IMPLEMENTATION OF SOR AND ADI METHODS TO NUMERICALLY SOLVE A UNIFORM INVISCID SUBSONIC FLOW HITTING A ONE PERIOD SINE WAVE WRINKLE IN THE METAL SKIN OF THE WIND TUNNEL

MAE 442/542 MIDTERM PROJECT

BY

SIDDHANT S. APHALE

DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING

UNIVERSITY AT BUFFALO, THE STATE UNIVERSITY OF NEW YORK

NOVEMBER 16, 2015

ABSTRACT:

This report discusses the implementation of SOR and ADI methods to Numerically solve a uniform inviscid subsonic flow hitting a One Period sine wave wrinkle in the metal skin of the wind tunnel. The maximum amplitude of the sine wave ε/L is small. Therefore, the Prandtl-Glauert equation for an inviscid, non-heat conducting perfect gas flow, which is only slightly perturbed by a thin body such that fluid motion satisfies the small perturbation assumption, can be considered as a governing equation. The given governing equation is an Elliptic Partial Differential Equation. To solve this equation numerically, Point Gauss Seidel Successive Over Relaxation (SOR) and the Alternating Direction Implicit (ADI) methods were implemented. The results achieved from these two methods were discussed and compared.

1. INTRODUCTION:

A wind tunnel is a tool used in aerodynamic research to study the effects of air moving past solid objects. A wind tunnel consists of a tubular passage in which the object to be tested is placed in the middle. Air is made to pass through the tunnel moving past the object by a very powerful source like fan. The object being tested has a set of suitable sensors mounted on it to measure aerodynamic forces, pressure distribution and other related characteristics. While performing such tests, it is very important to consider the consequences of a possible discontinuity in the metal skin of the wind tunnel. This discontinuity maybe responsible to affect the freestream velocity changes which we are using for the tests. In this problem, discontinuity in the form of a sine wave wrinkle in the metal skin of wind tunnel was examined.

1.1 Problem Definition:

A uniform inviscid subsonic flow U_∞ (M_∞<1) hits a one period sine wave wrinkle in the metal skin of the wind tunnel. The geometry of the configuration is given below. The maximum amplitude of the sine wave ε/L << 1 is small. Therefore, the Prandtl-Glauert equation for an inviscid, non-heat conducting perfect gas free flow, which is only slightly perturbed by a thin body such that fluid motion satisfies the small perturbation assumption, can be considered as the governing equation:

$$(1 - M_\infty^2)\varphi_{xx} + \varphi_{yy} = 0$$

i.e.

$$(1 - M_\infty^2)\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} = 0 \qquad (1)$$

where M_∞ is the free stream Mach number, and the perturbation potential is denoted by φ. The perturbation velocity components can be recovered by $u = \frac{\partial \varphi}{\partial x}$ and $v = \frac{\partial \varphi}{\partial y}$.

The Boundary conditions are Neumann type and are given as:

$$\frac{\partial \varphi}{\partial y}(x, 0) = U_\infty \frac{\partial y}{\partial x} = U_\infty \varepsilon \cos(x) \; at \; y = 0 \qquad (2)$$

$$\frac{\partial \varphi}{\partial n} = 0 \; on \; all \; the \; others. \qquad (3)$$

The given problem is non-dimensionalized and the values given are M_∞=0.5, U_∞=1, L=2π and ε=0.1.

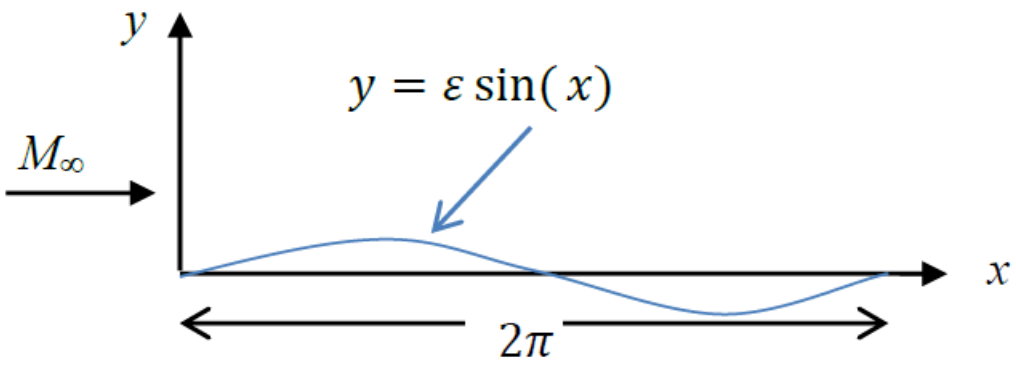

Figure 1: Problem Definition

1.2 Compatibility Condition:

The problem to have a solution must satisfy the following compatibility condition

$$\varphi_{xx} + \varphi_{yy} = f$$

$$i.e. \frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} = f \qquad (4)$$

Equation 4 is the general Laplace equation. The compatibility condition for this equation is given as

$$\int_V f dV = \oint_S g.ndS \qquad (5)$$

where *g* is the given Neumann boundary condition and *n* is the unit normal to the boundary. By comparing Eq. (1) and Eq. (4) we get *f=0*. Thus the left hand side of Eq. (5) is 0. We now need to check if the Right hand side of Eq. (5) holds true.

$$\oint_S g.ndS = \int_0^{2\pi} \left(\frac{\partial \varphi}{\partial x}(0, y) - \frac{\partial \varphi}{\partial x}(2\pi, y) \right) dy$$

$$\oint_S g.ndS = \int_0^{2\pi} \left(\frac{\partial \varphi}{\partial y}(x, y) - \frac{\partial \varphi}{\partial y}(x, 0) \right) dx$$

But from the boundary condition given in Eq. (3) $\frac{\partial \varphi}{\partial \mathrm{n}} = 0$. Thus we get

$$\oint_S g.ndS = \int_0^{2\pi} U_\infty \varepsilon \cos(x)\, dx = 0$$

Thus the Right hand side of the equation is also zero. Hence, the compatibility condition is satisfied and a solution exists to the problem under condition.

1.3 Discretization:

Considering a rectangular grid, we can write the governing equation as below.

$$(1 - M_\infty^2)\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} = 0$$

$$(1 - M_\infty^2)\frac{\varphi_{(i+1,j)} - 2\varphi_{(i,j)} + \varphi_{(i-1,j)}}{(\Delta x)^2} + \frac{\varphi_{(i,j+1)} - 2\varphi_{(i,j)} + \varphi_{(i,j-1)}}{(\Delta y)^2} \qquad (6)$$

$$\varphi_{(i+1,j)} - 2\varphi_{(i,j)} + \varphi_{(i-1,j)} = \beta\left(\varphi_{(i,j+1)} - 2\varphi_{(i,j)} + \varphi_{(i,j-1)}\right) \qquad (7)$$

where $\beta = \frac{(\Delta x)^2}{(1-M_\infty^2)(\Delta y)^2}$

Here we have done Second order accurate discretization in space using Taylor Series expansion. We use ghost nodes to change the discretized equation at boundaries using following equation for boundaries.

$$\varphi_{(i+1,j)} = \varphi_{(i-1,j)}$$

$$\varphi_{(i-1,j)} = \varphi_{(i+1,j)}$$

$$\varphi_{(i,j+1)} = \varphi_{(i,j-1)}$$

$$\varphi_{(i,j-1)} = \varphi_{(i,j+1)} - 2\Delta y \varepsilon \cos(x)$$

These equations are plugged into the governing equation to implement the boundary conditions at the edges of rectangular grids.

1.4 Grid Definition:

Here we define a rectangular grid. Each node on the grid is defined with “i,j” notation, where “i" is the notation used along X axis of the grid and “j” is used along the Y axis. Grid spacing is dependent on the number of nodes. It is inversely proportional to the number of nodes. As we increase the number of nodes, the grid becomes finer and computational cost increases.

2. METHODS:

The given governing equation is an Elliptical PDE. There are basically two methods to solve Elliptic PDE namely- Direct Methods and Iterative Methods. Some familiar Direct methods are Cramer’s rule and Gaussian elimination. However, in these methods, enormous amount of arithmetic operations is required to produce solution. Thus, using iterative methods is preferred over direct methods. In iterative methods, we obtain the solution by iteration. Usually, an initial

solution is guessed and new values are computed; based on the newly computed values, a newer solution is sought, and the procedure is repeated until a specified convergence criterion has been reached. To solve the given problem, we use Point Gauss Seidel Successive Over-Relaxation Method (PSOR) and the Alternating Direction Implicit (ADI) method.

2.1 Point Gauss Seidel Successive Over-Relaxation (PSOR) Method:

Point Gauss Seidel method is one of the methods used to solve linear system of equations. In this method, the current values of the dependent variable are used to compute the neighboring points as soon as they are available. The finite difference equation for Point Gauss Seidel method is given as follows:

$$\varphi_{(i,j)}^{(k+1)} = \frac{1}{2(1+\beta)}\left[\varphi_{(i+1,j)}^{(k)} + \varphi_{(i-1,j)}^{(k+1)} + \beta\left(\varphi_{(i,j+1)}^{k} + \varphi_{(i,j-1)}^{(k+1)}\right)\right] \qquad (8)$$

In Eq. 8, the superscripts *k* and *k+1*indicate the iteration step. The computation of $\varphi^{(k+1)}$ uses only the elements of $\varphi^{(k+1)}$that are already computed, and only the elements of φ^{k}that have not yet to be advanced to iteration at *k+1*. Thus, in this method we can store the vectors is same storage vectors by overwriting. Successive over-relaxation is a method of solving a linear system of solving a linear system of equations $\boldsymbol{Ax = b}$ derived by extrapolating the Gauss-Seidel method. This extrapolation takes the form of a weighted average between the previous iterate and the computed Gauss-Seidel iterate successively for each component,

$$\varphi_{(i,j)}^{(k+1)} = \varphi_{(i,j)}^{(k)} + \omega(\overline{\varphi}_{(i,j)}^{(k+1)} - \varphi_{(i,j)}^{(k)}) \qquad (9)$$

where $\overline{\varphi}_{(i,j)}^{(k+1)}$ denotes a Gauss-Seidel iterate and ω is the extrapolation factor. The idea is to choose a value for ω that will accelerate the rate of convergence of the iterates to the solution.

If ω=1, the SOR method simplifies to the Gauss-Seidel method.

If $0 < \omega < 1$, it is called under-relaxation and it helps in stability.

If $\omega > 1$, it is called over-relaxation and it can over shoot the solution and the extrapolate.

If $\omega > 2$, it is unstable.

Best performance is achieved for $\omega_{optimum}$. If $\omega < \omega_{optimum}$, convergence rate is monotonic. Rate of convergence increases with increase in ω. If $\omega > \omega_{optimum}$, convergence is oscillatory. Rate of convergence decreases with increase in ω.

2.2 Alternating Direction Implicit (ADI) Method:

Alternating Direction Implicit (ADI) method is a finite difference method used for solving parabolic, elliptic and hyperbolic equations. It is most notably used to solve the problem of heat conduction or diffusion equation in two or more dimensions. Crank-Nicolson method is generally used to solve the heat equation. However, this method involves solving complex equations and it increases the computational cost. ADI method is advantageous in this case. Equation that is to be solved in each step have a simpler structure and can be solved comparatively faster using a

tridiagonal matrix algorithm. In ADI method, we alternate the direction which results in the tridiagonal matrix in each direction to solve 1-D problem at each time step. In the first step we keep the y-direction fixed and in the second step x-direction is fixed. We take values from the previously solved x-direction. The resulting tridiagonal system is to be solved so as to conclude the iteration. This is to be done for all the rows and then followed by columns or vice versa. When we fix the column, it is called as X-Sweep. When the rows are kept fixed it is known as Y-Sweep. For 2-dimensional problem, model ADI equation is given by,

X-Sweep through all nodes:

$$\varphi_{(i-1,j)}^{\left(k+\frac{1}{2}\right)} - 2(1+\beta)\varphi_{(i,j)}^{\left(k+\frac{1}{2}\right)} + \varphi_{(i+1,j)}^{\left(k+\frac{1}{2}\right)} = -\beta(\varphi_{(i,j+1)}^{(k)} + \varphi_{(i,j-1)}^{\left(k+\frac{1}{2}\right)}) \qquad (10)$$

Y-Sweep through all nodes:

$$\beta\varphi_{(i,j-1)}^{(k+1)} - 2(1+\beta)\varphi_{(i,j)}^{(k+1)} + \beta\varphi_{(i,j+1)}^{\left(k+\frac{1}{2}\right)} = -\left(\varphi_{(i+1,j)}^{\left(k+\frac{1}{2}\right)} + \varphi_{(i-1,j)}^{(k+1)}\right) \qquad (11)$$

X-Sweep + Y-Sweep = 1 Iteration.

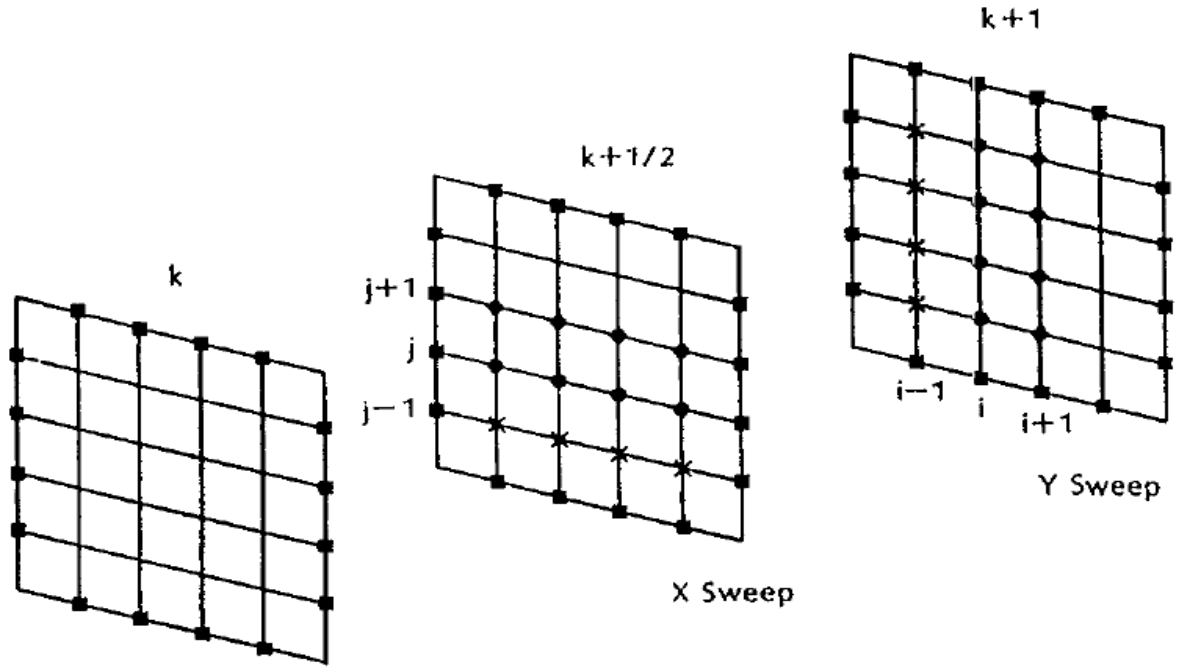

Figure 2: Representation of X-Sweep and Y-Sweep of ADI Method

2.3 Thomas' Algorithm for the Solution of a Tridiagonal System of Equations:

By implementing ADI X-Sweep and Y-Sweep we end up getting a Tridiagonal system of equations or a Tridiagonal matrix. These system of equations are solved using a well-known method Thomas' Algorithm. A tridiagonal system for 'n' unknowns may be written as

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

where $a_1 = 0$ and $c_n = 0$

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}.$$

For such systems, the solution can be obtained in O(n) operations instead of $O(n^3)$ required by Gaussian elimination. Thomas' algorithm is a 2 step method. A first sweep eliminates the a_i's and then backward substitution produces the solution.

3. RESULTS AND DISCUSSION:

3.1 Point Gauss Seidel with Successive Over-Relaxation:

Point Gauss Seidel with Successive Over-Relaxation was one of the methods used to solve the given problem. In this method the value of relaxation factor, ω was varied from 1 to 2 as we know that when ω=1, the method is identical to Gauss-Seidel, Jacobi, etc. If $0<\omega<1$, the system is said to be under relaxation which helps with stability. When $\omega>1$, the system is in over-relaxation and can over shoot the solution and extrapolate. However, when $\omega>2$, the system is unstable. The best performance is achieved for $\omega_{optimum}$.

If $\omega<\omega_{optimum}$, convergence rate is monotonic. In this case, the rate of convergence increases with increases in ω. If $\omega>\omega_{optimum}$, convergence is oscillatory. Rate if convergence decreases as we increase ω. In our problem, the value of ω is varied in between 1 to 2 with an increment of 0.1. The $\omega_{optimum}$ value is found out to be 1.65. Figure 3 shows the Analytical and Numerical Perturbation potential for Gauss-Seidel with Successive Over Relaxation with nodes size 40x40. From fig. 3 it can be observed that when the value of ω is above 1.7 there are oscillations. The iterations required for Successive Over-Relaxation ($\omega_{optimum}$=1.65) is 4837. For ω=1 i.e Gauss-Seidel Method, the number of iterations required are 7112.

The convergence criteria given for this problem is

$$\frac{\sum_{j=1}^{JM}\sum_{i=1}^{IM}\left\|(\varphi_{(i,j)}^{(k+1)}-\varphi_{(i,j)}^{k}\right\|}{\sum_{j=1}^{JM}\sum_{i=1}^{IM}\left\|\varphi_{(i,j)}^{(k)}\right\|} < \varepsilon \qquad (12)$$

where ε is a very small number. In our case ε=10^{-5}, *IM* and *JM* are the number of nodes in in X and Y-directions respectively.

Thus, we can clearly say that convergence is achieved faster when $\omega_{optimum}$=1.65

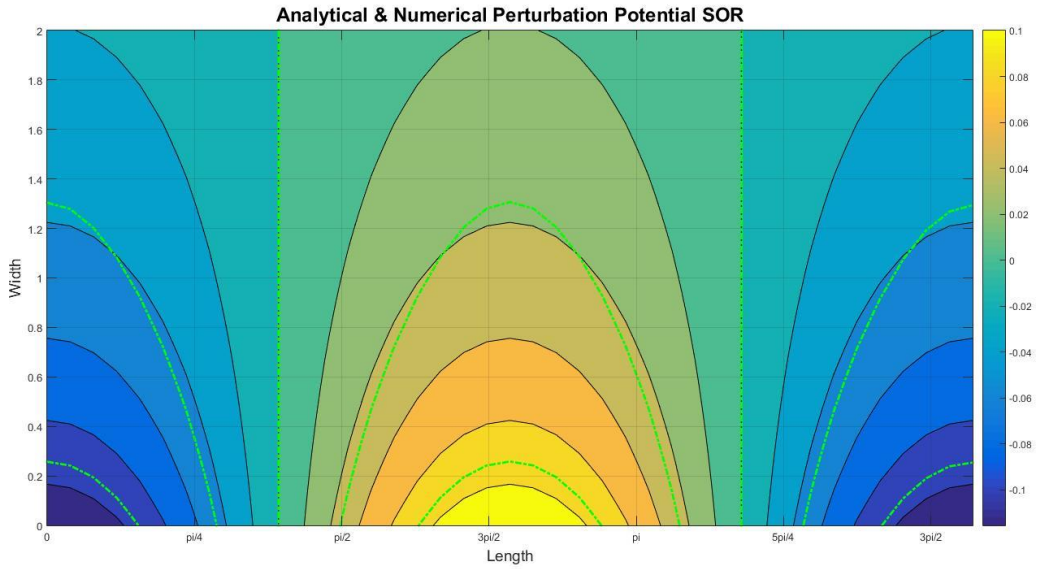

Figure 3: Analytical and Numerical Perturbation Potential

In fig. 3 we can see the analytical solution and superimposed numerical solution in green line color. The error involved in between both the solution is due to truncation error in Taylor series expansion and computing errors in the numerical solution.

Figure 4 shows the velocity vector plots for the converged solution using Point Gauss-Seidel SOR method. Figure 5 & 6 shows the contour plots for "u" velocity and "v" velocity respectively. These plots were obtained after 4837 iterations. It is computationally expensive as compared to ADI.

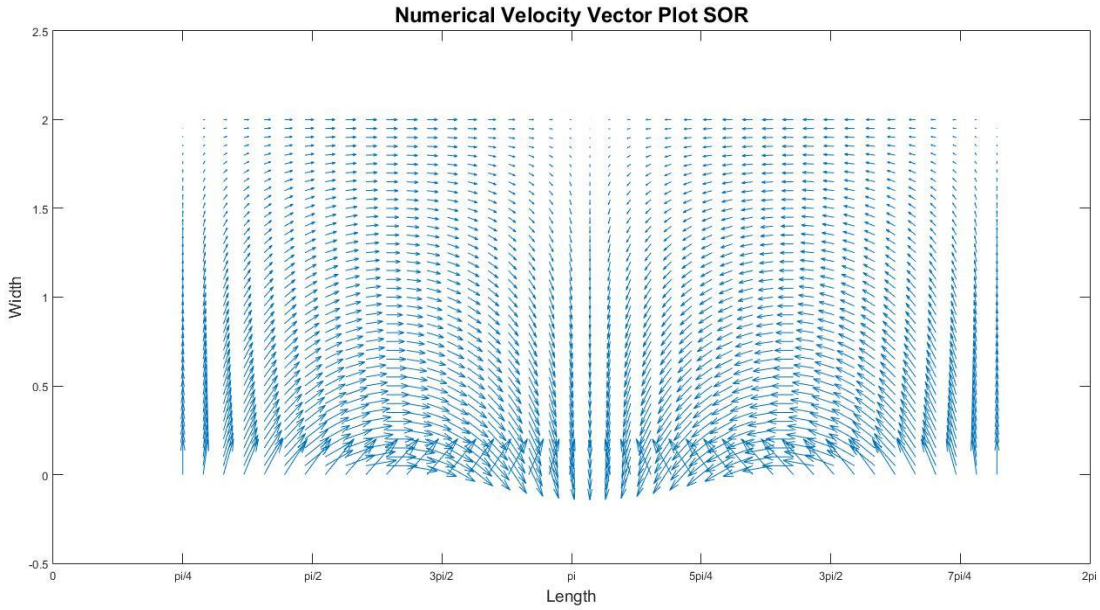

Figure 4: Velocity Vector Plot Gauss-Seidel SOR-4837 iterations.

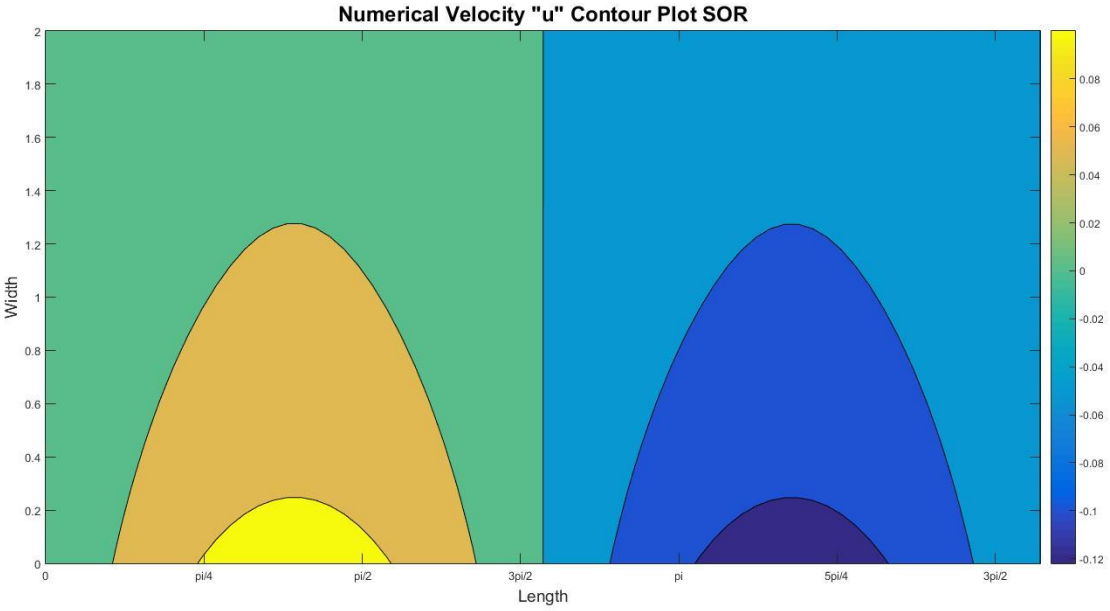

Figure 5: Velocity "u" Contour Plot Gauss-Seidel SOR-4837 iterations.

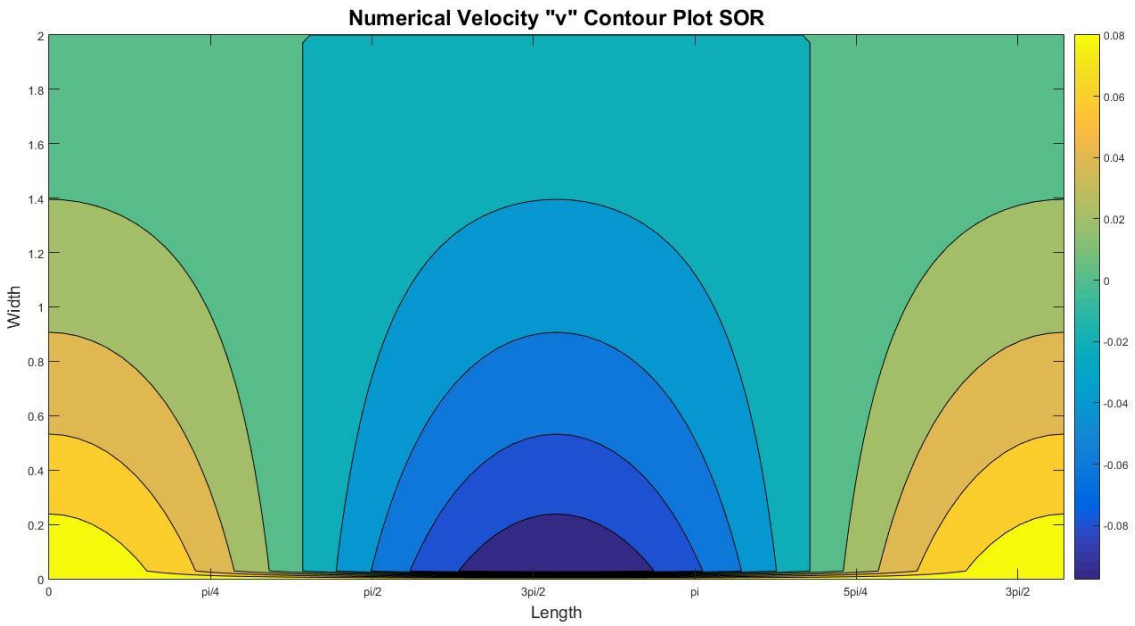

Figure 6: Velocity "v" Contour Plot Gauss-Seidel SOR-4837 iterations.

The above results are shown for same grid size with 40 nodes in both X & Y-direction. Thus the grid size is 20 x 20. The solution obtained from this grid is a converged solution. However, it is important to make the solution grid independent. Thus, the solution must not change if me go on making the grid finer. As we make the grid finer, we must achieve more accurate solution. However, by making the grid finer we are increasing the computational cost and the solution will take time to converge. Solution will be grid independent when there will be no significant change in the solution when we make the grid finer. This can be done by reaching the convergence and then increasing the grid size by increasing the number of nodes considered for the grid definition. Once we have achieved the solution for all the grid sizes we define we need to compare the results and errors of fine grids with the initial grid size. There will be a point for which solution won't change significantly even after we increase the grid size. This point is when we achieve a solution where it is independent of the grid. In our case we have initially selected 40,50,60,70 nodes. This will give us grid sizes as 20x20, 25x25, 30x30, 35x35. The perturbation plots for these grids is shown below for comparison. It can be observed from fig. 7, 8, 9 and 10 that there is a significant change in solution. All these plots are taken with ω=1. The solution was obtained in 7112 iterations. Now we will increase the grid size by a significant number so we can analyze if there is a significant change in the error or error has stabilized. We consider a grid of size 65x65 and 75x75 i.e 130 and 150 nodes respectively. It can be observed from fig. 11 & 12 that there is no significant change in the error. Thus we can say that the error has stabilized and the solution is now grid independent.

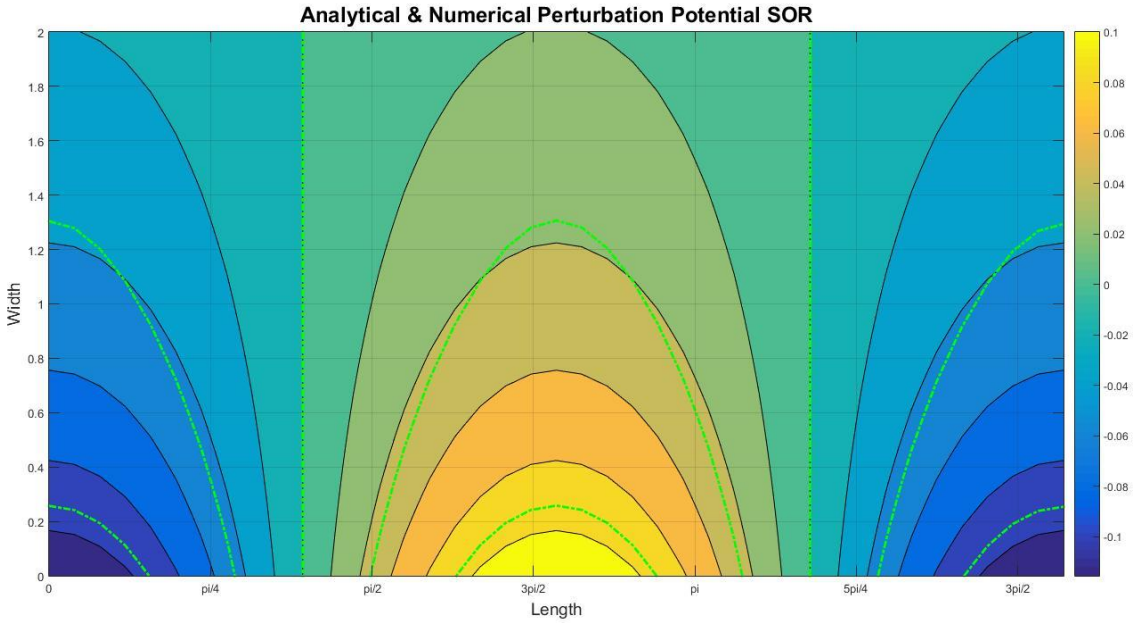

Figure 7: Analytical and Numerical Perturbation Potential 20x20 grid

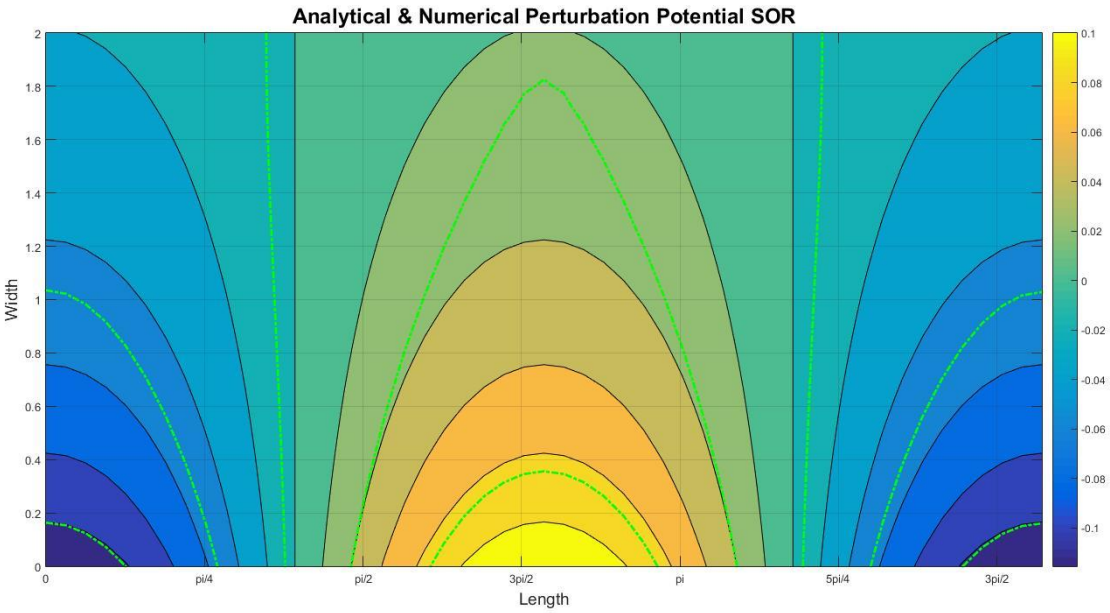

Figure 8: Analytical and Numerical Perturbation Potential 25x25 grid

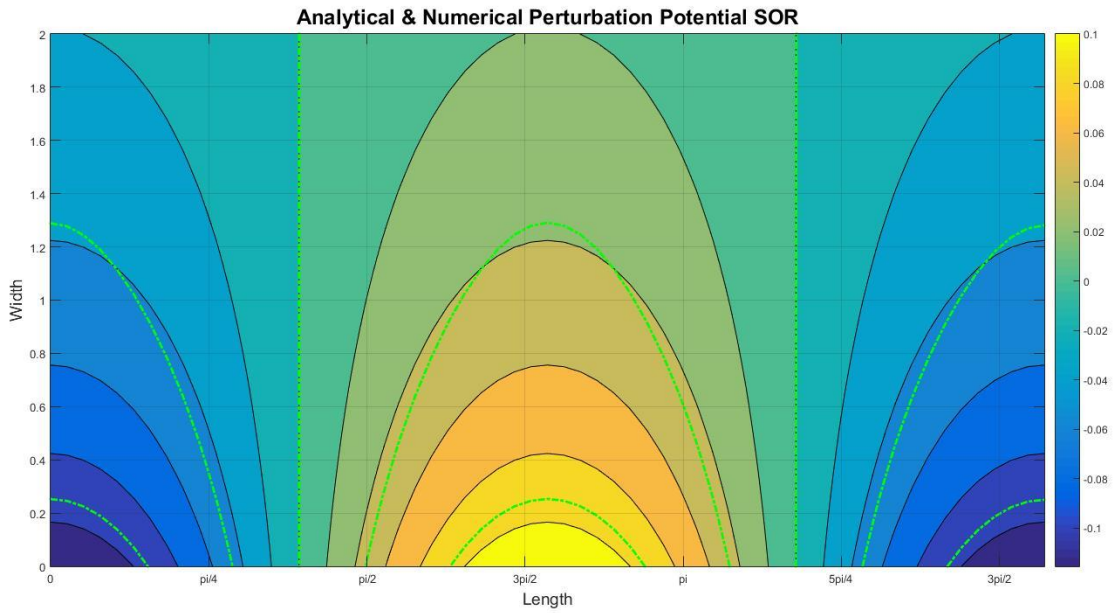

Figure 9: Analytical and Numerical Perturbation Potential 30x30 grid

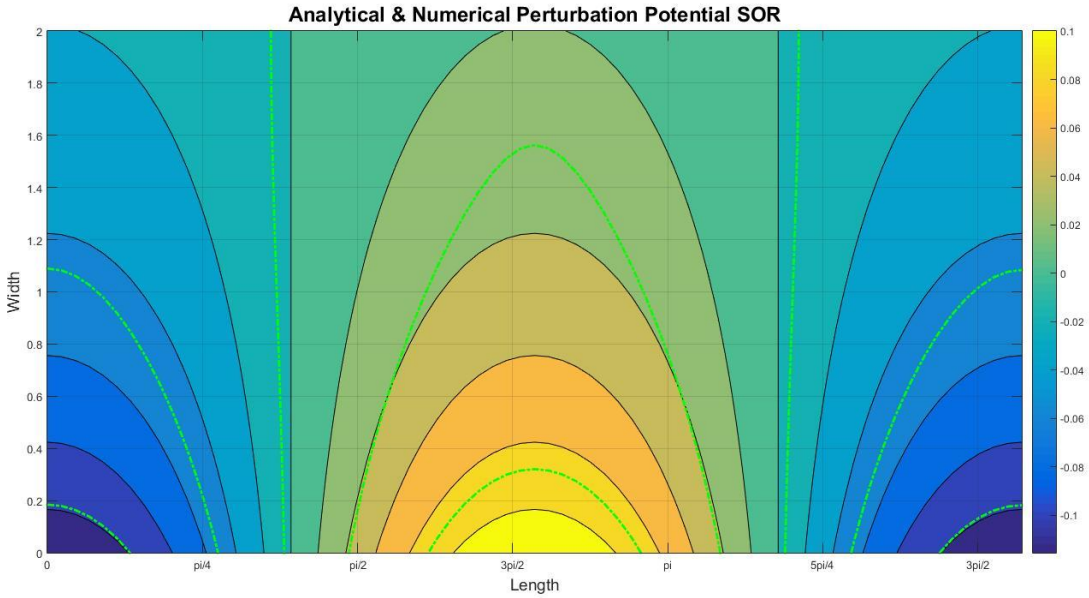

Figure 10: Analytical and Numerical Perturbation Potential 35x35 grid

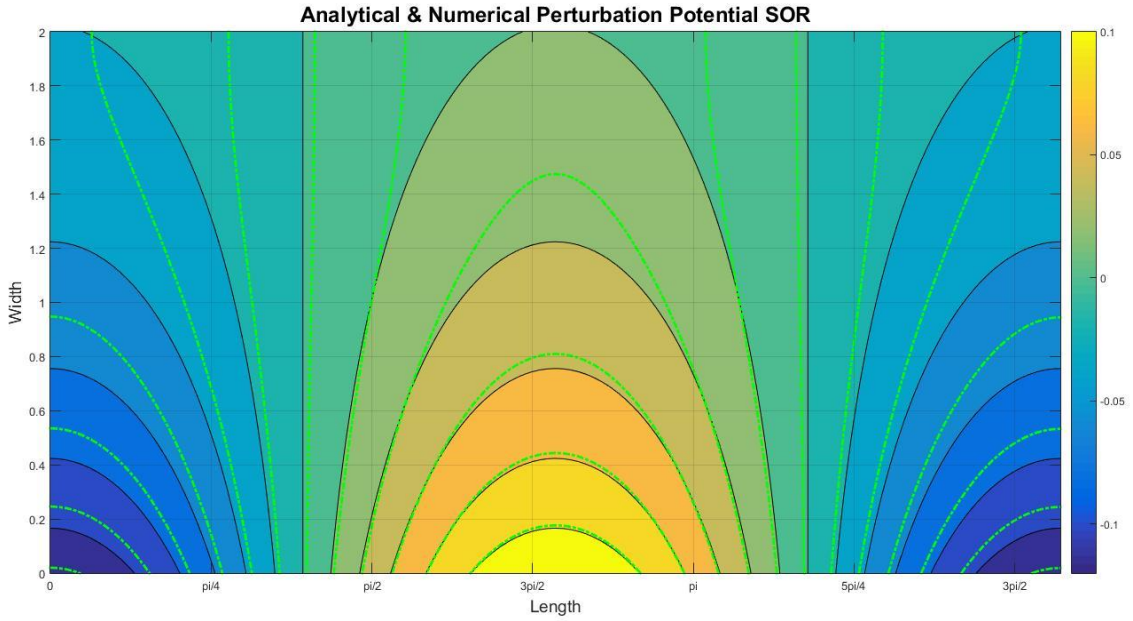

Figure 11: Analytical and Numerical Perturbation Potential 65x65 grid

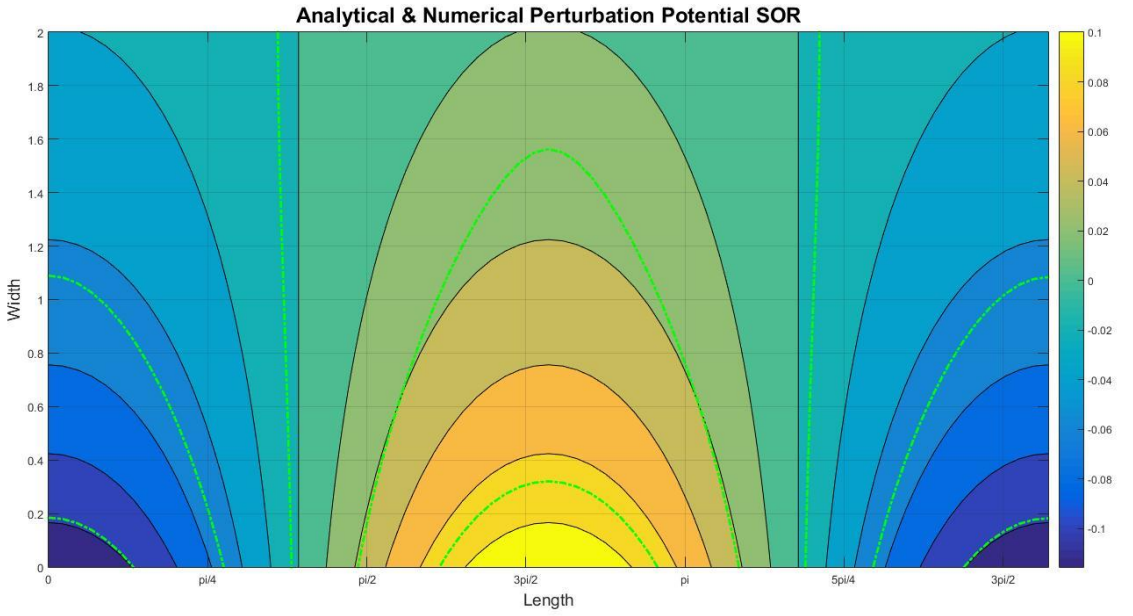

Figure 12: Analytical and Numerical Perturbation Potential 75x75 grid

The Gauss-Seidel with SOR method is a five-point formula. In this method we have used central differencing scheme to discretize the governing equation. While using the central difference scheme, we use Taylor Series expansion. Thus after implementing the Taylor series expansion, we truncate the higher order terms for convenience. Thus this truncation error is always involved in

numerical solution. The error is found out by comparing the numerical solution with analytical solution. The error is given by following equation

$$error = \frac{1}{JM*IM}\sqrt{\sum_{j=1}^{JM}\sum_{i=1}^{IM}(\varphi_{(i,j)}^{computed} - \varphi_{(i,j)}^{analyical})^2} \quad (13)$$

We know that the central difference scheme is second order accurate. The error is plotted on a log-log scale and is represented by fig. 13. This plot is log(delx) vs log(Error). It can be seen from the plot that the slope is 2. Thus we can conclude that the code is second order accurate.

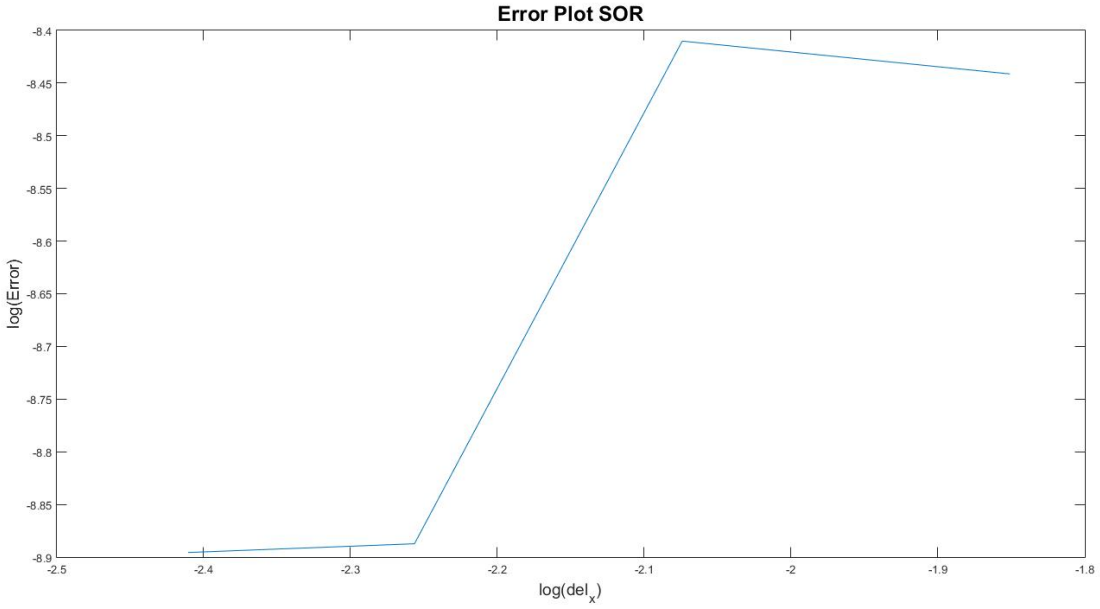

Figure 12: Error plot on log-log scale.

3.2 Alternating Direction Implicit:

Alternating Direction Implicit was the other method used to solve the given governing equation. The Gauss-Seidel SOR is an explicit method however ADI is an Implicit method. In ADI method we implement X-Sweep and Y-Sweep. In ADI method we solve tridiagonal matrix for each sweep. Thus for each iteration we need to solve two tridiagonal matrices. The convergence criteria are defined by following condition

$$\frac{\sum_{j=1}^{JM}\sum_{i=1}^{IM}\left\|(\varphi_{(i,j)}^{(k+1)} - \varphi_{(i,j)}^{k}\right\|}{\sum_{j=1}^{JM}\sum_{i=1}^{IM}\left\|\varphi_{(i,j)}^{(k)}\right\|} < \varepsilon$$

where ε is a very small number. In our case ε=10^{-5}, *IM* and *JM* are the number of nodes in in X and Y-directions respectively.

Figure 13 represents the Perturbation plot for ADI method. The plot consists of analytical solution and numerical solution is superimposed in green color line. The solution is achieved for 40x40 nodes in X and Y-directions. Thus the grid size is 20x20. The number of iterations involved in ADI are 1286. Thus the solution is achieved after solving 2572 tridiagonal matrices. Thus this increases the computational cost.

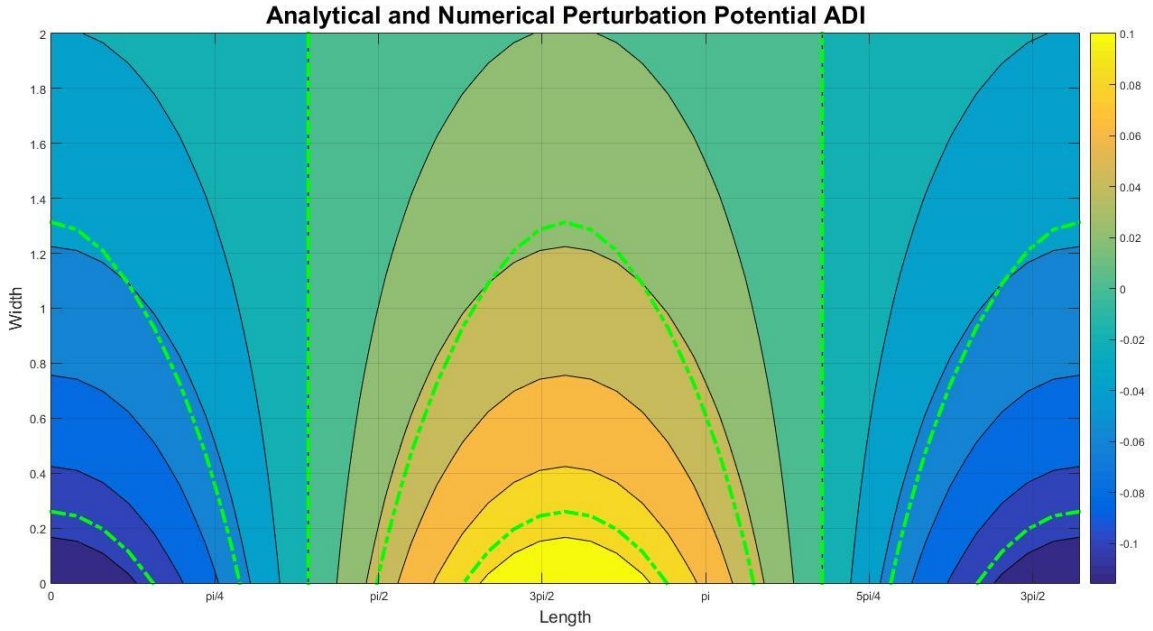

Figure 13: Perturbation plot for ADI method with 20x20 grid size.

Figure 14 represents the velocity vector plots recovered from perturbation potential of converged ADI method. Figure 15 and 16 represents the velocity contour plots for u and v respectively. These plots are achieved after 1286 iterations. Thus it is clear from the number of iterations that ADI method is computationally cheaper as compared to Point Gauss-Seidel with Successive Over-Relaxation method where the converged solution was achieved after 4837 iterations.

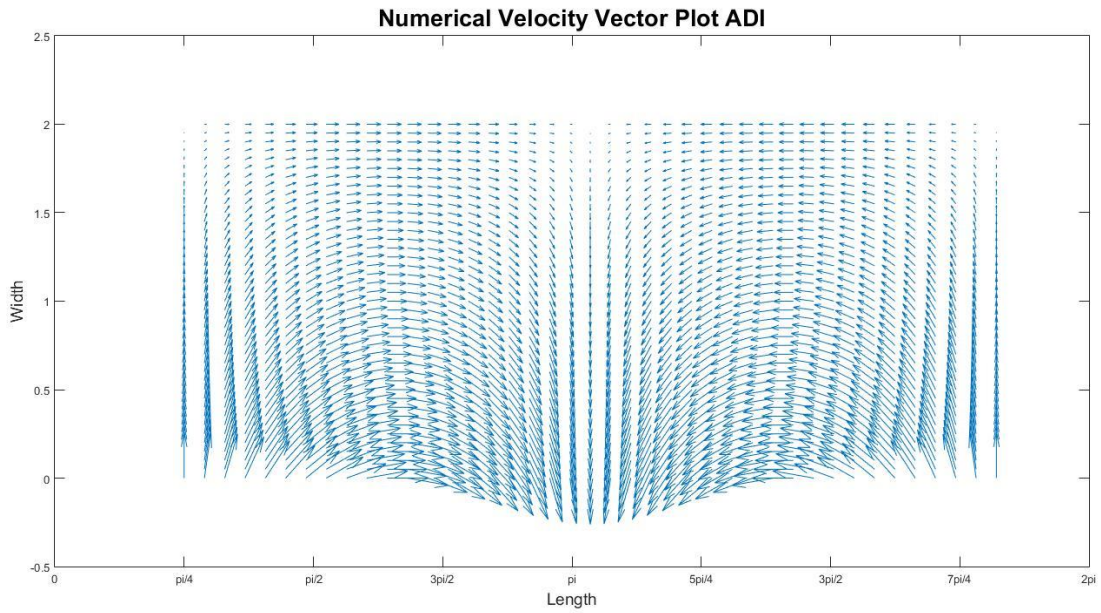

Figure 14: Velocity Vector Plots for ADI Method

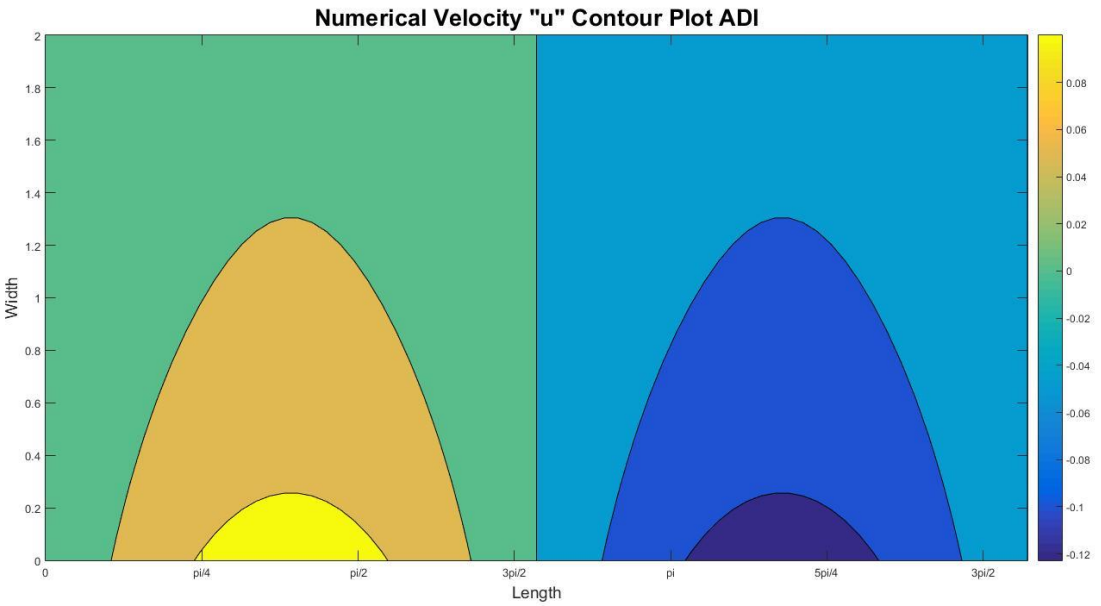

Figure 15: Velocity "u" contour plot by ADI Method

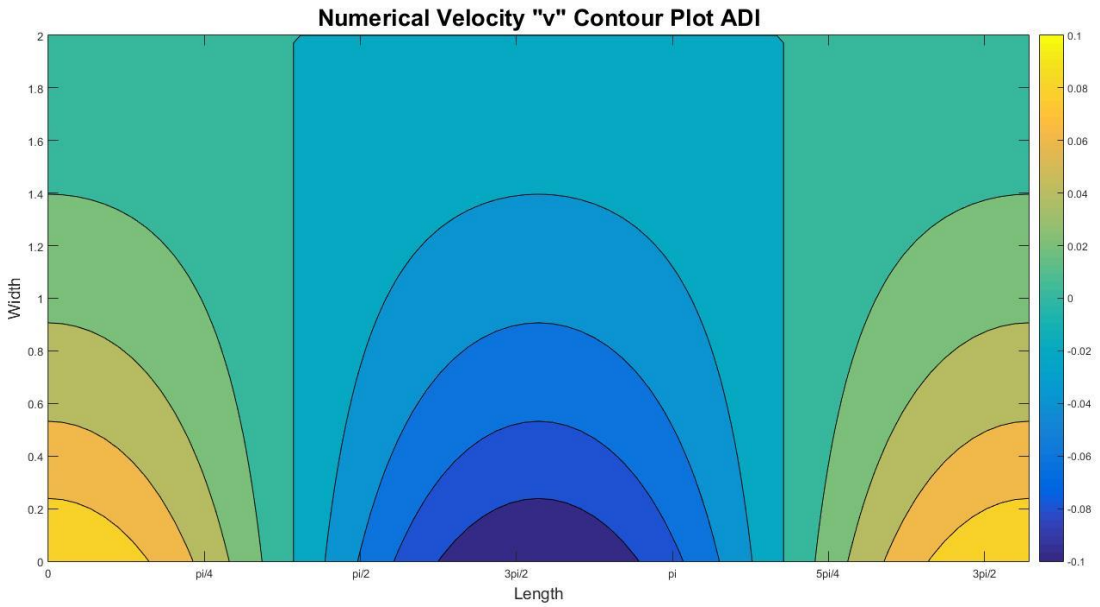

Figure 16: Velocity "v" Contour Plot by ADI Method

As discussed earlier, a solution must be grid independent. A similar analysis as in SOR was conducted for ADI method to check the grid dependency. Figure 17 to 21 shows the various perturbation plots for grid size 20x20, 25x25, 35x35, 75x75 and 80x80 respectively. It can be observed from fig. 17 to 19 that the error in the plots is significant. However, when we increase the grid size significantly, the error stabilizes as seen from fig. 20 & 21. Thus we can conclude that the solution is now grid independent.

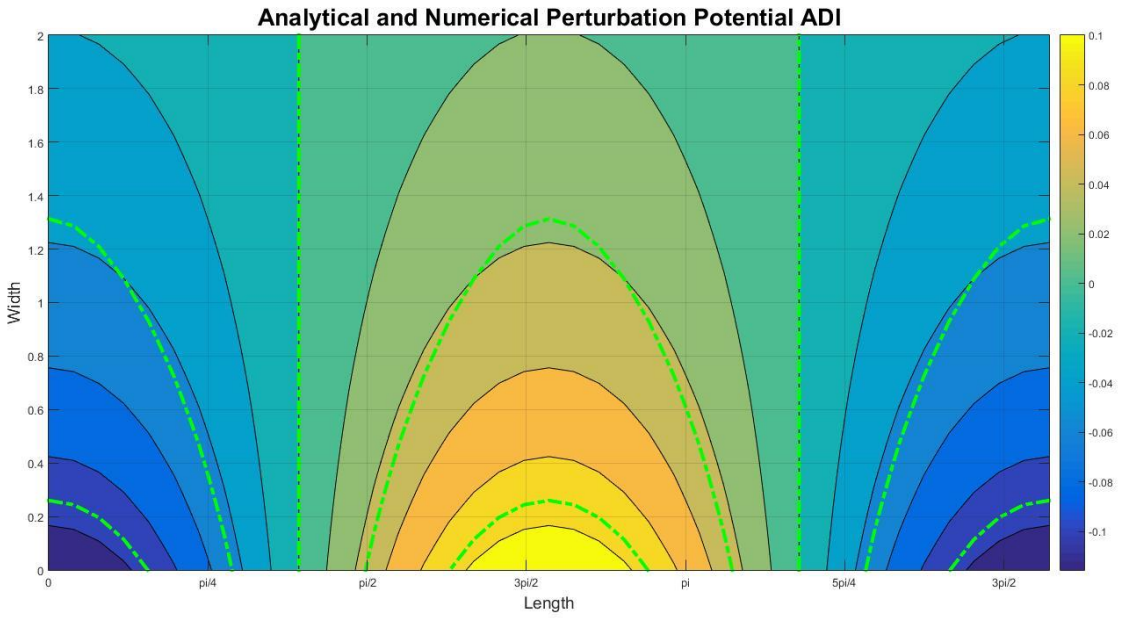

Figure 17: Perturbation plot for ADI method Grid Size 20x20

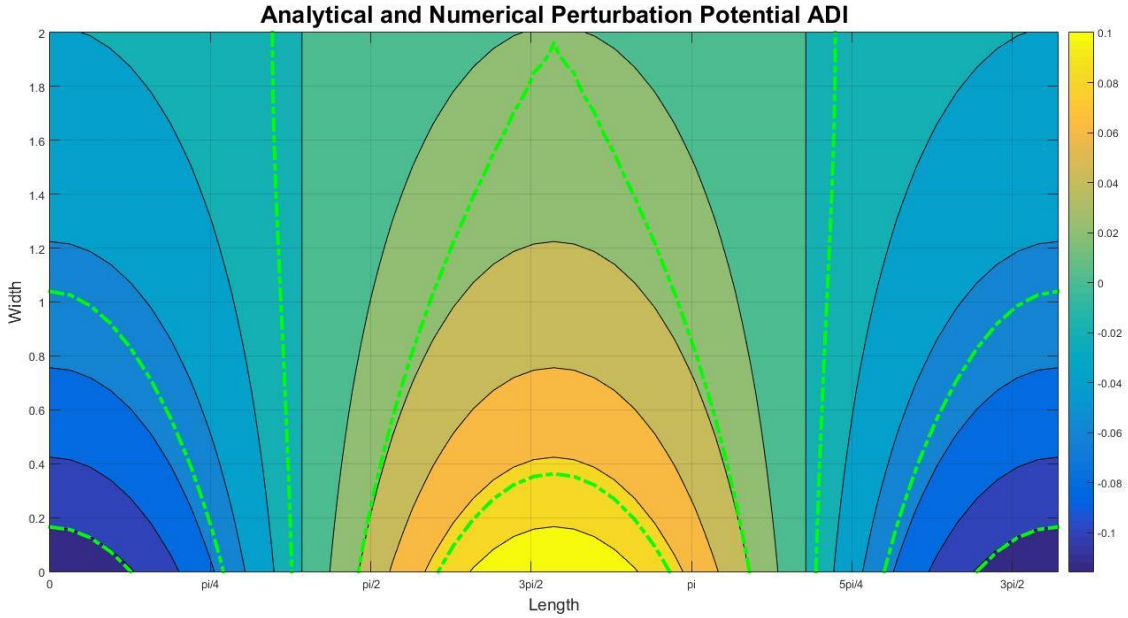

Figure 18: Perturbation plot for ADI method Grid Size 25x25

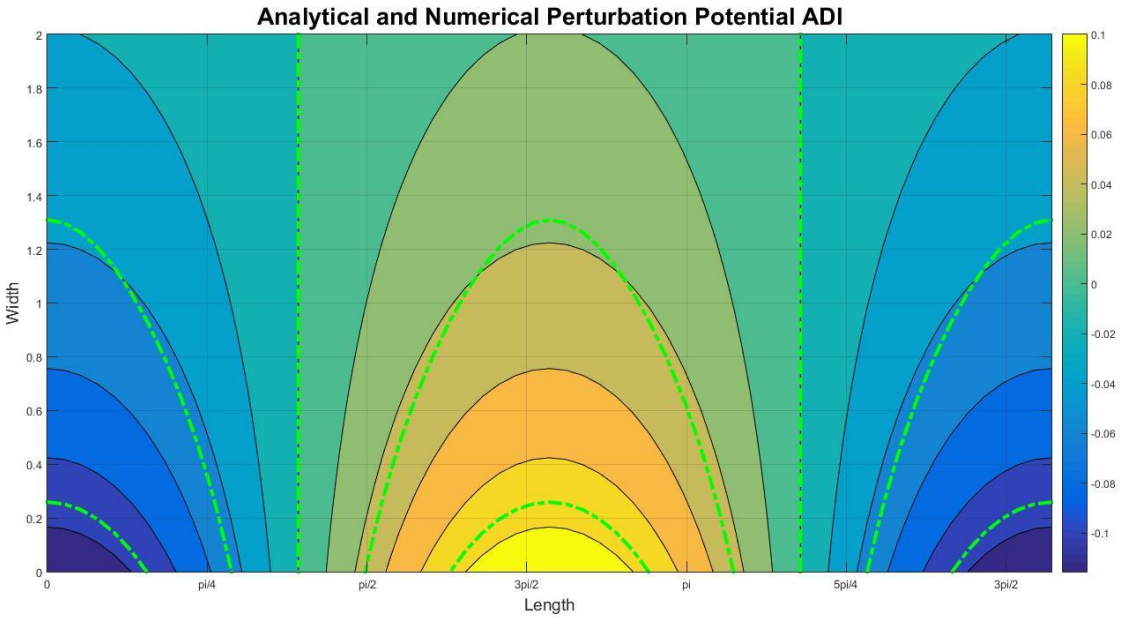

Figure 19: Perturbation plot for ADI method Grid Size 35x35

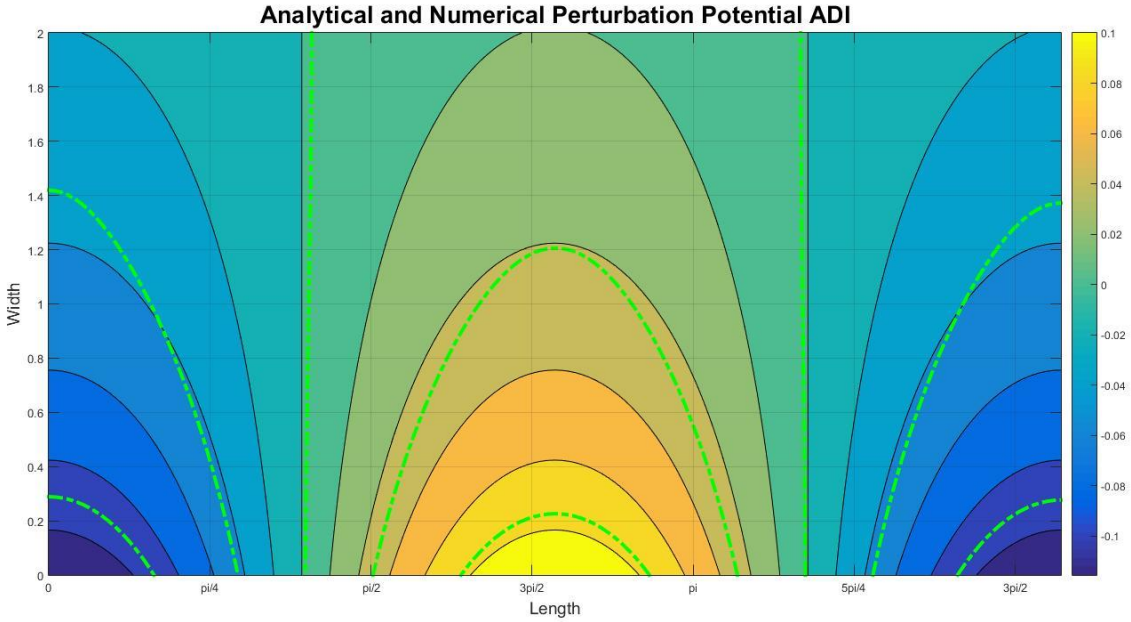

Figure 20: Perturbation plot for ADI method Grid Size 75x75

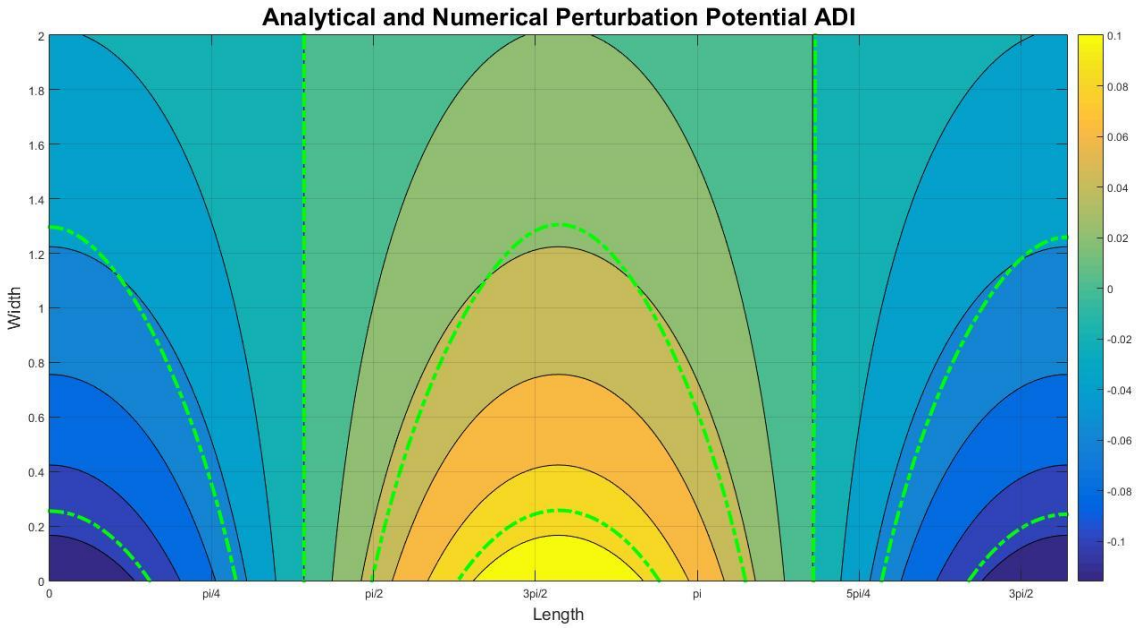

Figure 21: Perturbation plot for ADI method Grid Size 85x85

Similar to SOR method, error analysis was performed by Eq. 13. The error involved in calculating the numerical solution is because of the truncation error involved in the central difference of governing equation and computing errors. The error was plot on a log-log scale and is shown in Fig. 22. The central difference scheme is second order accurate. The program involved a step to calculate the slope of the error plot and it is equal to 2. Thus the solution is second order accurate in space.

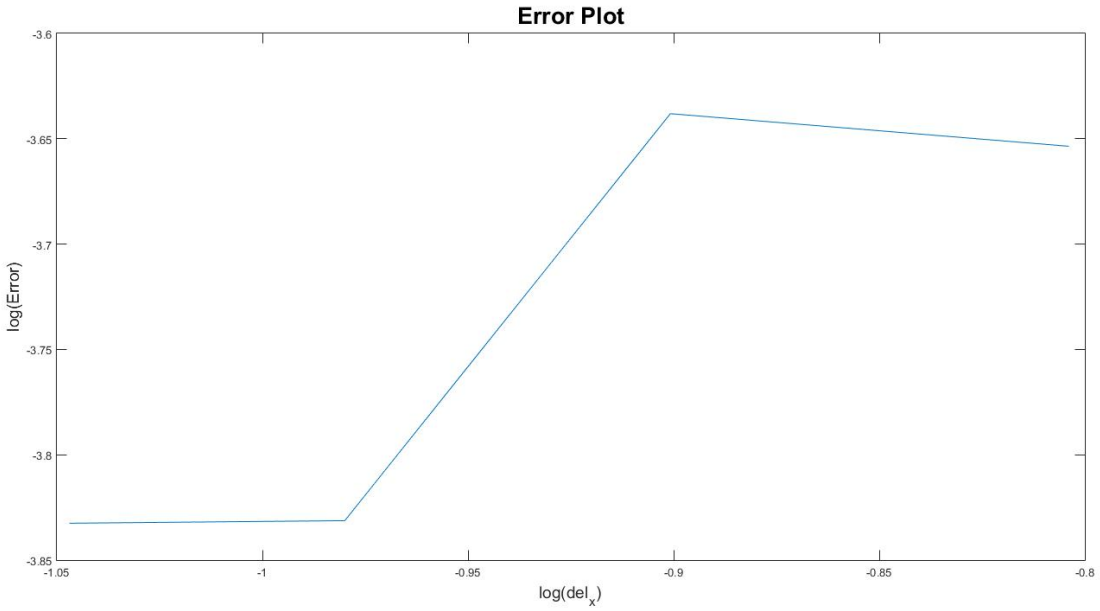

Figure 22: Error plot for log(delx) vs log(Error) for ADI Method.

4. CONCLUSION:

Two methods Point Gauss-Seidel with Successive Over-Relaxation and Alternating Direction Implicit method were presented to solve the perturbation potential for the given 2-dimensional domain. Neumann boundary conditions were involved to initialize the solution. The governing equation was an elliptic PDE and it satisfied the Compatibility condition. The numerical solution obtained from both these methods were compared with the analytical solution. Both the methods were checked for grid dependency by successive grid refinement. It was concluded that the solution is grid independent. The SOR method reached convergence in 4837 iterations whereas the ADI method reached the convergence criteria in 1286 iterations. Thus it was concluded that ADI method though requires two tridiagonal matrix solvers per iteration is faster than SOR. However, ADI method is computationally costly as compared to SOR. Comparing the perturbation plots and error plots, we can conclude that the solution obtained by numerical methods is of higher accuracy and we can efficiently use numerical methods to solve complex problems.

5. REFERENCES:

(1) Walter G. Vincenti (1959). Non-equilibrium flow over a wavy wall. Journal of Fluid Mechanics, 6, pp 481-496 doi:10.1017/S0022112059000775

(2) Hoffmann, K. A., & Chiang, S. T. (2000). {Computational fluid dynamics, Vol. 1}. Wichita, KS: Engineering Education System.

(3) Anderson, J. D. Computational Fluid Dynamics: The Basics with Applications. 1995. McGrawhill Inc.

(4) Boyd, J. P., & Flyer, N. (1999). Compatibility conditions for time-dependent partial differential equations and the rate of convergence of Chebyshev and Fourier spectral methods. Computer methods in applied mechanics and engineering, 175(3), 281-309.
(5) www.wikipedia.org
(6) http://inis.jinr.ru/sl/Simulation/Tannehill,_CFM_and_Heat_Transfer,2_ed/chap06.pdf
(7) http://www1.maths.leeds.ac.uk/~kersale/Teach/M3414/Notes/chap4.pdf
(8) http://web.stanford.edu/class/cs205b/lectures/lecture16.pdf
(9) http://www.mathworks.com/help/matlab/
(10) Class notes by Dr. Iman Borazjani

APPENDICES

A) Matlab Code for Point Gauss-Seidel with Successive Over Relaxation

```
%Point Gauss Seidel with SOR
close all; clear all; clc;
N=[40,50,60,70]; %Definition of Nodes for successive finer grid
for L=1:length(N)
    x=0:2*pi/N(L):2*pi;
    y=linspace(0,2,length(x));
    %Calculate the value of Analytical solution
    for j=1:length(y)
        for i=1:length(x)
            phi(j,i)=((-1*0.1)/(sqrt(1-(0.5.^2))))...
                *exp(-sqrt(1-(0.5^2))*y(j))*cos(x(i));
        end
    end
    [phi_min, phi_node]=min(min(abs(phi)));
    del_x(L)=x(2)-x(1);
    del_y=y(2)-y(1);
    B=(del_x(L))^2/((1-(0.5^2))*(del_y)^2);
    q=0;
    for omega=1.00
        q=q+1;
        Aold=zeros(length(y),length(x)); %Creating zero matrix for
intialization
        for k=1:10^7
            for j=1
                for i=1
                    Anew(j,i)=(-1/(2*(1+B)))*(-2*Aold(j,i+1)...
                        -2*B*Aold(j+1,i)+2*B*del_y*0.1*cos(x(1)));
                end
            end
            for j=1
                for i=2:length(x)-1
                    Anew(j,i)=(-1/(2*(1+B)))*(-Aold(j,i+1)...
                    -Anew(j,i-1)-2*B*Aold(j+1,i)+2*B*del_y*0.1*cos(x(i)));
                end
            end
            for j=1
                for i=length(x)
                    Anew(j,i)=(-1/(2*(1+B)))*(-2*Anew(j,i-1)...
                        -2*B*Aold(j+1,i)+2*B*del_y*0.1*cos(x(length(x))));
                end
            end
            for j=2:length(y)-1
                for i=1
                    Anew(j,i)=(-1/(2*(1+B)))*(-2*Aold(j,i+1)-B*Aold(j+1,i)...
                        -B*Anew(j-1,i));
                end
            end
            for j=length(y)
                for i=1
                    Anew(j,i)=(-1/(2*(1+B)))*(-2*Aold(j,i+1)-2*B*Anew(j-
1,i));
                end
```

```
            end
            for j=2:length(y)-1
                for i=2:length(x)-1
                    Anew(j,i)=(-1/(2*(1+B)))*(-Aold(j,i+1)-Anew(j,i-1)...
                           -B*Aold(j+1,i)-B*Anew(j-1,i));
                end
            end
            for j=2:length(y)-1
                for i=length(x)
                    Anew(j,i)=(-1/(2*(1+B)))*(-2*Anew(j,i-1)...
                        -B*Aold(j+1,i)-B*Anew(j-1,i));
                end
            end
            for j=length(y)
                for i=2:length(x)-1
                    Anew(j,i)=(-1/(2*(1+B)))*(-Aold(j,i+1)-Anew(j,i-1)...
                        -2*B*Anew(j-1,i));
                end
            end
            for j=length(y)
                for i=length(x)
                    Anew(j,i)=(-1/(2*(1+B)))*(-2*Anew(j,i-1)...
                        -2*B*Anew(j-1,i));
                end
            end
           Aconv=Anew(1,phi_node);
           for j=1:length(y)
               for i=1:length(x)
                   Anew(j,i)=Anew(j,i)-Aconv;
               end
           end

           %Now the SOR value is given as
           A_sor=((1-omega).*Aold)+(omega.*Anew);

           %Convergence criteria
           A_error=abs(A_sor-Aold);
           Error_a=(sum(sum(A_error)))/(sum(sum(abs(Aold))));

           if Error_a<(10^-5)
               break
           end
           Aold=A_sor;
       end
       itt(L)=k;
       figure;
       contourf(x,y,phi);
       %daspect([1,1,1]);
       grid on
       hold on
       [C H]=contour(x,y,A_sor,'-.g*');
       set(H,'linewidth',2);
       title('Analytical & Numerical Perturbation Potential
SOR','Fontsize',18);
       xlabel('Length','Fontsize',14);
       ylabel('Width','Fontsize',14);colorbar;
```

```
        hold off
        set(gca, 'XTickLabel',{'0', 'pi/4', 'pi/2',...
            '3pi/2', 'pi', '5pi/4', '3pi/2', '7pi/4', '2pi'});
    end
    Err=(A_sor-phi).^2;
    Error(L)=((1/(length(x)*length(y)))*(sqrt(sum(Err(:)))));
end
figure;
plot(log(del_x),log(Error))
title('Error Plot SOR','Fontsize',18);
xlabel('log(del_x)','Fontsize',14); ylabel('log(Error)','Fontsize',14);
figure;
contourf(x,y,phi);
%daspect([1,1,1]);
title('Analytical Perturbation Potential SOR','Fontsize',18);
xlabel('Length','Fontsize',14);
ylabel('Width','Fontsize',14);colorbar;
set(gca, 'XTickLabel',{'0', 'pi/4', 'pi/2',...
    '3pi/2', 'pi', '5pi/4', '3pi/2', '7pi/4', '2pi'});
figure;
contour(x,y,A_sor);
%daspect([1,1,1]);
title('Numerical Perturbation Potential SOR','Fontsize',18);
xlabel('Length','Fontsize',14);
ylabel('Width','Fontsize',14);colorbar;
set(gca, 'XTickLabel',{'0', 'pi/4', 'pi/2',...
    '3pi/2', 'pi', '5pi/4', '3pi/2', '7pi/4', '2pi'});
%For velocity plots
for c=1:length(y)
    u(c,1)=0;
    u(c,length(x))=0;
    for d=2:length(x)-1
        u(c,d)=(A_sor(c,d+1)-A_sor(c,d-1))/(2*del_x(L));
    end
end
for c=1
    for r=1:length(x)
        v(c,r)=1*0.1*cos(x(d));
    end
end
for c=2:length(y)-1
    for r=1:length(x)
        v(c,r)=(A_sor(c+1,r)-A_sor(c-1,r))./(2*del_y);
    end
end
for c=length(y)
    for r=1:length(x)
        v(c,r)=0;
    end
end
figure;
quiver(x,y,u,v,2); %Vector plot for velocities
title('Numerical Velocity Vector Plot SOR','Fontsize',18)
xlabel('Length','Fontsize',14);
ylabel('Width','Fontsize',14);
set(gca, 'XTickLabel',{'0', 'pi/4', 'pi/2',...
    '3pi/2', 'pi', '5pi/4', '3pi/2', '7pi/4', '2pi'});
```

```
figure;
contourf(x,y,u)
%daspect([1,1,1]);
title('Numerical Velocity "u" Contour Plot SOR','Fontsize',18)
xlabel('Length','Fontsize',14);
ylabel('Width','Fontsize',14);colorbar;
set(gca, 'XTickLabel',{'0', 'pi/4', 'pi/2',...
    '3pi/2', 'pi', '5pi/4', '3pi/2', '7pi/4', '2pi'});
figure;
contourf(x,y,v)
%daspect([1,1,1]);
title('Numerical Velocity "v" Contour Plot SOR','Fontsize',18)
xlabel('Length','Fontsize',14);
ylabel('Width','Fontsize',14);colorbar;
set(gca, 'XTickLabel',{'0', 'pi/4', 'pi/2',...
    '3pi/2', 'pi', '5pi/4', '3pi/2', '7pi/4', '2pi'});
```

B) Matlab Code for Alternating Direction Implicit Method

```
%Alternating Direction Implicit
close all; clear all; clc;
q=0;
N=[40,50,60,70]; %Definition of nodes for successive finer grid
for L=1:length(N)
    q=q+1;
    x=0:2*pi/N(L):2*pi;
    y=linspace(0,2,length(x));
    A_new_1=zeros(length(y),length(x));
    A_new_2=zeros(length(y),length(x));
    %Calculating the Analytical Solution
    for j=1:length(y)
        for i=1:length(x)
            phi(j,i)=((-1*0.1)/(sqrt(1-(0.5.^2))))...
                *exp(-sqrt(1-(0.5.^2)).*y(j)).*cos(x(i));
        end
    end
    [phi_min, phi_node]=min(min(abs(phi)));
    del_x(L)=x(2)-x(1);
    del_y=y(2)-y(1);
    Beta=(del_x(L))^2/((1-(0.5^2))*(del_y)^2);
    A_old=zeros(length(y),length(x));
    % Perform the X-Sweep
    %Initialization
    A=zeros(1,length(x));
    B=zeros(1,length(x));
    C=zeros(1,length(x));
    D=zeros(1,length(x));
    for k=1:10^5
        for j=1
            A(1)=0; B(1)=-2*(1+Beta); C(1)=2;
            D(1)=-Beta*(2*A_old(j+1,1)-2*del_y*0.1*cos(x(1)));
            for i=2:length(x)-1
                A(i)=1; B(i)=-2*(1+Beta); C(i)=1;
                D(i)=-Beta*(2*A_old(j+1,i)-2*del_y*0.1*cos(x(i)));
```

```
    end
    A(length(x))=2; B(length(x))=-2*(1+Beta); C(length(x))=0;
    D(length(x))=-Beta*(2*A_old(j+1,length(x))...
        -2*del_y*0.1*cos(x(length(x))));
    A_new_1(1,:)=trisol(1,length(x),A,B,C,D);
end
for j=2:length(y)-1
    A(1)=0; B(1)=-2*(1+Beta); C(1)=2;
        D(1)=-Beta*(A_old(j+1,1)+A_new_1(j-1,1));
    for i=2:length(x)-1
        A(i)=1; B(i)=-2*(1+Beta); C(i)=1;
            D(i)=-Beta*(A_old(j+1,i)+A_new_1(j-1,i));
    end
    A(length(x))=2; B(length(x))=-2*(1+Beta); C(length(x))=0;...
    D(length(x))=-Beta*(A_old(j+1,length(x))+A_new_1(j-1,length(x)));
    A_new_1(j,:)=trisol(1,length(x),A,B,C,D);
end
for j=length(y)
    A(1)=0; B(1)=-2*(1+Beta); C(1)=2; D(1)=-Beta*(2*A_new_1(j-1,1));
    for i=2:length(x)-1
        A(i)=1; B(i)=-2*(1+Beta); C(i)=1;
            D(i)=-Beta*(2*A_new_1(j-1,i));
    end
    A(length(x))=2; B(length(x))=-2*(1+Beta); C(length(x))=0;
        D(length(x))=-Beta*(2*A_new_1(j-1,length(x)));
    A_new_1(j,:)=trisol(1,length(x),A,B,C,D);
end
%Perform the Y-Sweep
%Initialization
A=zeros(1,length(y));
B=zeros(1,length(y));
C=zeros(1,length(y));
D=zeros(1,length(y));
for i=1
    A(1)=0; B(1)=-2*(1+Beta); C(1)=2*Beta;
        D(1)=-(2*A_new_1(1,i+1))+(2*Beta*del_y*0.1*cos(x(i)));
    for j=2:length(y)-1
        A(j)=Beta; B(j)=-2*(1+Beta); C(j)=Beta;
            D(j)=-2*(A_new_1(j,i+1));
    end
    A(length(y))=2*Beta; B(length(y))=-2*(1+Beta); C(length(y))=0;
        D(length(y))=-2*A_new_1(length(y),i+1);
    A_new_2(:, 1)=trisol(1,length(y),A,B,C,D);
end
for i=2:length(x)-1
    A(1)=0; B(1)=-2*(1+Beta); C(1)=2*Beta;
  D(1)=-(A_new_1(1,i+1)+A_new_2(1,i-1))+2*Beta*del_y*0.1*cos(x(i));
  for j=2:length(y)-1
      A(j)=Beta; B(j)=-2*(1+Beta); C(j)=Beta;
      D(j)=-(A_new_1(j,i+1)+A_new_2(j,i-1));
  end
  A(length(y))=2*Beta; B(length(y))=-2*(1+Beta); C(length(y))=0;
  D(length(y))=-(A_new_1(length(y),i+1)+A_new_2(length(y),i-1));
  A_new_2(:,i)=trisol(1,length(y),A,B,C,D);
end
for i=length(x)
    A(1)=0; B(1)=-2*(1+Beta); C(1)=2*Beta;
```

```
            D(1)=-(2*A_new_2(1,i-1))+2*Beta*0.1*del_y*cos(x(i));
            for j=2:length(y)-1
                A(j)=Beta; B(j)=-2*(1+Beta); C(j)=Beta;
                D(j)=-(2*A_new_2(j,i-1));
            end
            A(length(y))=2*Beta; B(length(y))=-2*(1+Beta); C(length(y))=0;
            D(length(y))=-(2*A_new_2(length(y),i-1));
            A_new_2(:,i)=trisol(1,length(y),A,B,C,D);
        end
        Aconv=A_new_2(1,phi_node);
        for j=1:length(y)
            for i=1:length(x)
                A_new_2(j,i)=A_new_2(j,i)-Aconv;
            end
        end
        %Convergence criteria
           A_error=abs(A_new_2-A_old);
           Error_a=(sum(sum(A_error)))/(sum(sum(abs(A_old))));

           if Error_a<(10^-5)
              break
           end
           A_old=A_new_2;
    end
    figure;
    contourf(x,y,phi);
    %daspect([1,1,1]);
    grid on
    hold on
    [c h]=contour(x,y,A_new_2,'-.g*');
    set(h,'Linewidth',3);
    hold off
    title('Analytical and Numerical Perturbation Potential
ADI','Fontsize',20);
    xlabel('Length','Fontsize',14); ylabel('Width','Fontsize',14); colorbar;
    set(gca, 'XTickLabel',{'0', 'pi/4', 'pi/2',...
    '3pi/2', 'pi', '5pi/4', '3pi/2', '7pi/4', '2pi'});
    Err=(A_new_2-phi).^2;
    Error(L)=((1/(length(x)*length(y)))*(sqrt(sum(Err(:)))));
    itt(q)=k;
end
plot(log10(del_x),log10(Error));
title('Error Plot','Fontsize',20);
xlabel('log(del_x)','Fontsize',14); ylabel('log(Error)','Fontsize',14);
%Velocity Plots
for c=1:length(y)
    u(c,1)=0;
    u(c,length(x))=0;
    for d=2:length(x)-1
        u(c,d)=(A_new_2(c,d+1)-A_new_2(c,d-1))/(2*del_x(L));
    end
end
for c=1;
    for r=1:length(x)
        v(c,r)=1*0.1*cos(x(r));
    end
end
```

```
for c=2:length(y)-1
    for r=1:length(x)
        v(c,r)=(A_new_2(c+1,r)-A_new_2(c-1,r))/(2*del_y);
    end
end
for c=length(y)
    for r=1:length(x)
        v(c,r)=0;
    end
end
figure;
quiver(x,y,u,v,2); %Vector plot for velocities
title('Numerical Velocity Vector Plot ADI','Fontsize',20)
xlabel('Length','Fontsize',14);
ylabel('Width','Fontsize',14);
set(gca, 'XTickLabel',{'0', 'pi/4', 'pi/2',...
    '3pi/2', 'pi', '5pi/4', '3pi/2', '7pi/4', '2pi'});
figure;
contourf(x,y,u)
%daspect([1,1,1]);
title('Numerical Velocity "u" Contour Plot ADI','Fontsize',20)
xlabel('Length','Fontsize',14);
ylabel('Width','Fontsize',14);colorbar;
set(gca, 'XTickLabel',{'0', 'pi/4', 'pi/2',...
    '3pi/2', 'pi', '5pi/4', '3pi/2', '7pi/4', '2pi'});
figure;
contourf(x,y,v)
%daspect([1,1,1]);
title('Numerical Velocity "v" Contour Plot ADI','Fontsize',20)
xlabel('Length','Fontsize',14);
ylabel('Width','Fontsize',14);colorbar;
set(gca, 'XTickLabel',{'0', 'pi/4', 'pi/2',...
    '3pi/2', 'pi', '5pi/4', '3pi/2', '7pi/4', '2pi'});
```

C) Matlab Code for Tridiagonal Solver using Thomas' Algorithm

```
function ans=trisol(istart,iend,A,B,C,D)
%Tridiagonal Matrix Sover using Thomas Algorithm (Ref-CFD by Anderson)
%Gaussian Elimination
for i=istart+1:iend
    M=A(i)/B(i-1);
    B(i)=B(i)-M*C(i-1);
    D(i)=D(i)-M*D(i-1);
end
%Backward Substitution Step
D(iend)=D(iend)/B(iend);
for i=iend-1:-1:istart
    D(i)=(D(i)-C(i)*D(i+1))/B(i);
end
ans=D;
```