ME 605 — Computational Fluid Dynamics Project 1

Tanmay Kumar Garg(23110330)

December 5, 2025

Contents

1 Problem Statement 2

1.1 Governing Equation . . . 2

1.2 Boundary Conditions . . . 2

1.3 Tasks . . . 3

2 Mesh Details and Approach for Discretization 3

2.1 Discretization Method . . . 3

3 Derivation and Presentation of the Final Form of the Discretised Equations 4

3.1 Approximation of Second Derivatives . . . 4

3.2 Discretized Equation for an Interior Node . . . 4

3.3 Discretized Equations for Boundary Nodes . . . 5

3.3.1 Dirichlet Boundaries . . . 5

3.3.2 Convective (Left) Boundary: First-Order Forward Difference . . . 5

3.3.3 Convective (Left) Boundary: Second-Order Central Difference (Ghost Node Method) . . . 5

4 Solution Methodology and Results 6

4.1 Gaussian Elimination Method . . . 6

4.1.1 Algorithmic Procedure . . . 7

4.1.2 Example: 5x5 Grid Discretization . . . 7

4.1.3 Results . . . 9

4.1.4 Time Complexity of Gaussian Elimination . . . 10

4.2 Gauss-Seidel (Iterative Method) . . . 11

4.2.1 Results . . . 11

4.2.2 Discussion . . . 12

4.3 Line-by-Line Method . . . 13

4.3.1 Row Sweep: . . . 13

4.3.2 Column Sweep: . . . 16

4.4 Alternating Direction Implicit (ADI) Method . . . 17

4.4.1 Results . . . 17

4.4.2 Discussion . . . 19

5 Concluding Remarks 20

1 Problem Statement

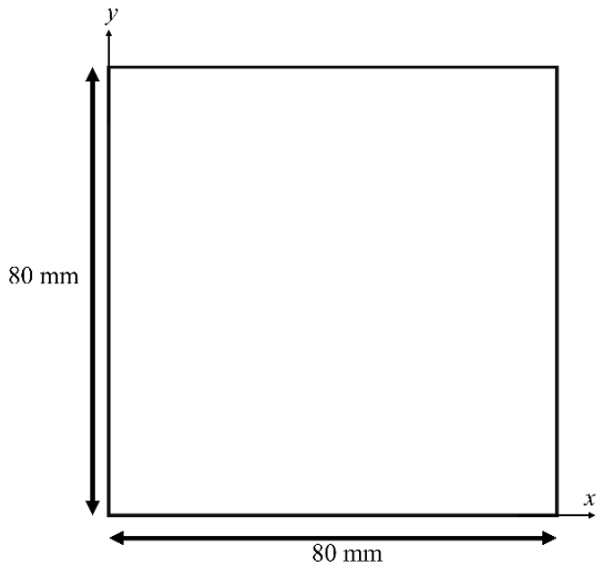
This project addresses the numerical solution of a two-dimensional, steady-state heat conduction problem within a defined physical domain, subject to a set of mixed boundary conditions.

1.1 Governing Equation

The physical process is governed by the 2D steady-state heat conduction equation with no internal heat generation. This is a form of the Laplace equation, an elliptic partial differential equation, expressed as:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{1}$$

where $T(x, y)$ represents the temperature field as a function of the spatial coordinates x and y.



1.2 Boundary Conditions

The temperature field within the domain is determined by the conditions imposed on its four boundaries. Three boundaries are subject to Dirichlet conditions (fixed temperature), while one is subject to a convective (Neumann-type) condition. The conditions are specified as follows:

$$\begin{aligned} T(x, y = 0) &= 323\,\mathrm{K} && \text{(Bottom Boundary: Dirichlet)} \\ T(x, y = 80\,\mathrm{mm}) &= 423\,\mathrm{K} && \text{(Top Boundary: Dirichlet)} \\ T(x = 80\,\mathrm{mm}, y) &= 473\,\mathrm{K} && \text{(Right Boundary: Dirichlet)} \\ h(T_\infty - T(x = 0, y)) &= -\lambda \left.\frac{\partial T}{\partial x}\right|_{x=0} && \text{(Left Boundary: Convective)} \end{aligned}$$

The parameters for the convective boundary condition are given as:

- Convective heat transfer coefficient, $h = 250$ W/m^{2}K
- Thermal conductivity of the material, $\lambda = 5$ W/mK
- Free stream temperature, $T_\infty = 573$ K

1.3 Tasks

1. Solve the governing equation using the Gaussian elimination method for the following grids: 11, 21, and 41 grid points in each direction. Show the computed T field as a contour plot for the finest grid. Plot the CPU run time vs total number of grid points and discuss the trend, and comment on the computational efficiency of the Gauss elimination method.

2. Solve the governing equation using the Gauss-Seidel iterative method for the following three grids: 11, 21, and 41 grid points in each direction. Show the computed T field as a contour plot for the finest grid. Plot the residual vs the number of iterations for the three grids in the same plot. Compare and discuss the dependence of the convergence rate on the total number of grid points. Plot the variation of CPU run time with the total number of grid points for the Gauss-Seidel iterative method and discuss the trend. How do the CPU run times of the Gauss-Seidel iterative method compare with those of the Gauss elimination method?

3. Solve the governing equation using the line-by-line method (row sweep) for the following three grids: 11, 21, and 41 grid points in each direction. Please note that you will need to use the TDMA solver to solve the resulting system of linear equations because of the tridiagonal structure of the resulting coefficient matrices. Show the computed T field as a contour plot for the finest grid. Plot residual vs number of iterations for the line-by-line method and for the Gauss-Seidel method for the 41×41 grid (both in the same plot). Compare and discuss the trends. Plot the variation of CPU run time with the total number of grid points for the line-by-line method. How do the CPU run times and the obtained scaling compare with those obtained for the Gauss elimination method and the point-wise iterative method?

4. Solve the governing equation using the Alternating Direction Implicit (ADI) method for the 21×41 grid and plot the residual vs number of iterations for row-wise sweep, column-wise sweep, and ADI method in the same plot. Discuss the trends.

2 Mesh Details and Approach for Discretization

To solve the continuous governing partial differential equation numerically, we have to discretise the domain of interest into a finite set of points. To do so, the Finite Difference Method (FDM) is used for this task.

2.1 Discretization Method

Note: In this report, the notation (i,j) refers to the element in the i-th row and j-th column of a matrix, not a coordinate point (x,y). The FDM approximates the continuous problem by replacing the partial derivatives with algebraic finite difference approximations, which introduces an error term in the solution. As we increase the number of nodes, the error term reduces. This process transforms the single PDE into a system of simultaneous algebraic equations, which are easier to solve. In this problem, a uniform Cartesian grid is superimposed over the 80 mm $\times$ 80 mm square domain. The nodes of this grid are indexed by a pair of integers (i, j), where i denotes the row and j denotes the column. Let N_x and N_y be the number of grid points in the x and y directions, respectively, and L_x and L_y be the corresponding domain lengths. The grid spacing in each direction is defined as:

$$\Delta x = \frac{L_x}{N_x - 1} \quad \text{and} \quad \Delta y = \frac{L_y}{N_y - 1}$$

3 Derivation and Presentation of the Final Form of the Discretised Equations

The following derivations are presented for the general case where $\Delta x \neq \Delta y$. The project requires simulations for three grid resolutions: 11×11, 21×21, and 41×41 for the first three problem statements. But task 4 requires a grid size of 21×41 points.

3.1 Approximation of Second Derivatives

The second partial derivatives in the Laplace equation are approximated using second-order accurate central difference schemes. These are derived from Taylor series expansions around a node (i, j). For the x-direction, the central difference formula is: ***Note: in $T_{i,j}$, i,j represent the row and column, respectively.***

$$\left.\frac{\partial^2 T}{\partial x^2}\right|_{i,j} = \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{(\Delta x)^2} + O((\Delta x)^2)$$

Similarly, for the y-direction:

$$\left.\frac{\partial^2 T}{\partial y^2}\right|_{i,j} = \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{(\Delta y)^2} + O((\Delta y)^2)$$

3.2 Discretized Equation for an Interior Node

Substituting these central difference approximations into the governing Laplace equation (Eq. 1) gives the finite difference equation for any interior node (i, j):

$$\frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{(\Delta x)^2} + \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{(\Delta y)^2} = 0$$

Multiplying by $(\Delta x)^2$ in LHS and RHS yields:

$$T_{i,j+1} - 2T_{i,j} + T_{i,j-1} + (\Delta x)^2 \left[\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{(\Delta y)^2}\right] = 0$$

Grouping the terms we get:

$$T_{i,j+1} - 2T_{i,j}\left[1 + \frac{(\Delta x)^2}{(\Delta y)^2}\right] + T_{i,j-1} + \frac{(\Delta x)^2}{(\Delta y)^2}T_{i+1,j} + \frac{(\Delta x)^2}{(\Delta y)^2}T_{i-1,j} = 0$$

multiply and divide the coefficient of $T_{i,j}$ by 2, we get:

$$T_{i,j+1} - 4T_{i,j}\left[\frac{(\Delta x)^2 + (\Delta y)^2}{2(\Delta y)^2}\right] + T_{i,j-1} + \frac{(\Delta x)^2}{(\Delta y)^2}T_{i+1,j} + \frac{(\Delta x)^2}{(\Delta y)^2}T_{i-1,j} = 0 \tag{2}$$

Let's call:

$$p_1 = \frac{(\Delta x)^2 + (\Delta y)^2}{2(\Delta y)^2}, p_2 = \frac{(\Delta x)^2}{(\Delta y)^2} \tag{3}$$

So, rewriting equation 2 in terms of p_1 and p_2:

$$T_{i,j+1} + T_{i,j-1} + p_2 T_{i+1,j} + p_2 T_{i-1,j} - 4p_1 T_{i,j} = 0 \tag{4}$$

Equation (4) is applied for all nodes where $i \in [2, N_y - 1]$, $\quad j \in [1, N_x - 1]$

For the special case where $\Delta x = \Delta y$, this equation simplifies to the well-known result that the temperature at an interior node is the arithmetic average of its four neighbors:

$$T_{i,j+1} + T_{i,j-1} + T_{i+1,j} + T_{i-1,j} - 4p_1 T_{i,j} = 0 \tag{5}$$

3.3 Discretized Equations for Boundary Nodes

3.3.1 Dirichlet Boundaries

For nodes on the top ($j = N_y$), bottom ($j = 1$), and right ($i = N_x$) boundaries, the temperatures are known constants:

$$\begin{aligned} T_{1,j} &= 323\,\mathrm{K} && \text{for } j = 1, \ldots, N_x \\ T_{N_y,j} &= 423\,\mathrm{K} && \text{for } j = 1, \ldots, N_x \\ T_{i,N_x} &= 473\,\mathrm{K} && \text{for } i = 2, \ldots, N_y - 1 \end{aligned}$$

3.3.2 Convective (Left) Boundary: First-Order Forward Difference

The convective boundary condition at $x = 0$ (corresponding to nodes with index $j = 1$) is:

$$h(T_\infty - T_{i,1}) = -\lambda \left.\frac{\partial T}{\partial x}\right|_{i,1}$$

A first-order forward difference scheme is employed for the gradient term:

$$\left.\frac{\partial T}{\partial x}\right|_{i,1} \approx \frac{T_{i,2} - T_{i,1}}{\Delta x}$$

Substituting this into the boundary condition:

$$h(T_\infty - T_{i,1}) = -\lambda \frac{T_{i,2} - T_{i,1}}{\Delta x}$$

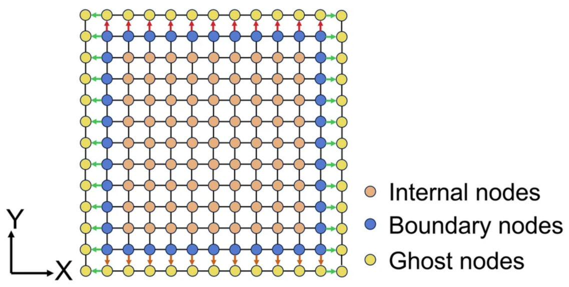
rearranging the terms:

$$T_{i,1}\left[-\lambda - h\Delta x\right] + \lambda T_{i,2} = -h\Delta x T_\infty$$

This equation says that at the left boundary point, the temperature at some point is a function of the temperature of the node next to it in the same row. But from equation (4) we have derived that the temperature at node depends on the temperature of nodes adjacent to it (i.e above, below, left, right), which is also applicable to the left convective boundary. So we need a more accurate numerical method that simulates the governing physics at the leftmost boundary more accurately.

3.3.3 Convective (Left) Boundary: Second-Order Central Difference (Ghost Node Method)

[1]



To achieve a more accurate, second-order approximation for the convective boundary condition, a central difference scheme can be used. This approach requires the introduction of a

ghost node $(i, 0)$ outside the physical domain. The central difference approximation for the gradient at the boundary node $(i, 1)$ is:

$$\left.\frac{\partial T}{\partial x}\right|_{i,0} \approx \frac{T_{i,2} - T_{i,0}}{2\Delta x}$$

Substituting this into the convective boundary condition:

$$h(T_\infty - T_{i,1}) = -\lambda \frac{T_{i,2} - T_{i,0}}{2\Delta x}$$

Write the ghost node temperature as:

$$T_{i,0} = T_{i,2} + \frac{2h\Delta x}{\lambda}(T_\infty - T_{i,1}) \tag{6}$$

We then apply the general discretised Laplace equation (4) at the boundary node $(i, 1)$:

$$T_{i,2} + T_{i,0} + p_2 T_{i+1,1} + p_2 T_{i-1,1} - 4p_1 T_{i,1} = 0$$

Substitute the ghost node temperature in the discretised Laplace equation:

$$2T_{i,2} + p_2 T_{i+1,1} + p_2 T_{i-1,1} + T_{i,1}\left[-4p_1 - \frac{2h\Delta x}{\lambda}\right] = -\frac{2h\Delta x T_\infty}{\lambda} \tag{7}$$

To simplify the appearance, let's call:

$$c_1 = \left[-4p_1 - \frac{2h\Delta x}{\lambda}\right], c_2 = -\frac{2h\Delta x T_\infty}{\lambda}$$

Rewriting equation 7 in using c_1 and c_2:

$$2T_{i,2} + p_2 T_{i+1,1} + p_2 T_{i-1,1} + c_1 T_{i,1} = c_2 \tag{8}$$

Equation (8) is applied for all nodes on the left boundary, from $i = 2$ *to* $i = N_y - 1$.

4 Solution Methodology and Results

The application of the finite difference method results in a large, sparse system of linear algebraic equations of the form $[A]\mathbf{T} = \mathbf{B}$. This project explores four distinct methods for solving this system.

4.1 Gaussian Elimination Method

Gaussian elimination is a fundamental and robust **direct method** for solving a system of linear algebraic equations of the form $[A]T = B$. Here, $[A]$ is the $N \times N$ coefficient matrix, T is the $N \times 1$ column vector of unknown temperatures, and B is the $N \times 1$ source vector containing known boundary condition information. As a direct solver, it finds the exact solution (to within machine precision) in a finite, predetermined number of steps. The algorithm is executed in two primary phases: Forward Elimination and Backward Substitution.

4.1.1 Algorithmic Procedure

Phase 1: Forward Elimination In this phase, we transform the original system into an equivalent system $[U]T = B'$, where $[U]$ is an **upper triangular matrix**. This is achieved by systematically creating zeros in all positions below the main diagonal of the matrix $[A]$.

The process operates column by column. For each column k (from 1 to $N-1$), the diagonal element $A(k,k)$ is called as the **pivot element**. For every row i below the pivot row (i.e., for $i = k+1, \dots, N$), a we define a multiplier as:

$$m = \frac{A(i,k)}{A(k,k)} \tag{9}$$

This multiplier is then used to perform a row operation:- the entire pivot row k is multiplied by m and subtracted from row i. This operation eliminates the element $A(i,k)$ (making it zero) and updates the remaining elements in row i as well as the corresponding element in the B. This can be mathematically shown as:

$$A(i,j) \leftarrow A(i,j) - m \times A(k,j) \quad \text{for } j = k, \dots, N \tag{10}$$

$$B(i) \leftarrow B(i) - m \times B(k) \tag{11}$$

This step is repeated until all the entries below the main diagonal become zero.

Phase 2: Backward Substitution Once we have the upper triangular form $[U]T = B'$, the solution vector T can be found by solving the equations from the last equation to the first.

The last unknown, T_N, is calculated from the last row of the system:

$$T_N = \frac{B'(N)}{U(N,N)} \tag{12}$$

The remaining unknowns are then found recursively for $i = N-1, N-2, \dots, 1$. The general formula for T_i involves substituting the already-known values of $T_{i+1}, \dots, T_N$:

$$T_i = \frac{1}{U(i,i)} \left(B'(i) - \sum_{j=i+1}^{N} U(i,j) T_j \right) \tag{13}$$

This process gives the solution vector T. At last, we need to map the 1D T vector to the 2D grid nodes to get out temperature field.

4.1.2 Example: 5x5 Grid Discretization

Consider a 5×5 grid ($N_x = 5, N_y = 5$). The boundary nodes at $i = 5$ (bottom), $i = 1$ (top), and $j = 5$ (right) are known from the Dirichlet conditions. The nodes at $j = 1$ are subject to the convective boundary condition. The interior temperatures are the unknowns.

The unknown nodes are at grid points (i,j) for $i \in [2,4], \quad j \in [1,4]$. This gives a total of $(5-2) \times (5-1) = 3 \times 4 = 12$ unknown temperatures. We map these 12 unknowns from the 2D grid to a 1D vector and call it $\mathbf{T}$ matrix, which has dimensions 12x1.

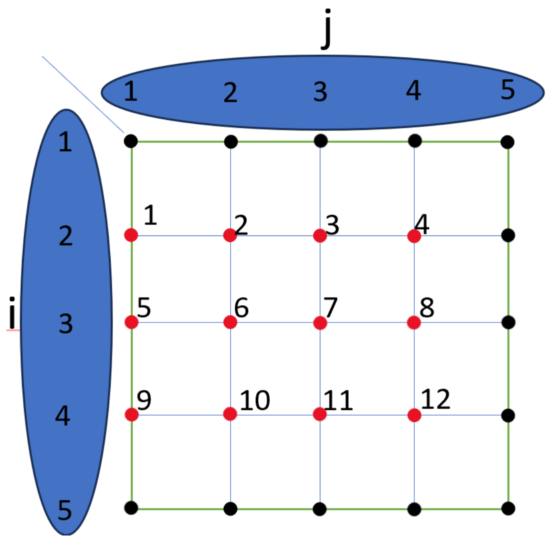


Figure 1: Nodal numbering for the 12 unknown temperatures on a 5x5 grid. Red nodes are unknowns, black nodes have known boundary values.

The system of equations $[A]T = B$ will be 12×12. The equations for each unknown $T_{i,j}$ are derived using equation (4) and (8). Here $c_1 = -(4p_1 + \frac{2\Delta xh}{\lambda})$, $c_2 = -\frac{2\Delta xh}{\lambda}T_\infty$, $p_1 = 1$, and $p_2 = 1$ as $\Delta x = \Delta y$.

Row i=2:

$$Node = 1 : c_1T_1 + 2T_2 + T_5 = c_2 - T_{top}$$

$$Node = 2 : T_1 - 4T_2 + T_3 + T_6 = -T_{top}$$

$$Node = 3 : T_2 - 4T_3 + T_4 + T_7 = -T_{top}$$

$$Node = 4 : T_3 - 4T_4 + T_8 = -T_{top} - T_{right}$$

Row i=3:

$$Node = 5 : T_1 + c_1T_5 + 2T_6 + T_9 = c_2$$

$$Node = 6 : T_2 + T_5 - 4T_6 + T_7 + T_{10} = 0$$

$$Node = 7 : T_3 + T_6 - 4T_7 + T_8 + T_{11} = 0$$

$$Node = 8 : T_4 + T_7 - 4T_8 + T_{12} = -T_{right}$$

Row i=4:

$$Node = 9 : T_5 + c_1T_9 + 2T_{10} = c_2 - T_{bottom}$$

$$Node = 10 : T_6 + T_9 - 4T_{10} + T_{11} = -T_{bottom}$$

$$Node = 11 : T_7 + T_{10} - 4T_{11} + T_{12} = -T_{bottom}$$

$$Node = 12 : T_8 + T_{11} - 4T_{12} = -T_{right} - T_{bottom}$$

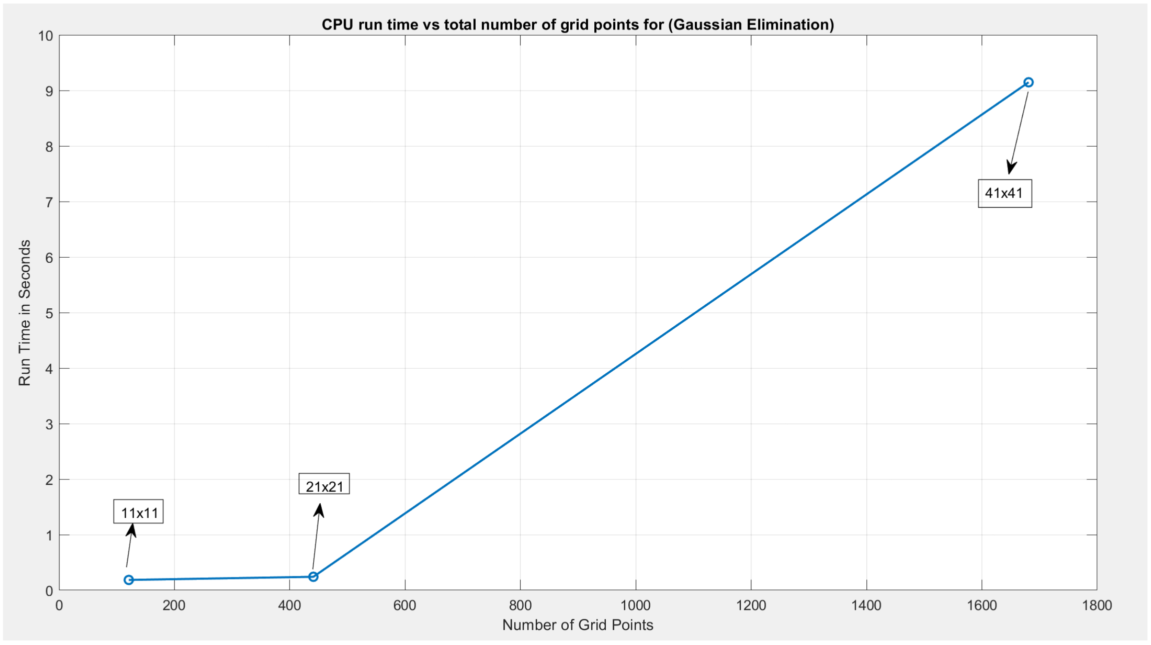
We can write this system of equations in matrix form, as a computer can also understand matrices, so we can write a code to solve this system of equations simultaneously. The resulting 12×12 coefficient matrix $[A]$ is sparse and has a penta-diagonal structure. The source vector B contains the known constant for the corresponding grid.

$$A = \begin{bmatrix} c_1 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & c_1 & 2 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & c_1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 \end{bmatrix}$$

$$B = \begin{bmatrix} c_2 - T_{\text{top}} \\ -T_{top} \\ -T_{top} \\ -T_{top} - T_{right} \\ c_2 \\ 0 \\ 0 \\ -T_{right} \\ c_2 - T_{bottom} \\ -T_{bottom} \\ -T_{bottom} \\ -T_{bottom} - T_{right} \end{bmatrix}$$

Now apply forward elimination, then backward substitution to get the unknown node temperatures.

4.1.3 Results



From this plot, the trend shows that as the number of points in the grid increased, the run time increased rapidly when we use Gaussian elimination. For an $n \times n$ grid, we have approximately

n^2 grid points where the temperature is unknown. This means we are solving a system of roughly $N = n^2$ linear equations.

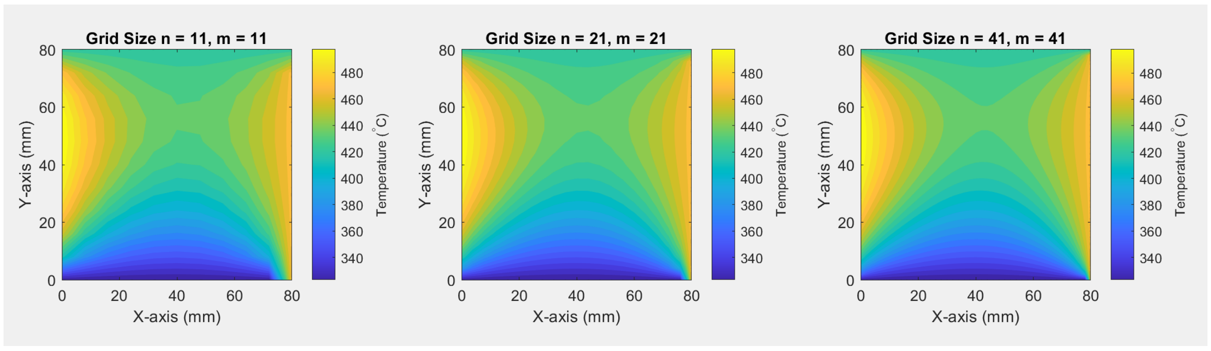


Figure 1: Temperature Distribution for Different Grid Sizes (Gaussian Elimination Method)

4.1.4 Time Complexity of Gaussian Elimination

1. **Forward Elimination:** In each step, we select a pivot row and use it to eliminate the entries below the pivot element. For the first pivot, about N^2 operations are required, for the second pivot about $(N-1)^2$, and so on. The total work is

$$\sum_{k=1}^{N-1} (N-k)^2 = O(N^3).$$

   Do, the elimination step costs $O(N^3)$.

2. **Back-Substitution:** Once the system is in upper triangular form, we compute the unknowns starting from the last equation. Also, we need to traverse all the elements of the upper triangular matrix once. This makes time complexity of it as

$$O(N^2).$$

3. **Overall Complexity:** As the elimination step dominates the cost, the total time complexity of Gaussian Elimination is

$$O(N^3).$$

4. **In terms of grid size (n):** Suppose we have square discritised grid as n x n in size then we have $N \approx n^2$, we substitute this into the time complexity formula:

$$O((n^2)^3) = O(n^6)$$

This means that if we double the number of grid points in each direction (e.g., from 20×20 to 40×40), the computation time for Gaussian elimination will increase by a factor of approximately $2^6 = 64$. This is a very rapid increase. This is why this method is not preferred in the case of a fine mesh.

4.2 Gauss-Seidel (Iterative Method)

The Gauss-Seidel method is a point-wise iterative technique. The general discretized equation is rearranged to solve for $T_{i,j}$ where $i \in [2, N_y - 1], \quad j \in [1, N_x - 1]$:

$$T(i,j) = \begin{cases} \dfrac{c_2 - 2T(i,2) - p_2 T(i-1,1) - p_2 T(i+1,1)}{c_1}, & \text{if } j = 1, \\ \dfrac{-p_2 T(i-1,j) - p_2 T(i+1,j) - T(i,j-1) - T(i,j+1)}{-4p_1}, & j \in [2, N_x - 1]. \end{cases}$$

In this implementation, the temperature values are updated in-place. Thus, when computing $T(i,j)$, the most recently updated neighbouring values are immediately used in the same iteration, This is known as the Gauss Seidel method. We generally define a tolerance at the start of the algorithm and in each iteration we check if the residual is within the tolerance or not, if not then we reiterate the procesure. Typically Gauss Seidel method converges faster than the Jacobi method, which uses the values from the previous iteration.

4.2.1 Results

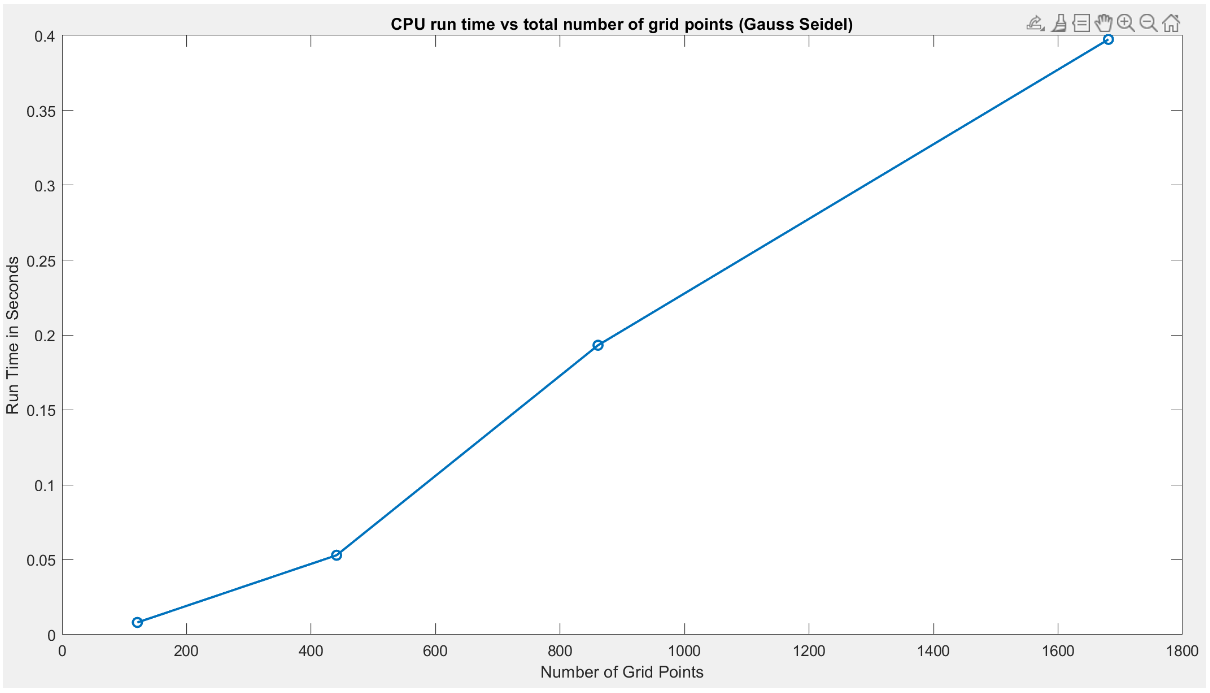


Figure 2: CPU run time vs total number of grid points (Gauss Seidel)

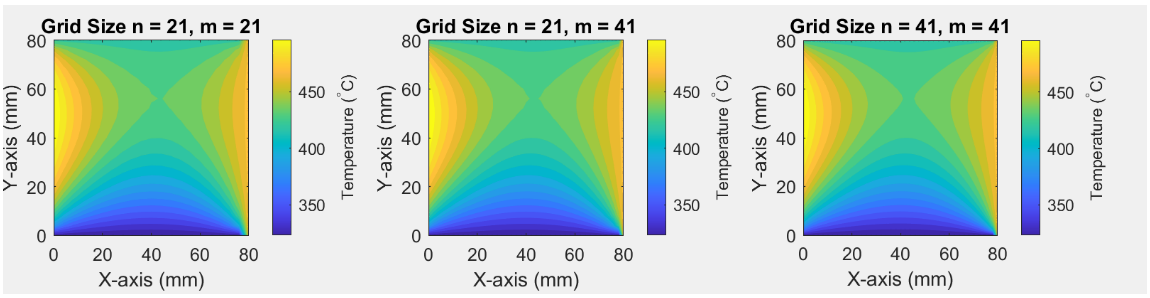


Figure 3: Temperature Distribution for Different Grid Sizes (Gauss Seidel Method)

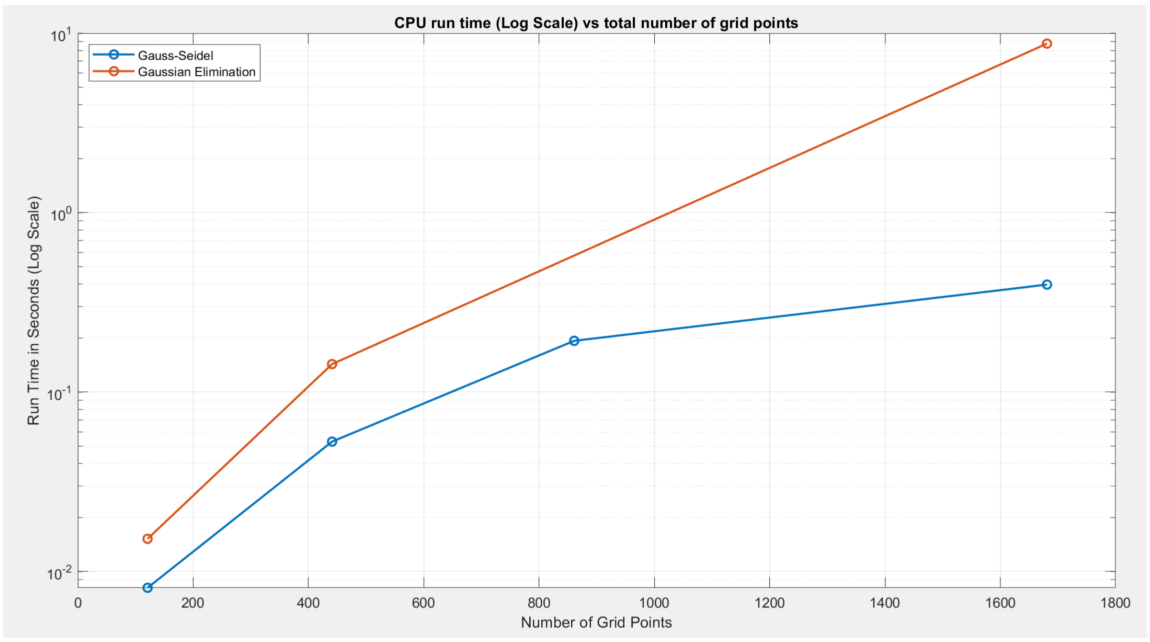


Figure 4: Temperature Distribution for Different Grid Sizes (Gauss Seidel Method)

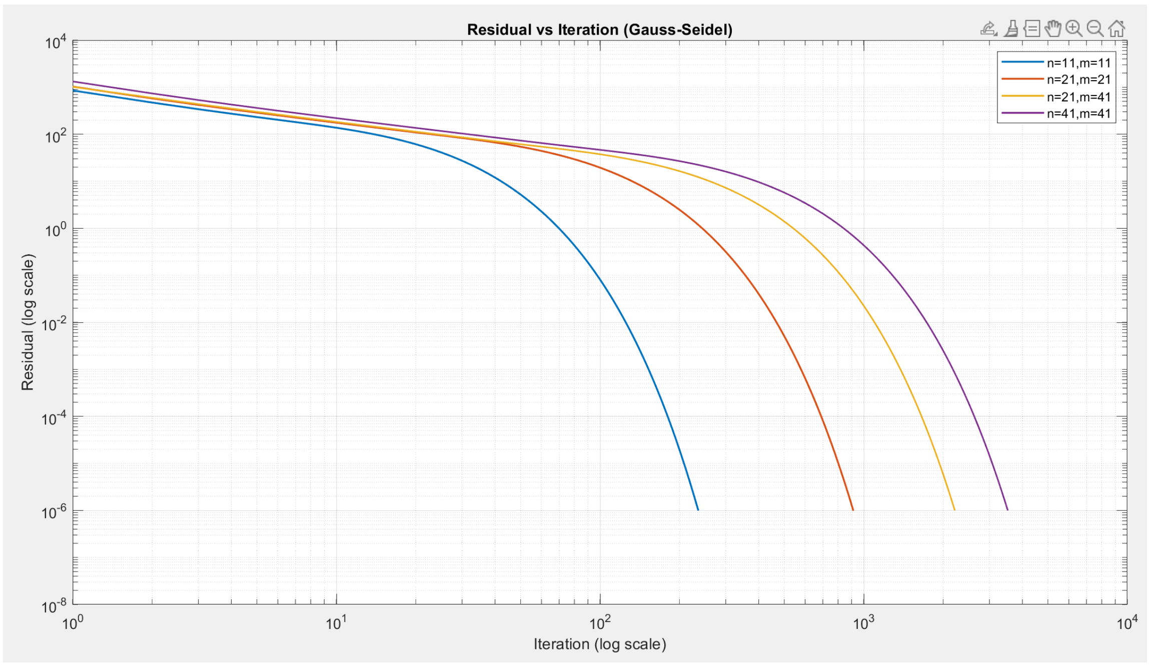


Figure 5: Temperature Distribution for Different Grid Sizes (Gauss Seidel Method)

4.2.2 Discussion

- **CPU run time with the total number of grid points:** The CPU run time is directly proportional to the number of grid points. This means more computing work is required for a larger number of grid points.

- **Dependence of the convergence rate on the total number of grid points:** From the "Residual vs Iteration" and "CPU run time vs total number of grid points" figures, we can clearly see that the rate of convergence is inversely proportional to the number of grid points. As the number of grid points increases, the solver takes a larger number of iterations to converge to a solution and hence results in an increase in run time. In summary, a larger number of grid points results in a slower rate of convergence.

- **Comparing the CPU run time of Gauss-Seidel and Gaussian Elimination:** From the graph we can see that for the 41×41 grid size, the Gaussian Elimination takes about 10 seconds to find the solution, whereas Gauss-Seidel takes about 0.4 seconds. That is, for the 41×41 grid, Gaussian Elimination takes approximately 25 times more time than Gauss-Seidel. This is why the iterative approach is preferred in comparison to Gaussian Elimination.

4.3 Line-by-Line Method

The line-by-line method is a hybrid approach that combines the Tridiagonal Matrix Algorithm (TDMA) with an iterative procedure. It solves for an entire line of nodes simultaneously. There are three types to implement it: (1) Row Sweep, (2) Column Sweep, (3) Alternating Direction. The core of the Line-by-Line method is to transform a 2D problem into a series of simpler 1D problems that are solved iteratively.

- **Traversal:** The grid is swept line-by-line, either by rows or by columns. For each individual line, the unknown temperatures on that line are solved simultaneously.
- **Knowns and Unknowns:** During the calculation for a single line, temperatures on that line are the unknowns. Temperatures on adjacent lines are treated as known values from the most recent update.
- **Tridiagonal System:** This setup creates a system of linear equations, $\mathbf{A}T = \mathbf{B}$, for each line where the coefficient matrix $\mathbf{A}$ is always tridiagonal.
- **TDMA Solver:** Each of these small, tridiagonal systems is solved very efficiently using the Thomas Algorithm (TDMA), which is faster than Gaussian Elimination solvers.
- **Iteration and Convergence:** The process of sweeping through all the lines constitutes one iteration. This entire process is repeated, updating the temperature field with each sweep, until the solution converges. Convergence is checked by L-2 norm of the matrix, which is obtained from subtracting the updated matrix from the last iteration matrix, and if it is less than the tolerance, we stop iteration, and report the temperature matrix as the solution.

4.3.1 Row Sweep:

Let us apply the "Line by Line(Row Sweep)" method on the same 5x5 grid, which is in figure 1. To generalise, let's consider $N_x \neq N_y$, there are $N_x - 1$ unknowns in each row. For i=3, we can rewrite equations 4(for first column) and 8(for other columns) as:

Row i=3:

$$Node = 5 : c_1T_5 + 2T_6 = c_2 - p_2T_1 - p_2T_9$$

$$Node = 6 : T_5 - 4p_1T_6 + T_7 = -p_2T_2 - p_2T_{10}$$

$$Node = 7 : T_6 - 4p_1T_7 + T_8 = -p_2T_3 - p_2T_{11}$$

$$Node = 8 : T_7 - 4p_1T_8 = -T_{right} - p_2T_4 - p_2T_{12}$$

Write in the form of matrices AT = B:

$$\mathbf{A} = \begin{bmatrix} c_1 & 2 & 0 & 0 \\ 1 & -4p_1 & 1 & 0 \\ 0 & 1 & -4p_1 & 1 \\ 0 & 0 & 1 & -4p_1 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} c_2 - p_2T_1 - p_2T_9 \\ -p_2T_2 - p_2T_{10} \\ -p_2T_3 - p_2T_{11} \\ -T_{right} - p_2T_4 - p_2T_{12} \end{bmatrix}$$

As we can see, the coefficient matrix is TriDiagonal and its elements do not depend on the row, so we can precompute it rather than computing it again and again in a loop. This makes the algorithm optimised because forming the matrix takes time of order $O(N_x)^2$. But the source matrix B is variable and depends on the row on which we are operating, so it has to be computed again and again inside the loop. As the coefficient matrix **A** is TriDiagonal, we can use the **TDMA** algorithm to solve it efficiently rather than using Gaussian Elimination.

Results

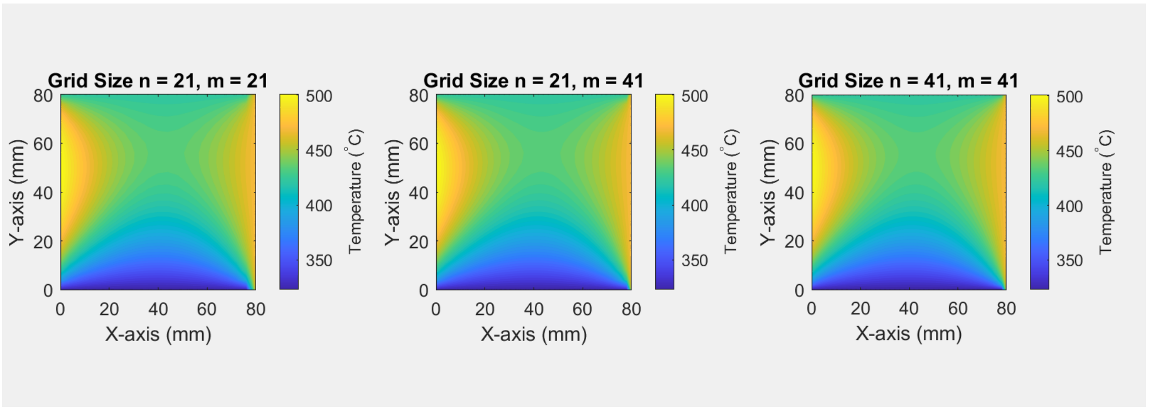


Figure 6: Temperature Distribution for Different Grid Sizes (line-by-line method ,row sweep)

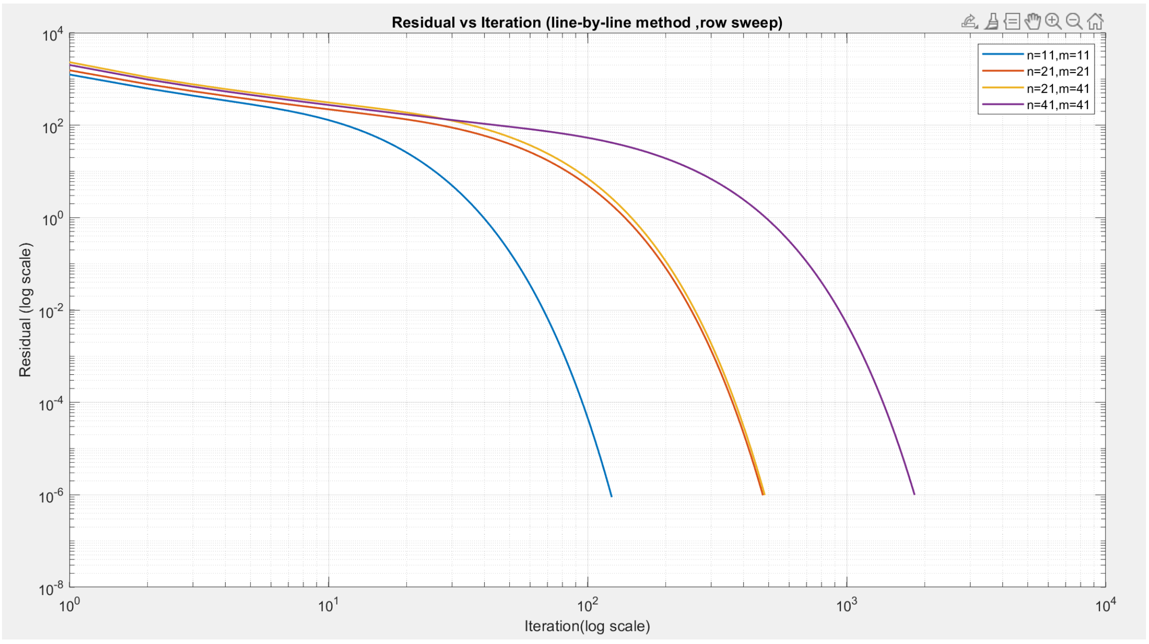


Figure 7: Residual vs Iteration for different sizes of grid(Row Sweep)

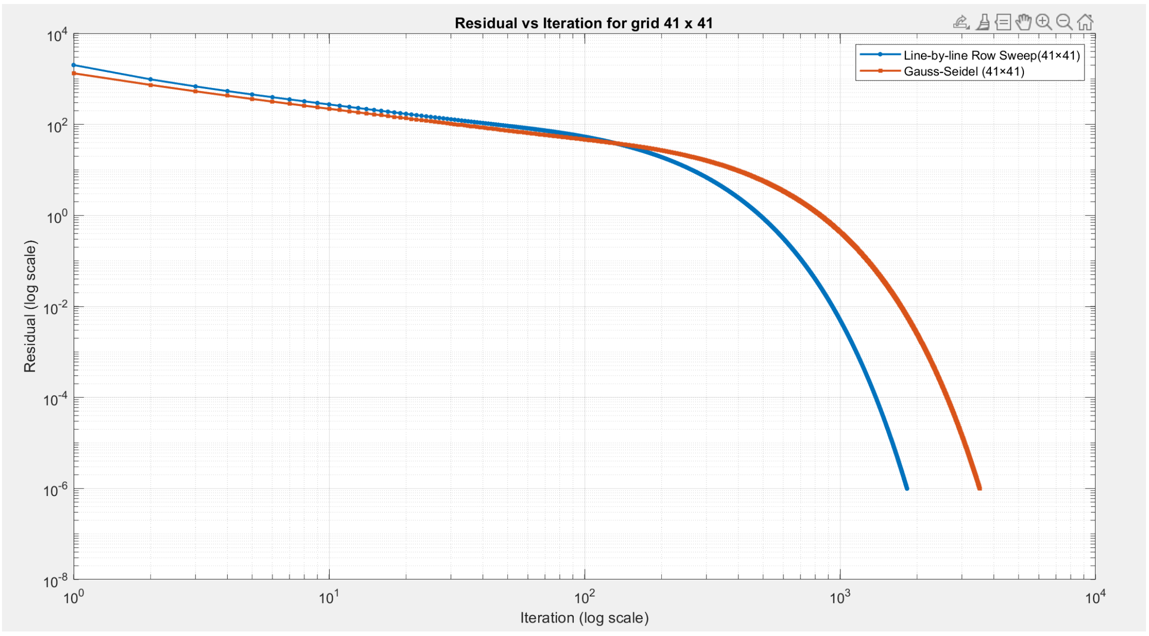


Figure 8: Residual vs Iteration Comparison, Grid Size: 41x41

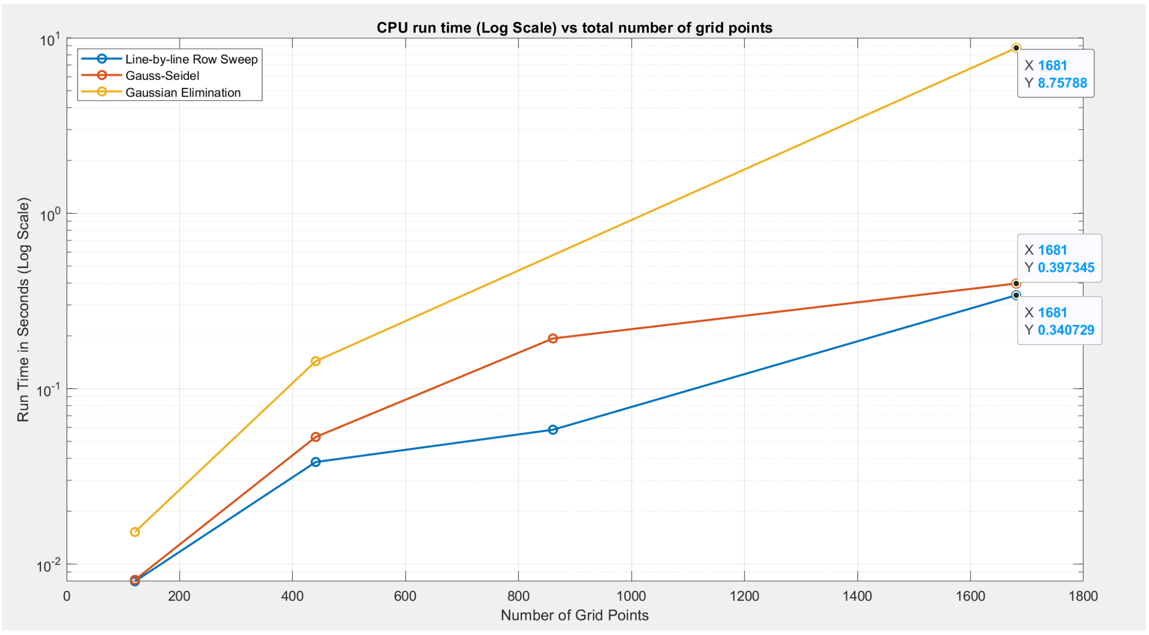


Figure 9: CPU Run Time vs Number of Grid Points Comparison

Discussion

- **CPU run times comparison:** As the number of grid points increases, the CPU run time also increases for all three methods, i.e., Line by Line (Row Sweep), Gauss-Seidel, and Gaussian Elimination. But for all numbers of grid points, the CPU run time follows this trend: "Gaussian Elimination" > "Gauss-Seidel" > "Line by Line (Row Sweep)". So the line-by-line sweep is more efficient in comparison to Gaussian Elimination and Gauss-Seidel.
- **Residual vs number of iterations:** The Line by Line method (row sweep) converges in fewer iterations compared to Gauss-Seidel. This is because we are solving the nodes in

one row simultaneously using the TDMA approach, which makes the Line by Line method converge more rapidly.

4.3.2 Column Sweep:

Let us again apply the "Line by Line(Column Sweep)" method on the same 5x5 grid, which is in figure 1. Here are two types of columns on 1st column(j=1), we have to use equation (8) and for $j > 1$ and $j < N_x$, we have to use equation 4. To generalise, let's consider $N_x \neq N_y$.

for column j=1:

$$Node = 1 : c_1T_1 + p_2T_5 = c_2 - 2T_2 - p_2T_{top}$$

$$Node = 5 : c_1T_5 + p_2T_1 + p_2T_9 = c_2 - 2T_6$$

$$Node = 9 : c_1T_9 + p_2T_5 = c_2 - 2T_{10} - p_2T_{bottomm}$$

Write in the form of matrices AT = B:

$$\mathbf{A} = \begin{bmatrix} c_1 & p_2 & 0 \\ p_2 & c_1 & p_2 \\ 0 & p_2 & c_1 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} c_2 - 2T_2 - p_2T_{top} \\ c_2 - 2T_6 \\ c_2 - 2T_{10} - p_2T_{bottomm} \end{bmatrix}$$

for column $j > 1$ and $j < N_x$: for example take j = 2:

$$Node = 2 : -4p_1T_2 + p_2T_6 = -T_1 - T_3 - p_2T_{top}$$

$$Node = 6 : -4p_1T_6 + p_2T_2 + p_2T_10 = -T_5 - T_7$$

$$Node = 10 : -4p_1T_10 + p_2T_6 = -T_{11} - T_9 - p_2T_{bottomm}$$

Write in the form of matrices AT = B:

$$\mathbf{A} = \begin{bmatrix} -4p_1 & p_2 & 0 \\ p_2 & -4p_1 & p_2 \\ 0 & p_2 & -4p_1 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} -T_1 - T_3 - p_2T_{top} \\ -T_5 - T_7 \\ -T_{11} - T_9 - p_2T_{bottomm} \end{bmatrix}$$

There are two types of coefficient matrix we have to consider because at the left boundary, the governing equation(8) is quite different from the interior temperature governing equation (4). But in both cases, we have a TriDiagonal Matrix whose elements are constant. so we can precompute them rather than computing them again and again in a loop. This makes the algorithm optimised because forming the matrix takes time of order $O(N_y)^2$. But the source matrix B is variable and depends on the column on which we are operating, so it has to be computed again and again inside the loop. As the coefficient matrix **A** is TriDiagonal, we can use the **TDMA** algorithm to solve it efficiently rather than using Gaussian Elimination.

4.4 Alternating Direction Implicit (ADI) Method

Note: In this report, one sweeping step is counted as one iteration. With this convention, if row sweeping is done in odd iterations and column sweeping is done in even iterations, the resulting scheme corresponds to the *Alternating Direction Implicit (ADI) Method.* Similarly, the order can be interchanged, i.e., starting with a column sweep in odd iterations and a row sweep in even iterations yields the same ADI scheme.

1. **Row Sweep:** The equations are solved simultaneously along each row. For a given row i, a tridiagonal system is formed for the unknown temperatures at an intermediate step. The terms involving vertical neighbours are treated as known values.
2. **Column Sweep:** The method alternates direction, solving unknowns simultaneously along each column. For a given column j, a tridiagonal system is formed for the new temperature values. The terms involving horizontal neighbours are treated as known values.

This alternating strategy takes into account the grid sizes Δx and Δy in each sweep. As the method uses both row sweep and column sweep in alternating iterations, its run time should be in between of them.

4.4.1 Results

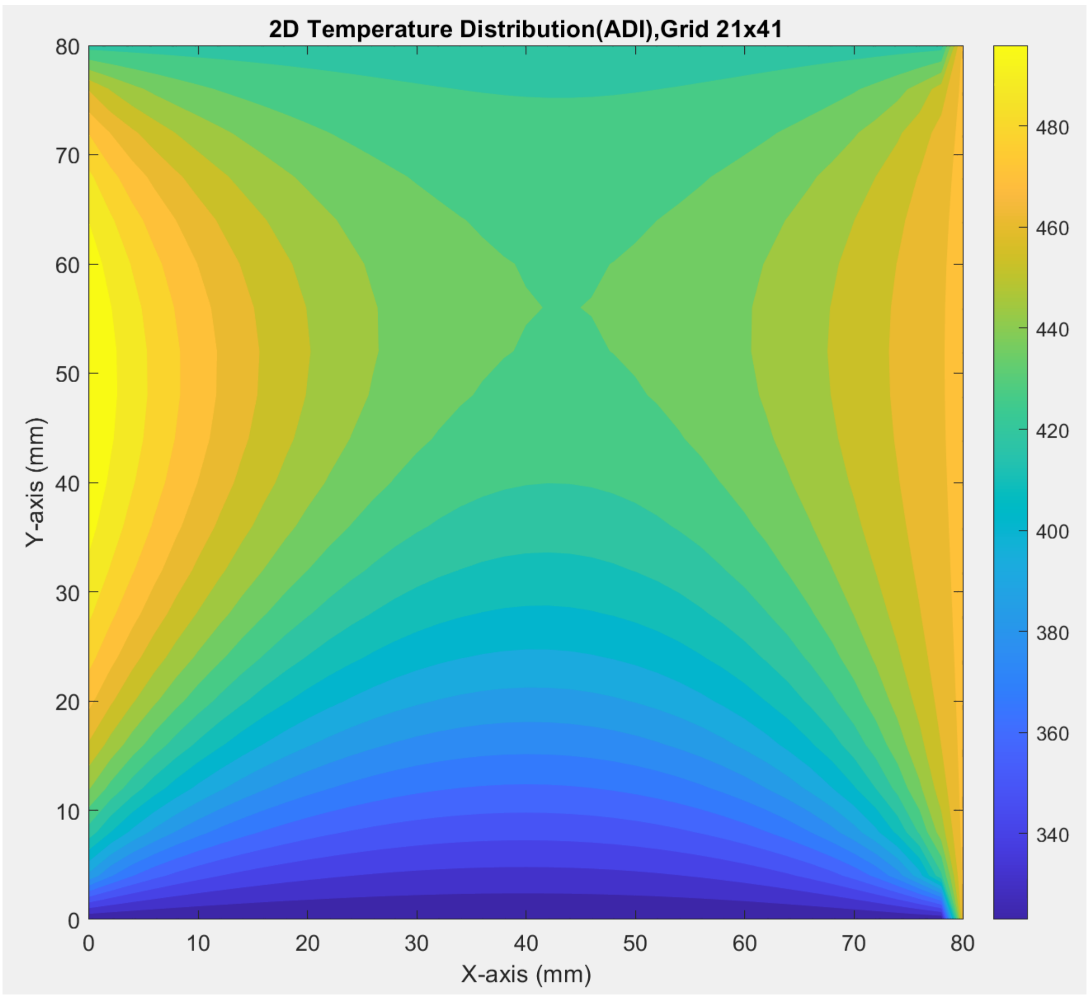


Figure 10: Contour plot of temperature distribution(ADI), Grid 21x41

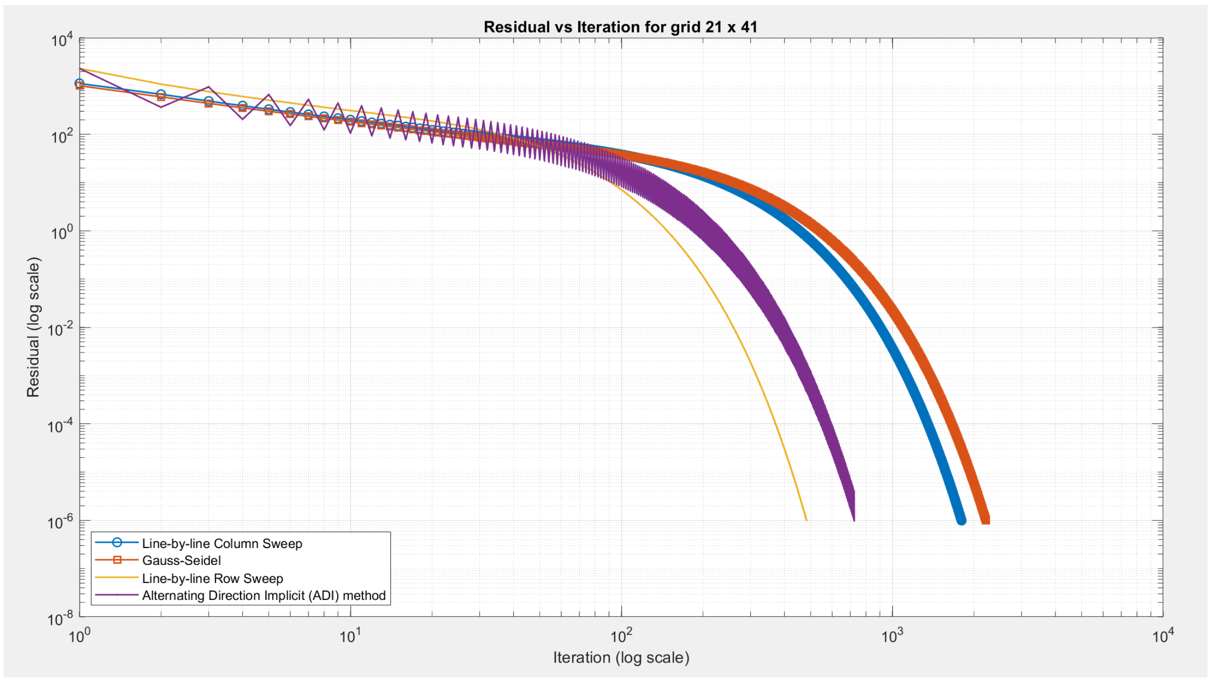


Figure 11: Residual vs Iteration Comparison, Grid Size: 21x41

ADI converses faster than the column sweep but slower than the row sweep. It is expected that, as the ADI is a mixture of row and column sweep, its time complexity should be between them.

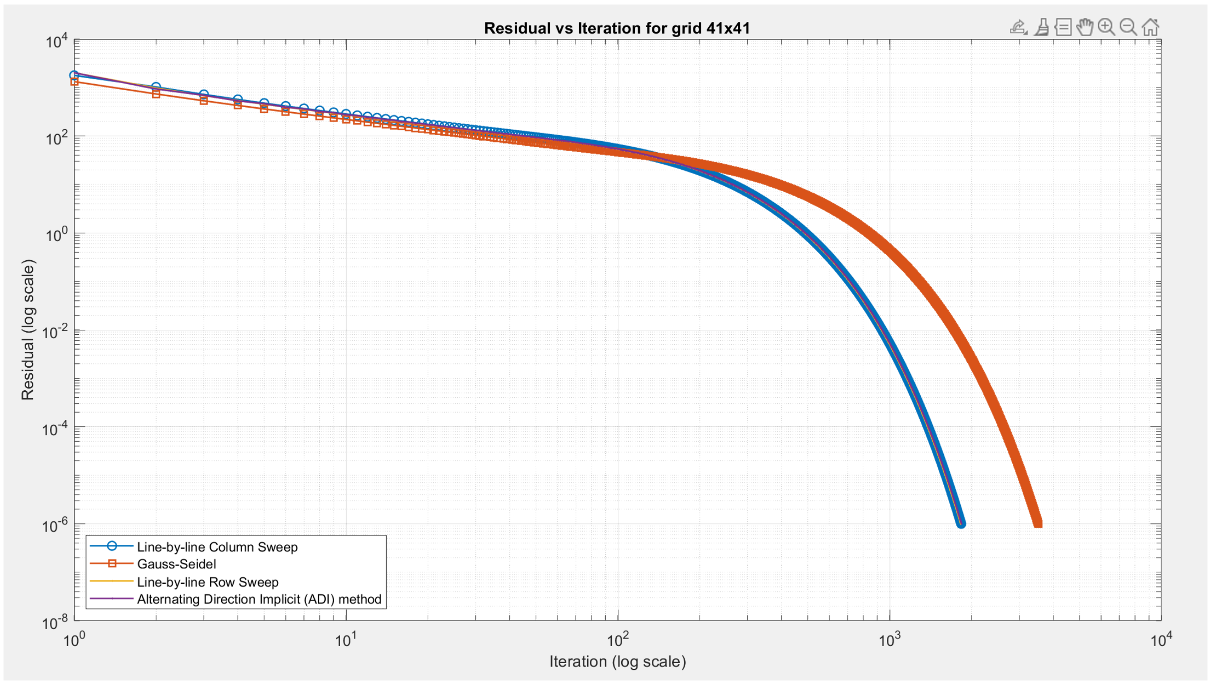


Figure 12: Residual vs Iteration Comparison, Grid Size: 41x41

We can see here that for 41x41 grid size, Row Sweep, Column Sweep and ADI all are converging at a similar rate, and hence the graph obtained has overlapping curves.

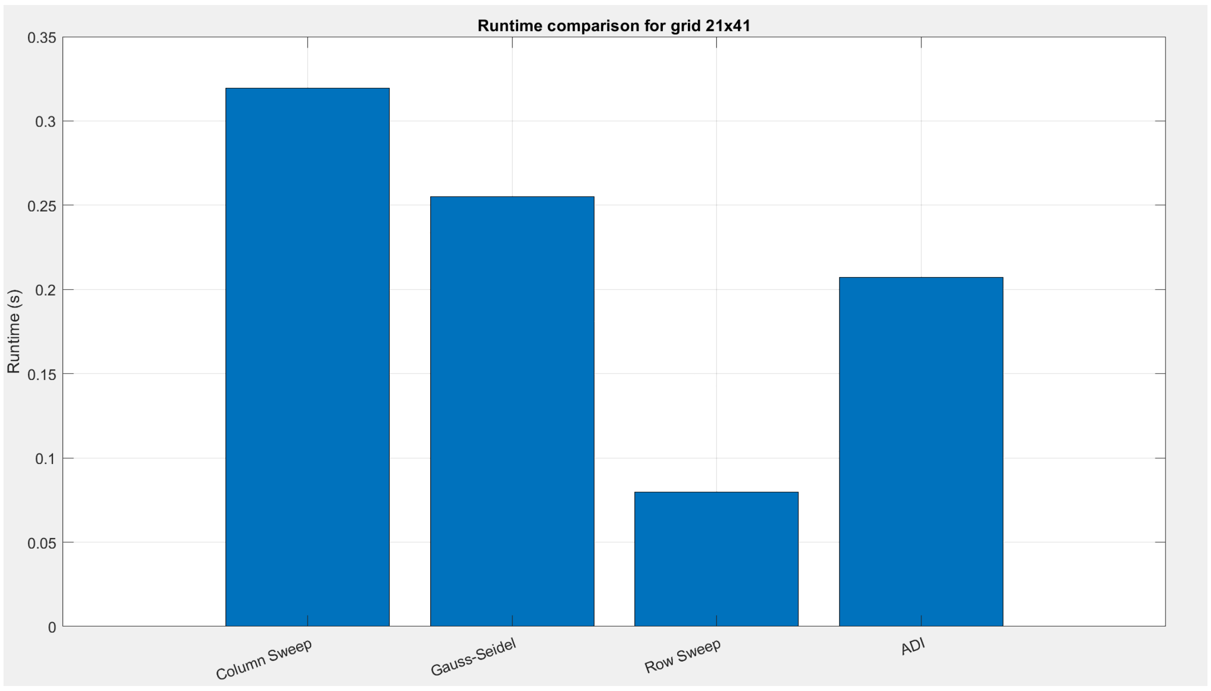


Figure 13: run time of different methods for Grid Size: 21x41

Run time of ADI is come out to be in between of column sweep and row sweep. Also Column Sweep is taking the longest time in the iterative method for 21x41, this is due to the fact that it requires two types of coefficient matrix, and to store them it requires more memory, resulting in higher run time.,

4.4.2 Discussion

The temperature contour is the same as obtained from the row sweep, column sweep and Gauss-Seidel. Also in this plot, we can see how the boundary condition has affected the interior node's state. In the residual vs iteration plot, we can see that the residual is fluctuating in the ADI method, but converges rapidly as compared to column sweep and Gauss-Seidel, in the case of 21x41 grid, it converges slowly as compared to Row Sweep, but for 41x41 grid, ADI, row sweep, and column sweep converge at the same rate. In theory ADI should be the fastest among all the methods discussed here, as the information propagates in both directions (row sweep and column sweep), but due to alternating direction, there is fluctuation in residual also, which can be seen in the figure(11). Due to this, it ends up converging slower than row sweep but faster than column sweep and Gauss-Seidel.

Table 1: CPU Runtimes (s) for Different Solvers and Grid Sizes

Solver Method	11x11 Grid (121 nodes)	21x21 Grid (441 nodes)	41x41 Grid (1681 nodes)
Gaussian Elimination	0.19	0.30	16.44
Gauss-Seidel	0.04	0.09	0.72
ADI	0.027	0.087	0.6

Note: The CPU run time may vary depending on the operating system, and even on the same operating system, it can fluctuate between each run. But, the overall order of magnitude remains the same on the same operating system.

5 Concluding Remarks

In this project, four numerical methods were compared: Gaussian Elimination, Gauss-Seidel, Line-by-Line TDMA, and the ADI method. The results clearly show that iterative methods are more efficient compared to the direct approach of the Gaussian Elimination method, also among the iterative methods, Gauss-Seidel is the slowest in general because it updates points one at a time, while Line-by-Line TDMA and ADI use TDMA and solve one line at a time, leading to much faster convergence. Overall, the Line by Line(row sweep) algorithm comes out the fastest in this problem statement.

References

[1] Mingxuan Tian, et al. *Physics-Informed Machine Learning-Based Real-Time Long-Horizon Temperature Fields Prediction in Metallic Additive Manufacturing.* Communications Engineering, vol. 4, no. 1, 29 Sept. 2025. Available at: https://doi.org/10.1038/s44172-025-00501-7. Accessed 2 Oct. 2025.