

Introduction

In this report a code will be used to calculate heat through an aluminium plate using the 2D diffusion equation and the explicit finite difference method. The primary aim is to prove the code against an analytical equation.

Further objectives of this report are to:

- Described the code step by step and critically analyse the inputs.
- Explain the equations and methodology behind the code.
- Explain the limitations of the analysis method used.
- Suggest improvements to the model.

Contents

Table of Contents

Introduction.....	1
Contents	1
Governing equation	2
Explicit finite difference form of the constitutive equation	3
Taylor Series	3
Taylor Series Approximations.....	3
Applying to the Heat Equation	3
Initial boundary conditions and analytical solution.....	4
Boundary condition	4
Analytical Solution	4
Codes' details and explanations	5
Analysis of the code results.....	8
Discussion as to the limitations of the model.....	10
Suggested improvements to the model and future developments.....	10
Conclusion.....	10
References	10
Appendix	11

Governing equation

The heat equation is a partial differential equation used to calculate the conduction of heat in solids. It was formulated by Joseph Fourier at the beginning of the 19th century and has since been modified for use in various conduction type applications in areas including biological science as well as social science. (T. N. Narasimhan, 1999).

The heat equation is described as follows:

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

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

Where:

- ∇^2 is the Laplacian defined by $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ in two dimensions
- t is time
- T is the temperature
- $\alpha = k/(\rho \cdot c)$ is the thermal diffusivity [$\text{m}^2 \cdot \text{s}^{-1}$]
- k is the thermal conductivity [$\text{W} \cdot \text{m}^{-1} \cdot \text{K}^{-1}$]
- ρ the density [$\text{kg} \cdot \text{m}^{-3}$]
- c the specific heat [$\text{J} \cdot \text{kg}^{-1} \cdot \text{K}^{-1}$].

Explicit finite difference form of the constitutive equation

Taylor Series

In certain cases, a function $f(x)$ can be rewritten in terms of an infinite sum of other functions such as:

$$f(x) = f(x_0) + f'(x_0) \cdot \frac{(x-x_0)^1}{1!} + f''(x_0) \cdot \frac{(x-x_0)^2}{2!} + f'''(x_0) \cdot \frac{(x-x_0)^3}{3!} + f^{(4)}(x_0) \cdot \frac{(x-x_0)^4}{4!} + \dots$$

which can be written in a more compact form using the summation notation:

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

This is referred to as a Taylor series expansion of $f(x)$ about point $x = x_0$

Taylor Series Approximations

The Taylor series can be simplified to make two approximations:

$$\text{Forward difference approximation } f'(x_0) \approx \frac{f(x_0+h) - f(x_0)}{h}$$

$$\text{Backward difference approximation } f'(x_0) \approx \frac{f(x_0) - f(x_0-h)}{h}$$

Both of these are only approximations, and they get more accurate as h gets smaller, however these two can be combined to get another more accurate approximation:

$$\text{Central difference approximation } f'(x_0) \approx \frac{f(x_0+h) - f(x_0-h)}{2h}$$

A similar process can be used to derive approximations from higher derivatives of the function to get the following:

$$\text{Central difference approximation of a second order derivative } f''(x_0) \approx \frac{f(x_0+h) - 2f(x_0) + f(x_0-h)}{h^2}$$

This allows for a higher accuracy approximations.

Applying to the Heat Equation

Applying the Taylor series approximations to the heat equation at any given point on the plate we can replace the temporal 1st derivative with the forward approximation and the spatial 2nd derivative with the central approximation to give the following:

$$\begin{aligned} \text{Temporal 1st derivative } \frac{\partial T}{\partial t} &= \frac{T_P^{t+\Delta t} - T_P^t}{\Delta t} \\ \text{Spatial 2nd derivative } \begin{cases} \frac{\partial^2 T}{\partial x^2} = \frac{T_E^t - 2T_P^t + T_W^t}{h^2} \\ \frac{\partial^2 T}{\partial y^2} = \frac{T_N^t - 2T_P^t + T_S^t}{h^2} \end{cases} \end{aligned}$$

Substituting these into the heat equation and simplifying we can get the following:

$$T_P^{t+\Delta t} = F_0 \left(T_E^t + T_W^t + T_N^t + T_S^t - \left(4 - \frac{1}{F_0} \right) \cdot T_P^t \right) \quad \text{where } F_0 \text{ is } F_0 = \frac{\alpha \cdot \Delta t}{h^2}$$

This is referred to as the Explicit Finite Difference Equation

As the Fourier number is dependent on α (thermal conductivity), t (time) and h (grid spacing) the time and grid spacing must be defined.

The main disadvantage of this method is that once α and h are fixed, there is a maximum permissible time-step size Δt which should not exceed the value imposed by the following stability criterion:

$$0 < F_0 \leq \frac{1}{4} \quad \text{in } 2D$$

Initial boundary conditions and analytical solution

Boundary condition

The initial boundary conditions applied to the code are as follows:

Initial domain condition:

$$T(x,y,0)=0 \quad 0 < x < 6 \quad 0 < y < 6$$

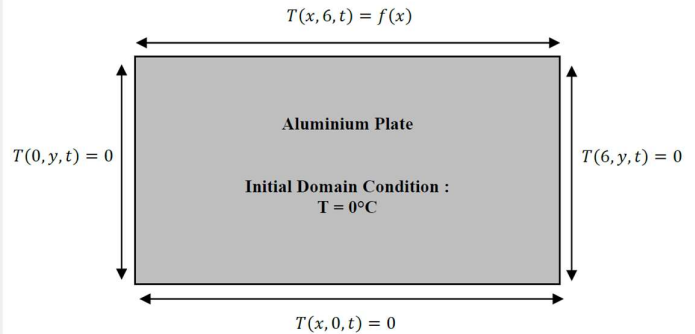
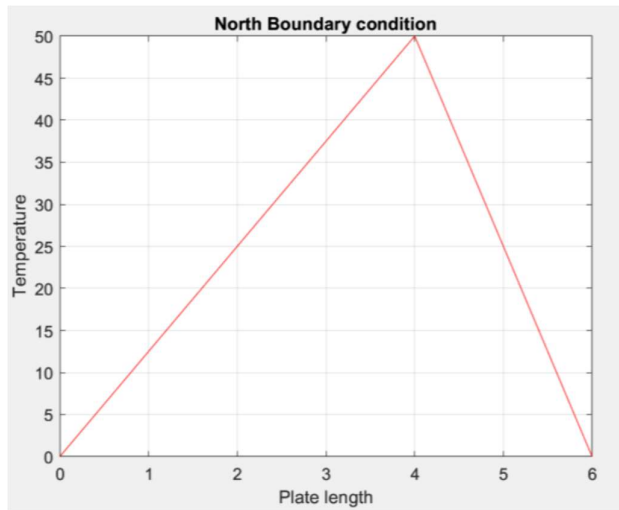
Constant boundary conditions:

$$T(0,y,t)=T(6,y,t)=0 \quad \text{for } 0 \leq y \leq 6 \text{ and } 0 \leq t \leq \text{thermal equilibrium}$$

$$T(x,0,t)=f(x) \quad T(x,6,t)=0 \quad \text{for } 0 \leq x \leq 6 \text{ and } 0 \leq t \leq \text{thermal equilibrium}$$

Where, $f(x) = \{12.5x \text{ if } 0 \leq x \leq (23 \times 6) - 25x + 150 \text{ if } (23 \times 6) < x \leq 6\}$

50°C is applied at 4 meters along the plate with a linear drop off towards the edges. This can be visualised in the following figures.



North boundary condition – shows the temperature profile applied to the north edge.

Initial domain conditions – shows the initial domain conditions of the plate before the temperature is applied.

Analytical Solution

The code is tested against the following analytical solution of the temperature at thermal equilibrium (Daileda R.C., 2012).

$$T(x,y) = \frac{450}{\pi^2} \cdot \sum_{n=1}^{\infty} \left(\frac{\sin\left(\frac{2 \cdot n \cdot \pi}{3}\right)}{n^2 \cdot \sinh(n \cdot \pi)} \cdot \sin\left(\frac{n \cdot \pi \cdot x}{6}\right) \cdot \sinh\left(\frac{n \cdot \pi \cdot y}{6}\right) \right)$$

This will act as a control to the numerical method being tested as it is a proven result.

Codes' details and explanations

The code is shown below split into sections with descriptions of what each sections is doing.

1	clc	
2	clear	
3		
4	%% Plate Dimensions	
5	x = 6	
6	y = 6	
7		
8	%% Aluminium Properties	
9	k = 143	
10	p = 2800	
11	c = 795	
12	a = k/(p*c)	
13		

Clearing the code window. Setting the plate dimensions and material properties.

14	%% User Inputs	
15	% Time step input	
16	t = input('Enter the time step in seconds (1hr=3600): '); % User input to time step (t)	
17	while t <= 0 % Checks if the time step provided is a positive value	
18	disp('time step must be greater than 0') % Displays error message if value is not positive	
19	t = input('Enter the time step in seconds (1hr=3600): '); % Requests new t dimension	
20	end	
21	% Grid spacing input	
22	minxy = min(x,y) % Determines the minimum dimension of the plate	
23	h = input('Enter grid spacing in meters: '); % User input for grid spacing (h)	
24	while h <= 0 h >= minxy % Checks if input is negative or greater than the minimum dimension of the plate	
25	fprintf('Grid spacing must be a positive value less than the minimum size of the plate (%g) \n',minxy)	
26	% Displays error message if grid spacing is incorrect	
27	h = input('Enter grid spacing in meters: '); % Asks user for new grid spacing	
28	end	
29		

Setting up the user inputs for time step (t) and grid spacing (h). Limits have been put in to ensure the values put in are realistic. Messages are displayed to guide the user as to what values should be used.

30	%% Fourier Calculation	
31	F0 = (a*t)/(h^2); % Fourier number calculation	
32	while F0 <= 0 F0 > 0.25 % Checks if inputs are out of stabilisation	
33	fprintf(['Fourier out of stabilisation at %g, please use a smaller time step or a larger grid spacing' ...	
34	' to achieve a Fourier number of <0.25 \n'],F0) % Informs user of Fourier number and prompts correction	
35	t = input('Enter the time step in seconds (1hr=3600): '); % Asks user new time step	
36	while t <= 0 % Checks if the time step provided is a positive value	
37	disp('time step must be greater than 0') % Displays error message if value is not positive	
38	t = input('Enter the time step in seconds (1hr=3600): '); % Requests new t dimension	
39	end	
40	% Grid spacing input	
41	minxy = min(x,y) % determines the minimum dimension of the plate	
42	h = input('Enter grid spacing in meters: '); % Asks user for new grid spacing	
43	while h <= 0 h >= minxy % while input is negative or greater than the minimum dimension of the plate	
44	fprintf('Grid spacing must be a positive value less than the minimum size of the plate (%g) \n',minxy)	
45	% Displays error message if grid spacing is incorrect	
46	h = input('Enter grid spacing in meters: '); % Asks user for new grid spacing	
47	end	
48	F0 = (a*t)/(h^2) % Calculates new Fourier number	
49	end	
50		

Calculates the Fourier number and checks whether it meets the stabilisation criteria (<0.25). If the Fourier number is not within stabilisation the code quotes the current Fourier number and target, then reruns the user inputs section.

51	%% Matrix	
52	NumRow = (y/h)+1 % Sets number of rows based on plate width and grid spacing	
53	NumCol = (x/h)+1 % Sets number of columns based on plate width and grid spacing	
54		
55	T1 = zeros(NumRow,NumCol) % Creation of blank Matrix	
56		
57	%% Boundary conditions	
58		
59	for Col = 1:NumCol % For the first row in each column in the matrix	
60	z = (Col-1)*h % Finds distance along the plate	
61	if z <= 2/3 * x	
62	T1(1,Col) = 12.5 * z; % For the first two third of the plate, 12.5°C is added for each meter of the plate	
63	else T1(1,Col) = -25 * z + 150 % For the last third of the plate the temperature drops off to 0	
64	end	
65	end	
66		

Creates the matrix based on the plate dimensions and grid step. Sets the boundary conditions; these are how the temperature is applied to the plate. In this case the heat is centred at the 4 metre mark and drops off towards the edges of the plate.

This boundary condition is only good for the given plate dimensions. If the plate dimensions could vary a more complex boundary condition would need to be used.

ACMA6012 - 2D Diffusion Equation model using the Explicit Finite Difference Method

```
67 %% Equilibrium & Figure
68
69 time = 0 % Sets time counter to 0
70 Equilib = false; % Sets condition for false equilibrium
71 while ~Equilib % While equilibrium is false
72     T2 = T1 % New matrix created for storing temperature values
73     for i = 2:NumRow-1 % For each row ignoring the last
74         for j = 2:NumCol-1 % For each column ignoring the last
75             T2(i,j) = F0 * (T1(i+1,j) + T1(i-1,j) + T1(i,j+1) + T1(i,j-1)) - (4-1/F0) * T1(i,j);
76             % Runs explicit finite difference equation
77         end
78     end
79     if max(max(abs(T1-T2))) < 1e-3
80         % Check if max temperature difference between the original matrix and the new matrix is less than 0.001 (tolerance)
81         Equilib = true % If the tolerance has been met the equilibrium condition is set to true and the loop ends
82     end
83     T1 = T2; % Equalises the two matrices per loop
84     time = time + t % Adds to the time counter per loop
85
86
87     figure(1); % Creation of the first figure/graph for the explicit finite difference method (numeric method)
88     contour(flip(T2));
89     colorbar;
90     clr = colorbar;
91     clr.Label.String = 'Temperature in C';
92     colormap(turbo);
93     title('Temperature across the plate at equilibrium');
94     xlabel('Grid spacing in the x direction');
95     ylabel('Grid spacing in the y direction');
96 end
97
98 fprintf('Time taken to equilibrium is %g seconds \n', time) % Displays time in seconds
99 fprintf('Time taken to equilibrium is %g minutes \n', time/60) % Displays time in minutes
100 fprintf('Time taken to equilibrium is %g hours \n', time/3600) % Displays time in hours
101 iterations = time/t; % Works out the number of calculation iterations based on the time counter
102 fprintf('The number of iterations until equilibrium is %g \n', iterations) % Displays the number of calculation iterations
103
```

Runs the explicit finite difference equation until equilibrium is reached. Each iteration uses the time step and grid spacing provided earlier in the code. The equation as applied to every point in the defined grid at the time step.

A smaller time step will lead to a smaller and more accurate temperature change per iteration, but more iterations will be needed to achieve equilibrium.

A time counter is used to track how long it would take to reach equilibrium, and hence to calculate how many iterations have been done.

Equilibrium is reached when there is no change from one calculation to the next, however as a small change will always be observed a tolerance is used. In this case a tolerance of 1E-3 or 0.001 is used. Once the change per iteration is less than this value the code treats equilibrium as true and stops the loop.

The second part of this section is the creation of a figure; a graph showing the change per iteration and the final heat distribution.

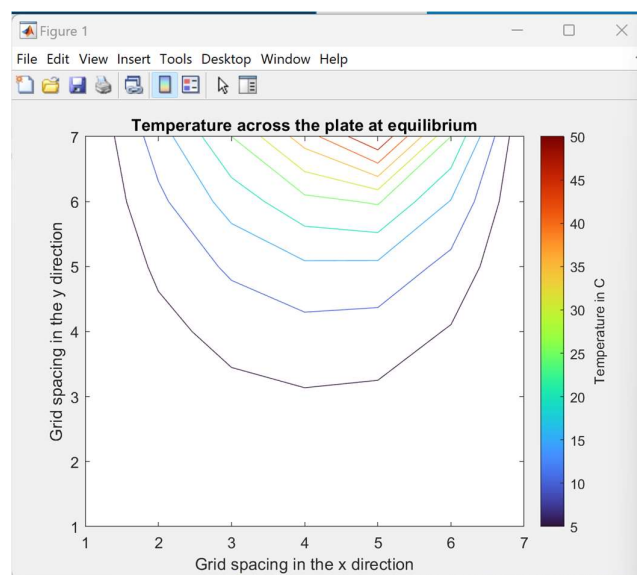


Figure 1 showing the resulting temperature map from the numerical method.

ACMA6012 - 2D Diffusion Equation model using the Explicit Finite Difference Method

```

104 %% Analytical Matrix
105
106 TA = zeros(NumRow,NumCol); % Creation of a new matrix for the analytical method
107
108 %% Boundary Conditions for Analytical method
109
110 for ColA = 1:NumCol; % For the first row in each column in the matrix
111     zA = (ColA-1)*h; % Finds distance along the plate
112     if zA <= 2/3 * x;
113         TA(1,ColA) = 12.5 * zA; % For the first two third of the plate, 12.5°C is added for each meter of the plate
114     else TA(1,ColA) = -25 * zA + 150; % For the last third of the plate the temperature drops off to 0
115     end
116 end
117

```

A second matrix is made to calculate the scenario using an analytical method to compare to the numerical method used above. A set of boundary conditions the same as those applied to the numerical method are applied to the new matrix.

```

118 %% Analytical Equation
119
120 TA = flip(TA);
121 for iA = 2:NumRow-1; % For each row ignoring the last
122     for jA = 2:NumCol-1; % For each row ignoring the last
123         for n = 1:200; % Run for 200 iterations
124             TA(iA,jA) = TA(iA,jA) + sin(2*n*pi/3)/(n^2*sinh(n*pi))...
125                 *sin(n*pi*(jA-1)*h/6)*sinh(n*pi*(iA-1)*h/6); % Analytical heat equation, iterative section
126         end
127         TA(iA,jA) = (450/pi^2)*TA(iA,jA); % Analytical heat equation, static section
128     end
129 end
130
131 %% Analytical Figure
132
133 figure(2); % Creation of the second figure/graph for the heat equation (analytical method)
134 contour(TA);
135 colormap(turbo);
136 colorbar;
137 title('Analytical Result - Temp Distribution');
138 xlabel('Grid spacing in the x direction');
139 ylabel('Grid spacing in the y direction');
140
141 TA = flip(TA);
142
143 figure(3); % Creation of the third figure/graph for the comparison of both methods
144 D=abs(T2-TA);
145 contour(flip(D));
146 colormap("turbo");
147 colorbar;
148 title('Numerical vs Analytical Temperature Distribution');
149 xlabel('Grid spacing in the x direction');
150 ylabel('Grid spacing in the y direction');
151

```

The heat equation for the analytical method is applied to the matrix over a set number of iterations, denoted by “n”. The set number chosen here is 200. A second and third figure are generated here. The second is the heat distribution from the analytical method, while the third is a difference map that compares the analytical and numerical methods.

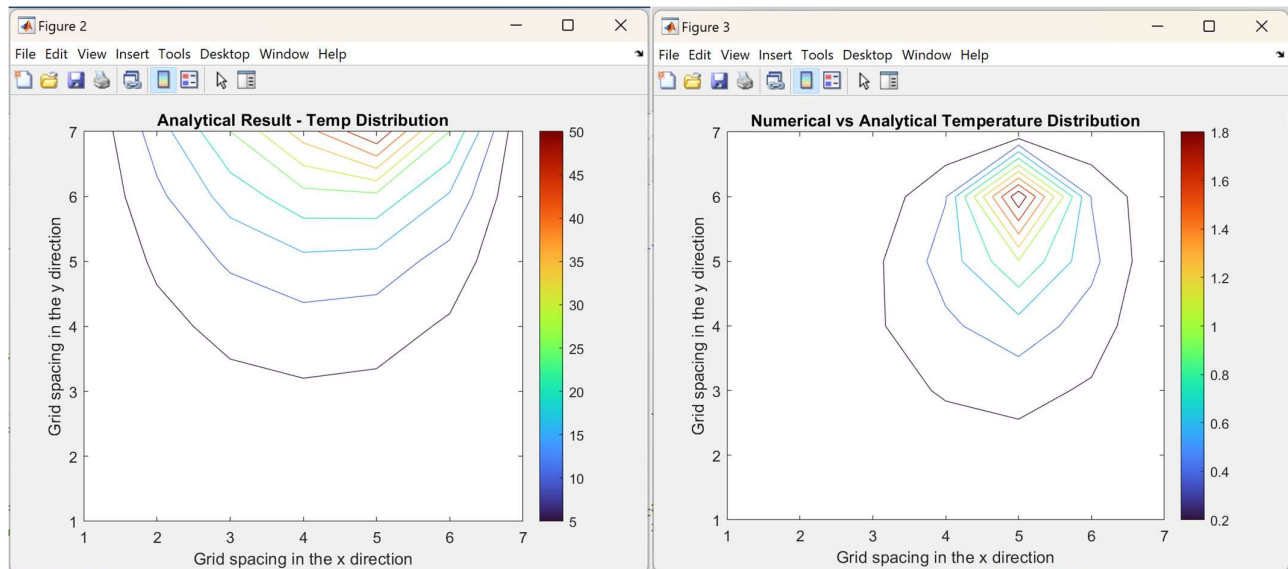


Figure 2 showing the resulting temperature map from the analytical method (left).

Figure 3 showing the resulting temperature difference map from both methods.

Analysis of the code results

In order to see how the output of the code changes with the inputs of Time Step and Grid Spacing, the following tests were done.

Starting from a grid spacing of 1, a number of time steps were ran from their highest stable value ($F0 \leq 0.25$) down to a value where the max temperature difference between the numerical and analytical methods becomes larger rather than smaller. A smaller resolution of grid spacing was used later on to see the change at each step.

The key dimension measured here was the maximum temperature difference between the numerical method and the analytical method. As the analytical method is used as a known control, this is a good indication that the code is working as intended.

Grid Spacing	Time Step	Tolerance	Equilibrium Time	Max Temp. Difference
1	4200	1E-02	Out of Stability	N/A
1	3600	1E-02	144,000	1.8
1	3000	1E-02	138,000	1.8
1	2400	1E-02	132,000	1.8
1	1800	1E-02	126,000	1.8
1	1200	1E-02	114,000	1.8
1	600	1E-02	94,800	1.6
1	400	1E-02	82,800	1.6
1	200	1E-02	62,800	1.6
1	100	1E-02	44,100	2.2
0.75	2200	1E-02	Out of Stability	N/A
0.75	2100	1E-02	128,100	0.25
0.75	2000	1E-02	126,000	0.25
0.75	1750	1E-02	122,500	0.25
0.75	1500	1E-02	118,500	0.25
0.75	1250	1E-02	113,750	0.2
0.75	1000	1E-02	108,000	0.22
0.5	1000	1E-02	Out of Stability	N/A
0.5	900	1E-02	104,400	0.9
0.5	800	1E-02	101,600	0.9
0.5	700	1E-02	98,000	0.9
0.5	600	1E-02	93,600	0.9
0.5	500	1E-02	88,000	0.8
0.5	400	1E-02	82,000	0.8
0.5	300	1E-02	73,800	0.8
0.5	200	1E-02	62,400	1.2
0.4	700	1E-02	Out of Stability	N/A
0.4	600	1E-02	93,000	0.7
0.4	500	1E-02	88,000	0.7
0.4	400	1E-02	81,600	0.6
0.4	300	1E-02	73,500	0.8
0.3	400	1E-02	Out of Stability	N/A
0.3	300	1E-02	73,500	0.9
0.3	200	1E-02	62,200	1.2

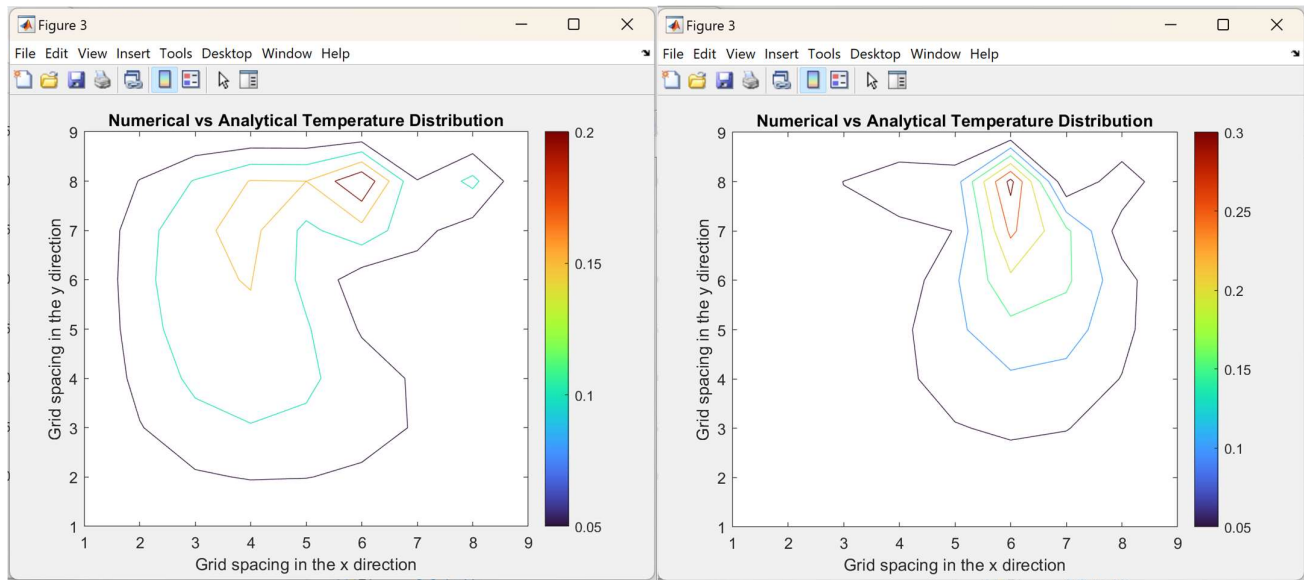
The best result from those tested was with a grid spacing of 0.75 and a time step of 1250 seconds. At this point the temperature maximum difference between the numerical and analytical methods was only 0.2°C.

Using this result the tolerance can then be tested to see what the results of the tolerance are on the ideal inputs. The following test was done. The ideal inputs were tested at values from 1E-01 (0.1) to 1E-08 (0.00000001). At a tolerance of 1E-03 a change was seen in the temperature map. It appears that the map shifted as the result got closer to a true equilibrium. Very little affect was seen on the difference map or the max temperature difference after this value. At a tolerance of 1E-1 the maximum temperature difference became dramatically worse.

Grid Spacing	Time Step	Tolerance	Equilibrium Time	Max Temp. Difference	Calculation Time (s)
0.75	1250	1E-01	50000	1.8	8
0.75	1250	1E-02	113750	0.2	19
0.75	1250	1E-03	178750	0.3	29
0.75	1250	1E-04	243750	0.3	41
0.75	1250	1E-05	309750	0.3	50
0.75	1250	1E-06	372500	0.3	60
0.75	1250	1E-07	437500	0.3	72
0.75	1250	1E-08	502500	0.3	82

Table showing results of tolerance testing on ideal inputs.

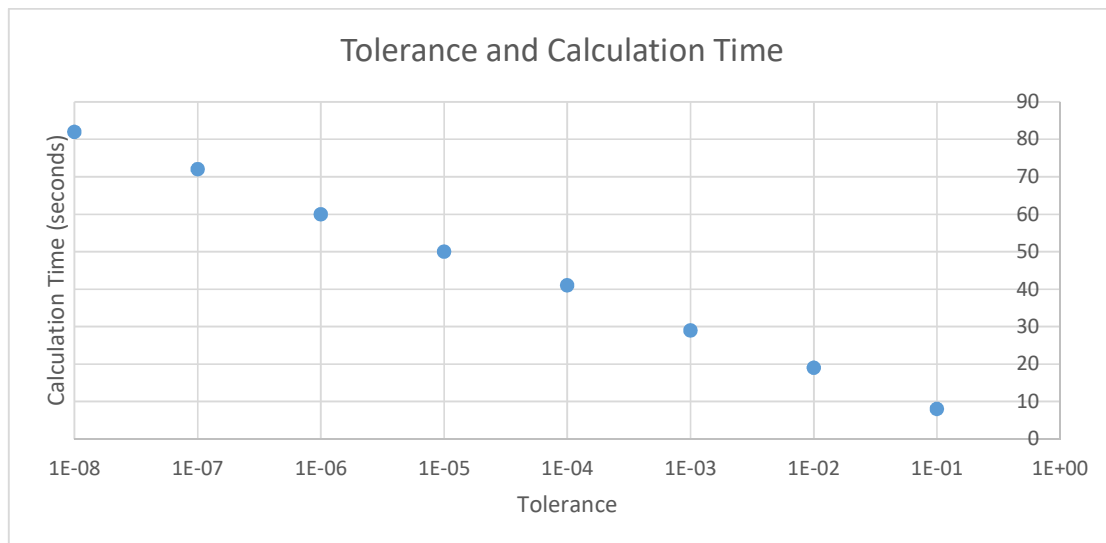
ACMA6012 - 2D Diffusion Equation model using the Explicit Finite Difference Method



Temperature difference map at $1E-2$ (left). Temperature difference map at $1E-3$ (right).

In conclusion, to get a result closer to the analytical solution the grid spacing must be maintained at a low value, which means the time step must also be low to maintain stabilisation. The tolerance should also be maintained to a good level of accuracy at least $1E-3$. This will put more stress on the computer and the calculation will become slow. This is referenced in a similar model done by an Oxford robotics team (Stephen Roberts, Alison Noble).

The real code completion time is directly proportional to the tolerance factor as shown in the following graph. For the $1E-08$ tolerance using a time step of 1250 and a grid spacing of 0.75 it took 82 seconds to process. Some computers may process this slower, and the time step and grid spacing are still quite large.



Discussion as to the limitations of the model

This model is limited by the method used due to a number of reasons. The first reason is potential error due to rounding and tolerances. The model relies on a limited resolution of the time step, grid spacing, and tolerance to be possible to calculate without major computational constraints. Due to this there will always be a level of error introduced where the ideal values may be between the chosen time and grid spacing, and the tolerance will never be tight enough to get to a perfect equilibrium, so it is an approximation.

Another limitation of this model is the complexity of the problems it can calculate in a realistic amount of time. Even with a simple model such as this one a more accurate approximation takes a considerable amount of time to process. With a more complex geometry or multiple boundary conditions to consider the model may quickly become unwieldy. Stability of the model may also become very difficult to maintain with a more complex problem. Non-standard geometries are often not possible to model with this method.

Practicality of the coding used is also a limitation. Many different methods of coding can be used to simulate physical scenarios, however they can easily introduce errors or may easily be off the mark with their results as shown with the wide varied of equilibrium times shown in this code, so they may not represent the physical scenario well. Complex problems may also become tiresome to code leading to more bugs and inaccuracies in the model.

Suggested improvements to the model and future developments

In order to improve the model different numerical methods could be employed. The explicit finite difference method inherently has issues with model stability due to the need to reference a previously known value within the stability criteria and hence has a limitation on the time step. The implicit finite difference method on the other hand does not rely on the previous value and instead the future step relies on itself so the method does not have a limitation on the time step and as such is more stable. (Cadence CFD, 2024)

If complex geometries are to be handled, other numerical alternatives could be used. One example is the finite element method which uses a mesh made of many which calculations can act on. This is similar to the grid in the finite difference method, but it can be applied to almost any geometry. (Wolfram, 2024)

Conclusion

From the work in this report it is clear that the explicit finite difference method is limited, but for simple scenarios like this one it performs well against the analytical method.

The outcome of the numeric method in the code showed vastly different times to reach equilibrium depending on the inputs which leads to the belief that it isn't accurately modelling the scenarios time, and that the outcome is heavily affected by the inputs.

Further work on the stabilisation and error could help to get a more reliable outcome, but it could also be beneficial to use a different numeric method.

References

- Cadence CFD. (2024). *Solve Problems With No Time Step Restriction Using the Implicit Finite Difference Method*.
<https://resources.system-analysis.cadence.com/blog/msa2023-solve-problems-with-no-time-step-restriction-using-the-implicit-finite-difference-method>
- Daleda R.C. (2012, March 6). *The Two Dimensional Heat Equation*. Trinity University, San Antonio, Texas, USA.
- T. N. Narasimhan. (1999). *FOURIER'S HEAT CONDUCTION EQUATION' HISTORY, INFLUENCE, AND CONNECTIONS*.
- Stephen Roberts, Alison Noble. *Second Year Engineering Mathematics Laboratory, Exercise 5: PDE's*. Accessible at:
<https://www.robots.ox.ac.uk/~sjrob/Teaching/Mathlab/ex5.pdf>.
- Wolfram. (2024). *Solving Partial Differential Equations with Finite Elements*.
<https://reference.wolfram.com/language/FEMDocumentation/tutorial/SolvingPDEwithFEM.html>

Appendix

Heat Transfer Code:

```
clc
clear

%% Plate Dimensions
x = 6
y = 6

%% Aluminium Properties
k = 143
p = 2800
c = 795
a = k/(p*c)

%% User Inputs
% Time step input
t = input('Enter the time step in seconds (1hr=3600): '); % User input to time step (t)
while t <= 0 % Checks if the time step provided is a positive value
    disp('time step must be greater than 0') % Displays error message if value is not positive
    t = input('Enter the time step in seconds (1hr=3600): '); % Requests new t dimension
end
% Grid spacing input
minxy = min(x,y) % Determines the minimum dimension of the plate
h = input('Enter grid spacing in meters: '); % User input for grid spacing (h)
while h <= 0 || h >= minxy % Checks if input is negative or greater than the minimum dimension
    of the plate
        fprintf('Grid spacing must be a positive value less than the minimum size of the plate (%g)
\n',minxy)
        % Displays error message if grid spacing is incorrect
        h = input('Enter grid spacing in meters: '); % Asks user for new grid spacing
    end

%% Fourier Calculation
F0 = (a*t)/(h^2); % Fourier number calculation
while F0 <= 0 || F0 > 0.25 % Checks if inputs are out of stabilisation
    fprintf(['Fourier out of stabilisation at %g, please use a smaller time step or a larger
grid spacing' ...
' to achieve a Fourier number of <0.25 \n'],F0) % Informs user of Fourier number and
prompts correction
    t = input('Enter the time step in seconds (1hr=3600): '); % Asks user new time step
    while t <= 0 % Checks if the time step provided is a positive value
        disp('time step must be greater than 0') % Displays error message if value is not
positive
        t = input('Enter the time step in seconds (1hr=3600): '); % Requests new t dimension
    end
    % Grid spacing input
    minxy = min(x,y) % determines the minimum dimension of the plate
    h = input('Enter grid spacing in meters: '); % Asks user for new grid spacing
    while h <= 0 || h >= minxy % while input is negative or greater than the minimum dimension
of the plate
        fprintf('Grid spacing must be a positive value less than the minimum size of the plate
(%g) \n',minxy)
        % Displays error message if grid spacing is incorrect
        h = input('Enter grid spacing in meters: '); % Asks user for new grid spacing
    end
    F0 = (a*t)/(h^2) % Calculates new Fourier number
end

%% Matrix
NumRow = (y/h)+1 % Sets number of rows based on plate width and grid spacing
NumCol = (x/h)+1 % Sets number of columns based on plate width and grid spacing
```

```
T1 = zeros(NumRow,NumCol) % Creation of blank Matrix

%% Boundary conditions

for Col = 1:NumCol % For the first row in each column in the matrix
    z = (Col-1)*h % Finds distance along the plate
    if z <= 2/3 * x
        T1 (1,Col) = 12.5 * z; % For the first two third of the plate, 12.5°C is added for each
meter of the plate
    else T1 (1,Col) = -25 * z + 150 % For the last third of the plate the temperature drops off
to 0
    end
end

%% Equilibrium & Figure

time = 0 % Sets time counter to 0
Equilib = false; % Sets condition for false equilibrium
while ~Equilib % While equilibrium is false
    T2 = T1 % New matrix created for storing temperature values
    for i = 2:NumRow-1 % For each row ignoring the last
        for j = 2:NumCol-1 % For each column ignoring the last
            T2 (i,j) = F0 * (T1(i+1,j) + T1(i-1,j) + T1(i,j+1) + T1(i,j-1) - (4-1/F0) *
T1(i,j));
            % Runs explicit finite difference equation
        end
    end
    if max(max(abs(T1-T2))) < 1e-3
        % Check if max temperature difference between the original matrix and the new matrix is
less than 0.001 (tolerance)
        Equilib = true % If the tolerance has been met the equilibrium condition is set to true
and the loop ends
    end
    T1 = T2; % Equalises the two matrices per loop
    time = time + t % Adds to the time counter per loop

    figure(1); % Creation of the first figure/graph for the explicit finite difference method
(numeric method)
    contour(flip(T2));
    colorbar;
    clr = colorbar;
    clr.Label.String = 'Temperature in C';
    colormap(turbo);
    title('Temperature across the plate at equilibrium');
    xlabel('Grid spacing in the x direction');
    ylabel('Grid spacing in the y direction');
end

fprintf('Time taken to equilibrium is %g seconds \n', time) % Displays time in seconds
fprintf('Time taken to equilibrium is %g minutes \n', time/60) % Displays time in minutes
fprintf('Time taken to equilibrium is %g hours \n', time/3600) % Displays time in hours
iterations = time/t; % Works out the number of calculation iterations based on the time counter
fprintf('The number of iterations until equilibrium is %g \n', iterations) % Displays the number
of calculation iterations

%% Analytical Matrix

TA = zeros(NumRow,NumCol); % Creation of a new matrix for the analytical method

%% Boundary Conditions for Analytical method
```

ACMA6012 - 2D Diffusion Equation model using the Explicit Finite Difference Method

```
for ColA = 1:NumCol; % For the first row in each column in the matrix
    zA = (ColA-1)*h; % Finds distance along the plate
    if zA <= 2/3 * x;
        TA (1,ColA) = 12.5 * zA; % For the first two third of the plate, 12.5°C is added for
each meter of the plate
    else TA (1,ColA) = -25 * zA + 150; % For the last third of the plate the temperature drops
off to 0
    end
end

%% Analytical Equation

TA = flip(TA);
for iA = 2:NumRow-1; % For each row ignoring the last
    for jA = 2:NumCol-1; % For each row ignoring the last
        for n = 1:200; % Run for 200 iterations
            TA(iA,jA)=TA(iA,jA)+sin(2*n*pi/3)/(n^2*sinh(n*pi))*...
                *sin(n*pi*(jA-1)*h/6)*sinh(n*pi*(iA-1)*h/6); % Analytical heat equation,
iterative section
        end
        TA(iA,jA) = (450/pi^2)*TA(iA,jA); % Analytical heat equation, static section
    end
end

%% Analytical Figure

figure(2); % Creation of the second figure/graph for the heat equation (analytical method)
contour(TA);
colormap(turbo);
colorbar;
title('Analytical Result - Temp Distribution');
xlabel('Grid spacing in the x direction');
ylabel('Grid spacing in the y direction');

TA = flip(TA);

figure(3); % Creation of the third figure/graph for the comparison of both methods
D=abs(T2-TA)
contour(flip(D));
colormap("turbo");
colorbar;
title('Numerical vs Analytical Temperature Distribution');
xlabel('Grid spacing in the x direction');
ylabel('Grid spacing in the y direction');
```