CFD Project-1

Abhijit Venkat (22110008) and Aditya Prasad (22110018)

Department of Mechanical Engineering, IIT Gandhinagar

Instructor: Prof. Dilip S. Sundaram

Contents

1. Problem Statement

This project focuses on solving the 2D steady-state diffusion equation, an elliptic partial differential equation, over a square domain of unit length using various numerical methods. The equation is given as:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = S_\phi \tag{1}$$

where S_ϕ is the source term and is 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 for the problem are as follows:

1. On the left boundary ($x = 0$):
$$\phi(0, y) = 500 \exp\left(-50\left[1 + y^2\right]\right)$$
2. On the right boundary ($x = 1$):
$$\phi(1, y) = 100(1-y) + 500 \exp\left(-50y^2\right)$$
3. On the bottom boundary ($y = 0$):
$$\phi(x, 0) = 100x + 500 \exp\left(-50\left[1-x\right]^2\right)$$
4. On the top boundary ($y = 1$):
$$\phi(x, 1) = 500 \exp\left(-50\left[(1-x)^2 + 1\right]\right)$$

The source term includes an exponential function of spatial coordinates. The primary objective is to solve this equation using different grid resolutions and methods such as Gauss elimination, Gauss-Seidel iterative method, line-by-line method with TDMA, and the Alternating Direction Implicit (ADI) method. Each method will be evaluated based on its computational efficiency, convergence behavior, and accuracy. Specifically, the study aims to analyze the computational cost and scalability of each approach by comparing CPU runtime and iteration residuals for various grid sizes. The results will include contour plots of the solution field for the finest grid resolution and performance comparisons across methods.

2. Analytical Solution

For the equation

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

The Analytical solution is given by:

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

The contour plot for the Analytical solution (For N = 161) looks something like:

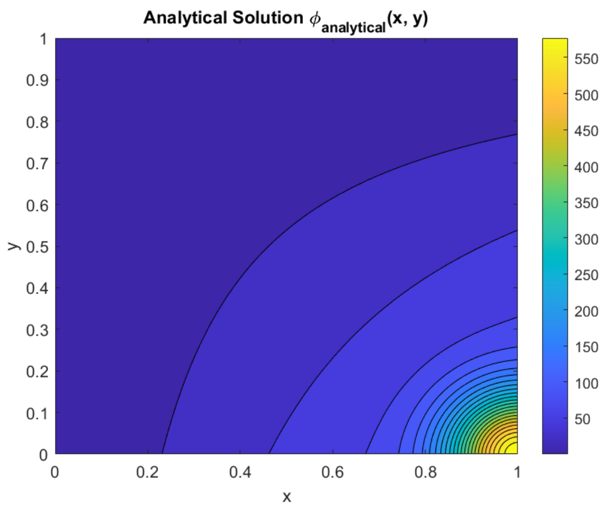

Figure 1. Contour plot of the Analytical Solution

3. Grid Discretization methods employed

Using the finite difference method, the second derivatives are approximated using central differences. Let $\phi_{i,j}$ be the value of ϕ at the grid point (i, j), where i and j are the indices for the x and y directions, respectively.

3.1. Discretization in the x-direction:

$$\frac{\partial^2 \phi}{\partial x^2} \approx \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2}$$

3.2. Discretization in the y-direction:

$$\frac{\partial^2 \phi}{\partial y^2} \approx \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2}$$

3.3. Combined Discretized Form:

By combining the above approximations, the discretized form of the 2D diffusion equation becomes:

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

Simplifying:

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

Or, equivalently:

$$A\phi_{i-1,j} + B\phi_{i,j-1} + C\phi_{i,j} + D\phi_{i+1,j} + E\phi_{i,j+1} = S_{\phi,ij}$$

where:

$$A = \frac{1}{\Delta x^2}, \quad B = \frac{1}{\Delta y^2}, \quad C = -(\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}), \quad D = \frac{1}{\Delta x^2}, \quad E = -\frac{1}{\Delta y^2}$$

The above Equation is the general discretized form of the equation given to us. We shall be modifying these equations according to our needs in the different parts of the problem.

- **Central Differences:** The equation uses central difference approximations to represent the second derivatives, which is common in finite difference methods for PDEs. This ensures a second-order accurate approximation of the spatial derivatives.
- **Coefficient Matrix:** The resulting equation at each grid point involves values from the neighboring points ($i-1$, $i+1$, $j-1$, $j+1$), which makes the coefficient matrix sparse with a banded structure. In this case, it becomes a 5-point stencil pattern, where each point $\phi_{i,j}$ is connected to its four immediate neighbors.
- **Source Term:** $S_{\phi,ij}$ is the source term evaluated at the grid point (i, j).
- **Mesh Details:** We have different cases of meshing for each type of algorithm that we implement. For the first three cases, we have a square mesh with the number of points (N) ranging from 21 to 161 in each of the two directions. Since the dimensions of the domain are of unit length, the grid spacing is given by $\Delta x = \Delta y = \frac{1}{N-1}$

4. Solution Obtained by Gauss Elimination

The first algorithm we implement is the Gauss Elimination method to solve this problem. In this method, we first form a coefficient matrix containing all the coefficients of the discretised equation (discussed in the above section) and also the RHS vector. Since we have N points in each direction, we have a total of N^2 points in the grid. The values of $\phi_{x,y}$ are known at all the boundaries. Therefore, in order to form the coefficient matrix, we used 2 loops: one for the row iteration and the other for the column iteration and filled the respective coefficients at their locations.

Special cases were considered for equations with boundary terms present.

We then turn the coefficient matrix into row-echelon form(upper triangular matrix). Once the system is in upper triangular form, we solve for the variables starting from the last equation and substituting the known values into the previous equations. This method is known as back substitution.

4.1. Results

We implemented this method for 3 different grid spacings, $N = 21$, $N = 41$, and $N = 81$. The following is the ϕ field Contour plot for the case of the results obtained from the Gauss Elimination method for $N = 81$.

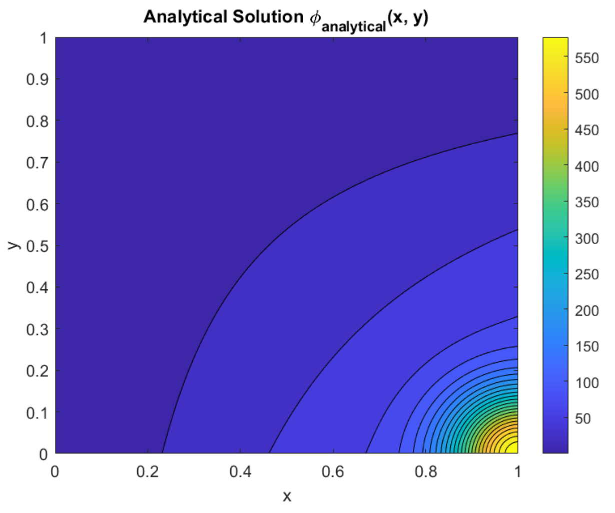

Figure 2. Contour plot of $\phi_{x,y}$ for N = 81 using Gaussian Elimination

We can clearly observe that the contour plot for $N = 81$ resembles the Analytical solution. We cannot tell the difference by just looking at the contour plot, but there will be some very minor differences between the two solutions. Now, here is the plot for CPU run time v/s N.

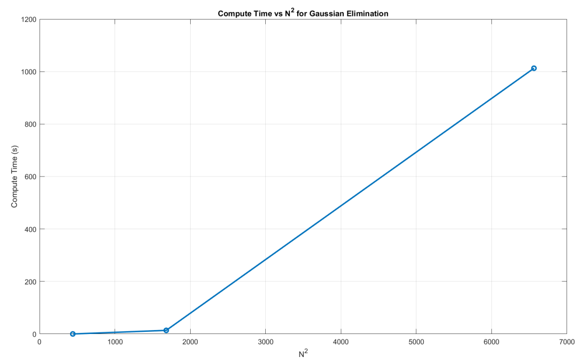

Figure 3. CPU run time (s) v/s N^2

The CPU runtime for $N = 21$ is 0.153209 seconds, for $N = 41$ is 13.799026 seconds and for $N = 81$ is 1012.5203 seconds. The time complexity of this algorithm is of the order of n^3, where n is the total number of grid points/ variables/ linear equations. This also means that since there are N^2 grid points in total in the entire domain, the complexity is of the order N^6. Therefore, when we increase N from 21 to 41, the number of grid points is increased from 441 to 1561, which is almost 4 times the previous case. Therefore, the process must theoretically take 4^3 times more time than the previous case. This gives us an expected time of 9.8 seconds. The time we got for $N = 41$ is 13.799026 seconds, which is comparable.

Moving on to $N = 81$, we expect the CPU run time to be 4^3 times more than that of $N = 41$. The expected CPU run time would, therefore, be $4^3 \times 13.799026$ seconds or 883.14 seconds. However, the CPU run time we got is 1012.5203 seconds. These two values are also quite comparable. The error might be due to the intrinsic workings of MATLAB and how it performs its operations. The time taken to run other minor parts of the code must have accumulated over the many iterations in this case.

5. Gauss-Seidel Method

The *Gauss-Seidel Method* is an iterative technique used to solve a system of linear equations of the form $A\phi = b$, where A is the coefficient matrix, ϕ is the solution vector, and b is the right-hand side vector.

At each iteration, we update the value of each unknown $\phi_{i,j}$ using the known values from both the previous and the current iteration, based on the discretisation of the governing PDE. The first step of this method is to guess the values of ϕ at all the nodes.

Let us assume that we are at the $(n+1)^{\text{th}}$ iteration. The values of ϕ at the interior grid points (i,j) are computed using the values of neighbouring points from the previous iteration (n) and the updated values from the current iteration $(n+1)$. The discretised form of the governing PDE at a general interior point (i,j) is given by:

$$\frac{\partial^2\phi}{\partial x^2} \approx \frac{\phi^n_{i-1,j} - 2\phi^{n+1}_{i,j} + \phi^{n+1}_{i,j}}{\Delta x^2}$$

$$\frac{\partial^2\phi}{\partial y^2} \approx \frac{\phi^n_{i,j-1} - 2\phi^{n+1}_{i,j} + \phi^n_{i,j+1}}{\Delta y^2}$$

Substituting these expressions into the governing PDE, we get:

$$\frac{\phi^n_{i-1,j} - 2\phi^{n+1}_{i,j} + \phi^n_{i+1,j}}{\Delta x^2} + \frac{\phi^n_{i,j-1} - 2\phi^{n+1}_{i,j} + \phi^n_{i,j+1}}{\Delta y^2} = S^n_{\phi,ij}$$

Rearranging the above equation to isolate the unknown $\phi^{n+1}_{i,j}$ on the left-hand side (LHS) and keeping the known terms on the right-hand side (RHS), we get:

$$\phi^{n+1}_{i,j} = \frac{1}{2\left(\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}\right)}\left(\frac{1}{\Delta x^2}\left(\phi^n_{i-1,j}+\phi^n_{i+1,j}\right)+\frac{1}{\Delta y^2}\left(\phi^n_{i,j-1}+\phi^n_{i,j+1}\right)-S^n_{\phi,ij}\right)$$

5.1. Residual Calculation and Convergence Check

At the end of each iteration, we compute the *residual* to measure how close the current solution is to the true solution. The residual is defined as:

$$\text{Residual} = ||A\phi - b||$$

Here, A is the coefficient matrix obtained from the discretisation of the PDE, ϕ is the current solution vector, and b is the right-hand side vector. The residual indicates the difference between the left-hand side and right-hand side of the equation $A\phi = b$.

The *L2 norm* of the residual is typically used as the measure of error. We stop the iteration process when the residual falls below a pre-specified tolerance ϵ. The stopping criterion can be expressed as:

$$\text{Residual} < \epsilon$$

5.2. Results

We got the following ϕ field Contour plot for $N = 161$ using the Gauss-Seidel or point-by-point method.

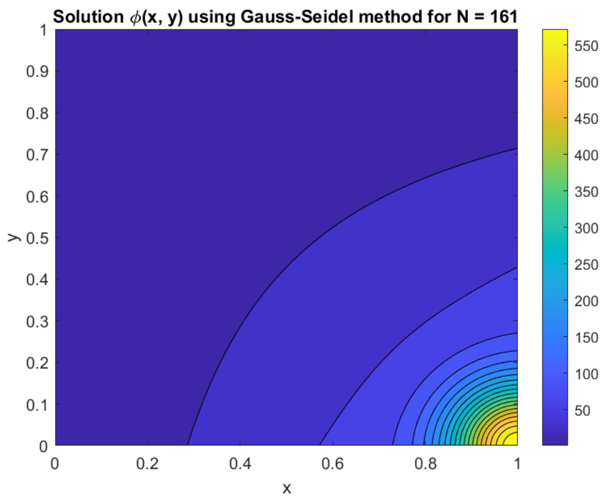

Figure 4. ϕ field for N = 161 using Gauss-Seidel method

Clearly, the solution contour plot matches that of the Analytical solution, similar to the Gauss Elimination case. Now, we move on to the CPU runtime plot. Here, we plot the time taken by the computer to run the simulation for each grid configuration v/s the number of grid points. The results are quite interesting.

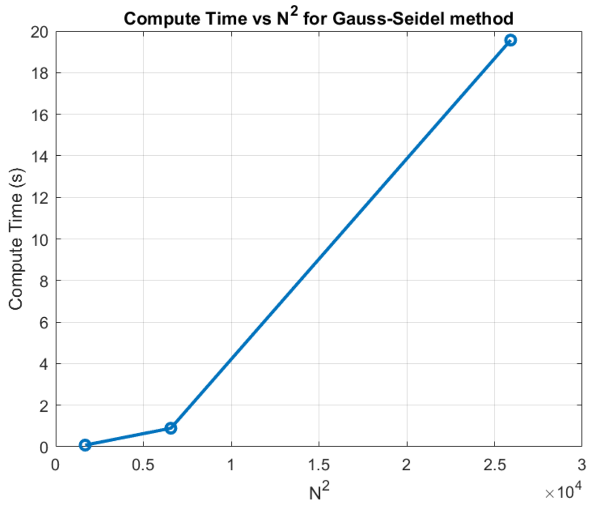

Figure 5. CPU run time (s) v/s N^2 for Gauss-Seidel method

The CPU runtime for $N = 41$ is 0.0802 seconds, for $N = 81$ is 0.8926 seconds, and for $N = 161$ is 19.566 seconds. The time complexity of this algorithm is of the order of N^3 where N is the number of grid points in each direction. This is because it takes a minimum of O(n) iterations for the information to flow from one end of the grid to the furthest point, and every iteration takes O(n^2) operations in the point-by-point approach. Since the time complexity is of the order of N^3, therefore when we increase N from 41 to 81. Therefore, the process must theoretically take 2^3 times more operations than the previous case. This gives us an expected time of 0.6416 seconds. The time we got for $N = 81$ is 0.8926 seconds, which is comparable.

Moving on to $N = 161$, we expect the CPU run time to be 2^3 times more than that of $N = 81$. Therefore, the expected CPU run time would be $2^3 \times 0.8926$ seconds or 7.1408 seconds. However, the CPU run time we got is 19.566 seconds. These two values are also quite comparable. The difference in the CPU run time and the expected run time can be attributed to Matlab's intrinsic workings. Also, some functions might be called only once, so they have a linear addition in the CPU run time.

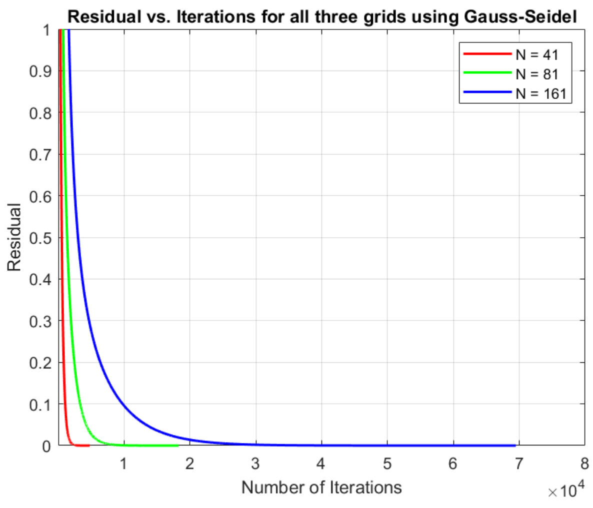

Figure 6. residual v/s number of iterations for Gauss-Seidel method

The convergence behaviour of the Gauss-Seidel method depends significantly on the grid size N. When comparing cases with $N = 41$, $N = 81$, and $N = 161$, the system with the fewest grid points ($N = 41$) tends to converge the fastest. This trend can be explained by the following key factors:

The total number of unknowns in a 2-D grid increases with the grid size as $N \times N$. For $N = 41$, there are $41^2 = 1681$ unknowns, while for $N = 81$, there are 6561 unknowns, and for $N = 161$, there are 25921 unknowns. As the number of unknowns increases, the system becomes larger and more complex, requiring more iterations for the solution to propagate throughout the grid. The grid spacing h is inversely proportional to N, given by $h = \frac{1}{N-1}$. A smaller N results in larger grid spacing, allowing information to propagate more quickly across the grid. For larger N, the finer grid spacing slows down the propagation of information, leading to slower convergence.

6. Line by Line methods

6.1. Row-sweep technique

The *row-sweep technique* is an iterative method where we solve for the values of ϕ at all points in a particular row using the known values from the previous iteration for neighbouring rows. The first step of the process is to guess the values of ϕ at all points. In our current context, let's assume that we are at the $(n+1)^{\text{th}}$ iteration, solving for the values in the j^{th} row. The indices i and j refer to the column and row indices, respectively, on the computational grid. To simplify, we will treat points in other rows as known values from the n^{th} iteration.

The discretised form of the PDE at the interior point (i, j) is as follows:

$$\frac{\partial^2\phi}{\partial x^2} \approx \frac{\phi^n_{i+1,j} - 2\phi^{n+1}_{i,j} + \phi^{n+1}_{i-1,j}}{\Delta x^2}$$

$$\frac{\partial^2\phi}{\partial y^2} \approx \frac{\phi^n_{i,j+1} - 2\phi^{n+1}_{i,j} + \phi^n_{i,j-1}}{\Delta y^2}$$

Substituting these into our PDE, we get:

$$\frac{\phi^n_{i+1,j} - 2\phi^{n+1}_{i,j} + \phi^{n+1}_{i-1,j}}{\Delta x^2} + \frac{\phi^n_{i,j+1} - 2\phi^{n+1}_{i,j} + \phi^n_{i,j-1}}{\Delta y^2} = S^n_{\phi,ij}$$

Rearranging this to isolate the unknowns (i.e., terms with $\phi^{n+1}_{i,j}$) on the left-hand side (LHS) and known terms (i.e., $\phi^n_{i,j}$) on the right-hand side (RHS), we obtain:

$$\frac{1}{\Delta x^2}\phi^{n+1}_{i-1,j} - 2\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right)\phi^{n+1}_{i,j} + \frac{1}{\Delta x^2}\phi^{n+1}_{i+1,j} = S^n_{\phi,ij} - \frac{\phi^n_{i,j+1} + \phi^n_{i,j-1}}{\Delta y^2}$$

This system of equations for the j^{th} row is reduced to a *Tri-diagonal system* because only three terms, $\phi^{n+1}_{i-1,j}$, $\phi^{n+1}_{i,j}$, and $\phi^{n+1}_{i+1,j}$, are unknown. We can solve this tri-diagonal system for each row using *Thomas Algorithm (TDMA)*. Once we finish solving for all rows, it concludes one complete iteration.

6.2. Column-sweep technique

The *column-sweep technique* follows a similar iterative procedure as the row-sweep but now solving for the points along each column. This method also reduces the system of equations along each column to a tri-diagonal system, which can then be solved using the *Thomas Algorithm (TDMA)*.

In the column-sweep approach, at iteration $n+1$, for each column i, the values of ϕ at all the points along the column are computed using the values from the previous iteration for the neighboring columns. At a given interior point (i, j), we can discretize the second derivatives as follows:

$$\frac{\partial^2\phi}{\partial x^2} \approx \frac{\phi^n_{i+1,j} - 2\phi^{n+1}_{i,j} + \phi^{n-1}_{i-1,j}}{\Delta x^2}$$

$$\frac{\partial^2\phi}{\partial y^2} \approx \frac{\phi^{n+1}_{i,j+1} - 2\phi^{n+1}_{i,j} + \phi^{n+1}_{i,j-1}}{\Delta y^2}$$

Substituting these into the PDE and rearranging to isolate the unknowns, we get:

$$\frac{1}{\Delta y^2}\phi^{n+1}_{i,j-1} - 2\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right)\phi^{n+1}_{i,j} + \frac{1}{\Delta y^2}\phi^{n+1}_{i,j+1} = S^n_{\phi,ij} - \frac{\phi^n_{i+1,j} + \phi^n_{i-1,j}}{\Delta x^2}$$

Similar to the row-sweep method, this system of equations for the i^{th} column is reduced to a tri-diagonal system. We solve this tri-diagonal system for each column using *TDMA*, completing one iteration.

6.3. Calculation of Residuals

After completing an iteration, we calculate the *residual* to check how close the current solution is to the true solution. This can be done by calculating the difference between the left-hand side and right-hand side of the equation $A\phi = b$. In terms of residuals, we calculate:

$$\text{Residual} = ||A\phi - b||$$

Here, A is the coefficient matrix derived from the discretisation of the PDE, ϕ is the current solution vector, and b is the known right-hand side vector (which includes the source terms and contributions from boundary conditions). The L2 norm of this residual is used as the measure of error, and we stop iterating when this residual falls below a predefined tolerance.

6.4. Thomas Algorithm (TDMA) for Solving Tri-Diagonal Systems

The *Thomas Algorithm* (also known as *TDMA* or *Tridiagonal Matrix Algorithm*) is a specialized form of Gaussian elimination optimized for solving systems where the coefficient matrix A is tri-diagonal. A tri-diagonal matrix has non-zero elements only on the main diagonal and the diagonals immediately above and below it.

For a system of equations of the form:

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i$$

Where a_i, b_i, and c_i are the sub-diagonal, main diagonal, and super-diagonal elements, respectively, the *TDMA* works as follows:

6.4.1. Forward Sweep

We first eliminate the lower sub-diagonal by performing a forward sweep, modifying the coefficients b_i and d_i as follows:

$$b'_i = b_i - \frac{a_i c_{i-1}}{b_{i-1}}, \quad d'_i = d_i - \frac{a_i d_{i-1}}{b_{i-1}}$$

This transformation eliminates a_i for all rows below the first row, reducing the system to a simpler upper triangular form.

6.4.2. Back Substitution

Once the forward sweep is complete, we can solve for x_i using *back substitution* starting from the last equation and moving upwards:

$$x_n = \frac{d'_n}{b'_n}$$

$$x_i = \frac{d'_i - c_i x_{i+1}}{b'_i}, \quad \text{for } i = n-1, n-2, \dots, 1$$

In this way, TDMA provides an efficient method for solving the system of equations with computational complexity $O(n)$, making it highly suitable for large grid systems encountered in iterative methods such as row-sweep and column-sweep.

6.5. Results

We got the following ϕ field Contour plot for $N = 161$ points in either direction:

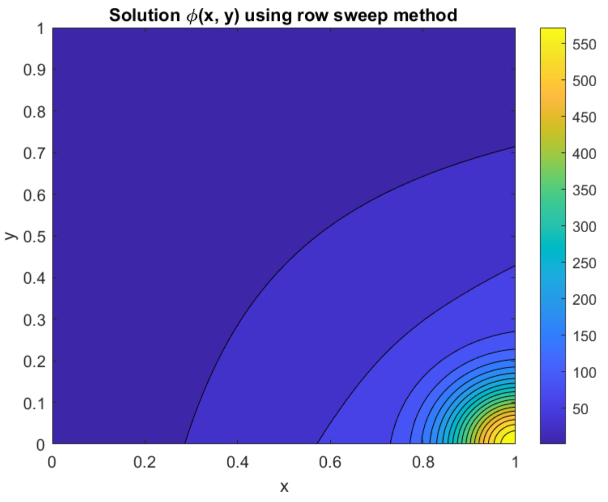

Figure 7. Contour plot of $\phi_{x,y}$ for $N = 161$ using Row-sweep method

We can see that this plot resembles the Analytical solution up to a very good extent. As we decrease the mesh sizing, we notice that the ϕ field approaches the Analytical ϕ field. Next, we plot the residual v/s number of iterations for each grid case.

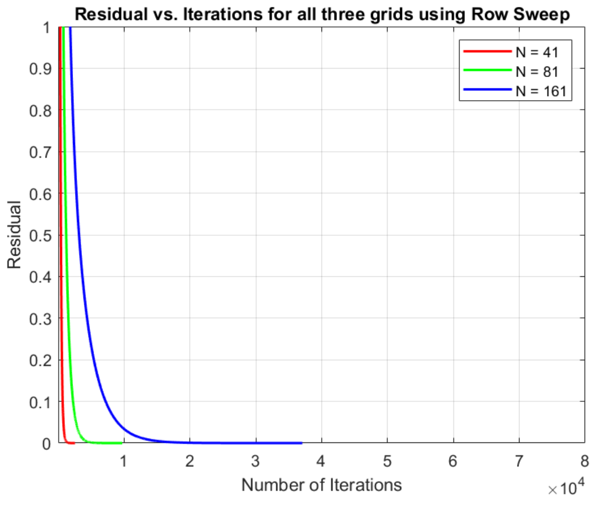

Figure 8. Residual v/s number of iterations for Row-sweep method

Here, it can concluded that the convergence rate decreases as the grid size N increases. When $N = 161$, the grid becomes finer, resulting in a larger number of unknowns and a smaller grid spacing $h = \frac{1}{N-1}$. As the number of rows increases, the information must propagate through more points, which slows down the convergence. Additionally, the larger system size makes the solution propagation more gradual, since the algorithm updates only one row at a time. The finer grid also introduces more interactions between adjacent rows, further slowing the propagation of the solution. This leads to a higher number of iterations to reduce the residual and, thus, slower convergence. In summary, the *larger number of unknowns*, *smaller grid spacing*, and the *serial nature* of the row sweep method contribute to the slower convergence for $N = 161$.

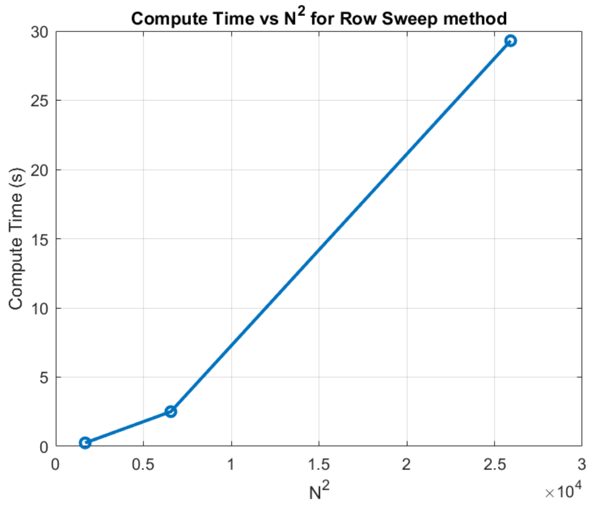

Figure 9. CPU run time (s) v/s N^2

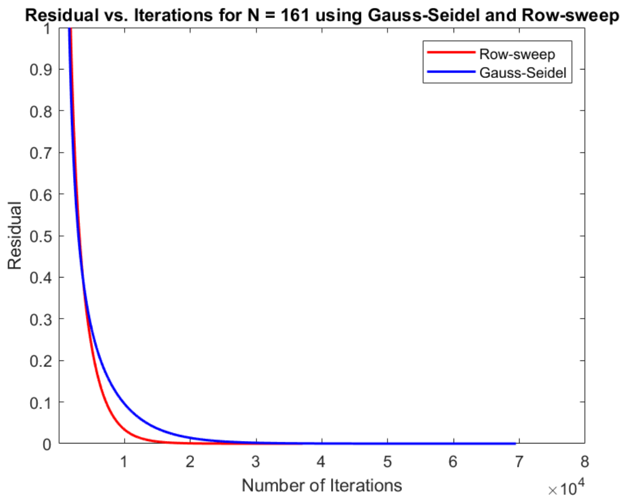

Figure 10. Residual v/s number of iterations for Row-sweep and Gauss-Seidel methods

Similar to the Gauss-Seidel method, the time complexity of the Row-sweep method is also of the order of N^3, where N is the number of grid points in one direction. This is because we have an external loop, which is almost of the order of N. Inside this loop, we have 2 loops: One for accessing the rows and the other for accessing the columns. As we fix a row, there are N points in it. The TDMA solver solves for all these points in an order $O(N)$. Since there are also N such rows, the overall order would be $O(N^3)$.

For N = 41, the CPU run time was 0.239864 s. Therefore, when we increase N from 41 to 81, we should approximately get an increase of 2^3 times than the previous case. Therefore, theoretically, for N = 81, the CPU run time should be 1.918912 s.

However, the actual run time we get for the case is 2.230190 s. And when we increase N from 81 to 161, we should get the theoretical CPU run time to be $2^3 \times 2.230190$ s or 17.8412 s. Even in this case ($N = 161$), we get a run time of 28.649861 s. These results are comparable because they are of the same order. The reason why it could be taking more time might be because as N increases, it takes more number of iterations to achieve convergence. Also, some functions in the MATLAB code are called only once, which also take up some computational time. They may have a constant time addition to the CPU run time. However, these are not included in our time analysis, which may cause the expected time and actual time required to not match.

From Figure 10. we can see that the row-sweep algorithm takes a lesser number of iterations to converge than the Gauss-Seidel method for $N = 161$. However, an interesting thing to note here is that according to our simulation, Gauss-Seidel is taking less time than the Row-sweep algorithm even though Row-sweep converges much faster (Almost twice as fast). We are unable to identify the reason for this discrepancy. As mentioned in the previous paragraph, it might be because of certain functions called and used within the algorithm that are repeatedly being called and causing time delay.

7. Solution Obtained by Alternating Direction Implicit method (ADI)

The Alternating Direction Implicit (ADI) scheme is an iterative method that alternates between performing a row sweep and a column sweep to solve a discretised partial differential equation (PDE). In each iteration of the ADI method, the row sweep is first applied to update the solution across all rows while treating the terms along the columns as constants. This is followed by the column sweep, where the updated solution from the row sweep is used to solve across all columns, treating the terms along the rows as constants. By alternating between these two sweeps, the ADI method effectively decouples the solution process in two directions, leading to faster convergence compared to standard iterative methods like Gauss-Seidel. The residual is calculated after each sweep, and the process continues until the residual falls below a specified tolerance.

7.1. Grid details

In this part, we need to discretise the domain into a non-square mesh of 41×81 points. This means that there are $N = 81$ points in the x direction and $M = 41$ points in the y direction. Therefore, the grid spacing would be $\Delta x = \frac{1}{N-1}$ and $\Delta y = \frac{1}{M-1}$. The total number of nodes would be $N \times M = 3321$.

7.2. Results

Three different methods had to be applied to this grid. The first method is the Row-sweep method, as discussed in the previous section; the second is the Column-sweep, and the third is the Alternating Direction Implicit (ADI) method. Since we have more points in each row than in each column, we would expect the Row-sweep method would converge much quicker than the Column-Sweep method. This is because when we fix a particular row, we apply TDMA to solve for all 81 points in that row, whereas, for a column, we only solve for 41 points using TDMA. The ADI method would be expected to converge slower than row sweep but faster than column sweep due to its alternating nature. Here is the plot for the residual v/s number of iterations:

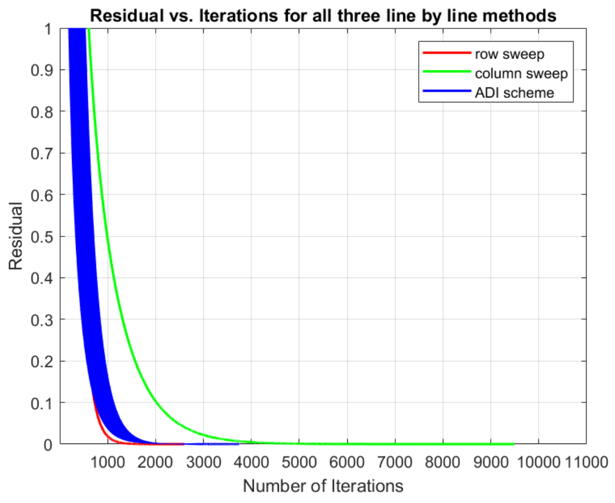

Figure 11. Residuals v/s iterations for the three line-by-line methods

As expected, we can see that row-sweep converges the fastest, followed by ADI and column sweep. The thick line we see for ADI is due to the nature of the residual. After each iteration of Row sweep and column sweep, the residuals are not always decreasing. For example, after the first row-sweep iteration, the residual is 100. In the next iteration, which is a column-sweep, the residual is 150. However, the overall trend is towards the decrease in residuals.

8. Acknowledgements

We would like to express our sincere gratitude to Professor Dilip Srinivas Sundaram for providing us with the opportunity to work on this project. We would also like to thank all the Teaching assistants for their invaluable help in sorting out all the bottlenecks throughout this project. Their dedication and expertise were crucial to the success of this project.