# Computer Solution of Elliptic PDEs

**Diya Mehta (22110078) and Nishit Mistry (22110172)**

Department of Mechanical Engineering, IIT Gandhinagar

Under Prof. Dilip Srinivas Sundaram for the course ME 605: Computational Fluid Dynamics

## 1. Problem Statement

The problem involves solving the 2D steady-state diffusion equation using the finite difference method (FDM) on a square domain of unit length. The governing equation is:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = S_\phi$$

With the source term given by:

$$S_\phi = 50000 \exp\left(-50\left([1-x]^2 + y^2\right)\right)\left(100\left([1-x]^2 + y^2\right) - 2\right)$$

The boundary conditions are provided for all edges of the domain as follows:

$$\phi(0, y) = 500 \exp\left(-50\left(1 + y^2\right)\right)$$
$$\phi(1, y) = 100(1-y) + 500 \exp\left(-50y^2\right)$$
$$\phi(x, 0) = 100x + 500 \exp\left(-50(1-x)^2\right)$$
$$\phi(x, 1) = 500 \exp\left(-50\left([1-x]^2 + 1\right)\right)$$

An analytical solution is:

$$\phi(x, y) = 500 \exp\left(-50\left([1-x]^2 + y^2\right)\right) + 100x(1-y)$$

The goal is to solve the equation using various numerical methods (Gaussian elimination, Gauss-Seidel iterative method, line-by-line method, and Alternating Direction Implicit (ADI) method) and analyze their computational efficiency.

## 2. Mesh Details and Approach for Discretization

To solve the 2D steady-state diffusion equation numerically, we discretize the square domain (with length 1) into a uniform grid. The size of the grid is varied, being $21 \times 21$, $41 \times 41$, $81 \times 81$ and $161 \times 161$ grid points in the $x$ and $y$ directions. A finer grid (i.e., more grid points) increases the accuracy of the solution but also increases the computational cost.

### 2.1. Grid Definition

Let the domain be divided into a uniform grid, where:

- $\Delta x$ is the spacing between grid points in the $x$-direction.
- $\Delta y$ is the spacing between grid points in the $y$-direction.

For an $N \times N$ grid, the step size is given by:

$$\Delta x = \Delta y = \frac{1}{N-1}$$

The grid points can be indexed as $(i, j)$, where $i = 1, 2, \ldots, N$ corresponds to the position in the $x$-direction, and $j = 1, 2, \ldots, N$ corresponds to the position in the $y$-direction.

### 2.2. Finite Difference Approximation

We use the finite difference method (FDM) to approximate the second-order partial derivatives in the governing equation. The central difference scheme is used for both the $x$- and $y$-derivatives, providing second-order accuracy.

The second derivative of $\phi$ with respect to $x$ at a grid point $(i, j)$ is approximated as:

$$\left.\frac{\partial^2 \phi}{\partial x^2}\right|_{(i,j)} \approx \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2}$$

Similarly, the second derivative of $\phi$ with respect to $y$ is approximated as:

$$\left.\frac{\partial^2 \phi}{\partial y^2}\right|_{(i,j)} \approx \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2}$$

### 2.3. Discretized Equation

The 2D steady-state diffusion equation:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = S_\phi$$

is discretized at the grid point $(i, j)$ using the central difference approximations as follows:

$$\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} = S_{\phi_{i,j}}$$

This gives the discretized form of the governing equation at any interior grid point.

For an $N \times N$ grid, $\Delta x = \Delta y = \Delta$, thus the discretized form of the equation becomes:

$$\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - 4\phi_{i,j} = S_{\phi_{i,j}} \Delta^2$$

### 2.4. Grid Boundary Conditions

The boundary conditions are applied directly to the boundary grid points:

- At $x = 0$, $\phi(0, y) = 500 \exp(-50(1 + y^2))$
- At $x = 1$, $\phi(1, y) = 100(1 - y) + 500 \exp(-50y^2)$
- At $y = 0$, $\phi(x, 0) = 100x + 500 \exp(-50(1 - x)^2)$
- At $y = 1$, $\phi(x, 1) = 500 \exp(-50([1 - x]^2 + 1))$

These values are applied directly at the respective boundary grid points. For example, for the left boundary ($x = 0$), the values of $\phi_{i,1}$ (where i denotes the row and j denotes the column in a matrix) are set based on the boundary condition $\phi(0, y)$, and similarly for the other boundaries.

## 3. Solution Methodology

The 2D steady-state diffusion equation is discretized using the finite difference method (FDM) and results in a system of linear algebraic equations. These equations can be solved using various numerical methods. We discuss the following methods in detail: Gaussian Elimination, Gauss-Seidel Iterative Method, Line-by-Line Method with TDMA, and Alternating Direction Implicit (ADI) Method.

### 3.1. Gaussian Elimination

Gaussian Elimination is a direct method used to solve systems of linear equations by transforming the coefficient matrix into an upper triangular form. The method involves two main steps:

1. Forward Elimination: This step involves manipulating the system of linear equations to make all elements below the diagonal of the coefficient matrix zero. This is achieved through a series of row operations, which systematically eliminate variables from each row, resulting in an upper triangular matrix.

2. Back Substitution: Once the matrix is in upper triangular form, the solution is found by solving for the variables starting from the last equation and moving upward. This is done by substituting the known values into the equations to find the unknowns.

For solving the diffusion equation, we generate a system of linear equations based on the discretization of the domain into grid sizes such as $21 \times 21$, $41 \times 41$, and $81 \times 81$.

Using the finite difference method, the continuous differential equation is approximated by a set of linear equations. Each grid point in the 2D domain corresponds to an equation in this system.

The system of equations is converted into a matrix equation of the form $A\phi = b$ where:

- A is a sparse matrix representing the coefficients of the finite difference approximation (with dimensions $N^2 \times N^2$ where N is the number of grid points in each direction).
- $\phi$ is the vector of unknowns at the grid points.
- b is the source term vector, incorporating the effects of the source term and boundary conditions.

The system $A\phi = b$ is then solved using Gaussian Elimination. This involves performing forward elimination to create an upper triangular matrix and then using back substitution to determine the values of $\phi$ at each grid point.

### 3.2. Gauss-Seidel Iterative Method

The Gauss-Seidel method is an iterative technique that solves the system of equations by sequentially updating the solution for each grid point based on its neighboring values. The method proceeds as follows:

1. Initial Guess: Start with an initial guess for the values of $\phi_{i,j}$ at all grid points. This guess can be any arbitrary set of values, here we have set the initial guess to zero to estimate the initial values.

2. Update Formula: For each grid point $(i, j)$, the value of $\phi_{i,j}$ is updated using the finite difference discretized equation:

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

The new values of $\phi$ are used as soon as they are computed, improving the convergence rate.

$$\phi_{i,j}^{(k+1)} = \frac{1}{4}\left(\phi_{i+1,j}^{(k)} + \phi_{i-1,j}^{(k+1)} + \phi_{i,j+1}^{(k)} + \phi_{i,j-1}^{(k+1)} - \Delta^2 S_{\phi_{i,j}}^{(k)}\right)$$

3. Iteration: This process is repeated, sweeping through all grid points until the solution converges. Convergence is achieved when the difference between successive iterations is smaller than a predefined tolerance, such as $|\phi_{i,j}^{(n)} - \phi_{i,j}^{(n-1)}| < \epsilon$ for all grid points $(i, j)$. Here, the convergence criteria is kept to be $\epsilon = 1e-6$ and maximum iterations = 100000.

The Gauss-Seidel method was applied to grids of sizes $41 \times 41$, $81 \times 81$, and $161 \times 161$.

### 3.3. Line-by-Line Method (Row Sweep) Using TDMA

The line-by-line method is a hybrid approach that combines aspects of both iterative and direct methods. The key idea is to reduce the 2D problem into a series of 1D problems, solving each row (or column) of the grid as a tridiagonal system. The method follows these steps:

1. Row Sweep: Sweep across the grid row by row. For each row, the system of discretized equations reduces to a tridiagonal system. This is because, in each row, the value of $\phi$ depends only on neighboring values within that row, making the matrix tridiagonal.

2. TDMA Solution: For each row, the tridiagonal system is solved using the Tridiagonal Matrix Algorithm (TDMA), also known as the Thomas algorithm. The TDMA solves the system in two steps:

- Forward sweep: The system is reduced to a simpler form.
- Back substitution: The solution is computed by back-substitution.

$$\phi_{i-1,j}^{(k+1)} - 4\phi_{i,j}^{(k+1)} + \phi_{i+1,j}^{(k+1)} = \Delta^2 S_{i,j}^{(k)} - \phi_{i,j+1}^{(k)} - \phi_{i,j-1}^{(k)}$$

3. Iteration: After solving each row or column, the updated values of $\phi$ are used to iterate through the next row/column. This process is repeated until the solution converges. Here, the convergence criteria is kept to be $\epsilon = 1e-6$ and maximum iterations = 100000.

The line-by-line method was applied to grid sizes $41 \times 41$, $81 \times 81$, and $161 \times 161$.

### 3.4. Alternating Direction Implicit (ADI) Method

The Alternating Direction Implicit (ADI) method is an implicit technique used for solving PDEs, including the diffusion equation. It breaks the multidimensional problem into multiple one-dimensional problems by alternating between directions (row-wise and column-wise) in each time step. The ADI method works as follows:

1. Row Sweep (Intermediate Solution): First, we solve the equation implicitly in the $x$-direction while treating the $y$-direction terms as constants. This results in a system of equations for each row:

$$\frac{\phi^*_{i+1,j} - 2\phi^*_{i,j} + \phi^*_{i-1,j}}{\Delta x^2} = \frac{\partial^2 \phi}{\partial y^2} + S_{\phi_{i,j}}$$

Here, $\phi^*$ represents an intermediate value of $\phi$ after the row sweep.

2. Column Sweep (Final Solution): Next, we solve the equation implicitly in the $y$-direction, using the intermediate solution $\phi^*$ from the row sweep. The equation for each column is:

$$\frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} = \frac{\partial^2 \phi^*}{\partial x^2} + S_{\phi_{i,j}}$$

3. Iteration: The process is repeated for multiple iterations until convergence. Each iteration involves solving in alternating directions, ensuring that both the $x$ and $y$ terms are updated implicitly. Here, the convergence criteria is kept to be $\epsilon = 1e{-}6$ and maximum iterations = 100000.

The ADI method was applied to the $41 \times 81$ grid.

## 4. Results and Discussions

The following analytical solution was plotted as a contour plot and compared with the numerical solution.

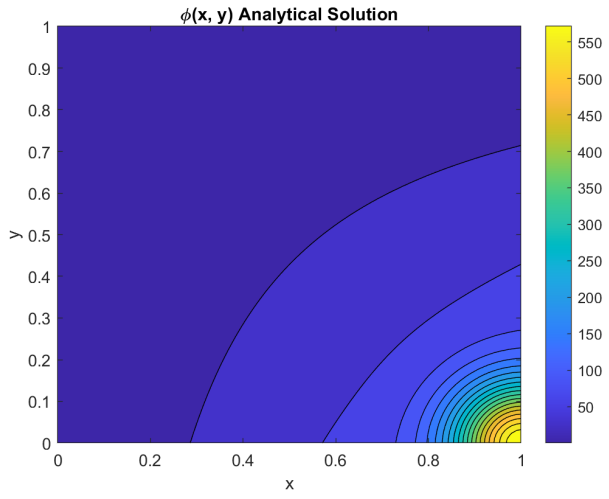


**Figure 1.** *Analytical Solution*

We have different grid sizes (21, 41, 81, 161), and the total number of grid points for these grids are:

- For $21 \times 21$, total points = $21^2 = 441$
- For $41 \times 41$, total points = $41^2 = 1681$
- For $81 \times 81$, total points = $81^2 = 6561$
- For $161 \times 161$, total points = $161^2 = 25921$

### 4.1. Gaussian Elimination

Gaussian elimination was performed for grid sizes of $21 \times 21, 41 \times 41, 81 \times 81$. The numerical solution was plotted as a contour plot for the finest grid showing a good agreement with the analytical solution.

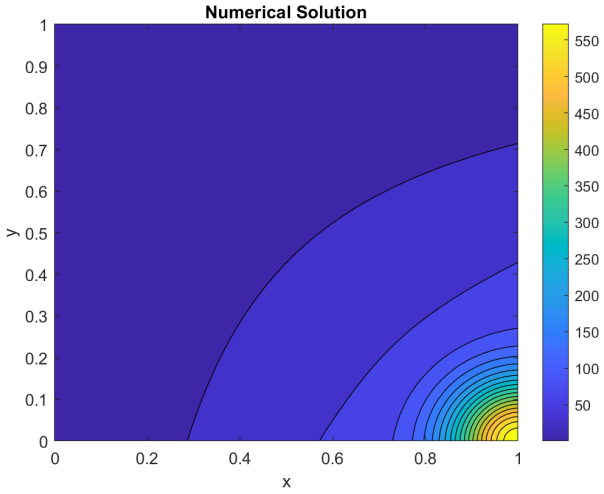


**Figure 2.** *Contour plot for grid 81x81 for Gaussian Elimination method*

The CPU run times for the different grid sizes were recorded, revealing an exponential increase as grid size increased.

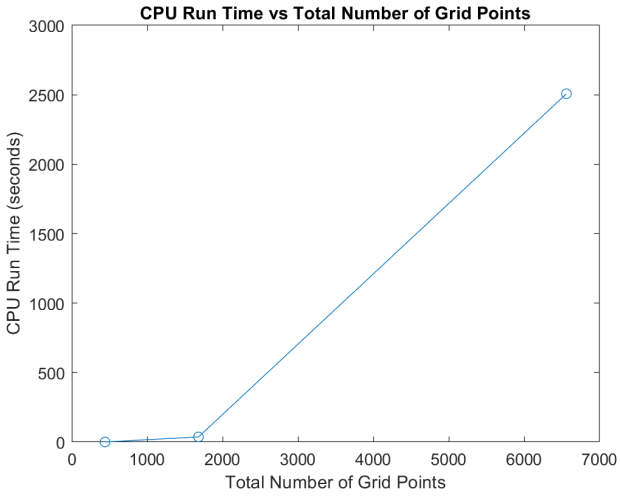


**Figure 3.** *CPU run time vs total number of grid points for Gaussian Elimination method*

For a 2D grid of size $N \times N$, the total number of unknowns is $n = N^2$. Gaussian elimination involves two main steps: forward elimination and back substitution.

- Forward elimination requires $n^3$ operations because it reduces a matrix of size $N^2 \times N^2$ i.e $n \times n$ to an upper triangular form.
- Back substitution requires $n^2$ operations as it solves the system starting from the last equation.

The dominant factor is the $n^3$ operations from forward elimination, making the overall time complexity: $O(n^3)$, which will be $O(N^6)$.

As the grid size increases, the computational cost grows disproportionately. For example, doubling the grid size in each direction (from $21 \times 21$ to $41 \times 41$) increases the total grid points by a factor of 4, but the CPU time grows by a factor closer to 8 or more. This highlights the inefficiency of Gaussian elimination for larger grids.

Although Gaussian elimination provides an exact solution, it is computationally prohibitive for large grid sizes. Its time complexity $O(N^6)$ makes it impractical for applications requiring fine grids or real-time computation.

This method is best suited for small grids where computational cost is not a significant concern.

### 4.2. Gauss-Seidel Iterative Method

The Gauss-Seidel method was applied to grid sizes $41\times41, 81\times81, 161\times161$. Residuals were tracked over the number of iterations for each grid size, and CPU run times were recorded.

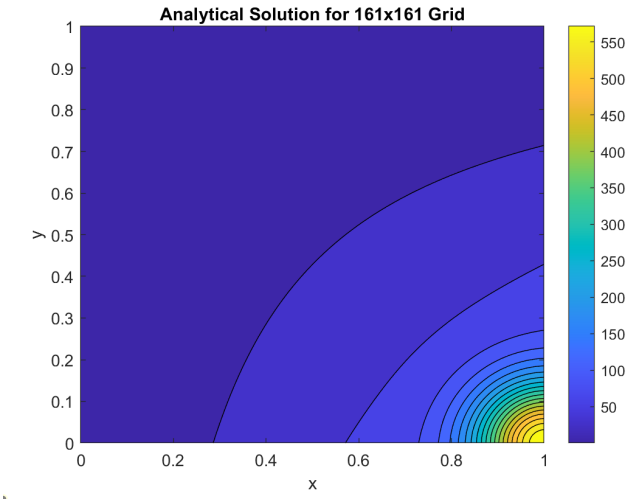


**Figure 4.** *Contour plot for grid 161x161 for Gauss-Seidel Iterative Method*

The residual decreases with the number of iterations. The convergence slows as the grid size increases, with larger grids requiring significantly more iterations to reach the same residual threshold.

As the number of grid points increases, the distance between grid points decreases, which means smaller updates to the solution at each iteration. This results in a slower reduction in the residuals for larger grids.

The CPU run time increases with grid size, but at a slower rate compared to Gaussian elimination.

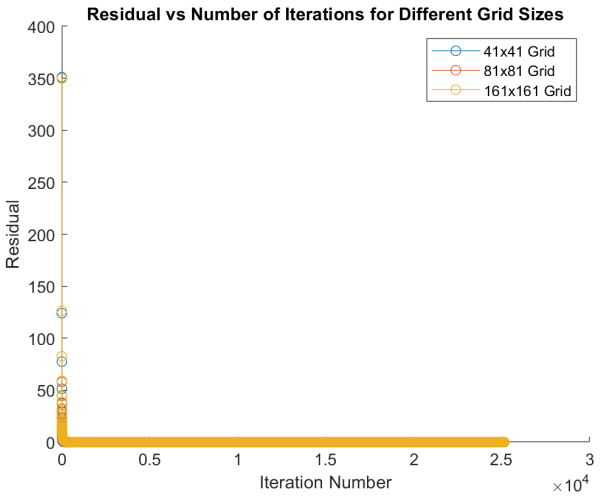


**Figure 5.** *Residual vs number of iterations for the three grids for Gauss-Seidel Iterative Method*

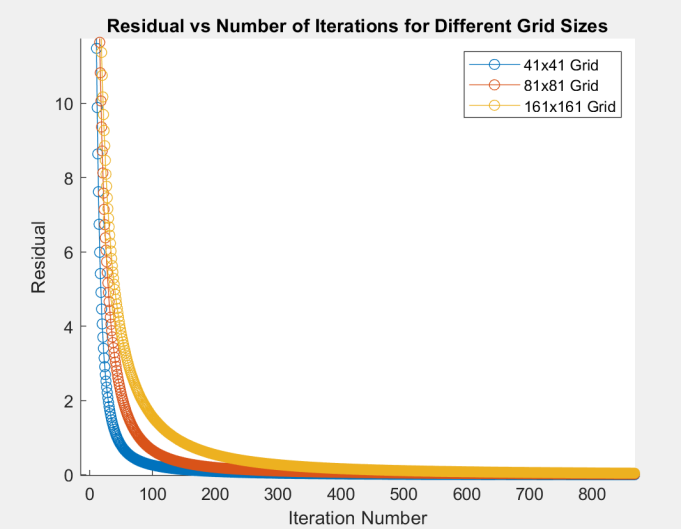


**Figure 6.** *Residual vs number of iterations for the three grids for Gauss-Seidel Iterative Method*

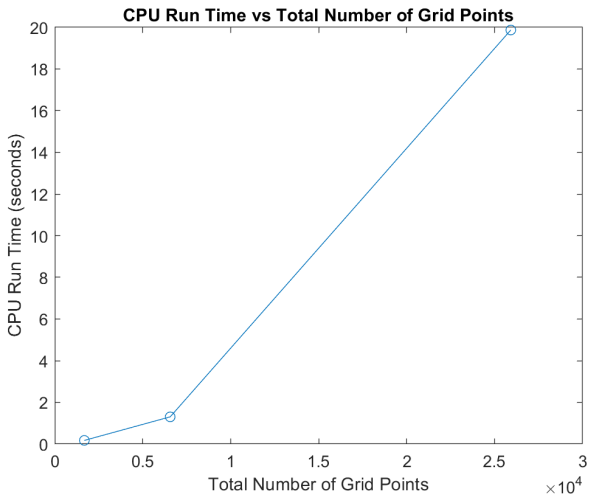


**Figure 7.** *CPU run time vs total number of grid points for Gauss-Seidel Iterative Method*

For a grid of size $N \times N$, the total number of unknowns is $N^2$. Gauss-Seidel iterates over all grid points, updating the value at each point based on its neighbors. In each iteration, updating all $N^2$ unknowns requires $O(N^2)$ operation, and with k iterations for convergence, the overall time complexity becomes $O(kN^2)$, which is a significant improvement over Gaussian elimination.

For smaller grids, Gauss-Seidel is competitive with Gaussian elimination, but as the grid size increases, Gauss-Seidel becomes far more efficient. Gauss-Seidel's lower time complexity makes it a better choice for larger grids, even though it requires more iterations.

### 4.3. Line-by-Line Method (Row Sweep) Using TDMA

The line-by-line method was implemented for grid sizes $41 \times 41, 81 \times 81, 161 \times 161$. The residuals were plotted against the number of iterations and compared to the Gauss-Seidel method. The CPU run time was recorded for each grid size, showing a much more efficient scaling compared to both Gauss-Seidel and Gaussian elimination.

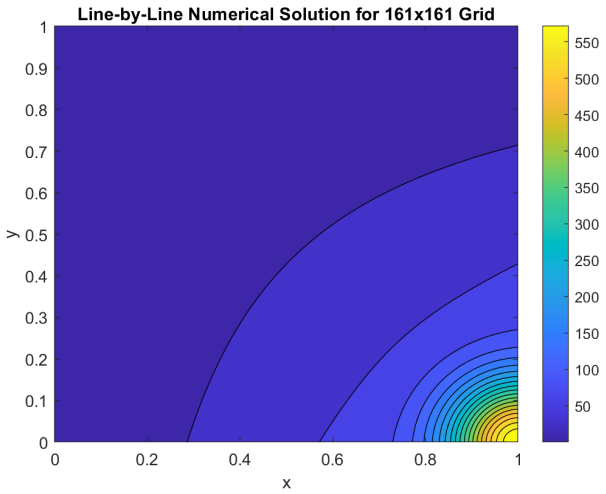


**Figure 8.** *Contour plot for grid 161x161 for Line-by-Line Method*

The rate of reduction of the residual is more rapid in this case of the Gauss-Seidel method, however the TDMA solver also effectively handles the tridiagonal system of equations in each row, thus we can conclude both to be comparable here in terms of convergence rate.

For a grid of size $N \times N$, the total number of unknowns is $N^2$. The line-by-line method solves the system of equations one row (or column) at a time using the Tridiagonal Matrix Algorithm (TDMA). Each row or column sweep involves solving a tridiagonal system of N equations in $O(N)$ time using TDMA. The number of sweeps (iterations) required for convergence is proportional to N, as each sweep processes one dimension of the system at a time.

Each iteration requires $O(N^2)$ operations (since there are N rows or columns, each taking $O(N)$ time). With k iterations for convergence, the overall time complexity becomes: $O(kN^2)$.

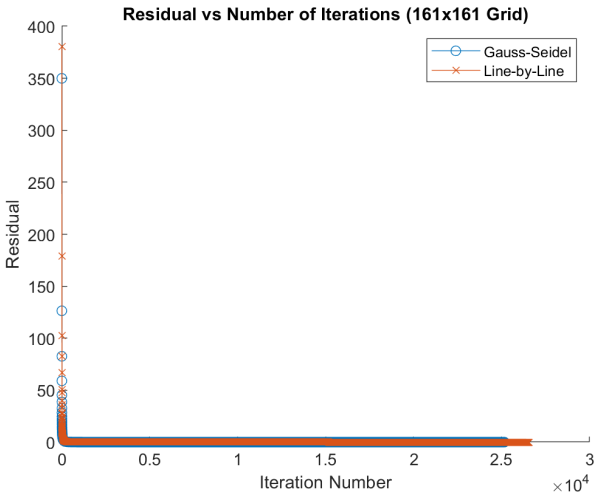


**Figure 9.** *Residual vs number of iterations for the line-by-line method and for the Gauss-Seidel method for the 161 x 161 grid*

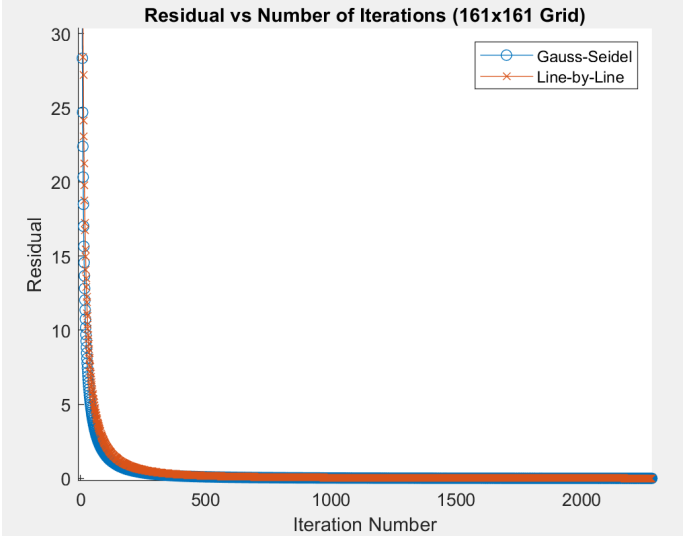


**Figure 10.** *Residual vs number of iterations for the line-by-line method and for the Gauss-Seidel method for the 161 x 161 grid*

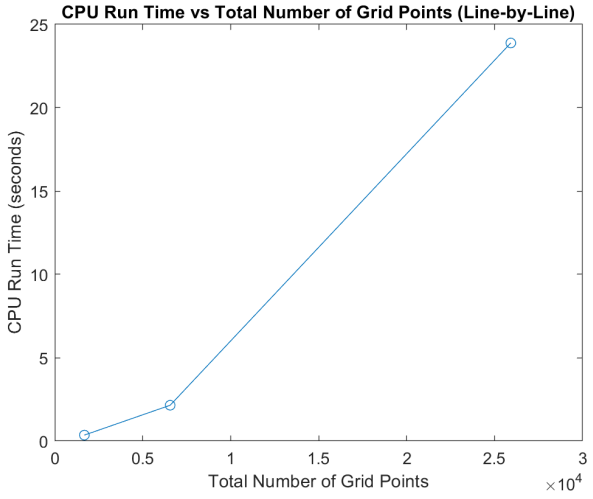


**Figure 11.** *CPU run time vs total number of grid points for Line-by-Line Method*

The line-by-line method and Gauss-Seidel both having the same time complexity are thus comparable, and both vastly outperforms Gaussian elimination in terms of computational cost for larger grids.

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

The ADI method was applied to a grid size of $41 \times 81$. The residuals were plotted for row-wise sweeps, column-wise sweeps, and the ADI method itself, with the column-sweep method showing the fastest convergence. In case, the grid size were taken as $81 \times 41$ then row-sweep would be showing the fastest convergence. In both the cases, ADI would show intermediate convergence.

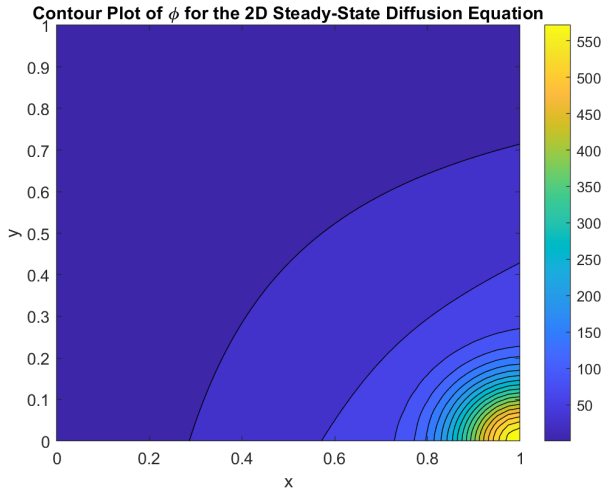


**Figure 12.** *Contour plot for grid 41x81 for ADI Method*

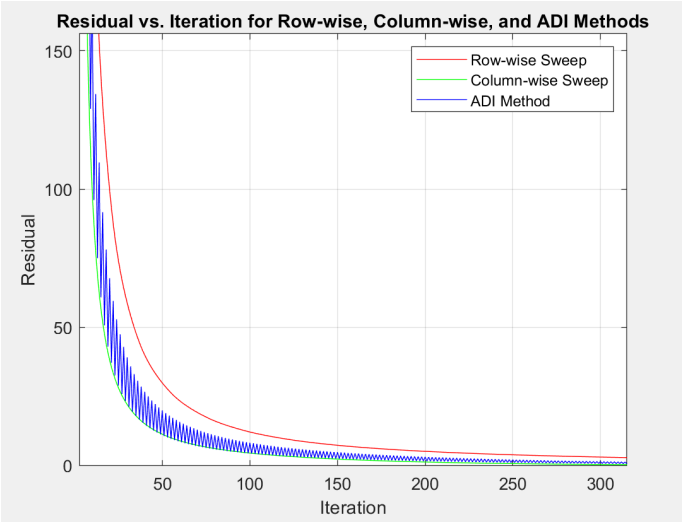


**Figure 13.** *Residual vs number of iterations for row-wise sweep, column-wise sweep, and ADI method*

For a grid of size $N \times N$, the total number of unknowns is $N^2$. The ADI method alternates between solving the system row-wise and column-wise. Each row-wise and column-wise sweep is solved using TDMA, which takes $O(N)$ operations per row or column. Therefore, each full iteration (including both row and column sweeps) requires $O(N^2)$ operations.

Each iteration takes $O(N^2)$ operations, and with k iterations for convergence, the overall time complexity becomes: $O(kN^2)$.

Each iteration takes $O(N^2)$ operations, and with k iterations for convergence, the overall time complexity becomes: $O(kN^2)$.

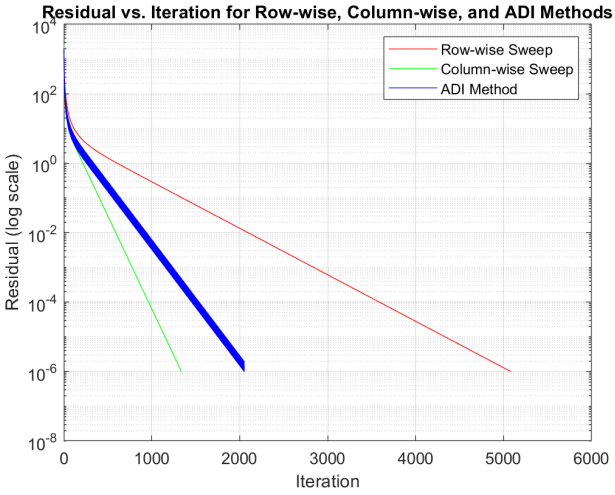


**Figure 14.** *Residual vs number of iterations for row-wise sweep, column-wise sweep, and ADI method (log-scale)*

In general, the ADI method would perform better since than the worst case which could be either row or column sweep depending on the grid size. By alternating between row-wise and column-wise sweeps, the ADI method effectively reduces the residual in both directions, leading to a comparatively faster convergence rate.

The ADI method accelerates convergence by solving in both directions during each iteration, allowing for a faster reduction in error compared to solving in just one direction at a time.

## 5. Concluding Remarks

In this analysis of the 2D steady-state diffusion equation, four numerical methods—Gaussian Elimination, Gauss-Seidel Iterative Method, Line-by-Line Method using TDMA, and the Alternating Direction Implicit (ADI) Method—were employed and compared in terms of computational efficiency, convergence rates, and CPU run times across various grid sizes.

- For small problems, where computational resources are not a concern, Gaussian elimination is a suitable method due to its straightforward implementation and exact solution. However, its inefficiency becomes apparent as grid sizes increase.
- The Gauss-Seidel iterative method is appropriate for medium-sized grids. It is easier to implement than more complex methods like line-by-line TDMA, but for large grids, it may require too many iterations to reach convergence, leading to longer CPU run times.

- The line-by-line TDMA method is also appropriate for medium-sized grids, as it has comparitively lesser computational cost than Gaussian elimination. It has comparable time complexity to Gauss-Seidel method.
- For larger grids, the ADI method are superior in terms of efficiency. The ADI method's ability to alternate between row and column directions results in the fastest convergence and lowest computational cost.
- The ADI method stands out as the best overall method for solving large-scale 2D steady-state diffusion equations. Its alternating sweep technique ensures that the residual decreases rapidly with fewer iterations, making it the most computationally efficient choice.

In conclusion, for large-scale applications, the ADI method offers the best combination of computational efficiency and accuracy. The line-by-line method is also an excellent option, particularly when implementing TDMA is feasible. For smaller problems or educational purposes, Gaussian elimination and Gauss-Seidel remain viable, though their use should be limited to problems where computational cost is not a primary concern.

## 6. Acknowledgements

We would like to express our sincere gratitude to Professor Dilip Srinivas Sundaram for their invaluable guidance and support throughout this project. We are also deeply thankful to the teaching assistants for their assistance and feedback.