

Heat Transfer in a 2-Dimensional Plate: A Computational Solution

COMPUTATION METHODS (ACMA6012)

Joshua U'Dell
BOC A LINDE COMPANY
UNIVERSITY OF WALES TRINITY SAINT DAVID

Contents

Contents.....	1
1.0 Introduction	2
2.0 Code's details and explanations.	3
2.1 MATLAB Workspace organisation.....	3
2.2 Defining the material properties.	3
2.3 Plate dimensions and grid spacing.	3
2.4 Time step and checking against Fourier number.	4
2.5 Generate a matrix for the initial state.	5
2.6 Boundary conditions.	6
2.7 Setting convergence criteria for the calculation.....	6
2.8 Calculation T2 using Finite Difference Method.	7
2.9 Visualising the temperature matrix T2.....	8
2.10 Outputting results from computed solution.....	9
2.11 Analytical solution check.	9
3.0 Sensitivity analysis of the code.	11
3.1 Grid spacing.	11
3.2 Time step.	12
3.3 Tolerance, Convergence Criteria.	13
4.0 Model limitations.	14
5.0 Conclusion.	14
6.0 References.....	15
7.0 Appendices.	16
7.1 MATLAB Script.	16

1.0 Introduction

This report will detail and explain how a computation solution has been generated to calculate the temperature distribution within an aluminium plate over time. The model will use the Explicit Finite Difference Method (EFDM) and consider the aluminium plate a two-dimensional plane.

The report will detail the design of the MATLAB script and how it calculates the required temperature diffusion matrix. It will also examine the mathematics behind the script and any limitations that might arise from using a computational approach to finding the solution.

Various tests have been run using the script, and the results and conclusions are clearly shown in the report.

2.0 Code's details and explanations.

2.1 MATLAB Workspace organisation.

The code starts with the `clc` command, which removes all the text from the command window, leaving it empty. The following command in the code is `clear`, which eliminates all variables in the current workspace and also removes them from the system's memory. Lastly, this section has the `close all` command, which closes any open figures created by previous scripts.

```
clc
clear
close all
```

Figure 1 MATLAB Code extract (Appendix 1)

This is a generic start to any MATLAB script as it functionally tidies up the environment and prepares it for a fresh script.

2.2 Defining the material properties.

The material properties of the aluminium plate are;

- Conductivity: $k = 143 \text{ W/m.C}$
-
- Specific Heat: $c = 795 \text{ J/kg.C}$

These numerical values are stored as variables in MATLAB so they can be called upon later in the script when required; this is done by giving the variable a unique name and using the `=` function to assign a value to the new variable.

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

Figure 2 MATLAB Code extract (Appendix 1)

A semicolon (;) is applied at the end of each line so the command does not appear in the command window; this keeps the command window clear and will only display text when required.

The thermal diffusivity of the material is calculated using the below formula.

$$a = \frac{k}{\rho \times c} \text{ (Roger \& Mayhew, 1980)}$$

This is stored in a variable using the equals sign followed by the relevant formula.

```
%Calculate alpha for aluminium
a = k/(p*c);
```

Figure 3 MATLAB Code extract (Appendix 1)

2.3 Plate dimensions and grid spacing.

The user is required to input the length and width of the aluminium plate in question; these values are stored in respective variables. The `input` function is used to display text in the command window, and it is paused until the user has typed in a value and enter is pressed. This is repeated twice for width and length (X and Y).

```
%get the size of the plate from the user
PlateSizeX = input("Enter the x size of the aluminium plate in meters: ");
PlateSizeY = input("Enter the y size of the aluminium plate in meters: ");
```

Figure 4 MATLAB Code extract (Appendix 1)

The user is then asked via another input function to set the grid spacing; this is how far apart each element in the grid will be. This value must be divided exactly into both the X and Y dimensions.

```
%Get the grid spacing from the user
h = input("Enter the grid spacing size for this simulation in meters: ");
```

Figure 5 MATLAB Code extract (Appendix 1)

A while loop is then used to check if the inputted grid spacing divides with no remainder into both the X and Y dimensions.

```
%check the grid spacing fits into the plate dimensions
hCheck = false;
while hCheck==false
    if rem(PlateSizeX,h)==0 && rem(PlateSizeY,h)==0 && h<PlateSizeX && h<PlateSizeY
        hCheck=true;
    else
        h = input("Grid spacing does not divide into plate sizing, Enter a different grid spacing size " + ...
            "for this simulation in meters: ");
    end
end
```

Figure 6 MATLAB Code extract (Appendix 1)

The rem function will output the number of remainders after a division has been applied, and an && function has been used within the if statement so that both statements must be true for the hcheck variable to become true. The while loop allows this to run until the user enters an appropriate grid spacing value. If the user enters a grid spacing that leaves a remainder for either of the plate dimensions, the else script will run from the if statement, and the user will be asked to input a new grid spacing, and the while loop will run again as the hcheck variable would remain false.

2.4 Time step and checking against Fourier number.

This section of the script requires the user to do the time step for the equation, which is done using the input function. The time step must result in the Fourier number being less than 0.25. A model with a Fourier number greater than 0.25 is said to be unstable.

The Fourier number is calculated using the below formula;

$$Fo = \frac{a * Dt}{h^2}$$

The formula can be rearranged as follows to calculate the minimum time step that can be inputted.

$$0.25 = \frac{a * DtMax}{h^2}$$

$$\therefore DtMax = \frac{h^2}{4a}$$

Calculating the max time step gives the user a reasonable starting point when selecting the desired time step. Once the max time step has been calculated, it is presented to the user within the command window, and the user is asked to input the desired time step. The script will then check this is less than equal to 0.25. A while loop is used again to achieve this.

```

%Get the time step from the user
Dtmax = h^2/(4*a);
fprintf("Maximum time step for a stable simulation is %g \n",Dtmax)
Dt = input("Enter the desired time step for this simulation in seconds (1 hour = 3600 seconds)");
%calculate Fourier Number
FoCheck = false;
Fo = (a * Dt) / h^2;
%check the time step does not cause stability issues, Fourier number 0<Fo<=0.25
fprintf("Fourier Number for the given information is %0.4f \n",Fo)
while FoCheck==false
    if Fo > 0 && Fo <= 0.25
        FoCheck=true;
    else
        Dt = input("Time step is causing stability issues, enter a smaller time step here in seconds or " + ...
            "stop the script and start again with a smaller grid spacing. (1 hour = 3600 seconds) new " + ...
            "time step in seconds: ");
        Fo = (a * Dt) / h^2;
        fprintf("Fourier Number for the given information is %0.4f \n",Fo)
    end
end

```

Figure 7 MATLAB Code extract (Appendix 1)

2.5 Generate a matrix for the initial state.

The two-dimensional plate will be represented in the script as a matrix; the size of this matrix depends on the grid spacing chosen in section 3.3 of this report. Below is a diagrammatic representation of the plate and how the matrix represents it.

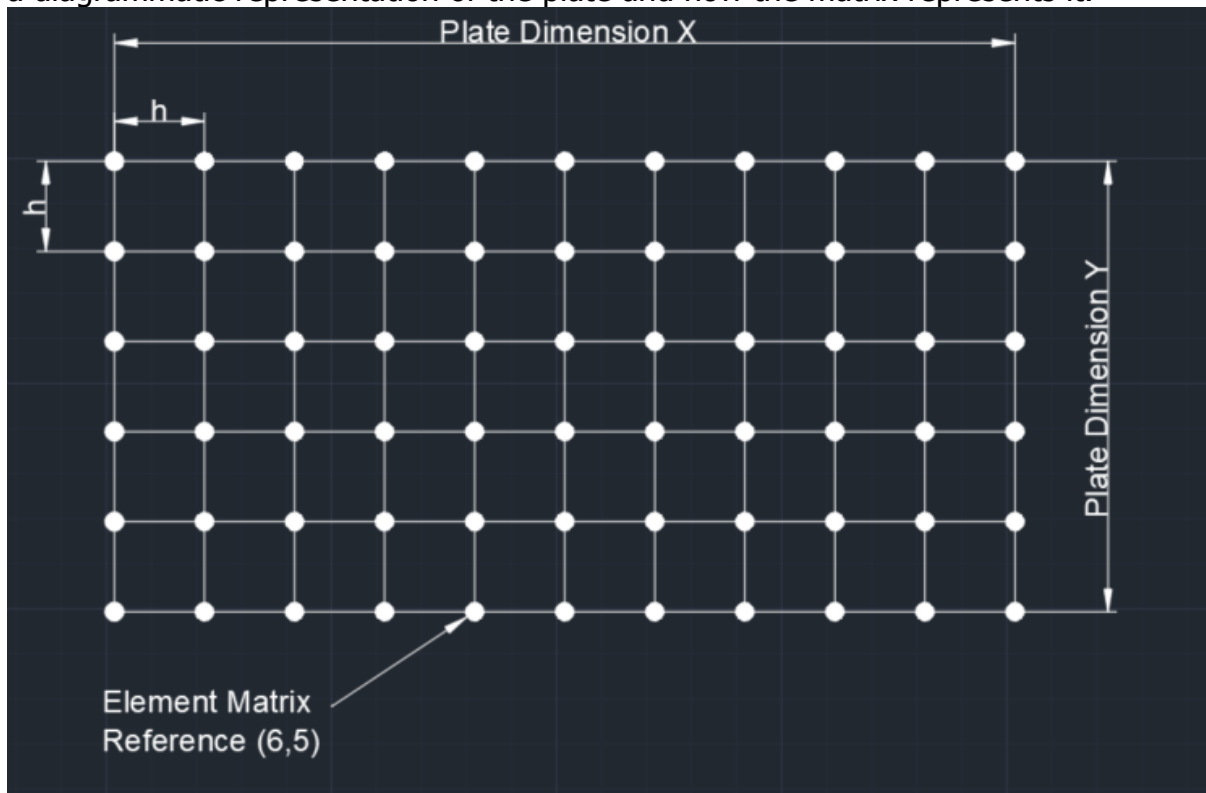


Figure 8 Matrix element spacing illustration (U'Dell J. 2025)

The matrix is made up of columns and rows; the amount of elements in each row is calculated by dividing the plate dimension by the grid spacing and then adding one element; the number of elements in the columns is computed using the same formula.

Once the matrix size is calculated, the script will create a matrix of the correct size but with each element value set to zero. This is done to make the script more efficient; if the matrix size is not set, then MATLAB will use more memory when using the matrix later on in the script.

```
%Generate Matrix for initial state
NoRows = (PlateSizeY/h)+1;
NoCols = (PlateSizeX/h)+1;
T1 = zeros(NoRows,NoCols);
```

Figure 9 MATLAB Code extract (Appendix 1)

The initial matrix is stored as T1. The Plate Size X and Y stores the plate dimensions in meters and the NoCols and NoRows stores the number of elements in each row and column. H is the grid spacing in Meters.

2.6 Boundary conditions.

The user is asked for the initial boundary conditions for the calculation as a temperature in Degrees Celsius; the script will apply this temperature to the first and the negative of this temperature to the last row of the matrix. The input function is used to gather the value from the user. To apply the value to the initial state matrix (T1), the matrix needs to be modified; MATLAB has many functions to modify matrices. To specify an element in a matrix, MATLAB uses a column and then a row coordinate system to modify an entire row or column. The colon (:) operator is used.

```
%Get boundary temprature from the user
T0 = input("Enter the Boundary temperature in deg C: ");
%Adjust T1 with boundary condition
T1(1,:) = T0;
T1(NoRows,:) = -T0;
T2 = T1;
```

Figure 10 MATLAB Code extract (Appendix 1)

The last line in the above script applied the same initial state stored in T1 to a new matrix called T2. Later in the script, T2 will be used to calculate the change in temperature after the time step.

2.7 Setting convergence criteria for the calculation.

The script is designed to calculate the final temperature distribution with set time steps once the temperature has converged. Convergence means the temperature value is no longer changing significantly enough to affect the calculation. To give the script something to aim for, a convergence criterion is applied; this is a numerical value decided by the user; this value is the minimum difference in temperature from state 1 to state 2 in the whole matrix; this means that the script will continue to calculate changes in the matrix from T1 to T2 with the set time step until every single element in T2 has a temperature value plus or minus the convergence criteria of the exact element location in T1. For instance, if the convergence criteria are set at 1, then the calculation will finish when each element is within 1 Degree Celsius of the same element location on the previous time step state.

Below is the section of the script that asks the user for convergence criteria value and stores it as a variable.

```
%Get convergence criteria from the user
CC = input("please enter your required convergence criteria, example 0.01: ");
```

Figure 11 MATLAB Code extract (Appendix 1)

2.8 Calculation T2 using Finite Difference Method.

The Finite Difference Method (FDM) is a well-known and widely used engineering technique. It allows engineers and mathematicians to approximate the answer to complex differential equations by approximating the derivatives in the equation. The conduction in solids is described in the below equation.

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

Figure 12 Conduction in solids Heat Equation (Croft D.R, Lilley D.G, 1977)

Where

- t = time in seconds
- T = temperature in degrees Celsius
- A = k/(p.c) is the thermal diffusivity in
- K = thermal conductivity in
- P = the density in
- C = specific heat in

Using Taylor series expansion, the heat equation is transformed into a set of solvable algebraic equations. Central differences and forward differences are used to

discretise $\frac{\partial T}{\partial t}$.

The discretised values for these are seen below.

$$\left\{ \begin{aligned} \frac{\partial^2 T}{\partial x^2} &= \frac{T_E^t - 2T_P^t + T_W^t}{h^2} \\ \frac{\partial^2 T}{\partial y^2} &= \frac{T_N^t - 2T_P^t + T_S^t}{h^2} \end{aligned} \right.$$

and

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

In the above equations, N, E, S, and W stand for the elements north, east, south, and west of point (P) at a distance of h.

The section of the script that handles this equation is encased within a while loop that will keep running until the convergent criteria are met. This is done using a variable called EndingCondition. While this variable is 0, the calculation will add another time step until the convergence criteria are met and the EndingCondition variable is set to 1.

Within the above-mentioned while loop, there is a for loop; a for loop will loop a set code for a set number of runs; in this instance, that amount of runs is equal to the number of rows in matrix T1 from the second row to the second to last row. This is due to the fact the first and last rows have a set temperature that will not change (T0). The current row is stored in the variable i. Within this for loop, another for loop runs through each column in the current row. This arrangement of a for loop within another for loop enables the code to perform the next calculation on each required element; once each component has been calculated in T2 from the initial state in T1, they can be compared. If the convergence criteria are not met, the while loop is still valid, T1 becomes equal to T2, and then T2 is recalculated.

The *co Density*: $\rho = 2.8 \times 10^3 \text{ kg/m}^3$ convergence criteria are assessed by first creating a new Matrix that is equal to the absolute value of T1 minus T2 and then using the max function within MATLAB to output the largest value within this matrix; using the absolute value allows the max function to find the largest difference in temperature values from T1 to T2 whether the value is larger or smaller.

```
%calculate temperature in T2 for each node in matrix
IterationCount = 0;
EndingCondition = 0;
while EndingCondition == 0
    IterationCount = IterationCount + 1;
    for i = 2:NoRows-1
        for j = 1:NoCols
            % Calculate T2 Value at point (i,j)
            if j == 1
                %calculate point temp value replacing west value with east value
                T2(i,j) = Fo*(T1(i-1,j)+T1(i+1,j)+T1(i,j+1)+T1(i,j-1)-(4-(1/Fo))*T1(i,j));
            elseif j == NoCols
                %calculate point temp value replacing east value with west value
                T2(i,j) = Fo*(T1(i-1,j)+T1(i+1,j)+T1(i,j-1)+T1(i,j+1)-(4-(1/Fo))*T1(i,j));
            else
                %calculate point temp value
                T2(i,j) = Fo*(T1(i-1,j)+T1(i+1,j)+T1(i,j+1)+T1(i,j-1)-(4-(1/Fo))*T1(i,j));
            end
        end
    end
end
```

Figure 13 MATLAB Code extract (Appendix 1)

This value is then stored in a variable called MaxDiff and compared to the Convergence Criteria, and if it is smaller than the convergence criteria, then the calculation has converged and can end.

```
%check if T1 and T2 are within the convergence criteria
TCC = abs(T2-T1);
MaxDiff = max(TCC(:));
if MaxDiff < CC
    EndingCondition = 1;
else
```

Figure 14 MATLAB Code extract (Appendix 1)

2.9 Visualising the temperature matrix T2.

As the script calculates the new temperature values in T2, the script prints out a line in the command window that states the current iteration number; this is the number of times the while loop has been cycled through. Followed by the convergence target value inputted by the user and the current MaxDiff value, along with a percentage of how close the script is to converging. The below code outputs this at the end of every while loop cycle.

```
fprintf("Current iteration %0g ", IterationCount)
```

Figure 15 MATLAB Code extract (Appendix 1)

```
fprintf("Convergence target is %g, current value is %0.5f, Solver is at %0.2f percent " + ...
    "completion\n", CC, MaxDiff, (CC/MaxDiff)*100)
```

Figure 16 MATLAB Code extract (Appendix 1)

The script also creates a visual graph that displays the latest T2 values in a colour projection. This is completed using the contour function; T2 Matrix is flipped so that

the bottom left of the graph displays the original matrix element (1,1). This is so that the grid references line up with the axis on the graph.

The visual colour is set to jet, this is a built-in setting that can be adjusted in MATLAB, the jet colour format depicts cold as blue and hot as red. The colorbar function is used to display the colour indicator key on the left of the visual, this helps the user see what colour is represented by what value. The title, X and Y axis labels are all added and the visual is loaded.

```
% Visualize the temperature grid
figure(1);
contourf(flip(T2));
colorbar;
title('Temperature Distribution on a 2D Plate computational method');
xlabel('X');
ylabel('Y');
colormap('jet');
```

Figure 17 MATLAB Code extract (Appendix 1)

2.10 Outputting results from computed solution.

Once the convergence criteria has been met the while loop will no longer run and the script will continue to the next line of code. If this happens the script will then calculate the time to reach convergence by multiplying the iteration count by the time step and then display this in the command window to the user.

```
fprintf("\nSolver finished\nEquilibrium met in %g iterations\n", IterationCount)
%calculate time to reach convergence
CTime = IterationCount*Dt;
fprintf("Equilibrium met in %g seconds (%0.2f Hours, %0.2f Days)\n", CTime, CTime/3600, CTime/85400)
```

Figure 18 MATLAB Code extract (Appendix 1)

2.11 Analytical solution check.

To verify the results from the computation solution the code will then compare the figures in the matrix to those of the analytical solution, not all heat problems have an analytical solution but in this case, there is one.

C.Y., Ma C.C., 2001),

$$T(x, y) = -T_0 + \frac{2 \cdot T_0}{Y_{dim}} \cdot y + \sum_{k=1}^{\infty} \left(\frac{1}{(100 \cdot k \cdot \pi)^2} \cdot [1 - (-1)^k] \cdot \frac{\sin\left(\frac{k \cdot \pi \cdot y}{Y_{dim}}\right)}{\sinh\left(\frac{k \cdot \pi \cdot X_{dim}}{Y_{dim}}\right)} \cdot \left[\cosh\left(\frac{k \cdot \pi \cdot x}{Y_{dim}}\right) - \cosh\left(\frac{k \cdot \pi \cdot (x - X_{dim})}{Y_{dim}}\right) \right] \right)$$

Figure 19 Analytic solution (Chang C.Y., Ma C.C., 2001),

This equation needs to be applied to each element in a matrix that represents the two-dimension aluminium plate. This is done by creating a new matrix T with the same number size as T1 and T2, the boundary conditions are then applied the same as the computational solution.

The analytical solution works by calculation the temperature at each node and adding a summation to this value, the summation is set from K=1 to k=infinity, this is not possible in practice, so the upper limit of k is set to 50 in this calculation. Each term in the summation is split out to be calculated separately purely for ease of coding. The summation is set inside a for loop that runs for 50 loops.

The analytical solution is then visualised in the same graphical format at the computational solution using the same functions.

```
%Check against analytical solution
T = zeros(NoRows,NoCols);
T(1,:) = -T0;
T(NoRows,:) = T0;
xdim = PlateSizeX;
ydim = PlateSizeY;
for i = 2:NoRows-1
    for j = 1:NoCols
        x = (j-1)*h;
        y = (i-1)*h;
        T(i,j) = -T0+((2*T0)/(ydim))*y;
        %Reset s for next node
        s=0;
        for k = 1:50
            term1 = 1/(100*k*pi)^2;
            term2 = 1-(-1)^k;
            term3 = sin((k*pi*y)/ydim)/sinh((k*pi*xdim)/ydim);
            term4 = cosh((k*pi*x)/ydim)-cosh((k*pi*(x-xdim)/ydim));
            s = s + term1*term2*term3*term4;
        end
        T(i,j) = T(i,j) + s;
    end
end

T=flip(T);

figure(2)
contourf(flip(T));
colorbar;
title('Temperature Distribution on a 2D Plate analytical method');
xlabel('X');
ylabel('Y');
colormap('jet');
```

Figure 20 MATLAB Code extract (Appendix 1)

Both solutions are then compared using the absolute function again and the max function as can be seen below. This is then displayed in the same graphical format so the user can see where the largest variations are in the matrices. Lastly the largest variation is printed in the command window.

```
TN = abs(T2-T);
TMaxDiff = max(TN(:));

figure(3)
contourf(flip(TN));
colorbar;
title('Value difference computational vs analytical methods');
xlabel('X');
ylabel('Y');
colormap('jet');
fprintf('The maximum difference between the analytical and computer model answer is %0.4f°C\n',TMaxDiff)
```

Figure 21 MATLAB Code extract (Appendix 1)

3.0 Sensitivity analysis of the code.

This section will look at the results from various test that have been run on the code to see how varying different parameters effects the results.

3.1 Grid spacing.

Firstly the grid spacing will be assessed, each calculation will comprise of a plate measuring 12 meters by 12 meters and a T0 of 50 deg C. The table below shows the results from the calculations.

Grid Spacing h	Time step (Fixed)	Tolerance (Fixed)	thermal equilibrium time (in seconds)
4	150	0.001	282600
3	150	0.001	278700
2	150	0.001	255150
1.5	150	0.001	258600
1.2	150	0.001	253200
1	150	0.001	254700
0.75	150	0.001	253350
0.6	150	0.001	252600
0.5	150	0.001	252300
0.4	150	0.001	251700
0.3	150	0.001	251850
0.25	150	0.001	251700
0.2	150	0.001	251700

Figure 22 Grid Spacing Vs Thermal Equilibrium Table (U'Dell J. 2025)

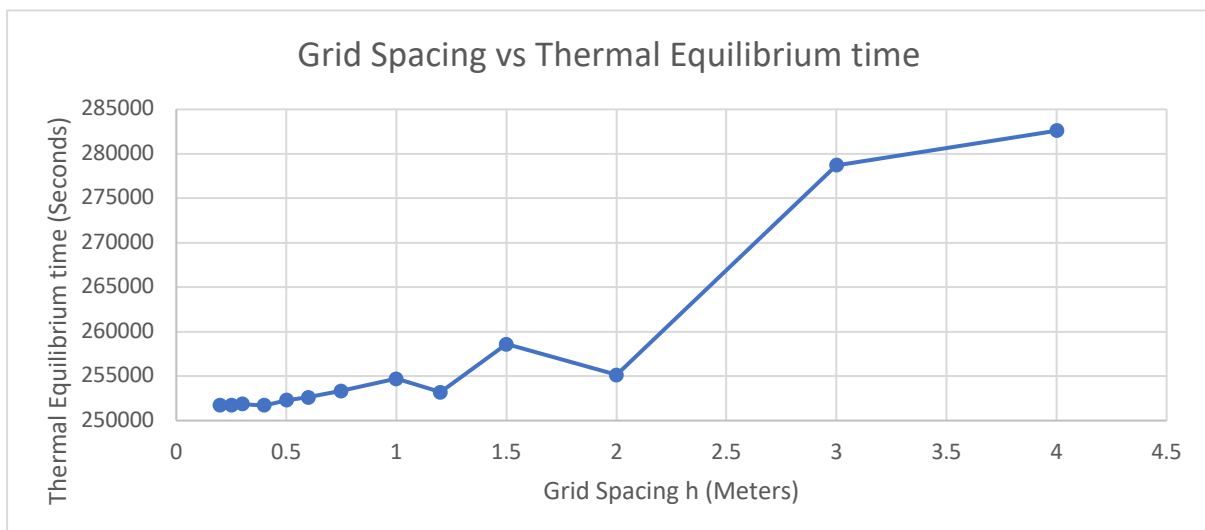


Figure 23 Grid Spacing Vs Thermal Equilibrium Graph (U'Dell J. 2025)

As can be seen from the above data, the smaller the grid spacing and therefore the h value, the less time the plate takes to reach thermal equilibrium. Having a smaller grid spacing will also cause the calculation to take longer to run as there will be

more elements to calculate on. The smaller the grid spacing the finer the mesh is said to be.

From the graph it can be said that there is not much difference from a 0.4 to a 0.2 meter grid spacing, a 0.2 mesh would have twice as many elements to calculate and therefore take longer to compute. This should be taken into account when selecting the grid spacing value.

3.2 Time step.

The time step is the theoretical time between each iteration, the below table and graph show the results from various time steps on the same plate.

Grid Spacing h (fixed)	Time step	Tolerance (Fixed)	thermal equilibrium time (in seconds)
0.75	2000	0.0001	528000
0.75	1500	0.0001	513000
0.75	1000	0.0001	492000
0.75	800	0.0001	480000
0.75	600	0.0001	463800
0.75	400	0.0001	441200
0.75	200	0.0001	402000
0.75	100	0.0001	362400
0.75	50	0.0001	322600
0.75	25	0.0001	282775

Figure 24 Time Step Vs Thermal Equilibrium Table (U'Dell J. 2025)

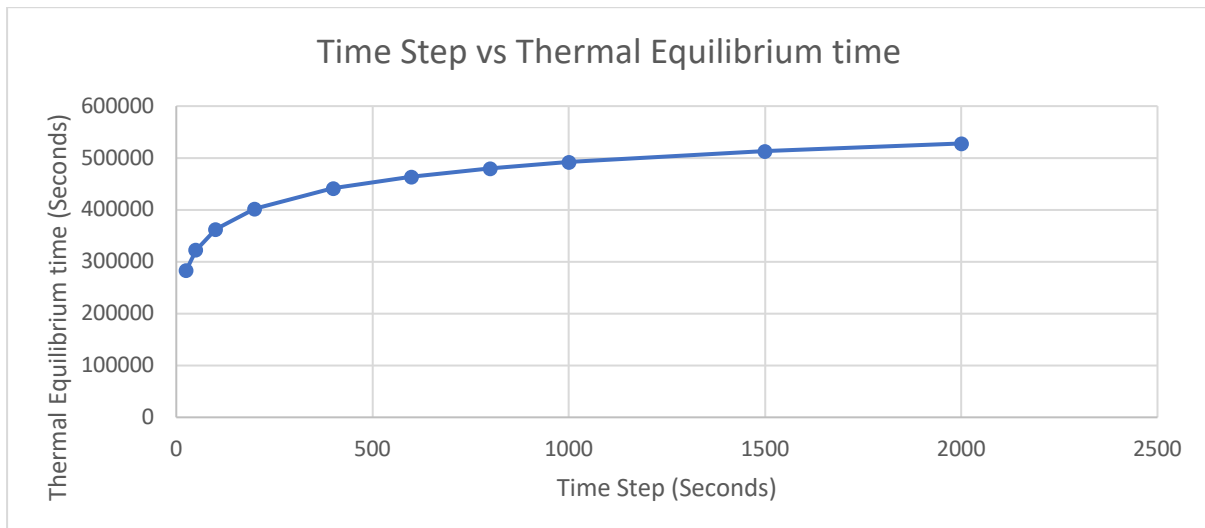


Figure 25 Time Step Vs Thermal Equilibrium Graph (U'Dell J. 2025)

The above data shows a clear correlation between time step value and time to reach thermal equilibrium, the smaller the time step the sooner the plate reaches thermal equilibrium, this is due to how time to reach thermal equilibrium is calculated. It is simply the number of iterations multiplied by the time step. If the time step is

smaller there will be more iterations, but thermal equilibrium will be reached with a higher accuracy in time and therefore the time will be shorter.

3.3 Tolerance, Convergence criteria.

As discussed in the code explanation, the convergence criteria or tolerance is the absolute difference value between T1 and T2 required for the calculation to be considered complete. The smaller the tolerance value the longer the simulation will need to run to converge and therefore the longer the theoretical thermal equilibrium time will be. This can be seen in the data below.

Grid Spacing h	Time step (Fixed)	Tolerance	thermal equilibrium time (in seconds)
1	500	0.1	57500
1	500	0.01	191000
1	500	0.001	324000
1	500	0.0001	457500
1	500	0.00001	590500
1	500	0.000001	723500

Figure 26 Tolerance Vs Thermal Equilibrium Graph (U'Dell J. 2025)

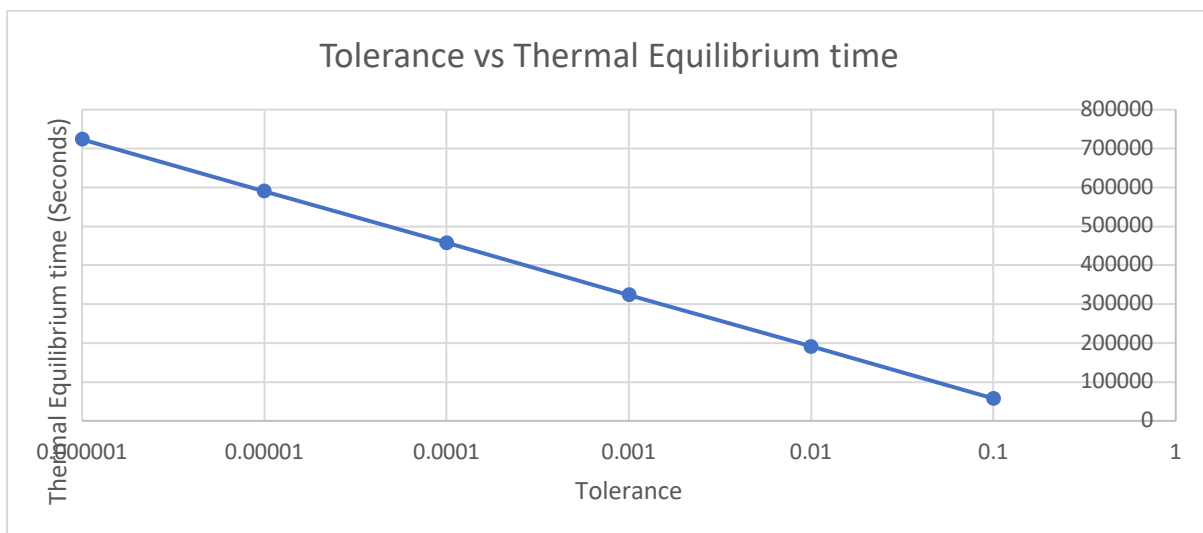


Figure 27 Tolerance Vs Thermal Equilibrium Graph (U'Dell J. 2025)

4.0 Model limitations.

This model, like all, has various limitations; there are simple limitations such as round-off and truncation errors, these are due to the fact that each equation completed by MATLAB will be rounded up to a set number of decimal points, and as more and more calculations are completed these rounding errors compile and build up.

There are also limitations with the fundamental set-up of the model, the equations used all assume that there is zero heat loss to the environment and that the material is perfect, to say the material has perfect heat transfer and exact material properties all the way through just isn't true to the real world. In reality, there are defects in materials, and however small, they can all lead towards slight anomalies in how the material behaves in the real world.

5.0 Conclusion.

In conclusion, the model will calculate the required temperature values with good precision, but it is not perfect; it has its limitations, and the user is required to make sound engineering decisions on the values used. This can lead to poor results due to poor values being inputted for a quick result. Any theoretical solution should be tested against real world results and verified by non biased peers.

There is large emphasis on the user to input appropriate values for grid spacing, time step and convergence criteria.

6.0 References.

Chang C.Y., Ma C.C. (2001). Transient thermal conduction analysis of a rectangular plate with multiple insulated cracks by alternating method. *International Journal of Heat and Mass Transfer*, 44, 2423-2437.

Croft D.R, Lilley D.G. (1977). *Heat Transfer Calculations using the Finite Difference Equations*. London: Applied Science Publishers Ltd.

Rogers & Mayhew (1980) 4th edition, *Engineering Thermodynamics Work & Heat Transfer*, Addison Wesley Longman Limited

7.0 Appendices.

7.1 MATLAB Script.

```
%Author: Joshua U'Dell
%Creation Date: 13/11/2024
%Last Edited Date: 27/01/2025
%This script will calculate the temperature distribution within an
aluminium
%plate through time
%Clean command window and workspace
clc
clear
close all
%Define Aluminium material properties
% • Conductivity: k = 143 W/m.C
% • Density: r = 2.8 ´ 10³ kg/m³
% • Specific Heat: c = 795 J/kg.C
k = 143;
p = 2.8 * 10^3;
c = 795;
%Calculate alpha for aluminium
a = k/(p*c);
%get the size of the plate from the user
PlateSizeX = input("Enter the x size of the aluminium plate in meters: ");
PlateSizeY = input("Enter the y size of the aluminium plate in meters: ");
%Get the grid spacing from the user
h = input("Enter the grid spacing size for this simulation in meters: ");
%check the grid spacing fits into the plate dimensions
hCheck = false;
while hCheck==false
    if rem(PlateSizeX,h)==0 && rem(PlateSizeY,h)==0 && h<PlateSizeX &&
h<PlateSizeY
        hCheck=true;
    else
        h = input("Grid spacing does not divide into plate sizing, Enter a
different grid spacing size " + ...
"for this simulation in meters: ");
    end
end
%Get the time step from the user
Dtmax = h^2/(4*a);
fprintf("Maximum time step for a stable simulation is %g \n",Dtmax)
Dt = input("Enter the desired time step for this simulation in seconds (1
hour = 3600 seconds)");
%calculate Fourier Number
FoCheck = false;
Fo = (a * Dt) / h^2;
%check the time step does not cause stability issues, Fourier number
0<Fo<=0.25
fprintf("Fourier Number for the given information is %0.4f \n",Fo)
while FoCheck==false
    if Fo > 0 && Fo <= 0.25
        FoCheck=true;
    else
        Dt = input("Time step is causing stability issues, enter a smaller
time step here in seconds or " + ...
```

```

        "stop the script and start again with a smaller grid spacing.
(1 hour = 3600 seconds) new " + ...
        "time step in seconds: ");
        Fo = (a * Dt) / h^2;
        fprintf("Fourier Number for the given information is %0.4f \n",Fo)
    end
end
%Generate Matrix for initial state
NoRows = (PlateSizeY/h)+1;
NoCols = (PlateSizeX/h)+1;
T1 = zeros(NoRows,NoCols);
%Get boundary temprature from the user
T0 = input("Enter the Boundary temperature in deg C: ");
%Adjust T1 with boundary condition
T1(1,:) = T0;
T1(NoRows,:) = -T0;
T2 = T1;
%Get convergence criteria from the user
CC = input("please enter your required convergence criteria, example 0.01:
");
%calculate temperature in T2 for each node in matrix
IterationCount = 0;
EndingCondition = 0;
while EndingCondition == 0
    IterationCount = IterationCount + 1;
    for i = 2:NoRows-1
        for j = 1:NoCols
            % Calculate T2 Value at point (i,j)
            if j == 1
                %calculate point temp value replacing west value with east
value
                T2(i,j) = Fo*(T1(i-1,j)+T1(i+1,j)+T1(i,j+1)+T1(i,j+1)-(4-
(1/Fo))*T1(i,j));
            elseif j == NoCols
                %calculate point temp value replacing east value with west
value
                T2(i,j) = Fo*(T1(i-1,j)+T1(i+1,j)+T1(i,j-1)+T1(i,j-1)-(4-
(1/Fo))*T1(i,j));
            else
                %calculate point temp value
                T2(i,j) = Fo*(T1(i-1,j)+T1(i+1,j)+T1(i,j+1)+T1(i,j-1)-(4-
(1/Fo))*T1(i,j));
            end
        end
    end
    fprintf("Current iteration %0g ",IterationCount)
    %disp(T2)
    %check if T1 and T2 are within the convergence criteria
    TCC = abs(T2-T1);
    MaxDiff = max(TCC(:));
    if MaxDiff < CC
        EndingCondition = 1;
    else
        % Visualize the temperature grid
        figure(1);
        contourf(flip(T2));
        colorbar;
        title('Temperature Distribution on a 2D Plate computational
method');
    end
end

```

```

        xlabel('X');
        ylabel('Y');
        colormap('jet');
        %Replace T1 with T2
        T1 = T2;
        fprintf("Convergence target is %g, current value is %0.5f, Solver
is at %0.2f percent " + ...
        "completion\n",CC,MaxDiff,(CC/MaxDiff)*100)
    end
end

fprintf("\nSolver finished\nEquilibrium met in %g
iterations\n",IterationCount)
%calculate time to reach convergence
CTime = IterationCount*Dt;
fprintf("Equilibrium met in %g seconds (%0.2f Hours, %0.2f
Days)\n",CTime,CTime/3600,CTime/85400)

%Check against analytical solution
T = zeros(NoRows,NoCols);
T(1,:) = -T0;
T(NoRows,:) = T0;
xdim = PlateSizeX;
ydim = PlateSizeY;
for i = 2:NoRows-1
    for j = 1:NoCols
        x = (j-1)*h;
        y = (i-1)*h;
        T(i,j) = -T0+((2*T0)/(ydim))*y;
        %Reset s for next node
        s=0;
        for k = 1:50
            term1 = 1/(100*k*pi)^2;
            term2 = 1-(-1)^k;
            term3 = sin((k*pi*y)/ydim)/sinh((k*pi*xdim)/ydim);
            term4 = cosh((k*pi*x)/ydim)-cosh((k*pi*(x-xdim)/ydim));
            s = s + term1*term2*term3*term4;
        end
        T(i,j) = T(i,j) + s;
    end
end

T=flip(T);

figure(2)
contourf(flip(T));
colorbar;
title('Temperature Distribution on a 2D Plate analytical method');
xlabel('X');
ylabel('Y');
colormap('jet');

TN = abs(T2-T);
TMaxDiff = max(TN(:));

figure(3)
contourf(flip(TN));
colorbar;
title('Value difference computational vs analytical methods');

```

```
xlabel('X');  
ylabel('Y');  
colormap('jet');  
fprintf("The maximum difference between the analytical and computer model  
answer is %0.4f°C\n",TMaxDiff)
```