Bauhaus-Universität Weimar

Computer Models for Physical Processes

WINTER SEMESTER 2023

Finite Difference Method for Stationary 2D heat Conduction

FINAL PROJECT

SUBMITTED TO:

Prof. Dr.-Ing. habil. Carsten Könke

SUBMITTED BY:

GROUP-13
Ronit Kumar Dadi Venkata Satya Sai-127
Furqan Shuja-126723
Emmanuel Olubunmi Ogunleye

25 January 2024

Table of Contents **2**

LIST OF FIGURES

LIST OF TABLES

1.0 Introduction

Computer models for Physical Processes is an associated subfield of Engineering Methods which can be used to formulate a numerical approximate solution for a problem in physics, i.e. heat flow problem or problem from structural mechanics by being able to establish the governing equations starting from energy formulations or conservation equations. It is used for transferring the strong form of a physical problem description into a weak form and will be able to solve finite difference methods or finite element methods.

In this project, finite-difference analysis serves as a numerical technique for addressing heat transfer challenges with limited analytical solutions. The method involves discretizing the system into nodal points, each representing the average temperature of a specific region. Accuracy hinges on the construction of the nodal network, with inherent trade-offs between finer meshes for enhanced resolution and the associated computational demands. The procedural steps encompass system representation through a nodal network, derivation of finite-difference equations, solving for unknown temperatures, and employing Fourier's Law to determine heat transfer. Essentially, finite-difference analysis facilitates a swift and approximate resolution of intricate heat transfer problem through MATLAB.

We modeled a finite difference term for a 2D heat conduction problem using the finite difference method. Subsequently, we employed MATLAB to simulate the FDM process and validated the results through comparisons with Ansys simulations and hand calculations. Following this, we interpreted the obtained results to imbue them with more meaningful insights.

2.0 Derivation of Finite Difference Term

2.1 Formulation of 2D Heat Equation

For the initial step we formulate the governing differential equations by applying Law of Conservation of Energy (Energy Balance), by considering rectangular domain.

We assume that the left side of the rectangle is at a constant higher temperature, the other three ends of the rectangle are at a lower temperature.

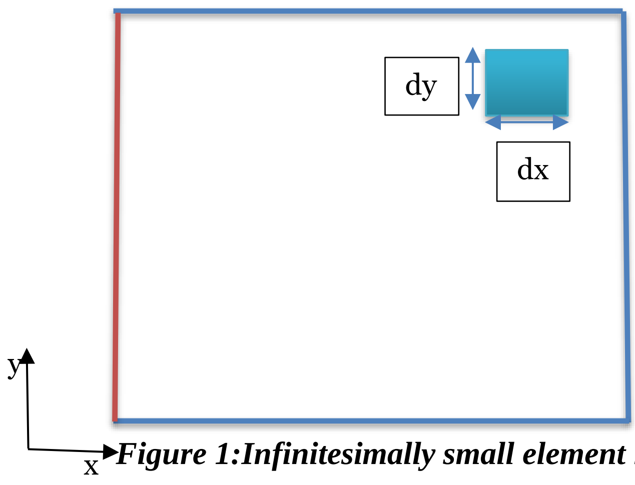


Figure 1:Infinitesimally small element in a Domain

We consider a infinitesimally small element in the rectangle with dimensions dx in x direction and dy in y direction. Taking the small element we assume it has mass dm.

Let's consider heat gets conducted from the left direction in the x direction (Q_x) and from the right face of the element heat comes out (Q_{x+dx}). Similarly heat gets conducted from the bottom of the element (Q_y) and comes out from the top (Q_{y+dy}).

If we use the energy balance equation.

$$\left(\frac{\text{Rate of}}{\text{Energy}}\right)_{\text{in}} - \left(\frac{\text{Rate of}}{\text{Energy}}\right)_{\text{out}} + \left(\frac{\text{Rate of}}{\text{Energy}}\right)_{\text{generated}} = \left(\frac{\text{Rate of}}{\text{Energy}}\right)_{\text{change}} \qquad \text{(Formula 2.1)}$$

Substituting the values,

$$\left(\dot{Q}_x + \dot{Q}_y\right) - \left(\dot{Q}_{x+dx} + \dot{Q}_{y+dy}\right) + \dot{Q}_{\text{gen}} = \text{dmCp}\,\frac{dT}{dt}$$

For steady state operation,

$$\frac{dT}{dt} = 0$$

$$\left(\dot{Q}_x + \dot{Q}_y\right) - \left(\dot{Q}_{x+dx} + \dot{Q}_{y+dy}\right) = dm\text{Cp}\frac{dT}{dt}$$

$$\left(\dot{Q}_x + \dot{Q}_y\right) - \left(\dot{Q}_{x+dx} + \dot{Q}_{y+dy}\right) = 0 \qquad \text{(Formula 2.2)}$$

Using Taylor series expansion,

$$\text{f}(\text{x} + \text{dx}) = \text{f}(\text{x}) + \text{f}'(\text{x})\frac{\text{dx}}{1!} + \text{f}''(\text{x})\frac{\text{dx}^2}{2!} + \text{f}'''(\text{x})\frac{\text{dx}}{3!} + \cdots.$$

$$\dot{Q}_{x+dx} = \dot{Q}_x + \dot{Q}_x'\frac{\text{dx}}{1!} + \dot{Q}_x''\frac{\text{dx}}{2!} + \dot{Q}_x{}''\frac{\text{dx}^3}{3!} + \cdots..$$

Neglecting the higher order terms,

$$\dot{Q}_{x+dx} = \dot{Q}_x + \dot{Q}_x'\frac{\text{dx}}{1!} \qquad \text{(Formula 2.3a)}$$

Similarly in the y direction,

$$\dot{Q}_{y+dy} = \dot{Q}_y + \dot{Q}_y\frac{dy}{1!} \qquad \text{(Formula 2.3b)}$$

Substituting Formula 2.3a and 2.3b in formula 2.2 we get,

$$\left(\dot{Q}_x + \dot{Q}_y\right) - \left(\dot{Q}_x + \dot{Q}_x'\frac{\text{dx}}{1!} + \dot{Q}_y + \dot{Q}_y'\frac{\text{d}y}{1!}\right) = 0$$

$$-\dot{Q}_x'\frac{\text{dx}}{1!} - \dot{Q}_y'\frac{\text{d}y}{1!} = 0$$

$$\dot{Q}_x'\frac{\text{dx}}{1!} + \dot{Q}_y'\frac{\text{d}y}{1!} = 0 \qquad \text{(Formula 2.4)}$$

Using Fourier's law of Heat Conduction,

Heat conduction in X-direction,

$$\dot{Q}_x = -\mathrm{KA}\frac{dT}{dx} \qquad \text{(Formula 2.5a)}$$

Heat conduction in Y-direction,

$$\dot{Q}_y = -\mathrm{KA}\frac{dT}{dy} \qquad \text{(Formula 2.5b)}$$

Where, K = Thermal conductivity $(\mathrm{W/mK})$, A = Area perpendicular to heat flow (m^2)

Substituting Formula 2.5a and 2.5b in 2.4,

$$\frac{d}{dx}(\dot{Q}_x)dx + \frac{d}{dy}(\dot{Q}_y)\mathrm{dy} = 0$$
$$\frac{d}{dx}\left(-\mathrm{KA}\frac{dT}{dx}\right)dx + \frac{d}{dy}\left(-\mathrm{KA}\frac{dT}{dy}\right)\mathrm{dy} = 0$$
$$\frac{d}{dx}\left(-\mathrm{K}dy\frac{dT}{dx}\right)dx + \frac{d}{dy}\left(-\mathrm{K}dx\frac{dT}{dy}\right)\mathrm{dy} = 0$$
$$\frac{d}{dx}\left(-\mathrm{K}\frac{dT}{dx}\right)dxdy + \frac{d}{dy}\left(-\mathrm{K}\frac{dT}{dy}\right)dxdy = 0$$
$$\frac{d}{dx}\left(\mathrm{K}\frac{dT}{dx}\right) + \frac{d}{dy}\left(\mathrm{K}\frac{dT}{dy}\right) = 0 \qquad \text{(Formula 2.6)}$$

If we assume the thermal conductivity in the element is constant, not varying because of position or distance. Then K is constant for entire element, therefore:-

$$\mathrm{K}\frac{d}{dx}\left(\frac{dT}{dx}\right) + \mathrm{K}\frac{d}{dy}\left(\frac{dT}{dy}\right) = 0$$
$$\frac{d}{dx}\left(\frac{dT}{dx}\right) + \frac{d}{dy}\left(\frac{dT}{dy}\right) = 0$$
$$\frac{d^2T}{dx^2} + \frac{d^2T}{dy^2} = 0 \qquad \text{(Formula 2.7)}$$

Formula 2.7 is the **Governing differential equation for 2D steady state heat conduction.**

2.2 Formulation of Finite Difference Term

For this we use Taylor series expansion,

$$f(x+dx) = f(x) + f'(x)\frac{dx}{1!} + f''(x)\frac{dx^2}{2!} + f'''(x)\frac{dx^3}{3!} + \cdots$$

If we neglect the higher order for dx as dx is a very small value and therefore the higher order term will become negligible,

$$f(x+dx) = f(x) + f'(x)\frac{dx}{1!} + f''(x)\frac{dx^2}{2!} \qquad \text{(Formula 2.8)}$$

Similarly,

$$f(x-dx) = f(x) - f'(x)\frac{dx}{1!} + f''(x)\frac{dx^2}{2!} \qquad \text{(Formula 2.9)}$$

If we add Formula 2.8 and 2.9,

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

$$f''(x) = (f(x+dx) + f(x-dx) - 2f(x))/dx^2 \qquad \text{(Formula 2.10)}$$

If we take Formula 2.10 for T with respect x and y,

$$\frac{d^2T}{dx^2} = \frac{T(x+dx,y) + T(x-dx,y) - 2T(x,y)}{dx^2} \qquad \text{(Formula 2.11a)}$$

$$\frac{d^2T}{dy^2} = \frac{T(x,y+dy) + T(x,y-dy) - 2T(x,y)}{dy^2} \qquad \text{(Formula 2.11b)}$$

3.0 Implement Finite Difference Method for Region

3.1 Establish Finite Difference Term for the Region

We divide the entire domain into small elements(subdomains), we divide it into small squares(subdomains). The intersections are called grid points. We use the Finite difference method to form the algebraic equation and we find the value of temperature at the grid points. Therefore, we can find the temperature distribution of the domain.

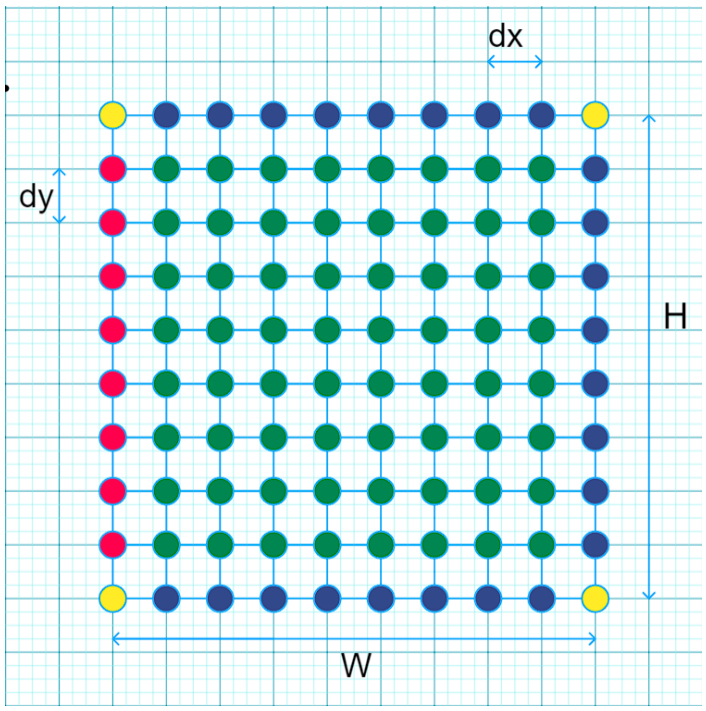


Figure 2: Distribution of temperature along the Domain

The Horizontal axis has index I,

i = 1: Nx

(Nx is number of grid points in x direction)

Similarly for vertical axis,

j = 1: Ny

(Ny is number of grid points in y direction)

Left side of the grid points on left side (red) are assigned with temperature values and the other sides are boundary grid points (blue). There are also corner grid points (yellow). There are also interior grid points (Green).

The size of the element in x direction is dx and dy in the y direction. The height of the entire domain is H and width is W. From these parameters we can find values of dx and dy that we can find in the x direction and y direction, which gives:

dx= W/ (Nx-1)

dy= H/(Ny-1)

We use this for simulation.

Now substituting Formula 2.11a and 2.11b in Governing differential equation,

(T(x+dx,y)+T(x-dx,y)-2T(x,y))/dx^2+(T(x,y+dy)+T(x,y-dy)-2T(x,y))/dy^2=0

Assuming, dx = dy,

$$T(x,y) = (\frac{T(x+dx,y)+T(x-dx,y)+T(x,y+dy)+T(x,y-dy)}{4}$$

$$T(i,j) = (\frac{T(i+1,j)+T(i-1,j)+T(i,j+1)+T(i,j-1)}{4} \qquad \text{(Formula 3.1)}$$

The equation is the average of all the surrounding grid points. We use this in code.

4.0 Boundary Conditions

Boundary conditions are applied on the edges. On the left and right edge, direct temperature values are assigned. The top and bottom edges are critical, since we have to design the terms which will be implemented in MATLAB later on.

For top edge: -

$$q = 0 \text{ (heat } flux \text{)}$$
$$\text{since, } q = -\frac{\Delta T}{\Delta y} \cdot k$$
$$\Delta T = 0$$

This gives,

$$T \text{ (edge node)} = T \text{ (previous node)}$$

For Bottom edge: -

Here, q is applied through Neumann Boundary condition
Again, q (heat flux) is given as

$$q = -k\frac{\Delta T}{\Delta y}$$

Where,

$$\Delta T = T\text{ (edge node) } - T\text{ (previous node)}$$

After simplifying.

$$T\text{ (edge node)} = T\text{ (previous node)} + \frac{q}{k} \cdot \Delta y$$

5.0 MATLAB solver

The MATLAB code is constructed using iterative solver based on derived FDM term and subdivided into:

1. Geometric Parameters
2. Material Properties
3. Boundary and Initial conditions
4. Computation
5. Plotting Error value achieved
6. Potting Temperature values

5.1 Geometric Parameters for Domain

```
%% Geometric Parameters for Domain
W = 2; % Width of domain
L = 1; % Length of domain
Nx = 41; % Number of nodes in x-direction
Ny = 21; % Number of nodes in y-direction
dx = W / (Nx - 1); % Step length in x-direction
dy = L / (Ny - 1); % Step length in y-direction
```

Figure 3: Code section for Geometric Parameters for Domain

Explanation:

- Set the width and length of the domain to be 2 units and 1 unit, respectively.
- Divide the domain into a grid with 41 points along the x-axis and 21 points along the y-axis.
- Calculate the spacing between grid points along each axis (dx and dy).

5.2 Material properties

```
%% Material Properties
k = 50; % Thermal conductivity coefficient
```

Figure 4:Code section signifying Material Properties

Explanation:

- Define the thermal conductivity of the material as 50.

5.3 Boundary Conditions

```
%% Boundary and Initial Conditions
Ti = 0; % Initial temperature
T = Ti * ones(Nx, Ny); % Temperature matrix with initial value (0 C)
T1 = 30; % Temperature at left edge
T2 = 10; % Temperature at right edge
Q1 = -420; % Heat flux at bottom edge

T(1, :) = T1; % left edge
T(Nx, :) = T2; % right edge
```

Figure 5:Code section for specifying the Boundary Conditions

Explanation:

- Set the initial temperature (Ti) of the entire domain to 0.
- Create a temperature matrix representing the domain with initial values set to Ti.
- Assign specific temperatures to the left and right edges of the domain (T1 and T2, respectively).
- Specify a heat flux (Q1) at the bottom edge.

5.4 Computation

```
%% Computation
error_history = [];
Epsilon = 1e-5;
Error = 5;

Iter = 0;
while (Error > Epsilon)
  Iter = Iter + 1;
  disp(Iter);
  Told = T;
  for j = 2:Ny-1
    for i = 2:Nx-1
        T(i, j) = (T(i + 1, j) + T(i - 1, j) + T(i, j + 1) + T(i, j - 1)) / (4);
    end
  end

  % Update boundary conditions
  T(2:end-1, 1) = T(2:end-1, 2) + Q1 * dy/k; % Neumann boundary condition Q = -420 W/m^2
  T(2:end-1, end) = T(2:end-1, end-1); % Adiabatic boundary condition Q = 0 W/m^2

  Error = sqrt(sumsqr(T - Told));
  disp(Error);
  error_history = [error_history, Error];
end
```

Figure 6:Code section for Computation of the Problem

Explanation:

- error history: A new array is initialized to store the history of errors during each iteration.
- Boundary conditions are updated separately after updating the interior points. This helps maintain clarity and ensures that boundary conditions are applied after the internal points have been updated.
- Inside the loop:
    - Store the current temperature matrix for comparison.
    - Update the temperature at each interior point using the finite difference method.
- The display statements show the iteration number and the corresponding error for better monitoring during the simulation.

5.5 Plotting Error values

```
%% Plotting the error curve
figure(1);
plot(1:Iter, error_history, '-o');
title('Convergence of Error Over Iterations');
xlabel('Iteration');
ylabel('Error');
grid on;
```

Figure 7:Code Section for plotting the error values.

Explanation:

- Use the plot function to generate a line plot of the error history.
- The x-axis represents the iteration number (1 to Iter), and the y-axis represents the corresponding error values.

5.6 Plotting 2D and 3D Temperature values:

```
%% Plotting 2D T result
figure(2);
x = 0:dx:W;
y = 0:dy:L;
colormap(jet);
contourf(x, y, T', 'LineColor', 'None');
colorbar;
title("2D Temperature Values");
xlabel("x-direction");
ylabel("y-direction");

%% Plotting 3D T result
% Different Visualization Approach
figure(3);

% Meshgrid for 3D surface plot
[X_vals, Y_vals] = meshgrid(0:dx:W, 0:dy:L);

% 3D Surface Plot
surf(X_vals, Y_vals, T', 'EdgeColor', 'none');
colormap(jet);
colorbar;

% Title and Labels
title("3D Temperature Values");
xlabel("X-direction");
ylabel("Y-direction");
zlabel("Temperature");

% View Perspective
view(45, 30);
```

Figure 8:Code section for Plotting 2D and 3D Temperature values

Explanation:

- Create a contour plot in 2D and 3D of the temperature matrix with labeled axes and a color bar.

6.0 Results & Discussions

6.1 Results

The code is structured to generate two types of graphs:

1. Displays the cumulative error computation and illustrates it against iterations.
2. Visualizes temperature in 2D.
3. Provides a 3D visualization of temperature.

Plot 1:

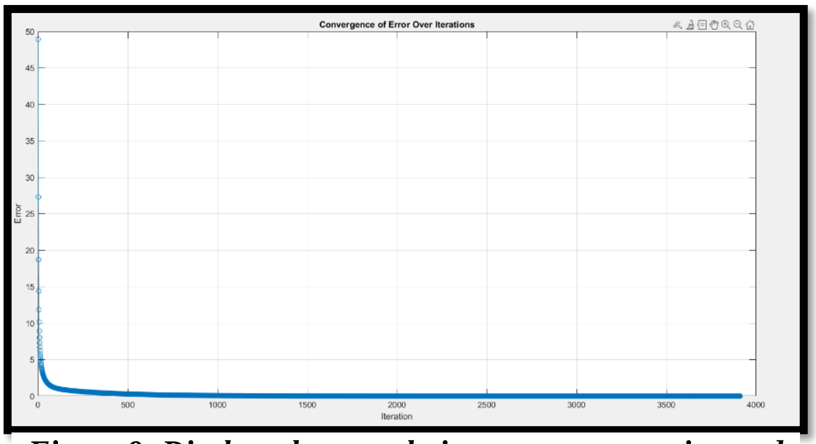


Figure 9: Displays the cumulative error computation and illustrates it against iterations.

Plot 2:

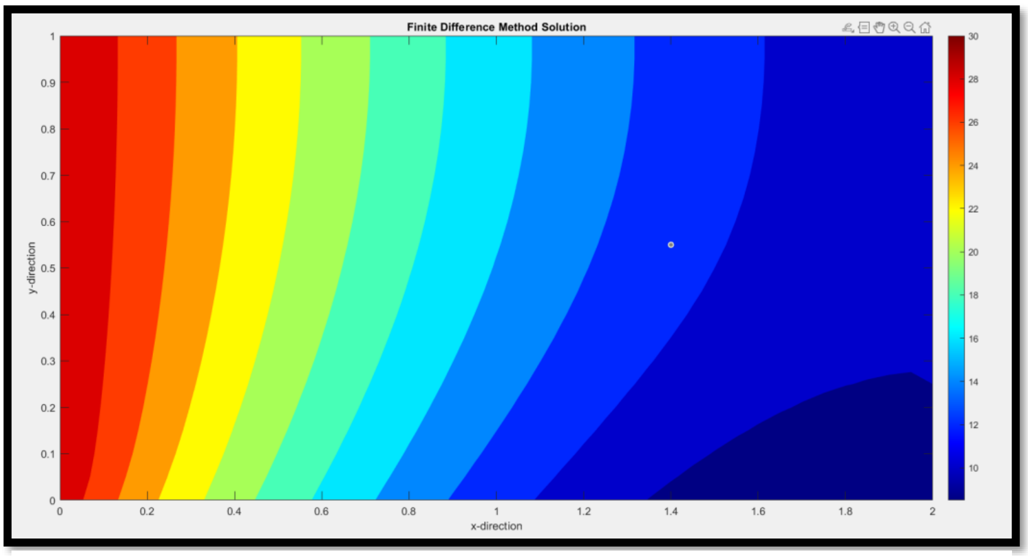


Figure 10: Visualizes temperature in 2D.

Plot 3:

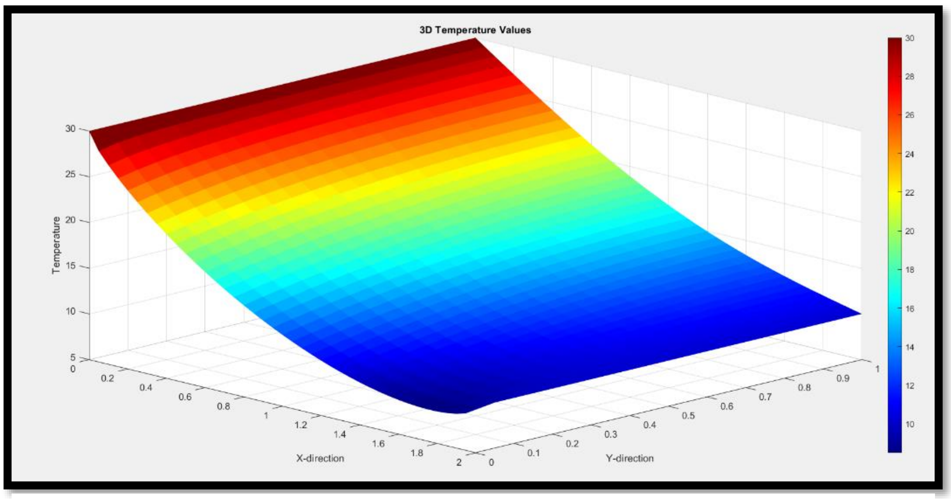


Figure 11: Provides a 3D visualization of temperature.

6.2 Discussion

Plot 1:

- Initially, the error is large, indicating the solution's inaccuracy in the early iterations.
- With continued iterations, the error decreases and reaches the threshold after 3911 iterations.
- The diminishing error suggests convergence, prompting the end of computation for temperature plotting.

Plot 2 & 3:

- Temperature Distribution Interpretation:
    - Top Region:
        - With zero heat exchange at the top edge (q=0), there's no vertical temperature difference.
        - Gradual color change horizontally confirms the linear heat flow.
    - Bottom Region:
        - Negative heat flux at the bottom edge causes a vertical temperature decrease.
        - Along a central horizontal line, a top-to-bottom temperature reduction aligns with the applied boundary condition.
    - Left Region:
        - The left region adheres to a temperature boundary condition of 30°C.
    - Right Region:
        - The right region follows a temperature boundary condition of 10°C.

This interpretation provides a clear and concise overview of the temperature distribution and convergence behavior based on the provided MATLAB solver results.

6.3 Validation of results

Validation of the obtained results is crucial to establish confidence in the accuracy of the model. Two validation methods will be employed:

6.3.1 Hand Calculation of Internal Node Values from MATLAB Results

- Extract a 4 x 4 node-based matrix from the temperature solution obtained in MATLAB.
- Conduct manual calculations to determine the internal node values.
- Validate these hand-calculated results against the MATLAB-derived results to ensure coherence and accuracy in the model's predictions.

6.3.2 Steady State 2D Heat Conduction Simulation in ANSYS

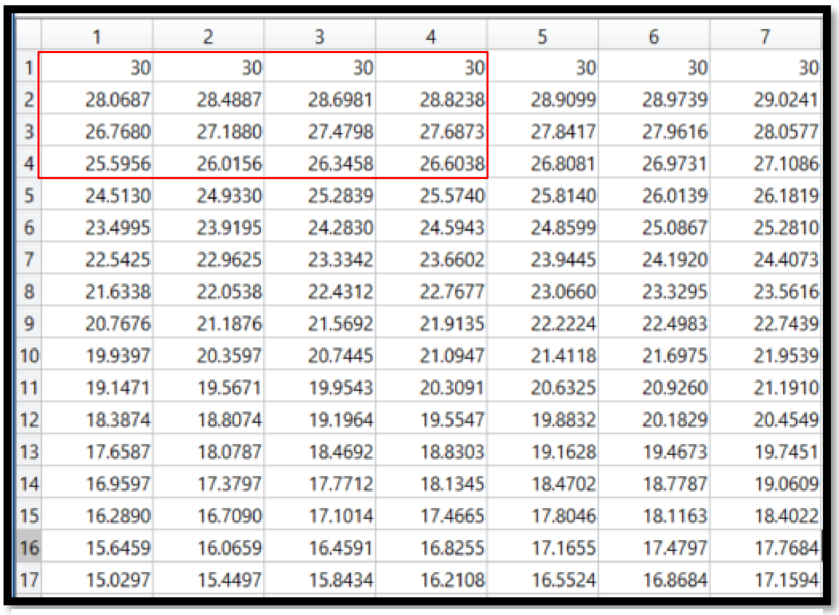
- Utilize ANSYS to perform a Steady State 2D Heat Conduction Simulation.
- Compare the ANSYS results with those obtained from MATLAB to validate the consistency of the simulation outcomes.

This comprehensive approach combines the precision of simulation software (ANSYS) with manual calculations, providing a thorough validation process for the temperature distribution model.

6.3.3 Hand Calculations:

A 4 x 4 node-based matrix is extracted from the MATLAB code and translated into a 2D domain to facilitate hand calculations.

The extracted node is as follows:



	1	2	3	4	5	6	7
1	30	30	30	30	30	30	30
2	28.0687	28.4887	28.6981	28.8238	28.9099	28.9739	29.0241
3	26.7680	27.1880	27.4798	27.6873	27.8417	27.9616	28.0577
4	25.5956	26.0156	26.3458	26.6038	26.8081	26.9731	27.1086
5	24.5130	24.9330	25.2839	25.5740	25.8140	26.0139	26.1819
6	23.4995	23.9195	24.2830	24.5943	24.8599	25.0867	25.2810
7	22.5425	22.9625	23.3342	23.6602	23.9445	24.1920	24.4073
8	21.6338	22.0538	22.4312	22.7677	23.0660	23.3295	23.5616
9	20.7676	21.1876	21.5692	21.9135	22.2224	22.4983	22.7439
10	19.9397	20.3597	20.7445	21.0947	21.4118	21.6975	21.9539
11	19.1471	19.5671	19.9543	20.3091	20.6325	20.9260	21.1910
12	18.3874	18.8074	19.1964	19.5547	19.8832	20.1829	20.4549
13	17.6587	18.0787	18.4692	18.8303	19.1628	19.4673	19.7451
14	16.9597	17.3797	17.7712	18.1345	18.4702	18.7787	19.0609
15	16.2890	16.7090	17.1014	17.4665	17.8046	18.1163	18.4022
16	15.6459	16.0659	16.4591	16.8255	17.1655	17.4797	17.7684
17	15.0297	15.4497	15.8434	16.2108	16.5524	16.8684	17.1594

Figure 12:4x4 node based matrix from the MATLAB code for Hand Calculations

The domain can now be designed as:

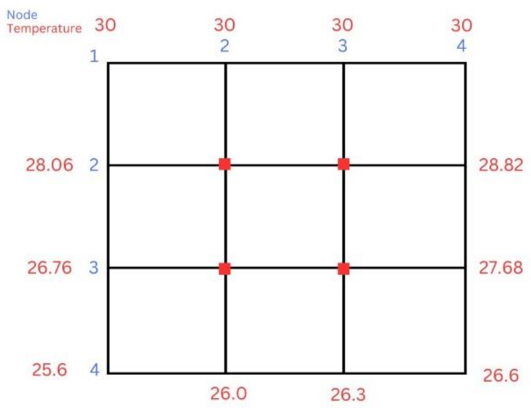


Figure 13:Domain designed from 4x4 matrix from MATLAB code.

Following hand calculations on the 2D domain, implement the finite difference method. This involves creating homogeneous equation and use Ax=b matrix system to solve them directly, making it easier to solve by hand:
Finite difference term used:

$$T_{m,n} = \frac{1}{4}\left(T_{m-1,n} + T_{m+1,n} + T_{m,n-1} + T_{m,n+1}\right)$$

Creating equations at internal nodes:

$$T_{2,2} = \frac{1}{4}\left(28.06 + 30 + T_{3,2} + T_{2,3}\right)$$
$$T_{3,2} = \frac{1}{4}\left(T_{2,2} + 30 + 28.82 + T_{3,3}\right)$$
$$T_{2,3} = \frac{1}{4}\left(26.76 + T_{2,2} + T_{3,3} + 26\right)$$
$$T_{3,3} = \frac{1}{4}\left(T_{2,3} + T_{3,2} + 27.68 + 26.3\right)$$

Converting $Ax = b$

$$\begin{bmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix}\begin{bmatrix} T_{2,2} \\ T_{3,2} \\ T_{2,3} \\ T_{3,3} \end{bmatrix} = \begin{bmatrix} 58.06 \\ 58.82 \\ 52.76 \\ 53.98 \end{bmatrix}$$

The values for the temperatures are solved using the direct method.
The following table compares the hand-calculated temperatures with the corresponding MATLAB-computed values and computes absolute error.

Table 1:Comparison of results from MATLAB code and Hand Calculations

Node Temperature	Value (Hand Calculation)	Value (MATLAB)	Absolute Error Value1 – Value2
$T_{2,2}$	28.48	28.48	0.00
$T_{3,2}$	28.69	28.69	0.00
$T_{2,3}$	27.17	27.18	0.01
$T_{3,3}$	27.46	27.47	0.01

A close match indicates that the MATLAB code accurately simulates the heat conduction problem.

ANSYS Simulation:
A steady-state thermal simulation was conducted in ANSYS to emulate the assumptions employed in the MATLAB computation. The comparison of results between the two solvers can be comprehended through two distinct approaches:

Visual Comparison:
Visual inspection involves scrutinizing the plots obtained from both ANSYS and MATLAB simulations.

MATLAB Plot:

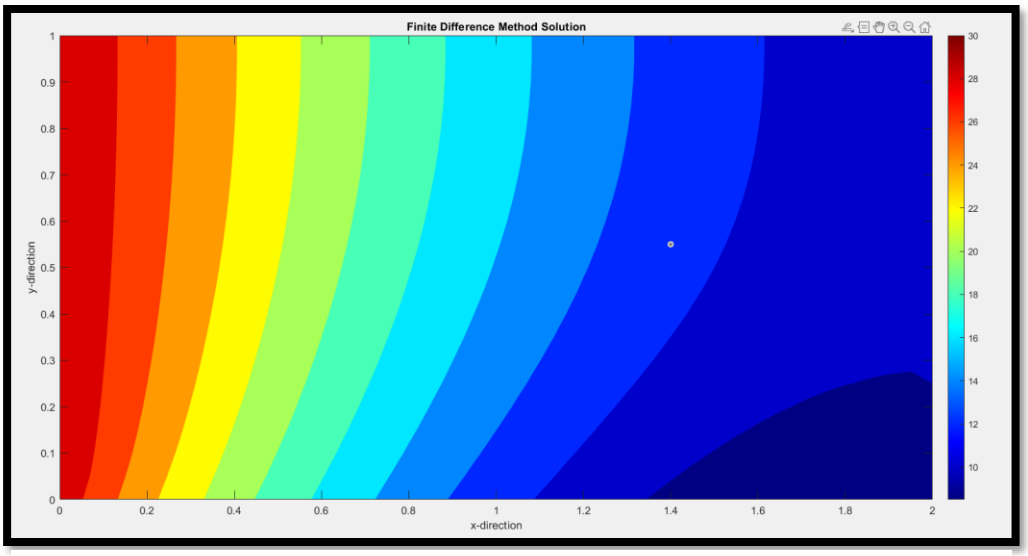


Figure 14:Plot derived from MATLAB.

ANSYS Plot:

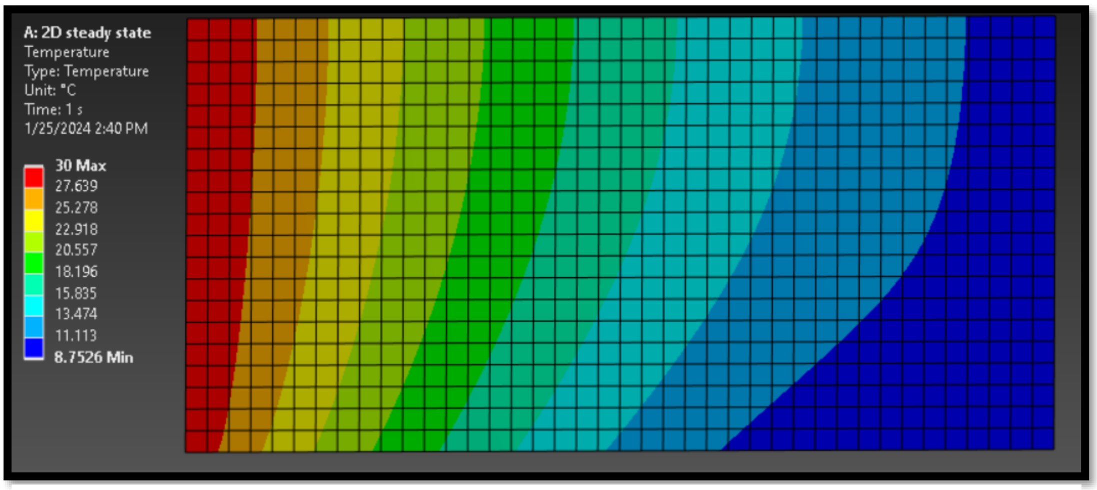


Figure 15:Plot derived from ANSYS

Interpretation:

Upon careful visual inspection, it is evident that the trends in heat flow and temperature values exhibit remarkable similarity between the MATLAB and ANSYS results. The visually aligned plots indicate a congruence in the thermal behavior predicted by both solvers.

Temperature Numerical Comparison:

To further validate the accuracy of the simulations, temperature values at randomly selected nodes were extracted from both ANSYS and MATLAB simulations, facilitating a direct numerical comparison.

This can be observed in the following table:

Table 2:Comparison of the results obtained from ANSYS and MATLAB

Node No.	ANSYS	MATLAB	Absolute Error
11	30	30	0
602	29.2	29.2	0.0
526	28.3	28.3	0.0
748	27.5	27.5	0.1
814	26.7	26.6	0.1
838	25.9	25.8	0.1
813	25.1	25.0	0.1
225	24.4	24.2	0.1
747	23.6	23.5	0.2
141	22.9	22.7	0.2

Interpretation:

Upon comparing the temperature values at these random nodes, a strong agreement between ANSYS and MATLAB results was observed. This not only complements the visual alignment of plots but also provides a quantitative measure of accuracy. The convergence in temperature values at specific locations enhances our confidence in the reliability of both computational models.

7.0 Conclusion

In summary, the MATLAB code effectively simulates 2D heat conduction using an iterative approach for computational efficiency with a large number of nodes. The well-structured setup of geometric parameters, material properties, and boundary conditions enhances the code's clarity. Validation through ANSYS simulations and hand calculations confirms the accuracy of the implemented finite difference method. The visualization tools, including error plotting, 2D temperature value plot, and 3D temperature value plot, provide clear insights into convergence, temperature distribution, and thermal behavior for a comprehensive understanding of the simulation results.

8.0 Bibliography

Lecture Notes

https://visualslope.com/Library/FDM-for-heat-transfering.pdf

Determining Temperature Distribution of a 2D Heated Plate , by Kristen Stewart