



University of Wales Trinity Saint David

School of Engineering

Title of the Coursework

Transient Thermal Diffusion Model using the Finite
Difference Method

Name and P Number

Danny Richardson

P151196

Module Name and Code

Advanced Computational Methods

M6X00408

Date

1/11/19

Lecturer's Name

Dr A. Marotin

Contents

1. Introduction.....	1
2. Constitutive Equation.....	2
1. Diffusion.....	2
2. Taylor series.....	4
3. Explicit finite difference method.....	5
3. Script modelling utilising the FDM equation.....	7
1. Validation plate boundary conditions - with script construction.....	7
2. Gaussian distribution method boundary conditions - with script construction.....	9
3. User inputs within the script.....	11
4. Validation plate – Script construction.....	13
5. Secondary plate – Script construction.....	14
6. Graphical representations.....	17
4. Analysis.....	19
5. Limitations and Improvements.....	22
6. Conclusion.....	23
7. References.....	24
8. Appendices.....	25
1. MATLAB Thermal Diffusivity Script.....	25

Table of Figures

Figure 1 Gaussian distribution method boundary conditioned plate.....	1
Figure 2 Representation of 'x' and 'y' points between the nodes.	2
Figure 3 Depiction of the node points showing their orientations from point 'P'.....	6
Figure 4 Depiction of the starting conditioned plate, with the additional temperatures distributed around the boundaries for the validation plate.	7
Figure 5 Script for laying out the starting conditions of the validation plate.....	8
Figure 6 Matrix of the starting plate using the Gaussian distribution method.	9
Figure 7 Gaussian distribution method of second plates boundary conditions script.	10
Figure 8 Example of Gaussian distribution with incorrect assignment in the while loop.	11
Figure 9 Correct assignment of exit criterion in the while loop used for the Gaussian distribution method.....	11
Figure 10 User input sections basic construction.	12
Figure 11 Line of code showing criterion of the while loop for the validation plate.....	13
Figure 12 Counter assignment for the rows and columns of the validation plate.	13
Figure 13 Calculation while loop for the linear heat transfer validation plate.....	14
Figure 14 'while' loop conditions for the secondary plate.....	14
Figure 15 Script section for the calculation across the main area and the left - right hand edges of the secondary plate.	15
Figure 16 Bottom row calculation line, with the exclusion of the 'corner' nodes for the secondary plate.....	16
Figure 17 Lines for the bottom corners of the secondary plate.	16
Figure 18 Lines of code used to create the 'figure' demonstrating the heat as it propagates through the plates.	17
Figure 19 Validation plate heat propagation - mid calculation.....	18
Figure 20 Validation plate heat propagation - final iteration.	18
Figure 21 15x15 dimension plate final temperature distribution.....	21

1. Introduction.

The experiment proposed within this paper is well known and is targeted with being able to predict how, in somewhat idealised situations, heat will be transferred through a two-dimensional metal plate.

This paper will therefore discuss the implementation of mathematical techniques upon the known equation that governs diffusivity. The effect of components within the then found derived equation, along with it's limits. Along with demonstrating how the software known as MATLAB can be used to aid in developing an automated 'tool' in which to both calculate and provide a visual representation of heat diffusing across the plate.

Objectives –

- Demonstrate an understanding of the mathematical principles used to derive the fundamental equation.
- Outline the boundary conditions used within the study and explain their effect upon the way heat will propagate.
- Produce a detailed description of how MATLAB was used to develop a user friendly 'script' to automate the calculation process.
- Explain the noticeable impacts of the inputted values.
- Discuss the potential limitations within this style of modelling, in terms of the logical based statements, used to assess the predicted model.

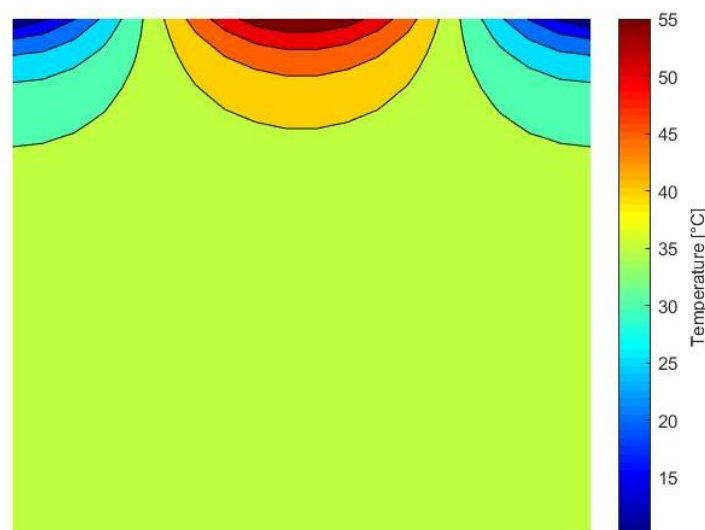


Figure 1 Gaussian distribution method boundary conditioned plate.

2. Constitutive Equation.

1. Diffusion.

The term diffusion can be defined as “Diffusion is the process by which matter is transported from one part of a system to another as a result of random molecular motions” (Crank, 1975). It’s also understood that heat transfer via conduction, is also governed by the same idea of ‘random molecular motions’ and that is why the following equation is also recognised as the “heat equation”;

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

(Langtangen & Linge, 2016).

Within the study, the consideration was however for a 2D metal plate, not a singular dimension that is described in the above equation. Therefore, the diffusion equation is to be shown as;

$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

(University of Münster, 2016).

Within the equation, the respective values used are; u is temperature, t is the time step, α is alpha and then of course the x & y are the steps between the nodes in the directions of. For ease, the figure below demonstrates how these values are to be thought of;

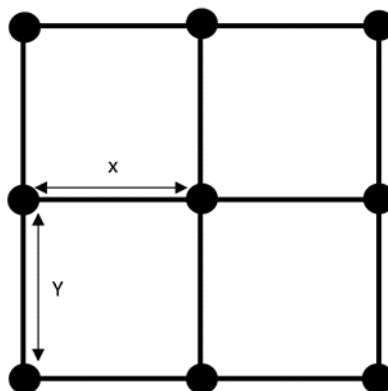


Figure 2 Representation of 'x' and 'y' points between the nodes.

With concerns to alpha, this is the constraint that remains constant within the study and is a product of $\alpha = \frac{k}{\rho \cdot c}$, with 'k' being the thermal conductivity of the material, 'ρ' is the density and then 'c' is the specific heat. The units of these are; W.m⁻¹.K⁻¹, kg.m⁻³ and J.kg⁻¹.K⁻¹ all respectively.

These constants play a major role in the stability of the mathematical model, governed under the term 'Fouriers number'. This is shown as;

$$F_o = \frac{\alpha \cdot \Delta t}{\Delta x^2}$$

(Langtangen & Linge, 2016)

This small equation is understood to be governed by the 'stability criterion', to which changes with each type of model. In the case of this study, the criterion that must be implemented is shown as; $0 \leq F_o \leq \frac{1}{4}$. This is because the model is considered as two dimensional and therefore now contains two fouriers numbers, in both directions of 'x' & 'y'. Hence each must be below or equal to one quarter, in order to still conform to;

$$Fo_x + Fo_y \leq \frac{1}{2}$$

(Langtangen & Linge, 2016)

If this criterion is met, the model will be considered to remain 'stable'.

2. Taylor series.

The Taylor series is understood to be, that for certain cases of a function of 'x' ('f(x)'), it can be rewritten as an infinite sum of other functions all with respect to 'x'.

Shown as;

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

This is definitive version of the 'Taylor series' and because of the multiple functions of the single variable, the curve created is known as a polynomial.

Within this study however, this fundamental required derivation in order to obtain the versions known as the 'forward difference' and the 'central difference' approximations.

With the forward difference approximation, if 'x = x₀ + h', it can be expressed as;

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

With this equation and only concerning the first order term, it can then be rearranged to show the forward approximation as;

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

As can be seen with this equation, the value of 'h' can be thought of as the error incurred. Hence, the smaller the value the more "accurate".

The term named the 'central difference approximation' was also required in order to complete the model, but it's required to include the second order derivative.

Starting with the approximation to the first order, the combination of the 'forward' & 'backward' difference approximations is required.

The backward difference approximation is shown as;

$$f'(x_0) \approx \frac{f(x_0) - f(x_0 - h)}{h}$$

So, with 'x = x₀ - h' being the case for the backward difference, the combination of the two can be condensed and written as;

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

Rearranged to become;

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0 - h)}{2. h}$$

This is therefore the central difference approximation equation, with respect to the first order.

As stated, the central difference approximation is required to include to second order term, because of the diffusivity equation for two-dimensional application includes the 'x' & 'y' terms as squared.

Again, using the combination of the forward and backward approximation equations, to yield them as;

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

Rearranging to present the second order function as the approximation of, it becomes;

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

These approximations can now be used in conjunction with the explicit finite difference method to obtain the key equation used for the experiment.

3. Explicit finite difference method.

Given the problem proposed in the experiment, the plate is two dimensional and therefore the equation of diffusivity (heat), is considered as;

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

Be defining the points of consideration if the figure underneath;

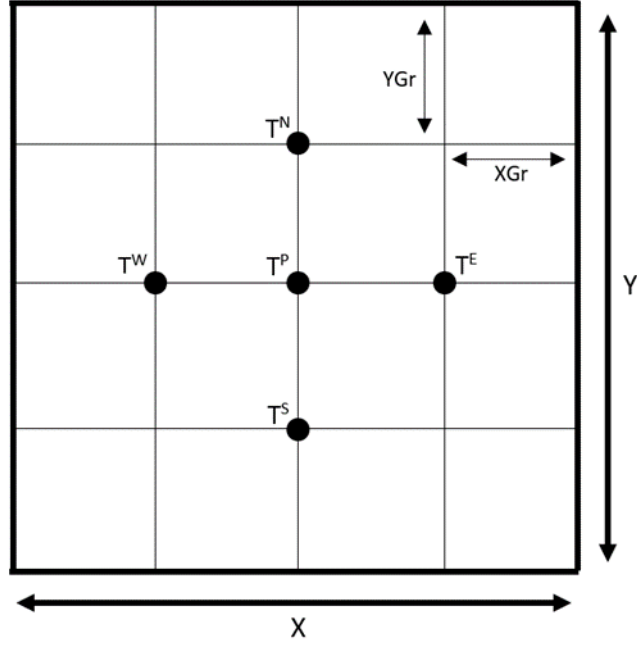


Figure 3 Depiction of the node points showing their orientations from point 'P'.

Upon implementing the explicit finite difference method, assigning new numerical approximations to the values within the diffusivity equation using the central and forward difference approximations. We show it as;

Central difference approximations [2nd order];

$$\frac{\partial^2 T_P}{\partial x^2} = \frac{T_E^n - 2 \cdot T_P^n + T_W^n}{h^2}$$

$$\frac{\partial^2 T_P}{\partial y^2} = \frac{T_N^n - 2 \cdot T_P^n + T_S^n}{h^2}$$

Forward difference approximation [1st order];

$$\frac{\partial T_P}{\partial t} = \frac{T_P^{n+\Delta t} - T_P^n}{\Delta t}$$

From these new definitions of terms, implementing them within the original diffusion equation, along with the defining 'Fouriers number', to get;

$$T_P^{n+\Delta t} = F_o \left(T_E^n + T_W^n + T_S^n + T_N^n - \left(4 - \frac{1}{F_o} \right) \cdot T_P^n \right)$$

This is the new approximation equation that will be utilised to demonstrate heat propagation through a 2D metal plate.

3. Script modelling utilising the FDM equation.

1. Validation plate boundary conditions - with script construction.

The modelling of the script was done under a confined set of boundary conditions, one for each of the plate types modelled. The first of which was the standard validation plate which was designed in the style of; A surrounded two-dimensional plate, with all nodes across the plate starting at 0 °C. This plate then had a series of temperatures attached to its edges, starting with the left-hand edge being a 50 °C constant input and then the right-hand being maintained at 0 °C. With the top and bottom edges of this plate having a linearly distributed heat source, dependent of grid spacing, decreasing from the starting 50 °C down to the maintained 0 °C.

These attached inputs were created in the form of an extra two rows and columns in addition from the chosen plate size. For example, in the figure below, the plate was specified as a 5x5 dimension plate with a grid spacing of 0.5. Hence the plate would be created as ten rows and ten columns ($\frac{5}{0.5} = 10$). And with the additional inputted boundary temperatures, creating in this case, a 12x12 matrix.

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

Figure 4 Depiction of the starting conditioned plate, with the additional temperatures distributed around the boundaries for the validation plate.

The script for the validation plate boundary conditions was written as such;

```
%% Starting Temperatures.

% Starting temperature placement around the plate.

% The following XGr and YGr are the variables that create the number of
% grid "lines" across the plate and in turn the correct amount of nodes.

XGr = round((x/Gr)+2,0);
YGr = round((y/Gr)+2,0);
TpsZerosXY = zeros(XGr,YGr)'; % Creates the correct correlating matrix of zeros.
LinearT = linspace(heatin,coldin,XGr); % function that will correctly space the decreasing
temperature intervals, dependent of the amount of 'x' nodes.
TpsZerosXY(1,:) = LinearT; % Assign's the linspace values across the "top edge" (row 1,
all coloumns) of the plate.
TpsZerosXY(YGr,:) = LinearT; % Assign's the linspace values across the "bottom edge" of the
plate.
TpsZerosXY(:,1) = heatin; % Assign's the heat in (50°C) to the "left hand edge" of the
plate.
TpsZerosXY(:,XGr) = coldin; % Assign's the cold in (0°C) to the "right hand edge" of the
plate.
PlateStart = TpsZerosXY; % Simply renaming for visualization.
Iterations = 1;
```

Figure 5 Script for laying out the starting conditions of the validation plate.

The starting variables and how the number of nodes is created, with the dimensions of the plate in the 'x' and 'y' directions divided by the chosen grid spacing. As can be seen, there is the addition of the extra two rows and columns here, shown as the +2 in the script. These are they heat source spaces within the matrix that is to be made.

The matrix is then created based of the requirement, as chosen by the user, and is filled by using the zeros function. The linspace function is then implemented so for any chosen dimension (within the criterion), the distribution will always be correct. In accordance with the starting temperatures at either end, linearly spaced across the length of the plate ('x' dimension).

The few lines left within this section are simply assigning the boundary conditions around the plate ready to start (Figure 1), along with renaming and the assignment of a counter variable.

2. Gaussian distribution method boundary conditions - with script construction.

With the secondary plate, using the Gaussian distribution method for the boundary condition heat sources, these were only to be inputted along the top edge of the plate. The target was to have the plate set with a completely insulated boundary across the left- & right-hand edges, along with the bottom edge also. With the heat source being supplied along the top edge, using the Gaussian method, such that the centre of the plate would be at 60 °C and then an evenly distributed input either side down to 10 °C.

The starting plate, using the same dimensions as the previous, would therefore start as shown in the following figure;

	1	2	3	4	5	6	7	8	9	10
1	10.0000	20.2963	34.5129	49.1688	58.6873	58.6873	49.1688	34.5129	20.2963	10.0000
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0

Figure 6 Matrix of the starting plate using the Gaussian distribution method.

The method of assigning an extra row for the heat source supplied, as incorporated in the previous plate, was also used in the model. Because this model is thought as the heat source being 'attached' against the edge of the plate, not an initial state of the plate.

The script for the boundary conditions implemented under the Gaussian distribution method can be seen below;

```
%% Gaussian Temperature distribution

% Because of the user input variable of the length of the plate, there is
% the change for a division that'll result in a remainder. Therefore, the following has been implemented to
% indefinitely ensure operation.

if Gauss == 1 % Continuation variable from the 'Continue' question box.
    XGr = x/Gr;
    XGr = round(XGr,0);
    Amplitude = 60; % Boundary conditions variables
    BoundaryEdges = 10; %
    TStartPoint = zeros(1,XGr);
    Column = 1;
    StandardDeviation = sqrt((-XGr - ((XGr + 1) / 2))^2 / (log( BoundaryEdges / Amplitude)^2));
    while Column < (XGr + 1)
        TStartPoint(1,Column) = Amplitude * exp((- (Column - (XGr + 1) / 2)^2) / (2 * (StandardDeviation)^2));
        Column = Column + 1;
    end
end

% The while loops are in place to enable the continual assignment of the
% symmetrical values to each column.
```

Figure 7 Gaussian distribution method of second plates boundary conditions script.

This section of the full script is used for creating the heat source row that will then be used to show the propagation through the secondary plate. The 'if' logic statement used here, is a continuation for the previous section that contains a user input (UI) question box to ask if they wish to continue [appendix 1. 1.11.]. If answered yes, this section will then begin, otherwise the script is stopped.

To be sure there was no issues carried through, the variable that denotes the required number of columns (XGr) is reassigned and rounded to be sure the value is an integer. Maximum and the boundary edge temperatures are assigned into the workspace.

An empty column vector is created, counter assigned, and the standard deviation equation can then be used to create a variable. Using the while loop allows the automatic assignment to each column along the 'x' axis, because it's done under such condition that the loop won't be broken until the counter has reached the number of columns required. Which is the number of nodes across the length of the plate, with the addition of one, because it starts at one and would otherwise meet the exit criterion before the correct number of columns has been assigned a value.

For example, without the addition of 1 to the variable XGr, the result is;

	1	2	3	4	5	6	7
1	10.0000	27.0587	49.1688	60	49.1688	27.0587	0

Figure 8 Example of Gaussian distribution with incorrect assignment in the while loop.

In comparison to the corrected of;

	1	2	3	4	5	6	7
1	10.0000	27.0587	49.1688	60	49.1688	27.0587	10.0000

Figure 9 Correct assignment of exit criterion in the while loop used for the Gaussian distribution method.

And of course, when this criterion is met the script will then move past this section and carry on into the section written to calculate the heat transfer through the secondary plate.

3. User inputs within the script.

The script was designed such that when ran, a user would be able to choose; the material used in the study, the dimensions of the plate being modelled, time step to be used and then the grid spacing that they desired.

With the user inputs within the script, they were done by using the standard function of 'input()', which will show the string of text and required the user to type in their chosen numbers, all within the command window. Although this could have been done using the function ['inputdlg()'], given the criterions needed within this model, the user may be required to choose their input again and therefore there could be multiple windows opened unnecessarily, therefore it was left alone.

Referring to appendix 1. 1.5., the script for this entire section of UI's can be seen. They've been incorporated within a 'while loop', with each required input within its own 'while loop' again. This was done specifically so that if the user had chosen a value outside of the criterion, the script wouldn't 'return' and require a restart for any and every time the criterion wasn't met.

Within the large loop, the sections are done using the 'if' logical statements because of the UI ['listdlg()'] at the start of the script, governing as to which criterion needs to be met, because the material properties have a large impact on the computing time required to run the model. Then the sections within are written as follows;

```
x = input("Please input a value for the length of the plate [Between 3 - 35]: ");
if x > 35
    error = msgbox('Error. Sorry but the length is too long and will
result in an unnesecary calculation time ','Error','error');
elseif x < 3
    error = msgbox('Error. Sorry but the length is to short to allow
for calculation ','Error','error');
end
if x < 3 || x > 35
    LeaveLength = 1;
elseif x > 3 || x < 35
    LeaveLength = 0;
end
```

Figure 10 User input sections basic construction.

Under the input function is the criterion that will check if the chosen value can be used. If incorrect, there will be a message box ['msgbox()'] displayed on the screen notifying the user that their value was either too large, or too small.

Depending on the value inputted, the following 'if' statement will assign a value to the variable 'LeaveLength' that is used as the exit criterion on the while loop. If the variable 'x' was incorrect 'LeaveLength' remains as 1 and the loop will run again. But if the value was fine, the value assigned is now 0, meaning the criterion is no longer true, therefore it's been met, allowing it to carry onto the sections for the height of the plate and the time step used, respectively.

This script also contains variations of dialogue boxes, to allow user interaction throughout the script. The script begins with a message box containing a 'string' of text to explain the purpose of the script along with a small informative section for the user before they start. Because the script includes four different materials in which to conduct the study upon, there is a list dialogue box included that assigns a number to the variable used in the "user inputs & criterions" section, depending on their chosen material (the standard aluminium is set as default). After the validation plate study has been conducted, there is also a question dialogue box ['questdlg()'] that lets the user choose to continue to the secondary plate when they wish, or they can end the study there.

4. Validation plate – Script construction.

(The full script written for the validation plate can be seen in appendix 1. 1.10.)

To be able to demonstrate the heat transfer across a 2D metal plate, there is obviously multiple iterations required, hence the code was written with the use of a while loop, to keep the same section running continuously until an exit criterion was met. The criteria used as material specific, but also dependent on the user inputs for dimensions and steps, these will be discussed further on.

Starting with the initial conditions of the loop, the criterion was set so that the second row and second column node (top corner edge of plate), has to be within a certain percentage of the node above (heat source being applied), with the idea that it reaches a level of thermal equilibrium possible when using the FDM.

```
while PlateStart(2,2) < (LinearT(1,2) - (LinearT(1,2)/stop))  
% Ending criterion check of the loop
```

Figure 11 Line of code showing criterion of the while loop for the validation plate.

Because of the iterative nature, counters had to be used to ensure that the calculation was continued across each node of the plate. These counters not only started the calculation in the correct positions, but they reassign the variable value at each iteration to continue doing so.

```
Column = 2;    % The row and column are 2, so that it's starting at the "actual" plate edge.  
while Column < XGr    % Stop condition such that it continues through all columns.  
    Row = 2;
```

Figure 12 Counter assignment for the rows and columns of the validation plate.

With these conditions set, the actual calculation loop had to be written. Using the variable 'Tstart', it was assigned continuously with the values at each node because of the counters for 'Row' & 'Column' having their value increased each time and therefore initially moving through all rows within the column, then continuing on within the next columns rows.

```

while Row < YGr % Stop condition such that it continues through all rows.
    Tstart = PlateStart(Row,Column);
    PlateStart(Row,Column) = Fo * (PlateStart(Row,Column+1) + PlateStart(Row,Column-1)
+ PlateStart(Row-1,Column) + PlateStart(Row+1,Column) - (4 - (1/Fo)) * Tstart );
    Row = Row + 1; % next row
end
Column = Column + 1; % next column

```

Figure 13 Calculation while loop for the linear heat transfer validation plate.

The row counter was contained within the loop to be sure that it would continue through each row without having moved column position. Hence the column counter being positioned outside of this loop to ensure the same principle, of remaining within the one column until all rows had been calculated.

Within the external loop with the exit criterion shown in figure 8, there is the small section containing another counter and a short 'if' statement. This section is here so that a graphical representation of the heat propagation is shown to the user in real time. Because of the sheer amount of calculations conducted within this loop, the counter titled 'iterations' is used, which will then update the figure being shown once a set amount had been ran through, this will be explained fully further on.

5. Secondary plate – Script construction.

The same style principles were used in construction of the script section concerning the secondary plate (Gaussian distribution method for heat source). A series of exit criterions were used, based upon the material used and plate "size" to govern when the script would stop, set with the hopes of reaching an acceptable point of 'thermal equilibrium' each time the script is ran.

Starting with conditions for the three 'while' loops used;

```

while StartPlate(YG,XG) < (TStartPoint(1,XG)*0.62)
    Column = 2;
    while Column < XGr % Calculation through the columns of the plate.
        Row = 2;
        while Row < YGr % Calculation through the rows of the plate.

```

Figure 14 'while' loop conditions for the secondary plate.

As can be seen, the same principles were used in the sense that, the external loop is the final 'convergence' loop deciding the point at which an acceptable level of heat transfer and thermal equilibrium has been reached. Then with the inner nestled loops

being the criteria ensuring all rows and columns that can be calculated, have been at each iteration. They've been set to 'less than' the bottom row (YGr) and last column (XGr), along with the column counter starting at 2, the reasons for which will be explained in the following.

Because of the boundary conditions of this plate and due to using the explicit FDM equation as discussed, the script required adjustment in order to calculate the left and right most columns.

```
while Row < YGr                                % Calculation through the rows of the plate.
    TStart = StartPlate(Row,Column);            % Main area of the plate.
    StartPlate(Row,Column) = Fo * (StartPlate(Row,Column+1) + StartPlate(Row,Column-
1) + StartPlate(Row+1,Column) + StartPlate(Row-1,Column) - (4-(1/Fo)) * TStart);

    TStart = StartPlate(Row,1);                % Left hand side of the plate.
    StartPlate(Row,1) = Fo * (StartPlate(Row,1+1) + StartPlate(Row,1+1) +
StartPlate(Row+1,1) + StartPlate(Row-1,1) - (4-(1/Fo)) * TStart);

    TStart = StartPlate(Row,XGr);              % Right hand side of plate.
    StartPlate(Row,XGr) = Fo * (StartPlate(Row,XGr-1) + StartPlate(Row,XGr-1) +
StartPlate(Row+1,XGr) + StartPlate(Row-1,XGr) - (4-(1/Fo)) * TStart);

    Row = Row + 1;
end
```

Figure 15 Script section for the calculation across the main area and the left - right hand edges of the secondary plate.

The first line is used to calculate the nodes that have a value for; north, south, east and west. Hence through the lines, there is both an addition and subtraction of 1 from the column and row number, corresponding to the co-ordinate positions.

The line underneath is to calculate the left-hand nodes, denoted within the 'TStart' assignment line with the use of column as one. This column doesn't have a value for the west node; therefore, the value is inputted as that of the east node within the equation because of its completely insulated boundary conditions.

Naturally the same conditions are used for the right-hand edge, shown by the final full line of code. With there now being a west node and not an east node, there is a slight rearrangement to correct the calculation.

Because of this issue regarding the lack of the west/east node, this column couldn't be calculated within the main area concerned, therefore the criterion of the loop was to start at the second column and stop before the last.

Once this loop has completed, the bottom row can be calculated. This again poses the issue of a missing 'South' node, hence it's replacement with the 'North' node in equation. Its outside the row loop because it's a singular row with a varying column counter, again starting at the second column whilst finishing before the final.

```
% Bottom row of the plate
StartPlate(YGr,Column) = Fo * (StartPlate(YGr,Column+1) + StartPlate(YGr,Column-
1) + StartPlate(YGr-1,Column) + StartPlate(YGr-1,Column) - (4-(1/Fo)) * TStart);

Column = Column + 1;
```

Figure 16 Bottom row calculation line, with the exclusion of the 'corner' nodes for the secondary plate.

The final lines within the section, both outside of the nestled 'while' loops, are to calculate the corners of the plate. Both of which have the issue of missing two nodes from the equation, South and either West (Left) or East (Right). Therefore, these had to again be adjusted to contain the correct value within all four of parts of the equation. They're outside of the loops simply because they only concern one point on the plate and remain as such.

```
% Bottom corners of the plate
StartPlate(YGr,1) = Fo * (StartPlate(YGr,1+1) + StartPlate(YGr,1+1) +
StartPlate(YGr-1,1) + StartPlate(YGr-1,1) - (4-(1/Fo)) * TStart);

StartPlate(YGr,XGr) = Fo * (StartPlate(YGr,XGr-1) + StartPlate(YGr,XGr-1) +
StartPlate(YGr-1,XGr) + StartPlate(YGr-1,XGr) - (4-(1/Fo)) * TStart);
```

Figure 17 Lines for the bottom corners of the secondary plate.

6. Graphical representations.

The script sections where the heat transfer is calculated contains a few lines of code that create a 'figure' showing the heat as it propagates through the plate.

```
if Iterations == TestI
    figure(1)
    contourf(PlateStart)
    axis off
    color = colorbar;
    color.Label.String = ' Temperature [°C] ';
    colormap(jet)
    Iterations = 1;
end
```

Figure 18 Lines of code used to create the 'figure' demonstrating the heat as it propagates through the plates.

It's done within an 'if' statement in order to control that number of times in which it is updated, simply because of the sheer amount of calculations that occurs it would be unnecessary to have it configure every time.

There is a small section that governs the iteration step, dependent upon material because of the large difference in calculation times.

Using the 'contourf' function of MATLAB allows it to show a smooth map of colours that correlate to temperature value, as opposed to simply a value within the matrix. The lines underneath are there to define how the 'figure' looks;

- 'axis off' is there to remove the values shown on the 'x' and 'y' axis.
- Assigning the variable color with the 'colorbar' function input the bar within the figure that corresponds to the temperature value at the nodes.
- Using the label function to assign a correct name to the figures color bar to inform the user.
- 'colormap' function decides the style of colours that are used within the figure. 'jet' was chosen because it was the most representable of heat propagation.
- Reassignment of the counter to ensure that the iterations step is corrected.

The effect of this can be seen in the figures below;

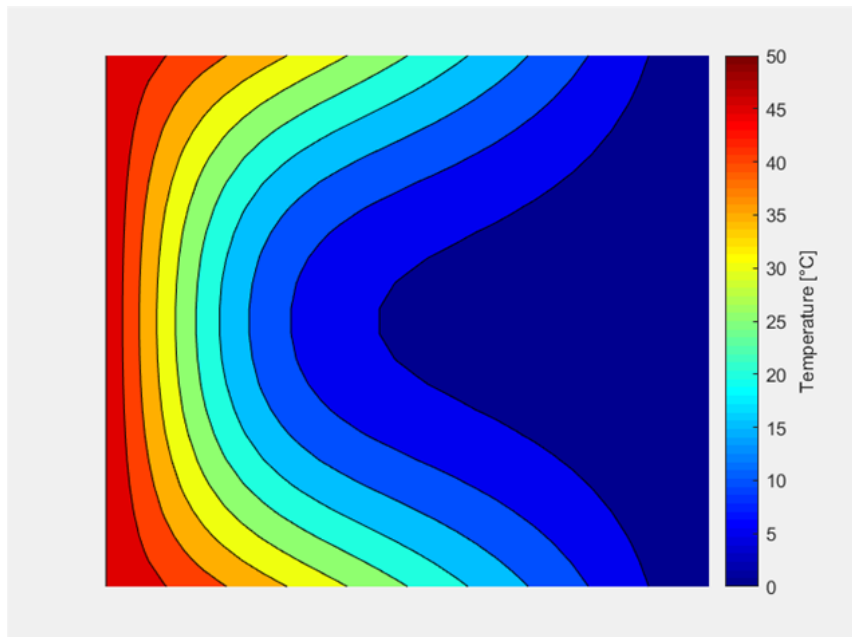


Figure 19 Validation plate heat propagation - mid calculation.

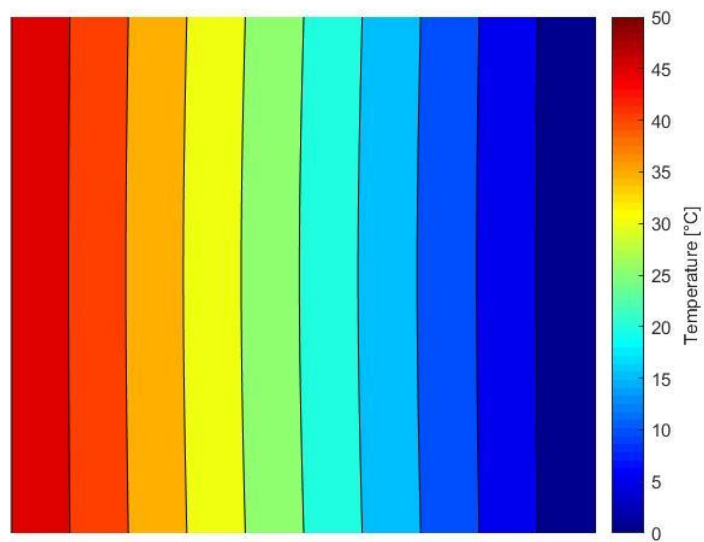


Figure 20 Validation plate heat propagation - final iteration.

4. Analysis.

When looking at the difference each input makes, it's clear that the time step and the square of grid spacing, are the most important factor regarding accuracy. This is because they are directly proportional to the error, hence using the 2D modelling the Fourier's criterion must be $0 \leq Fo \leq \frac{1}{4}$ where 'Fo' is shown to be $Fo = \frac{(a \cdot t)}{h^2}$. With 't' is time step and 'h²' is the grid spacing.

When producing a study of the effect of time step, using a square plate with dimensions 5x5 'meters' and a consistent grid spacing of 0.5. The effect upon calculation time in which it takes to reach the set level of equilibrium can be seen below;

Time step study	
Time step [s]	Calculation time [s]
0.5	34.6918
0.6	26.7845
0.7	23.3046
0.8	20.4893
0.9	17.1197
1	16.0328
1.5	10.2902
2	8.0413
3	5.1426
4	3.9335

Table 1 Time step study upon the effect of calculation time.

As can be seen, the time step adjustment has a major effect upon the calculation time when the dimension and grid spacing variables remain the same. This is because the number of calculations is dramatically dropped due to the fact of the 'Fourier number' being increased for each increase of in time. Therefore because of its implementation within the equation;

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

So, with the fouriers number being larger, the step in temperature calculated becomes larger and in turn, the difference between the nodes is a lot larger to that of when the time step is small. Because of this, although the time take is less, the error in calculation is greatly increased because of the gap between points calculated, where a change in heat transfer linearity could've have happened, therefore potentially invalidating the calculation.

Conducting a study upon the effect that grid spacing has on the model's calculation time can be seen below and for continuity the plate dimensions remained as 5x5 'meters' with a constant time step of 0.5;

Grid space study	
Grid space	Calculation time [s]
0.1	15.7455
0.2	19.6254
0.3	25.2507
0.4	30.1401
0.5	32.1199
0.75	43.8285
1	48.0813
1.5	54.8870
2	100.6323
2.5	102.1526

Table 2 Grid spacing study upon the effect on calculation time.

When assessing the data produced from the grid spacing study, the effect is that the larger the spacing the longer the test took, which seems odd when considering the impact of it being a drop in nodes across the plate, therefore reducing the calculations required. Because of this, it has not only highlighted to impact of error term being increased, but also the main component here is the exit criterion used within the coding. The fact that the same 'exit check' is used throughout, because there is less nodes, the temperature change is struggling to correlate correctly because the error was increase within the fouriers number. So, to retain accuracy, and in the case of this code, keeping the grid spacing to a minimum is key.

The last experiment carried out, was on the impact that the plate size made to the calculation time, when using a constant time step & grid spacing of 0.5. The effect is shown in the table below;

Plate size study	
Plate size	Calculation time [s]
5x5	31.3400
10x10	92.6236
15x15	171.3279

Table 3 Study on how the dimensions of the plate effect calculation time.

Again, as expected, the larger the plate was the greater the time required to calculate. With regards to accuracy, the largest plate (15x15) should've demonstrated the most, simply because of the difference in the number of nodes within the plate to be calculated. However, due to criterion used within the script, this wasn't the case;

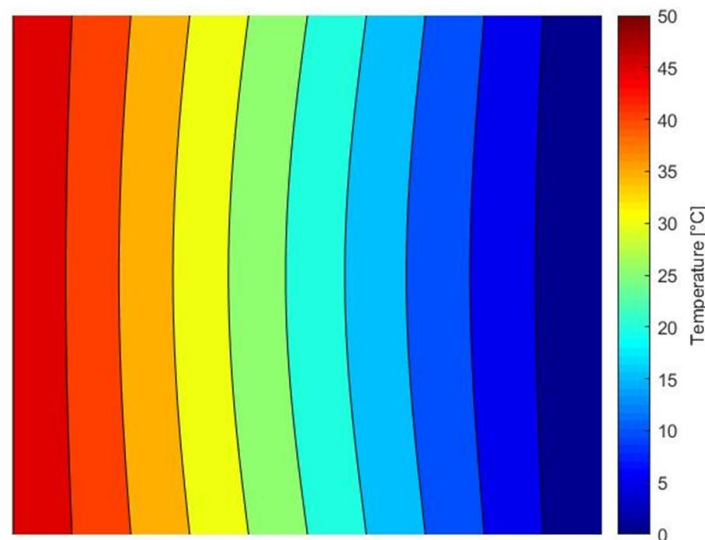


Figure 21 15x15 dimension plate final temperature distribution.

As can be seen, there is most definitely an issue within this code's criterion, this is now to be discussed within the following chapter.

5. Limitations and Improvements.

The nature of the heat propagation may not always be linear, although within this study the idea if there is no extra external inputs from convection or radiation, it's only conduction. Which, in itself, is a potential limit regarding accuracy of the model, but that comes more from a mathematical prospective than a coding limitation. However, due to the inconsistency of the heat transfer across the plate, based on the user inputs of; time step, grid spacing and actual dimensions of the plate, being able to decide the point that equilibrium is reached proves a difficult task.

As shown within the code, there was an attempt made to aid in controlling when the code can decide the suspected point in which 'equilibrium' is found. This was done via multiple assignments to the variable used, dependent primarily on material used, but with also the number of nodes implemented within the plate. The 'stop' variables used changes the value in which the gap from the top left node of the plate is from the external temperature being inputted. This has clearly been proven through multiple studies to not be the most effective way in which to govern thermal equilibrium within the plate.

Upon logical assessment of the modelling technique, it was thought that using an external check at the end of each iteration (still within the calculation 'while' loop), there was to be a secondary version of the original modelled plate that was constructed of 'zeros'. Throughout the calculations there would be a check criterion that would assess each node across the plate, not just one, and decide if it was within a certain value from the previous iteration, the value would become one. With all values across the secondary model plate being one, the exit criterion of the loop would then be met, and thermal equilibrium found.

Another unfortunate downfall within this modelling technique, is the simple factor that to increase model accuracy, you must incur the heavily increased calculation time.

Although it could be argued this is a fault from the mathematical equations imposed, because of the "error term" function within, is governed by the time step used and the square of the grid spacing.

6. Conclusion.

In summation, it's clear that there will always be some form of error within this mathematical approach of changing the variables into a dimensionless 'approximation' which is collected to be the term 'h' within the equations. This is related to the grid spacing across the plate, which defines the gaps between the temperature points used. It's obvious from even understanding the problem in hand, that the further apart the measured points, the larger the potential of misguided results because of the undefined linearity of heat transfer. Therefore, to enhance the accuracy of the model, the grid spacing must be correctly evaluated to reduce it as much as possible.

The downfall of this, as shown in the analysis, is that because of the computational method used, there is an addition of time taken to obtain an equilibrium across the plate. Meaning there is a potential trade off to be found, by the user, regarding the apparent accuracy required from the model and how long they're willing to compute for. Again, these computation times are seemingly only a downfall, given the computer hardware used.

But in terms of the mathematical approximations that are implemented within the construction of a user friendly 'script', to be used in MATLAB's coding software, they work to a high enough degree of accuracy. The only foreseeable issue within the script construction is the criteria used to decide when equilibrium is found, which can easily be corrected, via realistic inputs or through multiple iterations of conducting the experiment computationally.

7. References.

Crank, J. (1975). *THE MATHEMATICS OF DIFFUSION*. Oxford: CLARENDON PRESS.

Langtangen, H. P., & Linge, S. (2016). *Finite Difference Computing with PDEs - A Modern Software Approach*.

Langtangen, H. P., & Linge, S. (2016). *Finite difference methods for diffusion processes [Preliminary version]*.

University of Münster. (2016). *Diffusion equation*. Retrieved from uni-muenster:
https://www.uni-muenster.de/imperia/md/content/physik_tp/lectures/ws2016-2017/num_methods_i/heat.pdf

8. Appendices.

1. MATLAB Thermal Diffusivity Script.

```
%% Thermal Diffusion model across a thin plate, using
the finite Difference method.
```

1.1. %% Fresh start

```
clc
clear
```

1.2. %% Defined Inputs.

```
% If you wish to modify the script to run the single input
values for
% Aluminium 2024 [Standard material], then Comment out the
section
% underneath and Uncomment the '%%Standard material'
section's variables.
```

```
% Otherwise it's here to allow for a study upon four
materials,
% instead of just one.
```

```
Materials = {'Aluminum 2024 Temper T-351 [Assingment
standard material]', 'Aluminum 6061 Temper-O', 'Titanium', 'SAE
1010 Sheet Steel'};
```

```
[List,~] =
listdlg('ListString',Materials,'PromptString','Please select
the material to be used in the study:
','SelectionMode','Single','ListSize',[350 200]);
```

```
if isempty(List)
    return
elseif List == 1
    k = 143; p = (2.8*(10^3)); c = 795;
elseif List == 2
    k = 180; p = (2.71*(10^3)); c = (1.256*(10^3));
elseif List == 3
    k = 15.6; p = (4.51*(10^3)); c = 544;
elseif List == 4
    k = 63.9; p = (7.832*(10^3)); c = 434;
end
```

```
%           Material property units.
```

```
%      k =      % W/m.C           % Thermal Conductivity
%      p =      % Kg/m^3          % Density
%      c =      % J/Kg.C          % Specific heat
```

```
%           Ambient boundary Temperatures.
```

```
heatin = 50;                %Degrees/C
coldin = 0;                  %Degrees/C
```

```
%           Fourier's number
```

```
Alpha = k/(p*c);
```

1.3. %% Standard material

```
% This is the standard input variables, used to define the
properties of
% the material being using (Aluminium 2024).
```

```
%                               Aluminium Properties.
%           k = 143;                                % W/m.C
% Thermal Conductivity
%           p = (2.8*(10^3));                        % Kg/m^3
% Density
%           c = 795;                                % J/Kg.C
% Specific heat
```

1.4. %% User inputs & Criteriaions.

```
% Section that prompts the user to input a value for the
length and height of the plate in the command window.
% It is also where the user chooses the time step,
understood to be an aera concerning accuracy of the model.
```

```
% This section will also check that the inputted values for
the; length, height and time
% are within bounds that ensure the model retains accuracy,
whilst
% being able to compute within a sensible time period.
```

```
% If the result is out of the criterion [in all cases] an
error message box will appear
% on screen, to notify the user and it will end the script.
This is to be sure
% everything is reset and ready to try again.
```

```
CheckInput = 1;
LeaveLength = 1;
LeaveHeight = 1;
LeaveTime = 1;
```

```
while CheckInput == 1
```

```
    while LeaveLength == 1
```

```
        if List == 1 || List == 2
            x = input("Please input a value for the length
of the plate [Between 3 - 35]: ");
            if x > 35
                error = msgbox('Error. Sorry but the length
is too long and will result in an unessecary calculation
time ', 'Error', 'error');
            elseif x < 3
                error = msgbox('Error. Sorry but the length
is to short to allow for calculation ', 'Error', 'error');
            end
            if x < 3 || x > 35
                LeaveLength = 1;
```

```

        elseif x > 3 || x < 35
            LeaveLength = 0;
        end
    elseif List == 3
        x = input("Please input a value for the length
of the plate [Between 3 - 25]: ");
        if x > 25
            error = msgbox('Error. Sorry but the length
is too long and will result in an unnessecary calculation
time ', 'Error', 'error');
        elseif x < 3
            error = msgbox('Error. Sorry but the length
is to short to allow for calculation ', 'Error', 'error');
        end
        if x < 3 || x > 25
            LeaveLength = 1;
        elseif x > 3 || x < 25
            LeaveLength = 0;
        end
    elseif List == 4
        x = input("Please input a value for the length
of the plate [Between 3 - 30]: ");
        if x > 30
            error = msgbox('Error. Sorry but the length
is too long and will result in an unnessecary calculation
time ', 'Error', 'error');
        elseif x < 3
            error = msgbox('Error. Sorry but the length
is to short to allow for calculation ', 'Error', 'error');
        end
        if x < 3 || x > 30
            LeaveLength = 1;
        elseif x > 3 || x < 30
            LeaveLength = 0;
        end
    end
end

while LeaveHeight == 1

    if List == 1 || List == 2
        y = input("Please input a value for the height
of the plate [Between 3 - 35]: ");
        if y > 35
            error = msgbox('Error. Sorry but the height
is to large and will results in an unnessecary calculation
time ', 'Error', 'error');
        elseif y < 3
            error = msgbox('Error. Sorry but the height
is to short to allow for calculation ', 'Error', 'error');
        end
        if y < 3 || y > 35
            LeaveHeight = 1;
        elseif y > 3 || y < 35
            LeaveHeight = 0;
        end
    end
end

```

```

elseif List == 3
    y = input("Please input a value for the height
of the plate [Between 3 - 25]: ");
    if y > 25
        error = msgbox('Error. Sorry but the height
is to large and will results in an unessecary calculation
time ','Error','error');
    elseif y < 3
        error = msgbox('Error. Sorry but the height
is to short to allow for calculation ','Error','error');
    end
    if y < 3 || y > 25
        LeaveHeight = 1;
    elseif y > 3 || y < 25
        LeaveHeight = 0;
    end
elseif List == 4
    y = input("Please input a value for the height
of the plate [Between 3 - 30]: ");
    if y > 30
        error = msgbox('Error. Sorry but the height
is to large and will results in an unessecary calculation
time ','Error','error');
    elseif y < 3
        error = msgbox('Error. Sorry but the height
is to short to allow for calculation ','Error','error');
    end
    if y < 3 || y > 30
        LeaveHeight = 1;
    elseif y > 3 || y < 30
        LeaveHeight = 0;
    end
end
end

while LeaveTime == 1

    if List == 1 || List == 2
        Time = input("Please input a desired time step
in seconds [Greater than 0.1]: ");
        if Time > 3
            error = msgbox('Error. Sorry the time step
is too high and it`ll cause largely inaccurate results.
','Error','error');
        elseif Time < 0.1
            error = msgbox('Error. Sorry this time step
is too small resulting in to long a time for calculation.
','Error','error');
        end
        if Time < 0.1 || Time > 3
            LeaveTime = 1;
        elseif Time >= 0.1 || Time <= 3
            LeaveTime = 0;
        end
    elseif List == 3

```

```

        Time = input("Please input a desired time step
in seconds [Greater than 0.5]: ");
        if Time > 3
            error = msgbox('Error. Sorry the time step
is too high and it'll cause largely inaccurate results.
','Error','error');
        elseif Time < 0.5
            error = msgbox('Error. Sorry this time step
is too small resulting in to long a time for calculation.
','Error','error');
        end
        if Time < 0.5 || Time > 3
            LeaveTime = 1;
        elseif Time >= 0.5 || Time <= 3
            LeaveTime = 0;
        end
    elseif List == 4
        Time = input("Please input a desired time step
in seconds [Greater than 0.5]: ");
        if Time > 3
            error = msgbox('Error. Sorry the time step
is too high and it'll cause largely inaccurate results.
','Error','error');
        elseif Time < 0.5
            error = msgbox('Error. Sorry this time step
is too small resulting in to long a time for calculation.
','Error','error');
        end
        if Time < 0.5 || Time > 3
            LeaveTime = 1;
        elseif Time >= 0.5 || Time <= 3
            LeaveTime = 0;
        end
    end
end

if LeaveLength == 0 && LeaveHeight == 0 && LeaveTime ==
0
    CheckInput = 0;
end

end

```

1.5. %% Grid Spacing Criterion.

```

% The grid spacing is the section that governs the amount
and placement of
% the nodes that are to be used for the calculations using
the finite
% difference method.
% The stability criterion is to check that the time step
isn't out of range
% because it's governed by the Fourier's number 'Fo'. It's a
dimensionless

```

```

% time step for a change in temperature to occur. Concerning
2D modelling
% the bounds are between 0 < and ? ¼.

GRL = x/50;
GRH = y/50;
GridCheck = 1;
FoCheck = 1;

% These lines are here to aid the user in retaining accuracy
and ensuring
% that the model will work. A result of either; GRH, GRL or
both, whichever is
% larger, will appear in the command window to tell the user
the smallest
% number available to use and if the criterion isn't met
that can try again or restart.

while FoCheck == 1

    while GridCheck == 1

        if GRL > GRH
            GRL; fprintf('GRL = %4.2f',GRL)
        elseif GRH > GRL
            GRH; fprintf('GRH = %4.2f', GRH)
        elseif GRH == GRL
            GRH; fprintf('GRH = GRL = %4.2f', GRH)
        end

        % Again, these lines are here to allow the user to
input their desired
        % grid spacing value whilst simultaneously checking
to be sure it's sensible
        % and will work.

        if Time > 2.5
            Gr = input("\nPlease input a value for the grid
spacing across the plate that is larger than GRH or GRL and
less than 2: ");
        elseif Time < 2.5
            Gr = input("\nPlease input a value for the grid
spacing across the plate that is larger than GRH or GRL and
less than 4: ");
        end
        if Time > 2.5 && Gr > 2
            error = msgbox('Error. The grid spacing is to
large and will take far too long to calculate correctly.
','Error','error');
        elseif Time < 2.5 && Gr > 4
            error = msgbox('Error. The grid spacing is to
large and will take far too long to calculate correctly.
','Error','error');
        elseif Gr > (x/2) || Gr > (y/2)

```

```

        error = msgbox('Error. Sorry but the grid
spacing is to large and will not calculate correctly.
','Error','error');
    elseif Gr < (x/50) || Gr < (y/50)
        error = msgbox('Error. Sorry but the grid
spacing is to small to calculate. ','Error','error');
    elseif Gr >= GRH || Gr >= GRL
        GridCheck = 0;
    end

end

Fo = (Alpha*Time)/(Gr^2);

if Fo >= 0.25 || Fo <= 0
    Fouriers = questdlg({'The Fourier's number does is
not within the limits please change the Time step or Grid
spacing','','If you would like to change the Grid spacing
choosen please click Yes','','If you would like to change
the Time step press No to restart the
script'}, 'Change?', 'Yes', 'No', 'Yes');
    switch Fouriers
        case 'Yes'
            FoCheck = 1; GridCheck = 1;
        case 'No'
            FoCheck = 0;
            return
        end

    elseif Fo <= 0.25 || Fo >= 0
        FoCheck = 0; GridCheck = 0;
    end

end

% This has been placed in the while loop to enable
repetition of this
% section alone, instead of a complete restart.

```

1.6. %% Starting Temperatures.

```

% Starting temperature placement around the plate.

% The following XGr and YGr are the variables that create
the number of
% grid "lines" across the plate and in turn the correct
amount of nodes.

XGr = round((x/Gr)+2,0);
YGr = round((y/Gr)+2,0);
TpsZerosXY = zeros(XGr,YGr)'; % Creates the correct
correlating matrix of zeros.
LinearT = linspace(heatin,coldin,XGr); % function that
will correctly space the decreasing temperature intervals,
dependent of the amount of 'x' nodes.

```

```

TpsZerosXY(1,:) = LinearT;           % Assign's the linspace
values across the "top edge" (row 1, all coloumns) of the
plate.
TpsZerosXY(YGr,:) = LinearT;         % Assign's the linspace
values across the "bottom edge" of the plate.
TpsZerosXY(:,1) = heatin;             % Assign's the heat in
(50°C) to the "left hand edge" of the plate.
TpsZerosXY(:,XGr) = coldin;          % Assign's the cold in
(0°C) to the "right hand edge" of the plate.
PlateStart = TpsZerosXY;             % Simply renaming for
visualization.
Iterations = 1;

```

1.7. %% End criterion [Material dependent]

```

% An additional criterion in the hope to reduce the time
taken to calculate
% the heat transfer.

```

```

i = y+x;
if i <= 40
    f = 2500;
elseif i <= 50
    f = 3000;
elseif i <= 60
    f = 3500;
elseif i > 60
    f = 4000;
elseif i <= 10
    f = 1000;
elseif i <= 20
    f = 800;
elseif i <= 30
    f = 600;
end

if List == 1 && i <= 40
    stop = f;
elseif List == 1 && i <= 50
    stop = f;
elseif List == 1 && i <= 60
    stop = f;
elseif List == 1 && i > 60
    stop = f;
elseif List == 2 && i <= 40
    stop = f/2;
elseif List == 2 && i <= 50
    stop = f/2;
elseif List == 2 && i <= 60
    stop = f/2;
elseif List == 2 && i > 60
    stop = f/2;
elseif List == 3 && i <= 10
    stop = round(f/5);
elseif List == 3 && i <= 20

```

```

        stop = round(f/5);
elseif List == 3 && i <= 40
    stop = round(f/6);
elseif List == 3 && i > 40
    stop = round(f/8);
elseif List == 4 && i <= 10
    stop = round(f/4);
elseif List == 4 && i <= 20
    stop = round(f/4);
elseif List == 4 && i <= 40
    stop = round(f/6);
elseif List == 4 && i > 40
    stop = round(f/8);
end

```

1.8. %% Update criterion [Material dependent]

% This criterion is used within the following loop, as a counter of iterations, used for the % updating of the figure showing the heat transfer through the plate.

```

if List == 1 || List == 2
    TestI = 1000;
elseif List == 3
    TestI = 4000;
elseif List == 4
    TestI = 3000;
end

```

1.9. %% Temperature Propagation

```

tic

while PlateStart(2,2) < (LinearT(1,2) - (LinearT(1,2)/stop))
% Ending criterion check of the loop
    Column = 2; % The row and column are 2, so that it's
starting at the "actual" plate edge.
    while Column < XGr % Stop condition such that it
continues through all columns.
        Row = 2;
        while Row < YGr % Stop condition such that it
continues through all rows.
            Tstart = PlateStart(Row,Column);
            PlateStart(Row,Column) = Fo *
(PlateStart(Row,Column+1) + PlateStart(Row,Column-1) +
PlateStart(Row-1,Column) + PlateStart(Row+1,Column) - (4 -
(1/Fo)) * Tstart );
            Row = Row + 1; % next row
        end
        Column = Column + 1; % next column
    end
end

```

```

    Iterations = Iterations + 1;    % (The update conditions
for the figure) Once reached the input number the figure is
updated.

```

```

    if Iterations == TestI
        figure(1)
        contourf(PlateStart)
        axis off
        color = colorbar;
        color.Label.String = ' Temperature [°C] ';
        colormap(jet)
        Iterations = 1;
    end
end

```

```

toc

```

```

% This will close the last version of the figure and rename
it
% appropriately.

```

```

close(fgure(1))
figure('name','Heat transfer validation test')
contourf(PlateStart)
axis off
Color = colorbar;
Color.Label.String = ' Temperature [°C] ';
colormap(jet)

```

1.10. %% Continue

```

% The question box function is to allow the user to study
the first study
% and to continue when they wish, or stop is wished.

```

```

opS.Interpreter = 'tex';
opS.Default = 'Yes';
Continue = questdlg({'\fontsize{10}The Linear temperature
distribution experiment has completed!','','Would you like
to continue onto viewing the effect of the Gaussian
distribution method.'},'Continue?','Yes','No',opS);

switch Continue
    case 'Yes'
        Gauss = 1;
    case 'No'
        Gauss = 0;
    return
end

```

1.11. %% Gaussian Temperature distribution

```

% Because of the user input variable of the length of the
plate, there is

```

% the change for a division that'll result in a remainder.
Therefore, the following has been implemented to
indefinitely ensure operation.

```

if Gauss == 1                                % Continuation variable from
the 'Continue' question box.
    XGr = x/Gr;
    XGr = round(XGr,0);
    Amplitude = 60;                            % Boundary conditions
variables
    BoundaryEdges = 10;                        %

    TStartPoint = zeros(1,XGr);
    Column = 1;
    StandardDeviation = sqrt((-XGr - ((XGr + 1) /
2))^2) / (log( BoundaryEdges / Amplitude)*2));
    while Column < (XGr + 1)
        TStartPoint(1,Column) = Amplitude * exp(-(Column
- (XGr + 1) / 2)^2) / (2 * (StandardDeviation)^2));
        Column = Column + 1;
    end

end

% The while loops are in place to enable the continual
assignment of the
% symmetrical values to each column.

```

1.12. %% Update criterion [Material dependent]

% Like before, this is an attempt to reduce the time taken
for the script
% to complete the calculation.

```

if List == 1
    its = 550000;
elseif List == 2
    its = 600000;
elseif List == 3
    its = 850000;
elseif List == 4
    its = 700000;
end

```

1.13. %% Heat Transfer iterations

% Starting variables

```

TStartPoint;                                % Boundary condition
temperatures.
YGr = round((y/Gr)+1,0);                    % Additional row to
assign boundary conditions.
StartPlate = zeros(YGr,XGr);                % Starting matrix sized
'Plate size'.
StartPlate(1,:) = TStartPoint;              % Boundary condition
assignment.

```

```

Iterations = 1; % Criterion.
YG = round(YGr/2); %
XG = round(XGr/2); % -

if YGr > XGr % Update figure
criterion.
    IT = round(its/(YGr*2),0);
elseif XGr > YGr
    IT = round(its/(XGr*2),0);
elseif XGr == YGr
    IT = round(its/(XGr*2),0);
end % -

tic

while StartPlate(YG,XG) < (TStartPoint(1,XG)*0.62)
    Column = 2;
    while Column < XGr % Calculation through
the columns of the plate.
        Row = 2;
        while Row < YGr % Calculation through
the rows of the plate.
            TStart = StartPlate(Row,Column); %
Main area of the plate.
            StartPlate(Row,Column) = Fo *
            (StartPlate(Row,Column+1) + StartPlate(Row,Column-1) +
            StartPlate(Row+1,Column) + StartPlate(Row-1,Column) - (4-
            (1/Fo)) * TStart);
            TStart = StartPlate(Row,1); %
Left hand side of the plate.
            StartPlate(Row,1) = Fo * (StartPlate(Row,1+1)
+ StartPlate(Row,1+1) + StartPlate(Row+1,1) +
            StartPlate(Row-1,1) - (4-(1/Fo)) * TStart);
            TStart = StartPlate(Row,XGr); %
Right hand side of plate.
            StartPlate(Row,XGr) = Fo *
            (StartPlate(Row,XGr-1) + StartPlate(Row,XGr-1) +
            StartPlate(Row+1,XGr) + StartPlate(Row-1,XGr) - (4-
            (1/Fo)) * TStart);

            Row = Row + 1;
        end
        % Bottom row of the plate
        StartPlate(YGr,Column) = Fo *
        (StartPlate(YGr,Column+1) + StartPlate(YGr,Column-1) +
        StartPlate(YGr-1,Column) + StartPlate(YGr-1,Column) - (4-
        (1/Fo)) * TStart);

        Column = Column + 1;
    end
    % Bottom corners of the plate
    StartPlate(YGr,1) = Fo * (StartPlate(YGr,1+1) +
    StartPlate(YGr,1+1) + StartPlate(YGr-1,1) +
    StartPlate(YGr-1,1) - (4-(1/Fo)) * TStart);

```

```

        StartPlate(YGr,XGr) = Fo * (StartPlate(YGr,XGr-1) +
StartPlate(YGr,XGr-1) + StartPlate(YGr-1,XGr) +
StartPlate(YGr-1,XGr) - (4-(1/Fo)) * TStart);

        Iterations = Iterations + 1;
    if Iterations == IT
        figure(10)
        Plotting = flipud(StartPlate);
        contourf(Plotting)
        axis off
        colors = colorbar;
        colors.Label.String = ' Temperature [°C] ';
        colormap(jet)
        Iterations = 1;
    end
end

toc

% This will close the last version of the figure and
again, rename it
% appropriately.

close(figure(10))
figure('name','Heat propagation with Gaussian temperature
distribution method')
contourf(Plotting);
axis off
Colors = colorbar;
Colors.Label.String = ' Temperature [°C] ';
colormap(jet)

FinishStruct.Interpreter = 'tex';
FinishStruct.WindowStyle = 'modal';
Finish = msgbox({'\fontsize{10}Gaussian Experiment has
completed!','','Press OK to finish the
experiment.'},'Completed.','help',FinishStruct);
uiwait(Finish)

```