilibrium time is likely more accurate therefore increased grid spacing decreases accuracy.

5.3 Constant Fourier Number - Gaussian Boundary Conditions

Table 5 - Results: Constant Fourier Number

X = 8 Y = 6 Fo = 0.1				
Test	h (m)	Δt (s)	Time to equilibrium (h)	Computing time (s)
1	0.1	15.566	428.02	8.336
2	0.5	389.161	631.52	0.325
3	1.0	1556.643	720.81	0.339
4	2.0	6226.573	816.37	0.235

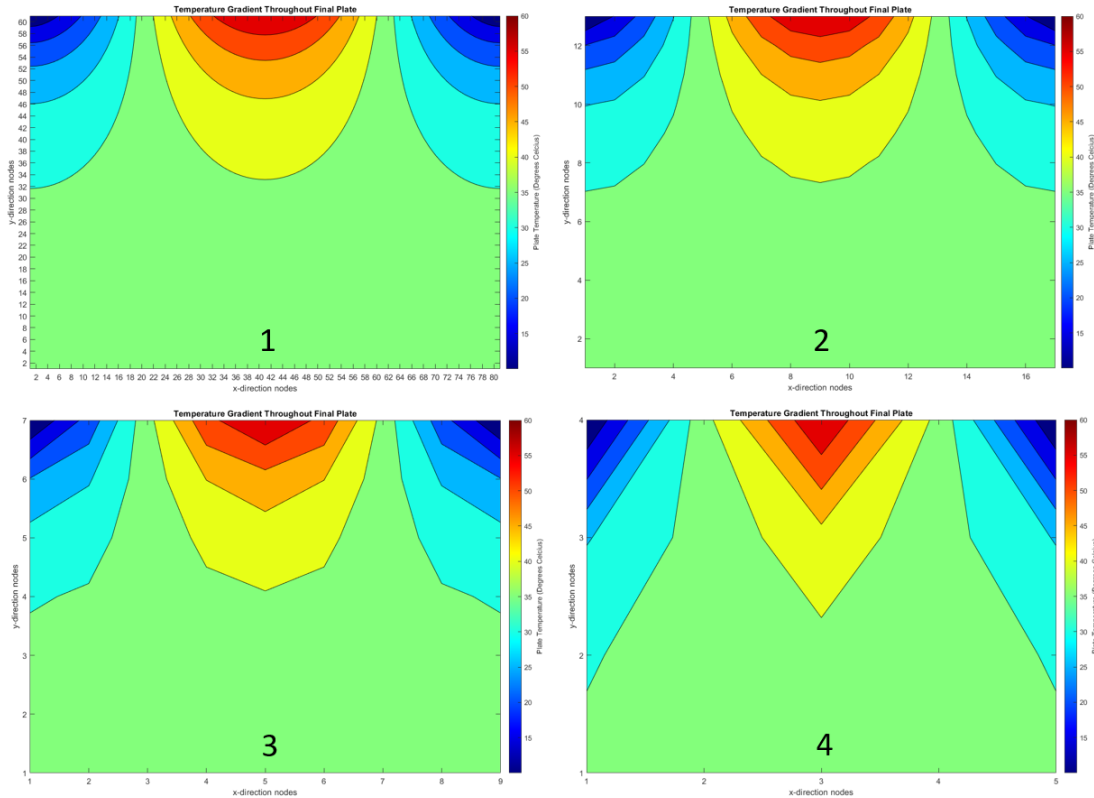


Figure 7 - Constant Fourier Number Graphical Results

- As per section 5.1, Larger grid spacing and timestep reduces accuracy as far less nodes to calculate from. From 5.1, its clear that timestep is the main contributor to this error. This is particularly clear in the gaussian plate.
- Picture 4 reached thermal equilibrium too early as yellow section (around 40°C) comes down too far - again showing inaccuracy.
- With a larger timestep, the point of equilibrium takes longer to reach, and the computing time decreases, agreeing with section 5.1 and 5.2.
- If grid spacing is too small, a very small timestep is required, thus the model will become inaccurate as too many calculations can interfere, causing thermal equilibrium to be reached too early (Causon and C.G., 2010). This will also increase computing time.
- If grid spacing is too large - it will take too long to reach thermal equilibrium, again showing inaccuracy.
- This shows there are negative impacts of a grid spacing too big or small, thus a figure in the middle should be chosen.

5.4 Increasing Fourier Number - Gaussian Boundary Conditions

Table 6 - Results: Increasing Grid Spacing

X = 8 Y = 6 h = 0.1				
Test	Fo (m)	Δt (s)	Time to equilibrium (h)	Computing time (s)
1	0.01	1.557	283.59	49.675
2	0.1	15.566	428.02	8.289
3	0.2	31.133	471.71	4.530
4	0.25	38.916	485.77	3.543

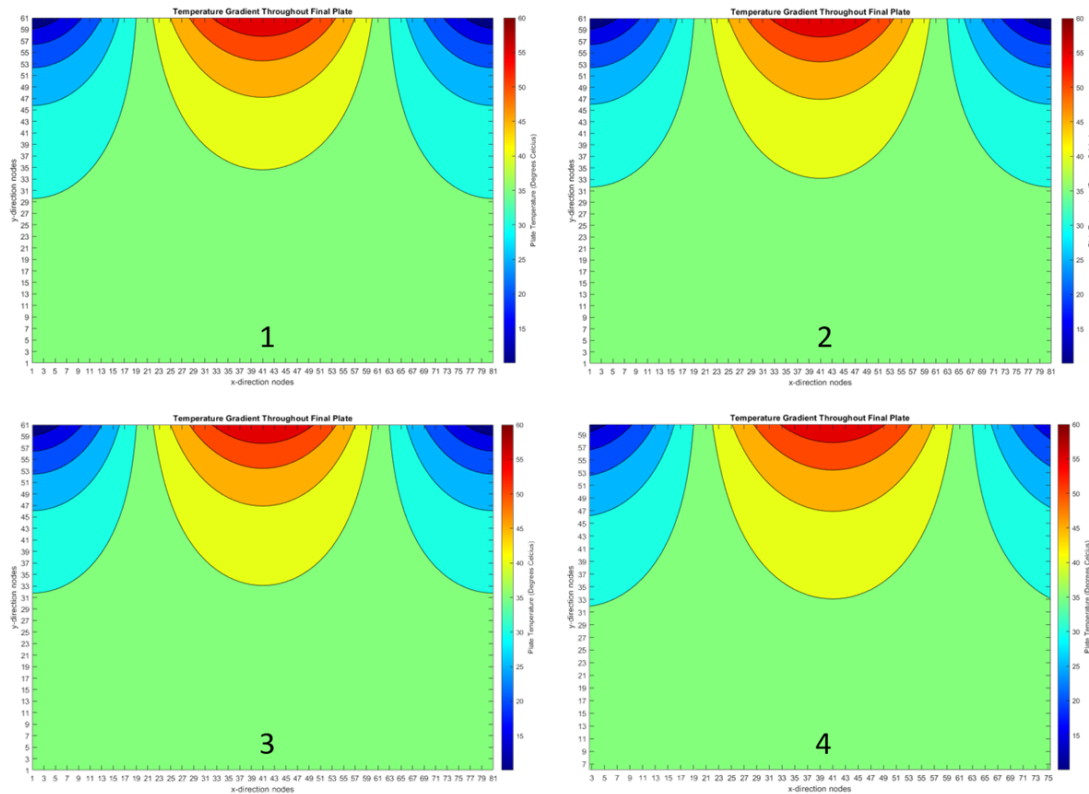


Figure 8 - Increasing Fourier Number Graphical Results

- When Fourier number = 0.1, the timestep is 15.566. This is a repeat of test 2 in section 5.3. The results are very similar, this proves consistency and validates results.
- With a constant grid spacing, changing Fourier number has no noticeable visual effect on reaching thermal equilibrium.
- A large increase in time to equilibrium is seen, which agrees with the findings of section 5.1, validating results.
- These tests show a higher Fourier number means higher equilibrium time and smaller solving time.

From these four tests, it is clear that too bigger timestep or too bigger grid spacing renders an inaccurately large equilibrium time. A larger time step means a bigger difference between node values in adjacent plates. This means the calculation is using more outdated data to calculate the current nodal value. This explains inaccuracy with a large timestep. If the timestep or grid spacing is incredibly small, the tolerance of the stop criterion (0.9999999 by default) may stop the calculation before equilibrium. Although this tolerance works in some situations, it may need to be altered for others.

5.5 Varying Stop Criterion Accuracy

Table 7 - Results: Varying Stop Criterion Accuracy

Test	Stop Criterion	Time to equilibrium (h)		Computing time (s)	
		Linear	Gaussian	Linear	Gaussian
1	0.999	0.33	0.63	0.040	0.651
2	0.999999	2.67	9.42	0.156	0.623
3	0.999999999	5.12	21.54	0.19	1.032
4	0.999999999999	7.55	33.67	0.236	1.264

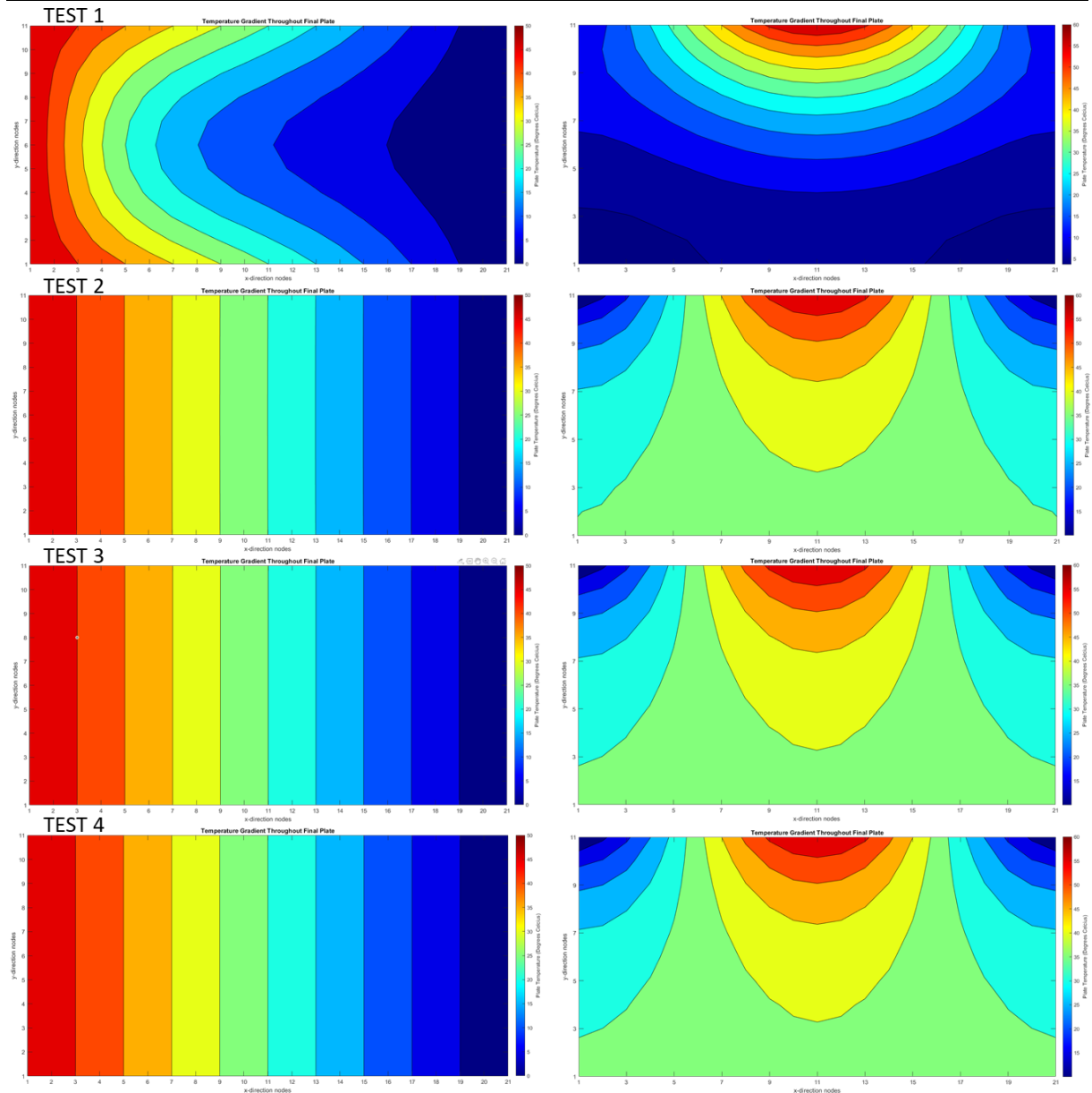


Figure 9 - Varying Stop Criterion Accuracy Graphical Results

The above results show the impact of varying the exit criterion. The lowest accuracy of convergence was set to 0.999 (99.9% accuracy). This could be set lower but would render the results completely inaccurate. This test shows that test one has reached thermal equilibrium 95% quicker than test four, figure 9 shows visually this is an inaccurate result. It is clear that visually, the linear plate is not affected as much as the Gaussian through tests 2-4.

The stop criterion should change depending on user input, as the user could then tune both result accuracy and computing time. As detailed in section 6, this was basically implemented into the app, but could be improved further.

6 Extra Work - Creating a MATLAB App

As has been mentioned through this report, the author undertook the extra task of creating a MATLAB App. This was created by selecting various user interface features and combining them with the existing code from the script. Details of each part are explained below. The specific script for the app is not included in this report but can be found within the app submitted on Turnitin.

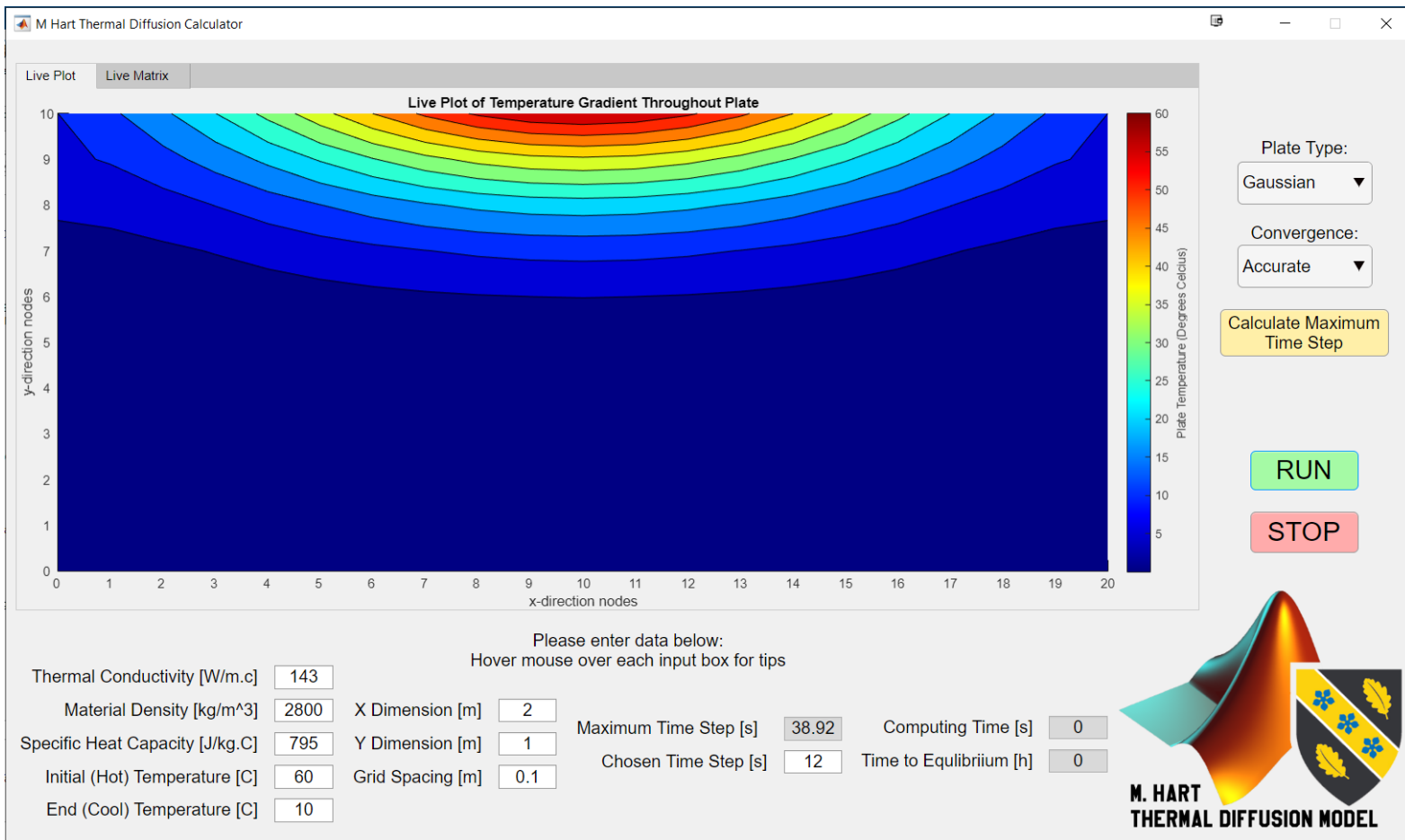


Figure 10 - Thermal Diffusion Model MATLAB App Interface

- Initially, global variables were assigned so they worked across each callback throughout the app.
- Fundamentally the only difference in coding within this app and the standard script is creating callback functions that are triggered when the user pushes a button, obtaining data from 'edit field' user input boxes, and obtaining data from a dropdown menu. An example of the former two is:

```
function CalculateMaximumTimeStepButtonPushed(app, event)    % Callback for button press

%% SETTING CONSTANTS
k = app.ThermalConductivityWmcEditField.Value;    % Conductivity from edit field box
p = app.MaterialDensitykgm3EditField.Value;    % Density from edit field box
c = app.SpecificHeatCapacityJkgCEditField.Value;    % Specific heat from edit field box
app.alpha = k / (p*c);    % Thermal diffusivity from edit field box

end
```

Here, function callback was created for when the user presses the 'calculate maximum time step' button. Data from user inputs were assigned to variables.

- An example of obtaining data from a drop-down menu is the user selecting different stop criterion values from the convergence drop-down selection:

```
%% SETTING STOP CRITERION VALUES
    if app.ConvergenceDropDown.Value == 'A'    % Sets stop criterion to accurate
        stop_criterion_comparison = 0.9999999;
    end

    if app.ConvergenceDropDown.Value == 'F'    % Sets stop criterion to fast
        stop_criterion_comparison = 0.8;
    end
```

- A summary of all app-specific code is:
 - Defining global variables
 - Creating button callback functions
 - Defining variables from edit field boxes
 - Selecting stop criterion from drop-down selection
 - Selecting linear or Gaussian script from drop-down selection
 - Ensuring user has not selected 0°C in Gaussian plate
 - Sending plot to app axes with 'app.UIAxes' command
 - Sending matrix to app table with 'app.UITable' command

Examples of all the above can be found in the code embedded within the app.

The app interface was created within the design view section of the app creator. The author could drag and drop each component into place, aligned with a grid system. Position and size of the workspace and each component could be assigned through a coordinate system. A custom logo was placed within the workspace and colours and formatting applied. The tool tip function was used so the user can hover the mouse over each component to be told what it does.

There are some errors and room for refinement within the app, namely the stop button which still requires the user to press Ctrl+C to stop the app, however as a first attempt this application works well, and produces reliable results. The convergence selection could also be refined, with a slider to allow the user to choose a larger variety of accuracy options. These improvements will be further researched and can be applied to version 2 in the future.

7 Code Limitations, Analysis and Improvements

The stop criterion within this code could be a source of error. By default, the node values between plates are assessed to see if they are within 0.9999999 of each other. This was chosen as it gives a consistently accurate solution for most inputs. This value, however, will always be a compromise. If the value was too small, the simulation would either take far too long to compute, or never stop itself due to errors within the calculations preventing thermal equilibrium ever actually happening; equally, if this value is too large, the model will reach early thermal equilibrium which is again inaccurate. As previously mentioned, this value was found through trial and error and is a compromise between these two situations. Moreover, if a very small timestep or grid spacing value is chosen, the default tolerance may be too large to give an accurate answer. To improve the script, the author should research how to allow the script to assess the relative size of the timestep or grid spacing and change the stop criterion tolerance accordingly, or allow user-input, like in the app.

A main source of error within this model comes from the fact the calculations are based on approximations. The error within the Taylor series expansion used lies within the value of h , as defined in section 2. With the central difference method, the error is to the order of grid spacing squared ($O(h^2)$). This means that if the grid spacing value is below one, the error gets smaller as it is being multiplied by less than one. Equally, if the grid spacing is above one, the error will increase exponentially. Because of this, the higher the grid spacing, the less accurate the result. However, it must also be noted that a very small grid spacing will mean many more calculations. Although a smaller error, they exist in much greater quantities and add up to make a large overall error, known as a truncation error (Langtangen, 2014).

This is also apparent in the forward difference method. Here the error lies within the timestep, and it is to an order of grid spacing ($O(h)$). As this is not squared, it is often a larger source of error than in the central difference method, and can be a contributing factor to inaccuracy within the model (Smolík *et al.*, 2018)

8 Referenced Work

Berciu, M. (2007) *Heat equation in a 2D rectangle*. Vancouver. Available at: <https://www.phas.ubc.ca/~berciu/TEACHING/PHYS312/LECTURES/FILES/2Dwave.pdf> (Accessed: 18 December 2019).

Causon, D. M. and C.G., M. (2010) *Introductory Finite Difference Methods for PDEs*. Ventus Publishing ApS. Available at: <http://www.cs.man.ac.uk/~fumie/tmp/introductory-finite-difference-methods-for-pdes.pdf> (Accessed: 18 December 2019).

Croft, D. R. and Stone, J. A. . (1989) *Heat transfer calculations using finite difference equations*. PAVIC Publications.

Davis, M. (2010) *Finite Difference Methods -Implicit and Explicit Comparison*. London. Available at: http://wwwf.imperial.ac.uk/~mdavis/FDM11/LECTURE_SLIDES2.PDF (Accessed: 18 December 2019).

Langtangen, H. P. (2014) *Truncation Error Analysis*. Available at: http://hplgit.github.io/INF5620/doc/pub/H14/trunc/html/_main_trunc000.html (Accessed: 23 December 2019).

Lewis, R. W. *et al.* (1996) *The Finite Element Method in Heat Transfer Analysis*. Chichester: John Wiley & Sons.

Magyar, P. (2003) *Taylor Series*. Lansing. Available at: <https://users.math.msu.edu/users/magyar/Math133/11.10-Taylor-Series.pdf> (Accessed: 18 December 2019).

Marotin, A. (2019) 'Assignment Specification'. University of Wales Trinity St. David.

Pei-baiZhou (2013) *Finite Difference Method*. Sian: Xi'an Jiao-tong University. Available at: https://link.springer.com/chapter/10.1007/978-3-642-50319-1_3 (Accessed: 18 December 2019).

Penn State (2006) *Probability Distributions - The Gaussian Distribution*. Available at: <http://www.cse.psu.edu/~rtc12/CSE586Spring2010/papers/prmlGaussian.pdf> (Accessed: 19 December 2019).

Pupeikis, D., Šeduikyte, L. and Bruzgevičius, P. (2012) 'The impact of the fourier number on calculation accuracy of transient heat transfer in homogeneous enclosures', *Medziagotyra*, 18(3). doi: 10.5755/j01.ms.18.3.2440.

Ran, A. (2012) *The implicit finite-difference method in matrix terms*. Amsterdam. Available at: <https://www.cs.vu.nl/~ran/LUmethod.pdf> (Accessed: 18 December 2019).

Salazar, A. (2003) *Thermal Diffusivity*. Bilbao. Available at:
https://www.researchgate.net/publication/231038795_On_thermal_diffusivity (Accessed: 18 December 2019).

Smolík, L. *et al.* (2018) *Limitations of finite difference methods in the computation of coupling forces prescribed by the Reynolds equation*. Pilsen, Czech Republic. Available at:
https://www.researchgate.net/publication/327393687_Limitations_of_finite_difference_methods_in_the_computation_of_coupling_forces_prescribed_by_the_Reynolds_equation (Accessed: 23 December 2019).

Széchenyi István University (2009) *FINITE DIFFERENCE METHODS Solution of the Second Order Differential Equations using Finite Difference Method*. Győr, Hungary. Available at:
<http://www.sze.hu/~lotfi/FINITE-DIFFERENCE-METHODS.pdf> (Accessed: 18 December 2019).

Tan, J. and Li, M. (2018) 'Solutions of fractional differential equations with p-Laplacian operator in Banach spaces', 2018. doi: 10.1186/s13661-018-0930-1.

9 Appendices

9.1 Appendix 1a - User Selection Script

```
1 - clear
2 - clc
3
4 - %      *** PLEASE RUN SCRIPT TO START SIMULATION ***
5
6 - choose_script = questdlg(' Which simulation would you like to run? ',... % Asks user which thermal diffusion model to simulate
7 -     'Script Selection','Linear Distribution','Gaussian Distribution',... % Sets question, Sets dialogue box title, Sets option 1, Sets option 2, Sets default option if user presses Enter
8 -     'Linear Distribution');
9 - switch choose_script
10 -     case 'Linear Distribution' % Assigns value if user chooses 'Linear'
11 -         choose_script = 0;
12 -     case 'Gaussian Distribution' % Assigns value if user chooses 'Gaussian'
13 -         choose_script = 1;
14 - end
15
16 - if choose_script == 0 % Assesses whether user wants to see Linear model
17 -     Thermal_Diffusion_Model_Linear
18 - elseif choose_script == 1 % Assesses whether user wants to see Gaussian model
19 -     Thermal_Diffusion_Model_Gaussian
20 - end
```

9.2 Appendix 1b - Linear Plate Script

<pre> 1 - clc, clear 2 - 3 - %% CONSTANTS 4 - 5 - k = 143; 6 - p = 2.8e3; 7 - c = 795; 8 - 9 - t1 = 50; 10 - t2 = 0; 11 - 12 - alpha = k / (p*c); 13 - 14 - %% USER INPUTS & VALIDATION 15 - 16 - % PLATE SIZE INPUT AND VALIDATION 17 - disp(' Welcome. Please enter the required information below: ') 18 - 19 - x_dim = input(' X-axis dimension (m) = '); 20 - while isempty(x_dim) == 1 21 - warning '* X-axis dimension undefined. Please re-enter *' 22 - clear x_dim 23 - x_dim = input(' X-axis dimension (m) = '); 24 - end 25 - 26 - y_dim = input(' Y-axis dimension (m) = '); 27 - while isempty(y_dim) == 1 28 - warning '* Y-axis dimension undefined. Please re-enter *' 29 - clear y_dim 30 - y_dim = input(' Y-axis dimension (m) = '); 31 - end 32 - 33 - % GRID SPACING INPUT AND VALIDATION 34 - grid_space = input(' Grid spacing (m) = '); 35 - while isempty(grid_space) == 1 36 - warning '* Grid spacing undefined. Please re-enter *' 37 - clear grid_space 38 - grid_space = input(' Grid spacing (m) = '); 39 - end 40 - 41 - while grid_space >= y_dim grid_space >= x_dim grid_space <= 0 42 - warning '* Grid Spacing must be less than X and Y dimensions & > 0 *' 43 - clear grid_space 44 - grid_space = input(' Please re-enter grid spacing (m) = '); 45 - end </pre>	<pre> % Aluminium Conductivity (W/m.C) % Aluminium Density (kg/m^3) % Aluminium Specific heat (J/kg.C) % Start Temperature (C) % End Temperature (C) % Thermal diffusivity (m^2/s) % X axis user input % Checks if value assigned to X-axis % Clears incorrect variable % X axis user re-input % Y axis user input % Checks if value assigned to Y-axis % Clears incorrect variable % Y axis user re-input % Grid spacing user input % Checks if value assigned to grid spacing % Clears incorrect variable % Grid spacing user re-input % Ensures grid spacing is not greater than or equal to dimensions - '>' = no values, '=' = display of just boundary conditions - not correct % Clears incorrect variable % Grid spacing user re-input </pre>
--	---

```

46
47 % CREATING ROWS AND COLUMNS TO CHECK GRID SPACING VALUE
48 x_points = (x_dim/grid_space + 1);
49 y_points = (y_dim/grid_space + 1);
50 x_check = x_points == round(x_points);
51 y_check = y_points == round(y_points);
52
53 while x_check == 0 || y_check == 0
54     warning '* Grid spacing is incompatible. Please choose a factor of both X & Y dimensions *'
55     clear grid_space & x_points & y_points & x_check & y_check
56     grid_space = input(' Please re-enter grid spacing (m) = ');
57     x_points = (x_dim/grid_space + 1);
58     y_points = (y_dim/grid_space + 1);
59     x_check = x_points == round(x_points);
60     y_check = y_points == round(y_points);
61 end
62
63 % TIME STEP INPUT AND VALIDATION
64 disp ' '
65 max_time_step = ((grid_space^2)/(4*alpha));
66 fprintf('Time step must be between 0 and %7.5f seconds',max_time_step)
67 disp ' '
68
69 time_step = input(' Time step (s) = ');
70 while isempty(time_step) == 1
71     warning '* Time step undefined. Please re-enter *'
72     clear time_step
73     time_step = input(' Time step (s) = ');
74 end
75
76 %% FOURIER VALIDATION
77
78 Fo = (alpha*time_step) / (grid_space^2);
79
80 while Fo >= (1/4) || Fo < 0
81     warning '* Stability criterion broken. Please try different timestep value *'
82     fprintf('Time step must be between 0 and %7.5f seconds',max_time_step)
83     disp ' '
84     clear time_step & Fo
85     time_step = input(' Please re-enter time step (s) = ');
86     Fo = (alpha*time_step) / (grid_space^2);
87 end
88

```

```

% Calculates number of columns (+1 to include starting at zero)
% Calculates number of rows (+1 to include starting at zero)
% Checks grid space x ratio is an integer (checks it's a factor)
% Checks grid space y ratio is an integer (checks it's a factor)

% Checks for integers, replace values if needed
% Clears user's old, incorrect values
% Grid spacing user re-input
% Re-calculates number of columns (+1 to include starting at zero)
% Re-calculates number of rows (+1 to include starting at zero)
% Re-checks new grid spacing value against X dimension
% Re-checks new grid spacing value against Y dimension

% Displays a space between lines - easier to read for user
% Calculates maximum time step
% Displays maximum time step to user

% Time step user input
% Checks if value assigned to time step

% Clears incorrect variable
% Time step user re-input

```

```

% Fourier number (dimensionless)

% Checking stability criterion, tells user what maximum values are

```

89	%% USER RESULTS PREFERENCE	
90		
91	plot_display = questdlg(...	% Asks the user if live plot should be displayed via dialogue box
92	' Would you like to see a live plot of the simulation? ', 'Yes','No');	
93	switch plot_display	
94	case 'Yes'	% Assigns value = True if user chooses 'yes'
95	plot_display = 1;	
96	case 'No'	% Assigns value = False if user chooses 'No'
97	plot_display = 0;	
98	case 'Cancel'	% Assigns value = False if user chooses 'Cancel'
99	plot_display = 0;	
100	end	
101	%% CREATING HEAT TRANSFER MATRIX	
102		
103		
104	plate = zeros([y_points,x_points]);	% initially makes correct size matrix of zeros
105		
106	boundary = linspace (t1,t2,x_points);	% Defines linear gradient from t1 to t2
107	plate(1,:) = boundary;	% Defines first row boundary conditions
108	plate(end,:) = boundary;	% Defines end row boundary conditions
109	plate(:,1) = t1;	% Defines first column boundary conditions
110		
111	%% SIMULATING TIME-STEP HEAT TRANSFER MATRIX	
112		
113	stop_checker = ones([y_points x_points]);	% Creates initial convergence checking matrix (referred to as 'stop criterion')
114	stop_checker(2:end-1,2:end-1) = 0;	% Sets middle domain values to zero, so while loop can change them as simulation runs
115	exit_condition = ones([y_points x_points]);	% Sets the exit conditon to be when stop cheker matrix is all 1's
116		
117	stop_criterion_comparison = 0.9999999;	% Sets accuracy of stop criterion
118	stop_criterion = 0;	% Sets initial stop criterion value
119	iteration_counter_time = 1;	% Counts how many plates used in calculation - for time
120	iteration_counter_plot = 1;	% Counts how many plates used in calculation - for plot
121		
122	if(time_step/max_time_step) >0 && (time_step/max_time_step) <=0.1	% If very small timestep is chosen, graph will be plotted every 10000 iterations
123	plot_iteration = 100;	
124	elseif(time_step/max_time_step) >0.1 && (time_step/max_time_step) <=0.5	% If medium timestep is chosen, graph will be plotted every 1000 iterations
125	plot_iteration = 10;	
126	elseif(time_step/max_time_step) >0.5 && (time_step/max_time_step) <=1	% If large timestep is chosen, graph will be plotted every 100 iterations
127	plot_iteration = 10;	
128	end	
129		
130	tic	% Starts counting computing time

132 -	while sum(stop_checker, 'all') < sum(exit_condition, 'all')	% Sets calculation to stop when stop checker = all 1's
133 -	plate_old = plate;	% Saves previously calculated plate as 'old plate', to calculate new values from
134 -	iteration_counter_time = (iteration_counter_time + 1);	% Counts how many plates used in calculation - for time
135 -	iteration_counter_plot = (iteration_counter_plot + 1);	% Counts how many plates used in calculation - for plot
136 -	column = 2;	% Sets start point at column 2, as column 1 values are already set
137 -	while column < x_points	% Calculates all columns through to end
138		
139 -	row = 2;	% Sets start point at row 2, as row 1 values are already set
140 -	while row < y_points	% Calculates all rows through to end
141		
142 -	Tp = plate(row,column);	% Sets start node for each iteration
143 -	plate(row,column) = Fo * (plate_old(row + 1,column) + ...	% Equation to find adjacent nodes (N,E,S,W)
144	plate_old(row,column + 1) + plate_old(row - 1,column) + ...	
145	plate_old(row,column - 1) - (4 - (1/Fo)) * Tp);	
146		
147 -	stop_criterion = plate_old(row,column) / plate(row,column);	% Calculates change of values in each node between old plate and current plate - the closer the values get, the closer this variable gets to one.
148 -	if stop_criterion > stop_criterion_comparison	% Assesses if change in values is greater than chosen acceptable limit
149 -	stop_checker(row,column) = 1;	% Sets current node, if above acceptable limit, to 1 in stop checker matrix
150 -	end	
151 -	row = row + 1;	% Moves calculation to next column
152 -	end	
153 -	column = column + 1;	% Moves calculation to next column
154 -	end	
155		
156	%% GRAPH PLOT	
157		
158 -	if plot_display == 1 && iteration_counter_plot == plot_iteration	% Assesses whether or not user wants to see plot, and only plots every 100 iterations
159 -	plot_plate = flipud(plate);	% Flips matrix on Y axis, as 'contourf' function plots upside-down
160 -	f1= figure (1);	% Assigns plot to figure 1
161 -	f1.WindowState = 'maximized';	% Sets figure to open fullscreen for user convenience
162 -	set(gcf, 'color','w');	% Sets background colour to white
163		
164 -	colormap(jet)	% Sets up a changing colourmap with respect to timestep - 'jet' colourscheme
165 -	contourf(plot_plate)	% Displays plate matrix as image - 'scaled' sets values to range between the min and max colour limits
166 -	key = colorbar;	% Displays Key on Plot
167 -	key.Label.String = 'Plate Temperature (Degrees Celcius)';	% Sets text lable on colourbar
168 -	title('Live Plot of Temperature Gradient Throughout Plate')	% Adds title
169 -	axis equal tight	% Makes both axes equal length
170 -	xticks(1:1:x_points)	% Sets X axis label locations
171 -	yticks(1:1:y_points)	% Sets Y axis label loactions
172 -	xlabel('x-direction nodes')	% Sets X axis titles
173 -	ylabel('y-direction nodes')	% Sets Y axis titles
174 -	iteration_counter_plot = 1;	% Resets iteration counter to 1
175 -	drawnow	% Prints value at each loop - creating moving colourmap
176 -	end	

177		
178	end	
179	toc	% Finishes counting computing time
180		
181	%% FINAL PLATE VISUAL DISPLAY	
182		
183	if plot_display == 0	% If user selects not to see live plot, graphic of final plate is displayed
184	plot_plate = flipud(plate);	% Flips matrix on Y axis, as 'contourf' function plots upside-down
185	f1= figure (1);	% Assigns plot to figure 1
186	f1.WindowState = 'maximized';	% Sets figure to open fullscreen for user convenience
187	set(gcf, 'color','w');	% Sets background colour to white
188		
189	colormap(jet)	% Sets up a changing colourmap with respect to timestep - 'jet' colourscheme
190	contourf(plot_plate)	% Displays plate matrix as image - 'scaled' sets values to range between the min and max colour limits
191	key = colorbar;	% Displays Key on Plot
192	key.Label.String = 'Plate Temperature (Degrees Celcius)';	% Sets text lable on colourbar
193	title('Temperature Gradient Throughout Final Plate')	% Adds title
194	axis equal tight	% Makes both axes equal length
195	xticks(1:l:x_points)	% Sets X axis label locations
196	yticks(1:l:y_points)	% Sets Y axis label loactions
197	xlabel('x-direction nodes')	% Sets X axis titles
198	ylabel('y-direction nodes')	% Sets Y axis titles
199	drawnow	% Prints value at each loop - creating moving colourmap
200	end	
201		
202	%% END DISPLAY	
203	total_time = ((iteration_counter_time)*(time_step))/(3600);	% Calculates total time to reach equilibrium
204	end_msg = sprintf...	% Sets first line of message to be shown in dialogue box
205	('Simulation complete. Time until thermal equilibrium: %4.2f hours.'...	
206	,total_time);	
207	end_msg = [end_msg newline 'Open variable "plate" to view matrix.'];	% Sets second line of message to be shown in dialogue box
208	end_msgbox = msgbox(end_msg);	% Displays dialogue box
209		

9.3 Appendix 1c - Gaussian Plate Script

1 -	clc, clear	
2 -		
3 -	%% CONSTANTS	
4 -		
5 -	k = 143;	% Aluminium Conductivity (W/m.C)
6 -	p = 2.8e3;	% Aluminium Density (kg/m^3)
7 -	c = 795;	% Aluminium Specific heat (J/kg.C)
8 -		
9 -	t1 = 10;	% Start Temperature (C)
10 -	t2 = 60;	% Mid Temperature (C)
11 -		
12 -	alpha = k / (p*c);	% Thermal diffusivity (m^2/s)
13 -		
14 -	%% USER INPUTS & VALIDATION	
15 -		
16 -	% PLATE SIZE INPUT AND VALIDATION	
17 -	disp(' Welcome. Please enter the required information below: ')	
18 -		
19 -	x_dim = input(' X-axis dimension (m) = ');	% X axis user input
20 -	while isempty(x_dim) == 1	% Checks if value assigned to X-axis
21 -	warning '** X-axis dimension undefined. Please re-enter **'	
22 -	clear x_dim	% Clears incorrect variable
23 -	x_dim = input(' X-axis dimension (m) = ');	% X axis user re-input
24 -	end	
25 -		
26 -	while x_dim <= 0	% Checks if X-axis dimension greater than 0
27 -	warning '** X-axis dimension must be > 0 **'	% X axis user re-input
28 -	clear x_dim	
29 -	x_dim = input(' X-axis dimension (m) = ');	
30 -	end	
31 -		
32 -	y_dim = input(' Y-axis dimension (m) = ');	% Y axis user input
33 -	while isempty(y_dim) == 1	% Checks if value assigned to Y-axis
34 -	warning '** Y-axis dimension undefined. Please re-enter **'	
35 -	clear y_dim	% Clears incorrect variable
36 -	y_dim = input(' Y-axis dimension (m) = ');	% Y axis user re-input
37 -	end	
38 -		
39 -	while y_dim <= 0	% Checks if Y-axis dimension greater than 0
40 -	warning '** Y-axis dimension must be > 0 **'	% Y axis user re-input
41 -	clear y_dim	
42 -	y_dim = input(' Y-axis dimension (m) = ');	
43 -	end	
44 -		

```

45 % GRID SPACING INPUT AND VALIDATION
46 grid_space = input(' Grid spacing (m) = ');
47 while isempty(grid_space) == 1
48     warning '* Grid spacing undefined. Please re-enter *'
49     clear grid_space
50     grid_space = input(' Grid spacing (m) = ');
51 end
52
53 while grid_space >= y_dim || grid_space >= x_dim || grid_space <= 0
54     warning '* Grid Spacing must be less than X and Y dimensions and > 0 *'
55     clear grid_space
56     grid_space = input(' Please re-enter grid spacing (m) = ');
57 end
58
59 % CREATING ROWS AND COLUMNS TO CHECK GRID SPACING VALUE
60 x_points = (x_dim/grid_space + 1);
61 y_points = (y_dim/grid_space + 1);
62 x_check = x_points == round(x_points);
63 y_check = y_points == round(y_points);
64
65 while x_check == 0 || y_check == 0
66     warning '* Grid spacing is incompatible. Please choose a factor of both X & Y dimensions *'
67     clear grid_space & x_points & y_points & x_check & y_check
68     grid_space = input(' Please re-enter grid spacing (m) = ');
69     x_points = (x_dim/grid_space + 1);
70     y_points = (y_dim/grid_space + 1);
71     x_check = x_points == round(x_points);
72     y_check = y_points == round(y_points);
73 end
74
75 % TIME STEP INPUT AND VALIDATION
76 disp ' '
77 max_time_step = ((grid_space^2)/(4*alpha));
78 fprintf('Time step must be between 0 and %7.5f seconds',max_time_step)
79 disp ' '
80
81 time_step = input(' Time step (s) = ');
82 while isempty(time_step) == 1
83     warning '* Time step undefined. Please re-enter *'
84     clear time_step
85     time_step = input(' Time step (s) = ');
86 end
87

```

% Grid spacing user input
 % Checks if value assigned to grid spacing
 % Clears incorrect variable
 % Grid spacing user re-input
 % Ensures grid spacing is not greater than or equal to dimensions - '>' = no values, '=' = display of just boundary conditions - not correct. Or zero.
 % Clears incorrect variable
 % Grid spacing user re-input
 % Calculates number of columns (+1 to include starting at zero)
 % Calculates number of rows (+1 to include starting at zero)
 % Checks grid space x ratio is an integer (checks it's a factor)
 % Checks grid space y ratio is an integer (checks it's a factor)
 % Checks for integers, replace values if needed
 % Clears user's old, incorrect values
 % Grid spacing user re-input
 % Re-calculates number of columns (+1 to include starting at zero)
 % Re-calculates number of rows (+1 to include starting at zero)
 % Re-checks new grid spacing value against X dimension
 % Re-checks new grid spacing value against Y dimension
 % Displays a space between lines - easier to read for user
 % Calculates maximum time step
 % Displays maximum time step to user
 % Time step user input
 % Checks if value assigned to time step
 % Clears incorrect variable
 % Time step user re-input

88	%% FOURIER VALUE VALIDATION	
89		
90	Fo = (alpha*time_step) / (grid_space^2);	% Fourier number calculation
91		
92	while Fo >= (1/4) Fo < 0	% Checks stability criterion
93	warning 'Stability criterion broken. Please try different timestep value '	
94	fprintf('Time step must be between 0 and %7.5f seconds',max_time_step)	% Reminds user of maximum time step
95	disp ' '	% Displays a space between lines - easier to read for user
96	clear time_step & Fo	% Clears user's old, incorrect values
97	time_step = input(' Please re-enter time step (s) = ');	% Time step user re-input
98	Fo = (alpha*time_step) / (grid_space^2);	% Fourier number re-calculation
99	end	
100		
101	%% USER RESULTS PREFERENCE	
102		
103	plot_display = questdlg(...	% Asks the user if live plot should be displayed via dialogue box
104	' Would you like to see a live plot of the simulation? ', 'Yes','No');	
105	switch plot_display	
106	case 'Yes'	% Assigns value = True if user chooses 'yes'
107	plot_display = 1;	
108	case 'No'	% Assigns value = False if user chooses 'No'
109	plot_display = 0;	
110	case 'Cancel'	% Assigns value = False if user chooses 'Cancel'
111	plot_display = 0;	
112	end	
113		
114	%% CREATING HEAT TRANSFER MATRIX - BOUNDARY CONDITIONS	
115		
116	plate = zeros([y_points x_points]);	% initially makes correct size matrix of zeros
117	x_0 = ((x_points+1)/2);	% Finds mid-point of matrix to assign to t2
118	Sd = sqrt((-x_points-x_0)^2)/(2*(log(t1/t2)));	% Finds standard deviation
119		
120	temp_count = 1;	
121	while temp_count <= x_points	% Calculates all values in Gaussian distribution
122		
123	plate(1,temp_count) = (t2*exp(-((temp_count-x_0)^2)/(2*(Sd^2))));	% Equation to put values into first row of plate matrix
124		
125	temp_count = temp_count + 1;	
126		
127	end	
128		

129	%% CREATING HEAT TRANSFER MATRIX - NODE TEMPERATURE VALUES	
130		
131	stop_checker = zeros([y_points x_points]);	% Creates initial convergence checking matrix (referred to as 'stop criterion')
132	stop_checker(1,:) = 1;	% Sets first row of matrix as 1's, as convergence checker ignores first row.
133	exit_condition = ones([y_points x_points]);	% Sets the exit condition to be when stop checker matrix is all 1's
134		
135	stop_criterion = 0;	% Sets initial stop criterion value
136	iteration_counter_time = 1;	% Counts how many plates used in calculation - for time
137	iteration_counter_plot = 1;	% Counts how many plates used in calculation - for plot
138	stop_criterion_comparison = 0.9999999;	% Sets accuracy of stop criterion
139		
140	%% SETTING PLOT ITERATIONS	
141		
142	if(time_step/max_time_step) >0 && (time_step/max_time_step) <=0.1	% If very small timestep is chosen, graph will be plotted every 1000 iterations
143	plot_iteration = 1000;	
144	elseif(time_step/max_time_step) >0.1 && (time_step/max_time_step) <=0.5	% If medium timestep is chosen, graph will be plotted every 100 iterations
145	plot_iteration = 100;	
146	elseif(time_step/max_time_step) >0.5 && (time_step/max_time_step) <=1	% If large timestep is chosen, graph will be plotted every 10 iterations
147	plot_iteration = 10;	
148	end	
149		
150	%% TEMPERATURE EQUATIONS	
151	tic	% Starts counting computing time
152		
153	while sum(stop_checker, 'all') < sum(exit_condition, 'all')	% Sets calculation to stop when stop checker = all 1's
154	%% INTERIOR VALUES	
155	plate_old = plate;	% Saves previously calculated plate as 'old plate', to calculate new values from
156	iteration_counter_time = (iteration_counter_time + 1);	% Counts how many plates used in calculation - for time
157	iteration_counter_plot = (iteration_counter_plot + 1);	% Counts how many plates used in calculation - for plot
158	column = 2;	% Sets start point at column 2, as column 1 values are already set
159	while column < x_points	% Calculates all columns through to end-1
160		
161	row = 2;	% Sets start point at row 2, as row 1 values are already set
162	while row < y_points	% Calculates all rows through to end-1
163		
164	Tp = plate(row,column);	% Sets start node for each iteration
165	plate(row,column) = Fo * (plate_old(row -1,column) + ...	% Equation to find adjacent nodes (N,E,S,W)
166	plate_old(row,column + 1) + plate_old(row + 1,column) + ...	
167	plate_old(row,column - 1) - (4 - (1/Fo)) * Tp);	
168		
169	stop_criterion = plate_old(row,column) / plate(row,column);	% Calculates change of values in each node between old plate and current plate - the closer the values get, the closer this variable gets to one.
170	if stop_criterion > stop_criterion_comparison	% Assesses if change in values is greater than chosen acceptable limit
171	stop_checker(row,column) = 1;	% Sets current node, if above acceptable limit, to 1 in stop checker matrix
172	end	

173 174 - 175 - 176 177 - 178 - 179 180 - 181 - 182 - 183 184 - 185 - 186 187 188 189 - 190 - 191 - 192 - 193 194 - 195 - 196 197 - 198 - 199 - 200 201 - 202 - 203 204 205 206 - 207 - 208 - 209 - 210 211 - 212 - 213 214 - 215 - 216 - 217	row = row + 1; end column = column + 1; end %% LEFT EDGE column = 1; row = 2; while row < y_points Tp = plate(row,column); plate(row,column) = Fo * (plate_old(row - 1,column) + ... plate_old(row,column + 1) + plate_old(row + 1,column) + ... plate_old(row,column + 1) - (4 - (1/Fo)) * Tp); stop_criterion = plate_old(row,column) / plate(row,column); if stop_criterion > stop_criterion_comparison stop_checker(row,column) = 1; end row = row + 1; end %% RIGHT EDGE column = x_points; row = 2; while row < y_points Tp = plate(row,column); plate(row,column) = Fo * (plate_old(row - 1,column) + ... plate_old(row,column - 1) + plate_old(row + 1,column) + ... plate_old(row,column - 1) - (4 - (1/Fo)) * Tp); stop_criterion = plate_old(row,column) / plate(row,column); if stop_criterion > stop_criterion_comparison stop_checker(row,column) = 1; end row = row + 1; end %% BOTTOM EDGE row = y_points; column = 2; while column < x_points	% Moves calculation to next row % Moves calculation to next column % Sets calculation to column 1 % Sets start point at row 2, as row 1 values are already set % Calculates all rows through to end-1 % Sets start node to be row 2 column 1 % Equation to find adjacent nodes (N,E,S,W) where W = E because equilibrium at boundary % Stop criterion calculations % Moves calculation to next row % Sets calculation to end column % Sets start point at row 2, as row 1 values are already set % Calculates all rows through to end-1 % Sets start node to be row 2 column 1 % Equation to find adjacent nodes (N,E,S,W) where W = E because equilibrium at boundary % Stop criterion calculations % Moves calculation to next row % Sets calculation to end row % Sets start point at column 2 % Calculates all columns through to end-1
---	---	---

218 -	Tp = plate(row,column);	% Sets start node to be row end, column 1
219 -	plate(row,column) = Fo * (plate_old(row - 1,column) + ...	% Equation to find adjacent nodes (N,E,S,W) where S = N & W = E because equilibrium at boundary
220	plate_old(row,column + 1) + plate_old(row - 1,column) + ...	
221	plate_old(row,column - 1) - (4 - (1/Fo)) * Tp);	
222		
223 -	stop_criterion = plate_old(row,column) / plate(row,column);	% Stop criterion calculations
224 -	if stop_criterion > stop_criterion_comparison	
225 -	stop_checker(row,column) = 1;	
226 -	end	
227		
228 -	column = column + 1;	% Moves calculation to next column
229 -	end	
230	%% BOTTOM LEFT NODE	
231 -	Tp = plate(y_points,1);	% Sets start node to bottom left
232 -	plate(y_points,1) = Fo * (plate_old(y_points-1,1) + ...	% Equation to find adjacent nodes (N,E,S,W) where S = N & W = E because equilibrium at boundary
233	plate_old(y_points,2) + plate_old(y_points-1,1) + ...	
234	plate_old(y_points,2) - (4 - (1/Fo)) * Tp);	
235		
236 -	stop_criterion = plate_old(y_points,1) / plate(y_points,1);	% Stop criterion calculations
237 -	if stop_criterion > stop_criterion_comparison	
238 -	stop_checker(y_points,1) = 1;	
239 -	end	
240	%% BOTTOM RIGHT NODE	
241 -	Tp = plate(y_points,x_points);	% Sets start node to bottom right
242 -	plate(y_points,x_points) = Fo * (plate_old(y_points-1,x_points) + ...	% Equation to find adjacent nodes (N,E,S,W) where S = N & E = W because equilibrium at boundary
243	plate_old(y_points,x_points-1) + plate_old(y_points-1,x_points) ...	
244	+ plate_old(y_points,x_points-1) - (4 - (1/Fo)) * Tp);	
245		
246 -	stop_criterion = plate_old(y_points,x_points)/plate(y_points,x_points);	% Stop criterion calculations
247 -	if stop_criterion > stop_criterion_comparison	
248 -	stop_checker(y_points,x_points) = 1;	
249 -	end	
250		
251	%% GRAPH PLOT	
252		
253 -	if plot_display == 1 && iteration_counter_plot == plot_iteration	% Assesses whether or not user wants to see plot, and only plots every set number of iterations
254 -	plot_plate = flipud(plate);	% Flips matrix on Y axis, as 'contourf' function plots upside-down
255 -	f1 = figure (1);	% Assigns plot to figure 1
256 -	f1.WindowState = 'maximized';	% Sets figure to open fullscreen for user convenience
257 -	set(gcf, 'color','w');	% Sets background colour to white
258		

259 - colormap(jet)	% Sets up a changing colourmap with respect to timestep - 'jet' colourscheme
260 - contourf(plot_plate)	% Displays plate matrix as image - 'scaled' sets values to range between the min and max colour limits
261 - key = colorbar;	% Displays Key on Plot
262 - key.Label.String = 'Plate Temperature (Degrees Celcius)';	% Sets text lable on colourbar
263 - title('Live Plot of Temperature Gradient Throughout Plate')	% Adds title
264 - axis equal tight	% Makes both axes equal length
265 - xticks(0:1:x_points)	% Sets X axis label locations
266 - yticks(0:1:y_points)	% Sets Y axis label loactions
267 - xlabel('x-direction nodes')	% Sets X axis titles
268 - ylabel('y-direction nodes')	% Sets Y axis titles
269 - drawnow	% Prints value at each loop - creating moving colourmap
270 - iteration_counter_plot = 1;	% Resets iteration counter to 1
271 - end	
272	
273 - end	
274 - toc	% Finishes counting computing time
275	
276 %% FINAL PLATE VISUAL DISPLAY	
277	
278 - if plot_display == 0	% If user selects not to see live plot, graphic of final plate is displayed
279 - plot_plate = flipud(plate);	% Flips matrix on Y axis, as 'contourf' function plots upside-down
280 - f1= figure (1);	% Assigns plot to figure 1
281 - f1.WindowState = 'maximized';	% Sets figure to open fullscreen for user convenience
282 - set(gcf, 'color','w');	% Sets background colour to white
283	
284 - colormap(jet)	% Sets up a changing colourmap with respect to timestep - 'jet' colourscheme
285 - contourf(plot_plate)	% Displays plate matrix as countour image
286 - key = colorbar;	% Displays Key on Plot
287 - key.Label.String = 'Plate Temperature (Degrees Celcius)';	% Sets text lable on colourbar
288 - title('Temperature Gradient Throughout Final Plate')	% Adds title
289 - axis equal tight	% Makes both axes equal length
290 - xticks(1:2:x_points)	% Sets X axis label locations
291 - yticks(1:2:y_points)	% Sets Y axis label loactions
292 - xlabel('x-direction nodes')	% Sets X axis titles
293 - ylabel('y-direction nodes')	% Sets Y axis titles
294 - end	
295	
296 %% END DISPLAY	
297 - total_time = ((iteration_counter_time)*(time_step))/(3600);	% Calculates total time to reach equilibrium
298 - end_msg = sprintf...	% Sets first line of message to be shown in dialogue box
299 - ('Simulation complete. Time until thermal equilibrium: %4.2f hours.'...	
300 - ,total_time);	
301 - end_msg = [end_msg newline 'Open variable "plate" to view matrix.'];	% Sets second line of message to be shown in dialogue box
302 - end_msgbox = msgbox(end_msg);	% Displays dialogue box
303	