2-D Poisson's Equation

Project Code: Apc2-6

Scientific Computing for Mechanical Engineers: MECE 5397

May 3, 2017

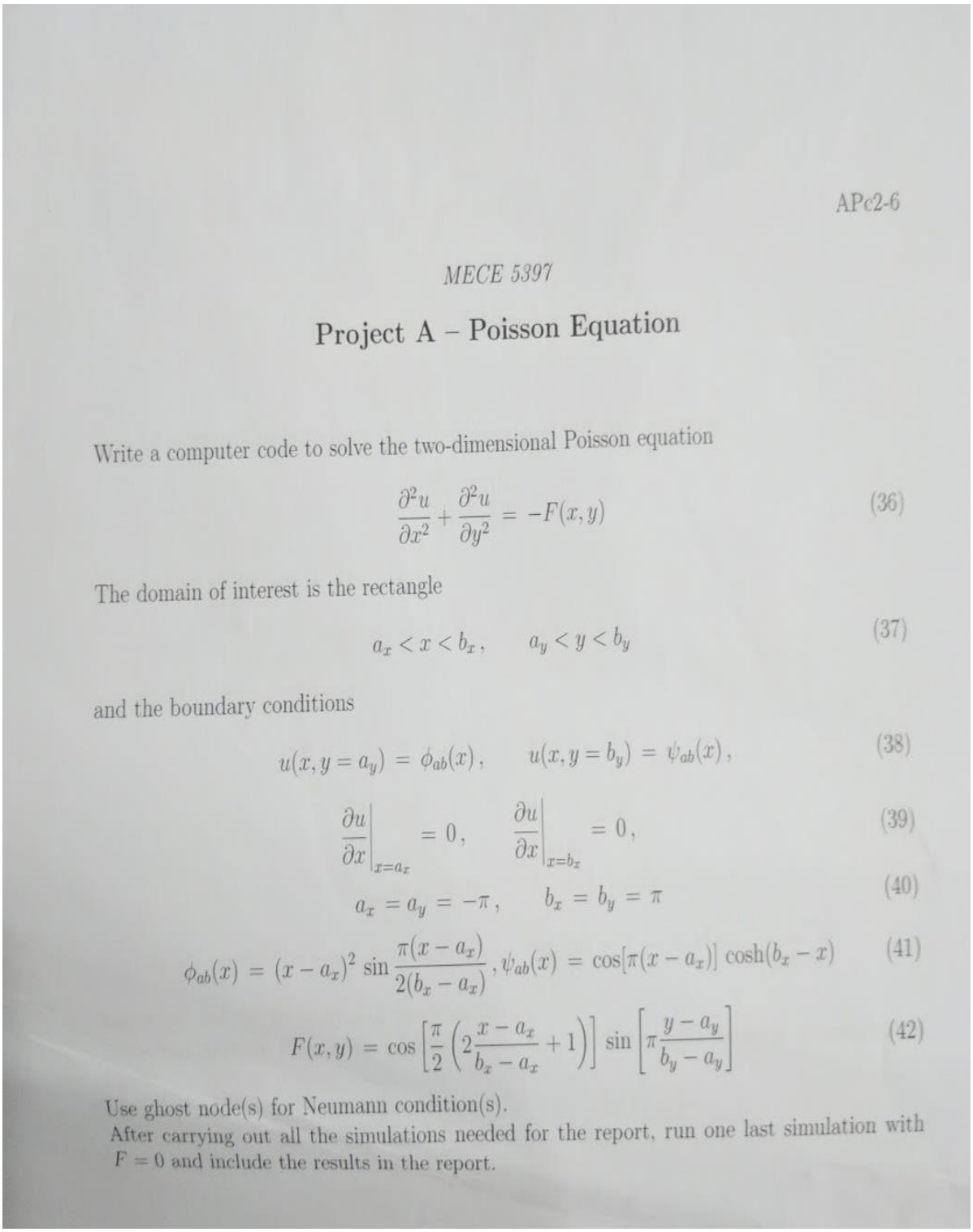
Author Name: Jose Chavez

University of Houston

Abstract

This report contains computational methods in solving a 2-D Poisson partial differential equation by using centered difference discretization indices using MATLAB programming. The purpose of this project is to gain experience writing scientific code using Dirichlet and Neumann boundary conditions to solve the given equations and explore interest in partial differential equations. A mesh grid 3-D surface plot will illustrate how boundary conditions will behave with given equations and include some amount of discretization error. Discretization can help improve performance of algorithms in transforming continuous values of a variable to discrete ones. This has important implications for the analysis of high dimensional data derived from experiments. Thus, numerical iterative methods such as Gauss Seidel and Gauss elimination can be used to solve diagonally dominant linear systems of equations. The report includes the Gauss Seidel method and Gauss Seidel method with successive over-relaxation for values of 2-D dimensional Poisson Equation to converge rapidly. The MATLAB Script considers all the conditions provided in the problem statement which is displayed below in Figure 1.

Mathematical Statement



APc2-6

MECE 5397

Project A – Poisson Equation

Write a computer code to solve the two-dimensional Poisson equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -F(x,y) \tag{36}$$

The domain of interest is the rectangle

$$a_x < x < b_x\,, \qquad a_y < y < b_y \tag{37}$$

and the boundary conditions

$$u(x, y = a_y) = \phi_{ab}(x)\,, \qquad u(x, y = b_y) = \psi_{ab}(x)\,, \tag{38}$$

$$\left.\frac{\partial u}{\partial x}\right|_{x=a_x} = 0\,, \qquad \left.\frac{\partial u}{\partial x}\right|_{x=b_x} = 0\,, \tag{39}$$

$$a_x = a_y = -\pi\,, \qquad b_x = b_y = \pi \tag{40}$$

$$\phi_{ab}(x) = (x - a_x)^2 \sin\frac{\pi(x - a_x)}{2(b_x - a_x)}\,, \psi_{ab}(x) = \cos[\pi(x - a_x)]\cosh(b_x - x) \tag{41}$$

$$F(x,y) = \cos\left[\frac{\pi}{2}\left(2\frac{x - a_x}{b_x - a_x} + 1\right)\right]\sin\left[\pi\frac{y - a_y}{b_y - a_y}\right] \tag{42}$$

Use ghost node(s) for Neumann condition(s).
After carrying out all the simulations needed for the report, run one last simulation with $F = 0$ and include the results in the report.

Figure 1: Mathematical Problem Statement

Discretized Versions of Equations

The following discretized equations were obtained by using second derivative and centered difference formulas in relating the Poisson general form equation to obtain the Neumann and General $U_{i,j}$ equations as shown in Figure 2 and Figure 3. Ghost Nodes were applied in using nodes that are not within the grid to relate unknown U values in the range of the given conditions. Figure 3 contains the working process in using ghost nodes to obtain the U values included in the grid. Thus, the equations for Neumann Boundary conditions are applied in the "left" and "right" side of the rectangle. In this problem statement, the condition to use the centered difference of the derivative at the point is in respect to the change of X. The given Dirichlet boundary conditions are applied along the X axis for the "top" and "bottom" of the rectangle. With all the equations solved for specific regions of the rectangle, a pseudocode is to write the ideas in what the script file will contain by using numerical methods in solving the 2-D poisson's equation under two Dirichlet condtions and two Neumann conditions.

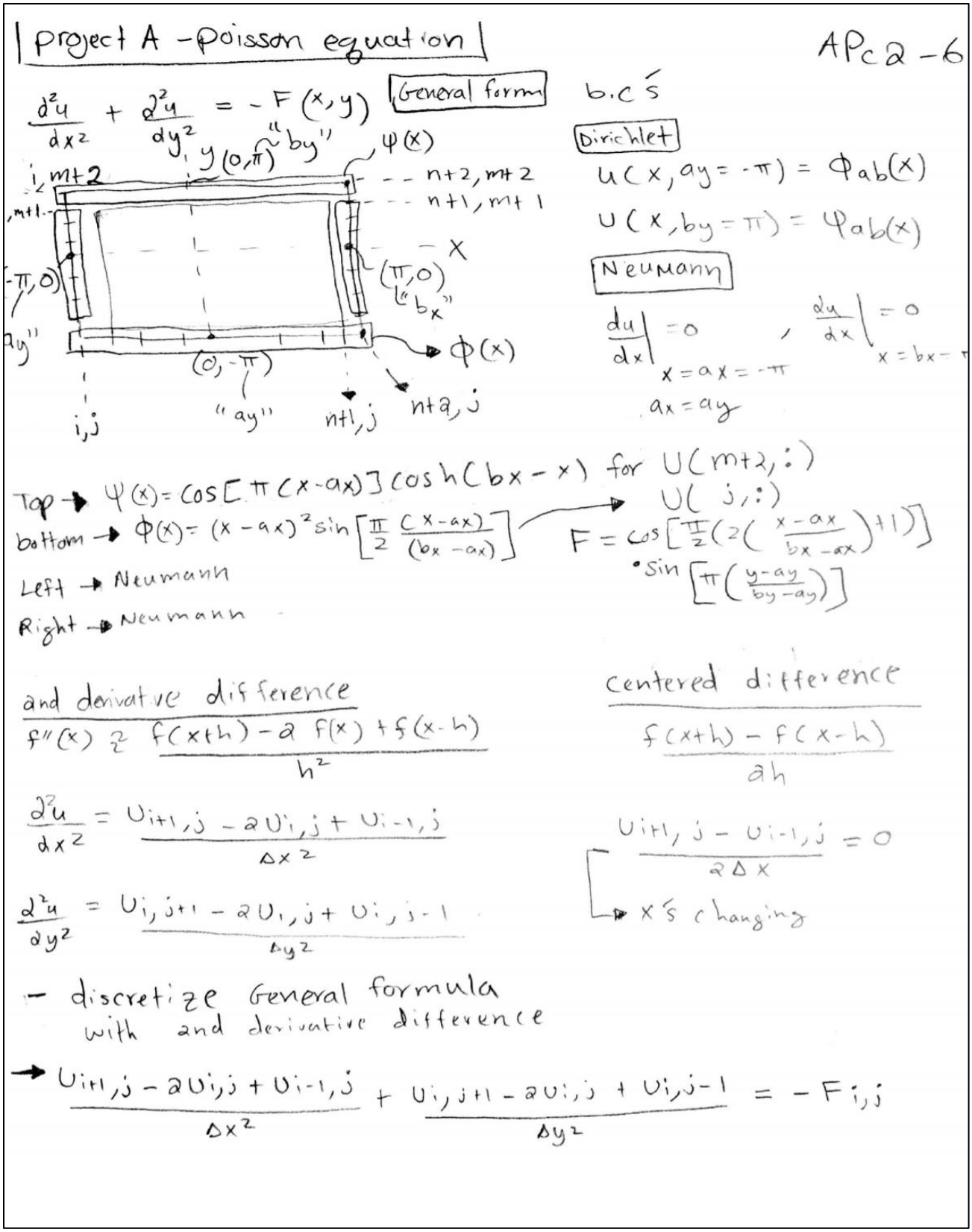


Figure 2: Mathematical Approach

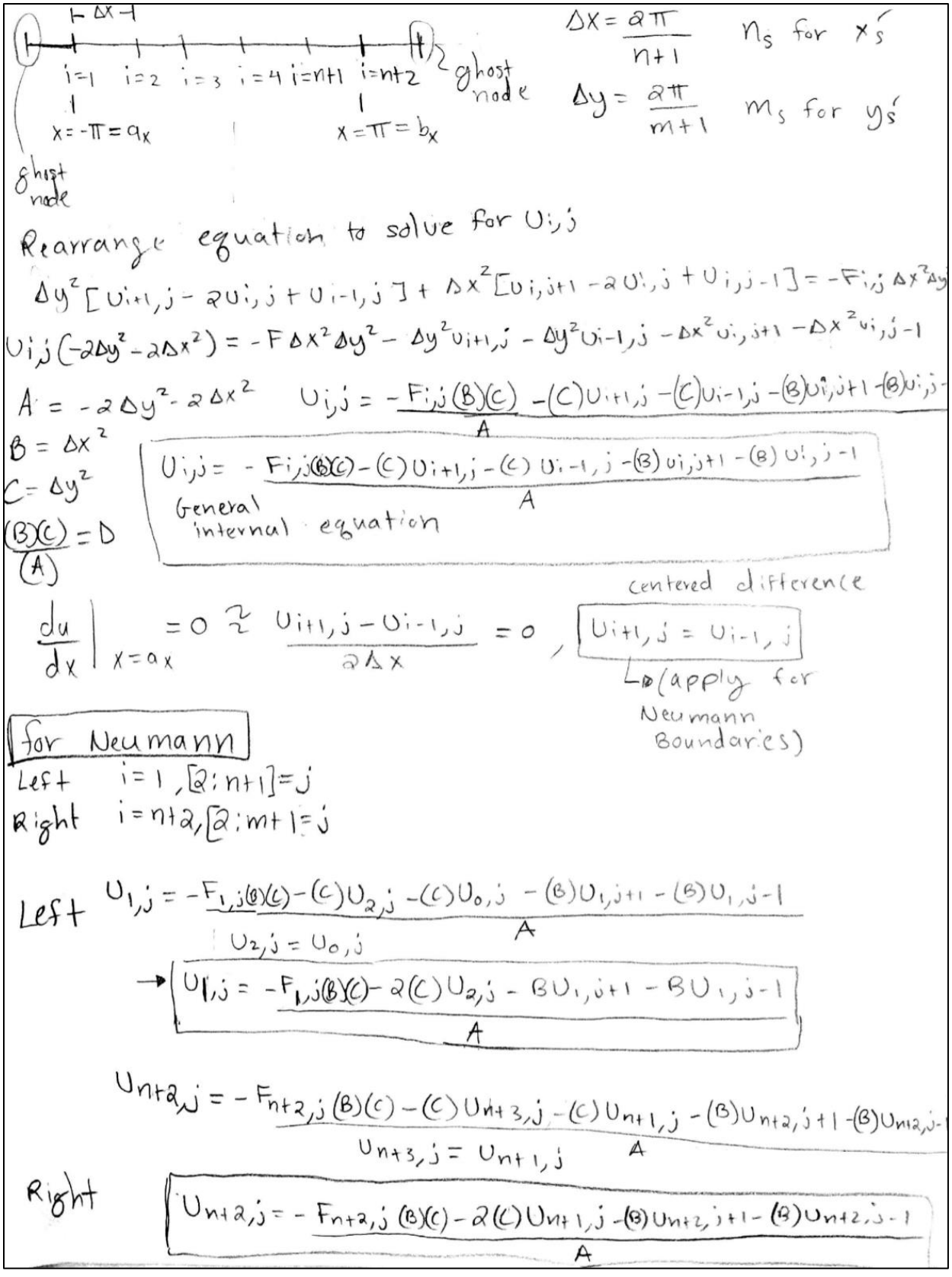


Figure 3: Discretized Versions of Equations

Description of The Numerical Methods

Numerical methods are techniques by which mathematical problems are formulated so that they can be solved with arithmetic operations. In many problems, this implies producing a sequence of approximations by repeating the procedure. The rate of convergence, accuracy, and completeness of the response are the following issues in solving problems. Model applications can be solved using appropriate numerical methods and software, and can determine the values of some of its major parameters. The two methods used in the project are the Gauss-Seidel and the Successive Over Relaxation methods.

The Gauss-Seidel approach is an iterative method that computes values of the next iteration using elements from that new iteration after guessing the initial iteration. The Successive Over Relaxation method is much faster in computing than the Gauss-Seidel method. The method uses the same Gauss-Seidel equation, but includes a constant factor λ. When $\lambda = 1$, the result is the Gauss-Seidel method, but when the value is within a range between $1 < \lambda < 2$ the solutions will ultimately converge faster. It can be seen in Table 2; the rate is increased by using the Over Relaxation Method by setting a value for λ equal to 1.5.

Below is the following Pseudocode for the following problem statement.

Pseudocode:

- Define given values and conditions
- Input N and M values for number of internal nodes
- Set stopwatch tic
- Set incremental values of X and Y
- Initialize guess for all unknown values
- Input all X and Y into F, Phi, and Psi equation

- Index the Phi and Psi equation
- Initialize guess for error
- Set iteration to zero
- Use while loop with an error estimate that will reevaluate solutions for U until it converges onto solution satisfies error parameters.
- Set an iteration counter to take note and time to complete convergence for each method and display visuals for comparison and grid convergence study.
- Provide checkpointing within code.

Technical Specifications of The Computer Used

The code was written on a personal computer tablet used for portable applications. The specifications are shown below for the Surface Pro 4.

Table 1.

CPU	Intel(R) Core(TM) i5-6300U CPU @ 2.40Ghz 2.50 Ghz
RAM	4.00 GB
Graphics	Intel (R) HD Graphics 520
Storage	128 GB Samsung PM951 SSD
Audio	Realtek High Defintion Audio (SST)

All the specification determined the amount of time and iterations computed by running MATLAB, which can found in Table 2.

Results

In all simulations, the parameters used were $a_x = a_y = -\pi$ & $b_x = b_y = \pi$. The following simulations are broken down in a way to compare the time elapsed and total iterations by using

Gauss Seidel and Gauss Seidel with Successive Over Relaxation for each case with different N & M internal node points as shown in Figure(s) 4,5,6,7,8,9,10…..17.

1.	F= $\cos[\frac{\pi}{2}(2\frac{x-ax}{bx-ax}+1)]\sin[\pi\frac{y-ay}{by-ay}]$	U=0	Give Function
2.	F= -6	U=1+x^2+2y^2	Method of Manufacturing Equation
3.	F=0	U=0	Last Simulation

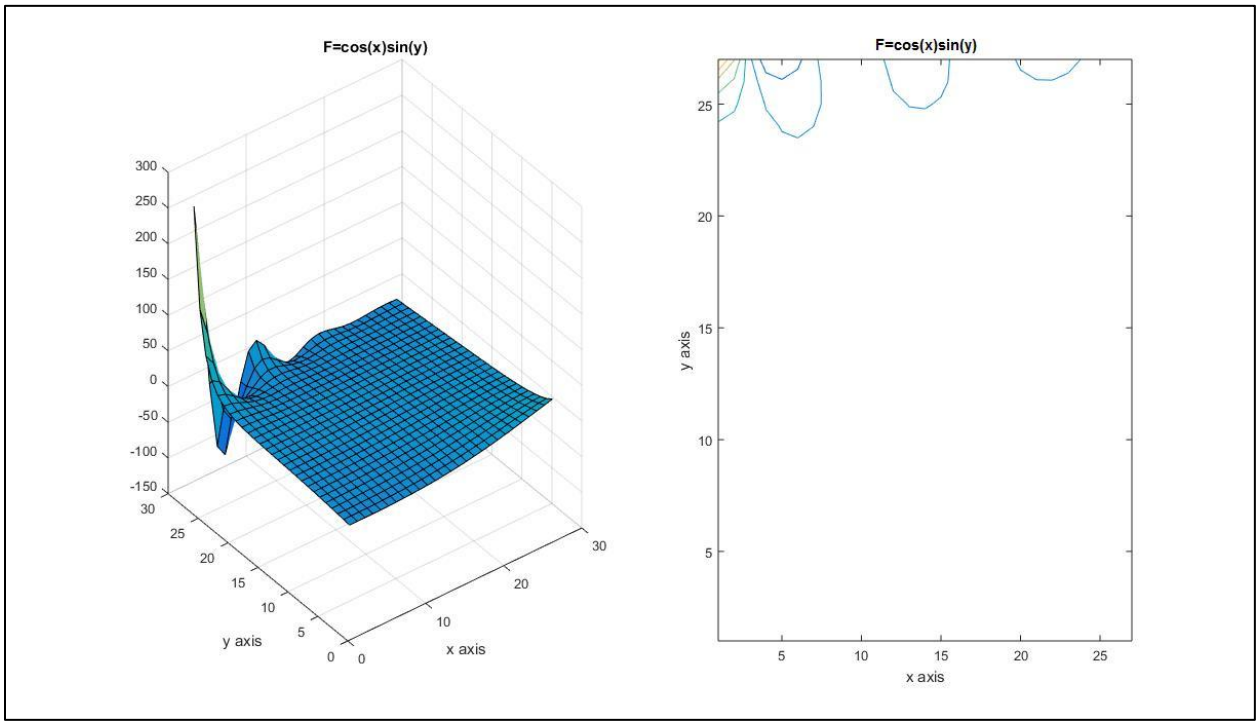


Figure 4: Gauss Seidel 25 x 25 Simulation Case 1.

Figure 4 shows the 3-D mesh grid of the Gauss Seidel Method with the given F equation.

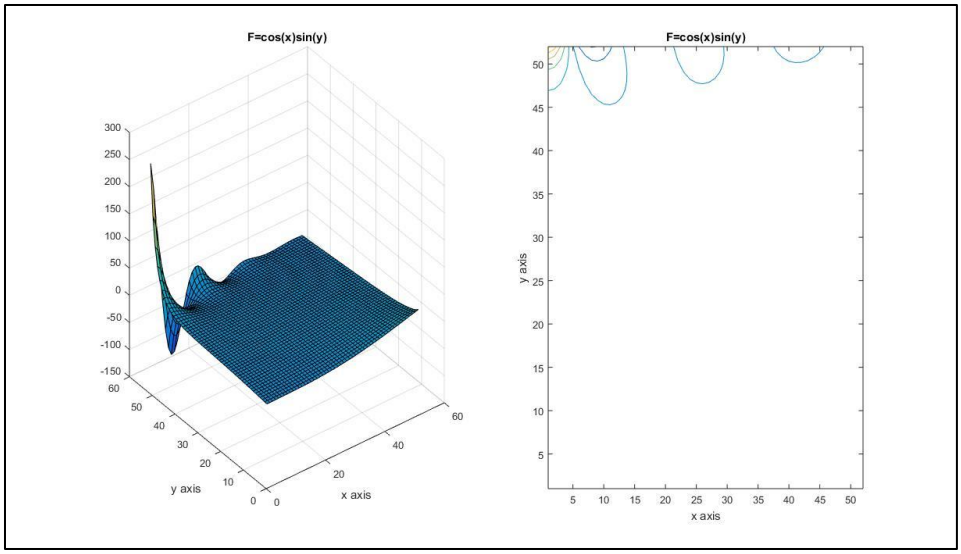


Figure 5: Gauss Seidel 50 x 50 Simulation Case 1.

More grid points generate a smoother 3-D mesh of the Gauss Seidel method in Figure 5, but it can be seen in the contour requires more points to have a better approximation.

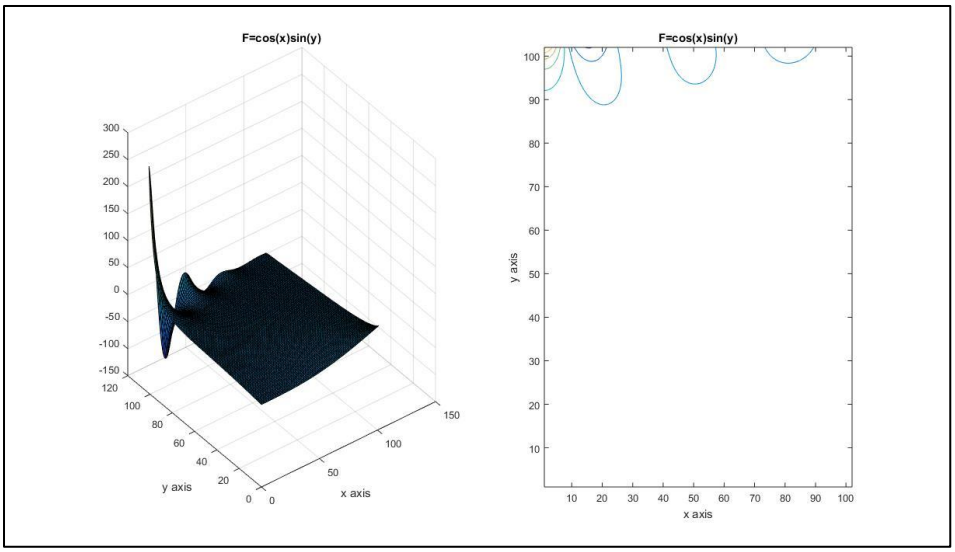


Figure 6: Gauss Seidel 100 x 100 Simulation Case 1.

Figure 6 contains Twice the number of points of Figure 5 and produced the smoothest finer curve.

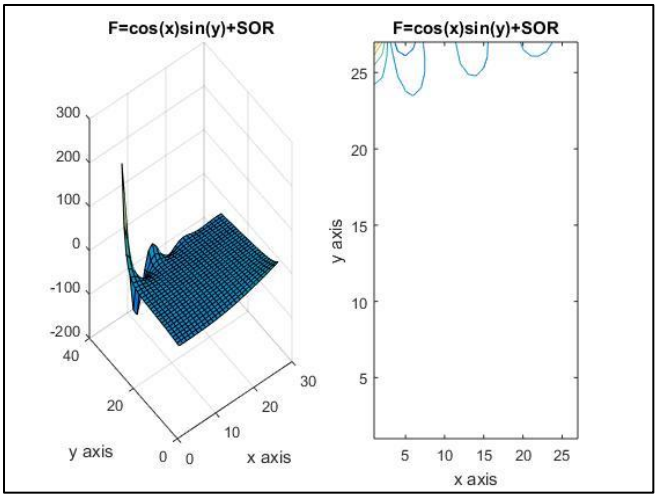


Figure 7: Gauss Seidel SOR 25 x25 Simulation Case 1.

The Gauss Seidel Successive Over Relaxation (SOR) method is used in this simulation. The SOR method allows the iterative numerical method to converge rapidly by introducing a multiplier λ to be multiplied with the old value get this from the notes. The following Figures repeats the simulation with the Gauss Seidel Method and Successive Over Relaxation method for Figure(s) 9, 10, 11, 12...17.

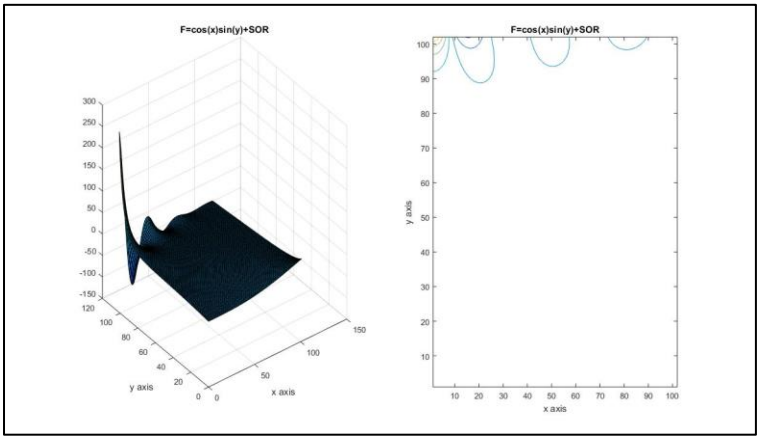


Figure 9: Gauss Seidel SOR 100 x 100 Simulation Case 1.

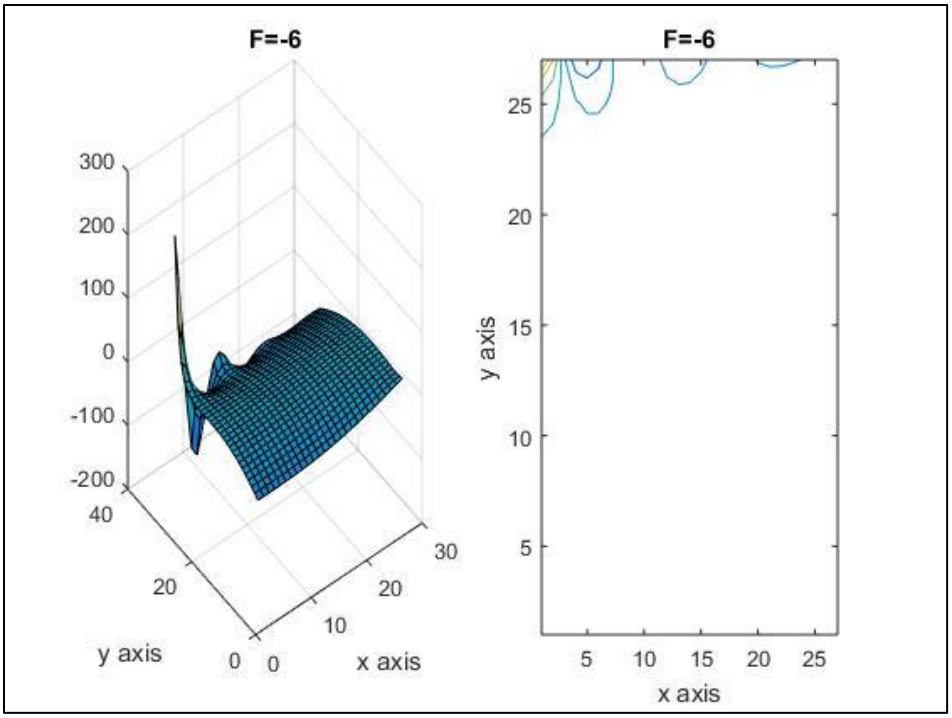


Figure 10: Gauss Seidel 25 x 25 Simulation Case 2.

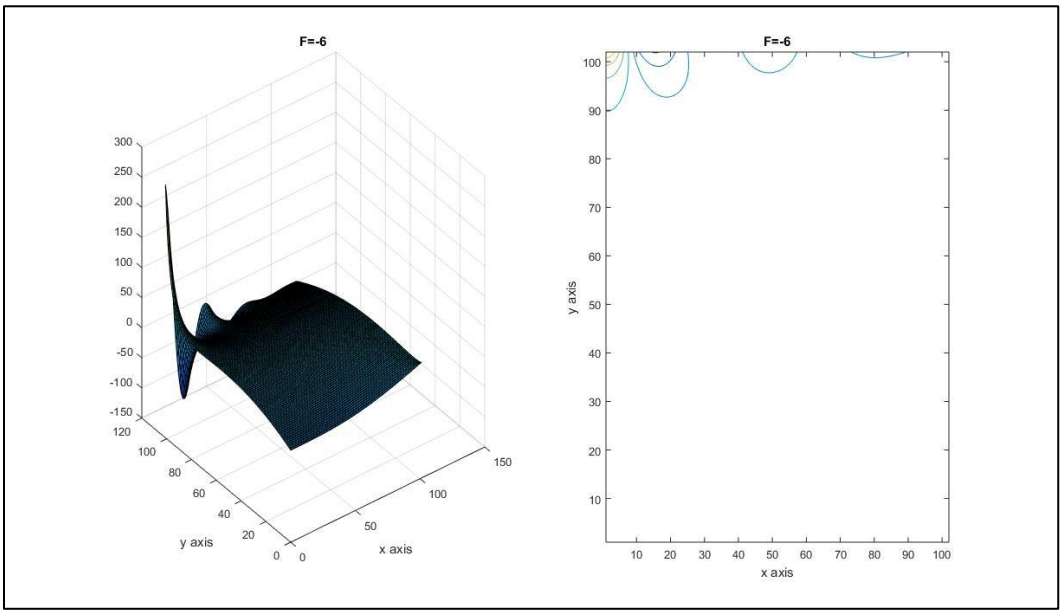


Figure 11: Gauss Seidel 100 x 100 Simulation Case 2.

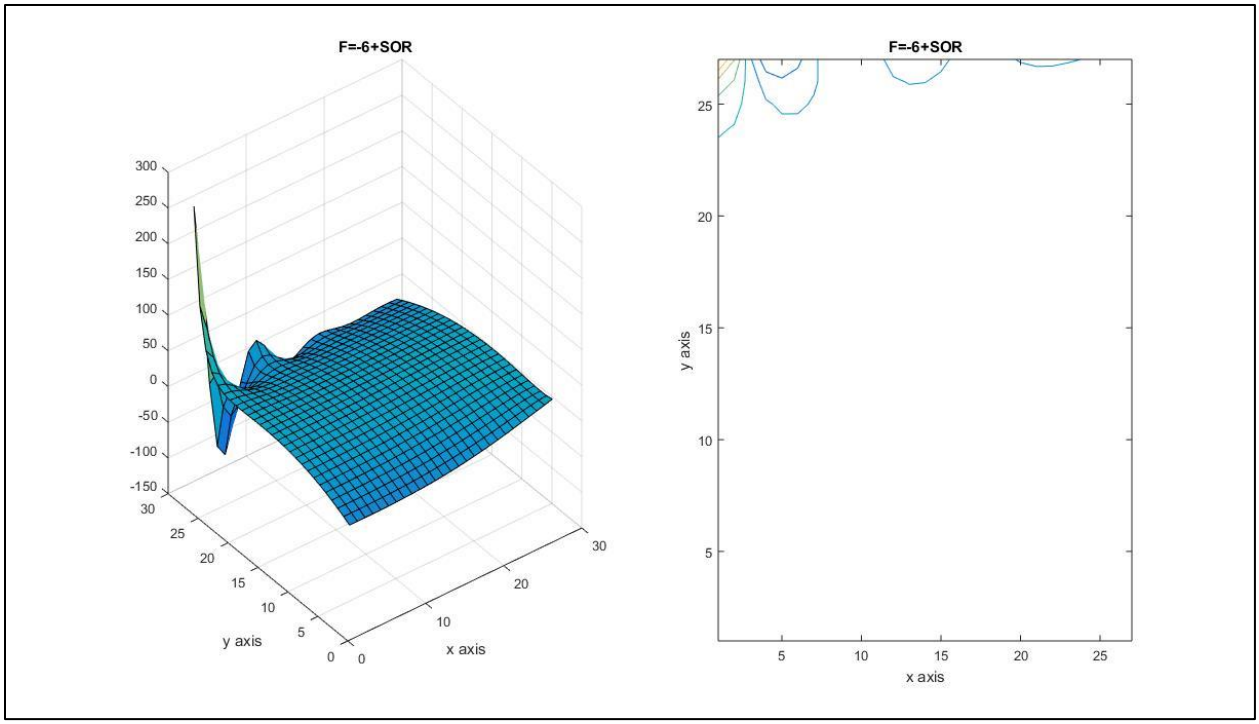


Figure 12: Gauss Seidel SOR 25 x 25 Simulation Case 2.

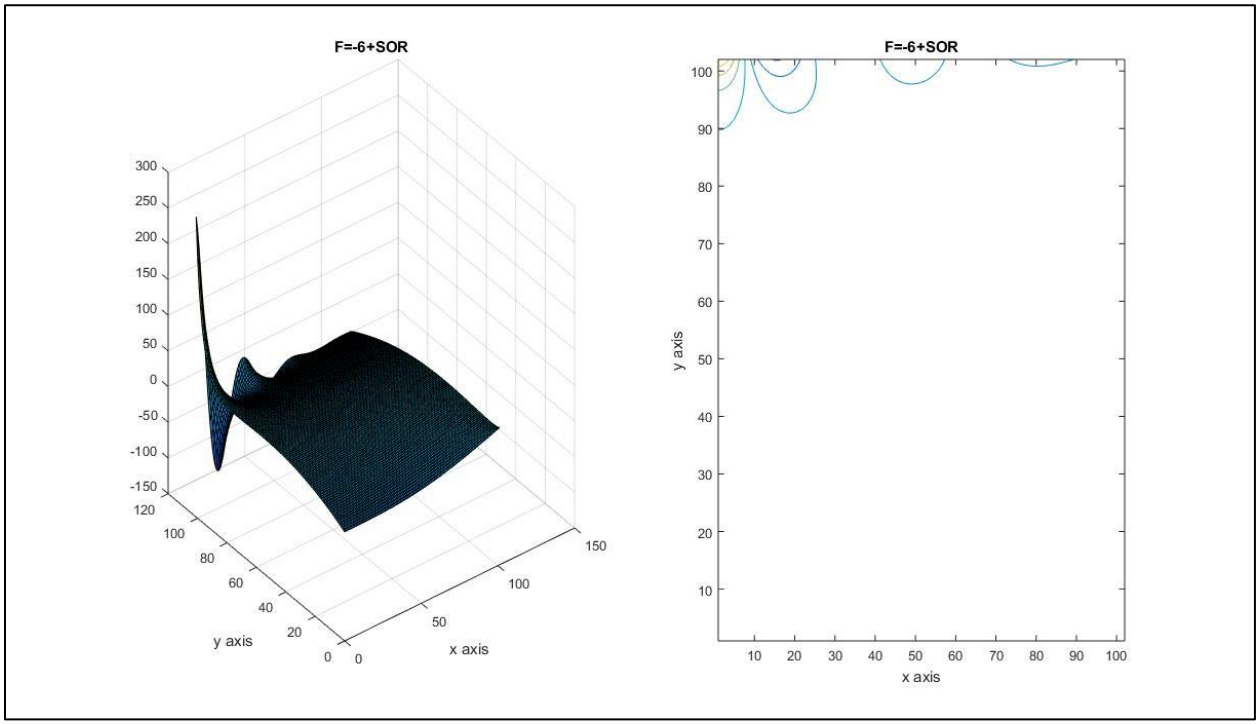


Figure 13: Gauss Seidel SOR 100 x 100 Simulation Case 2.

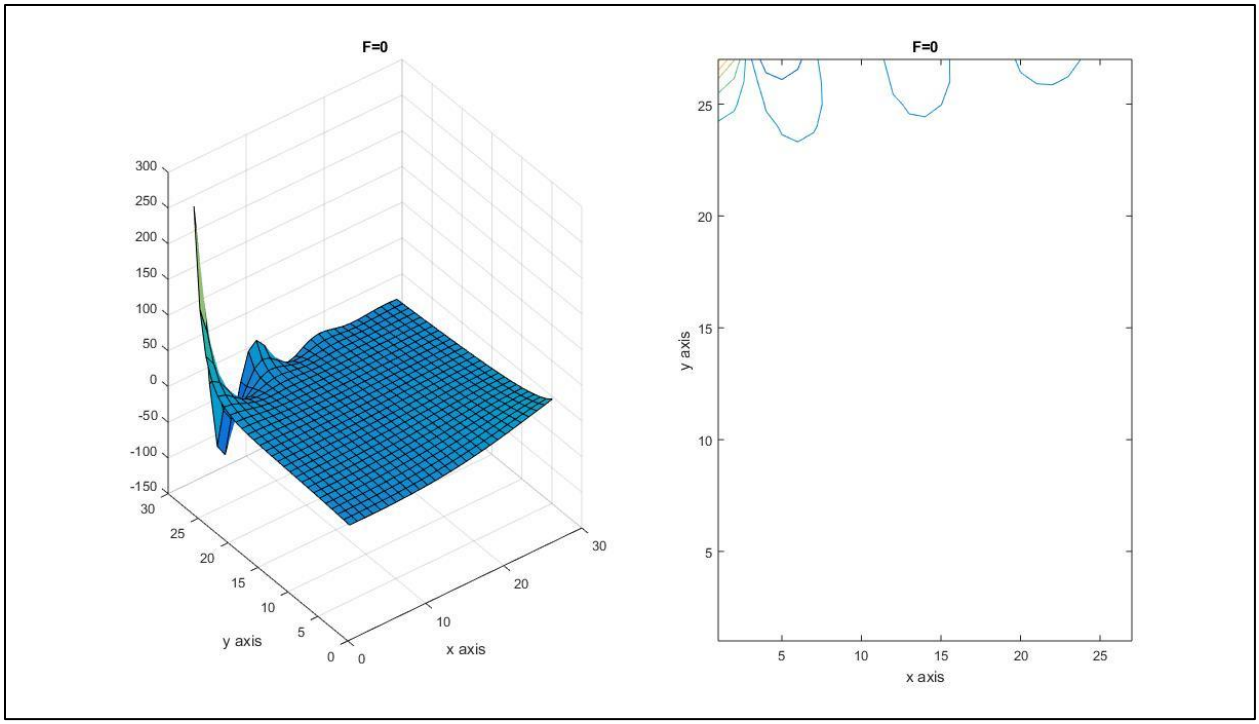


Figure 14: Gauss Seidel 25 x 25 Simulation Case 3.

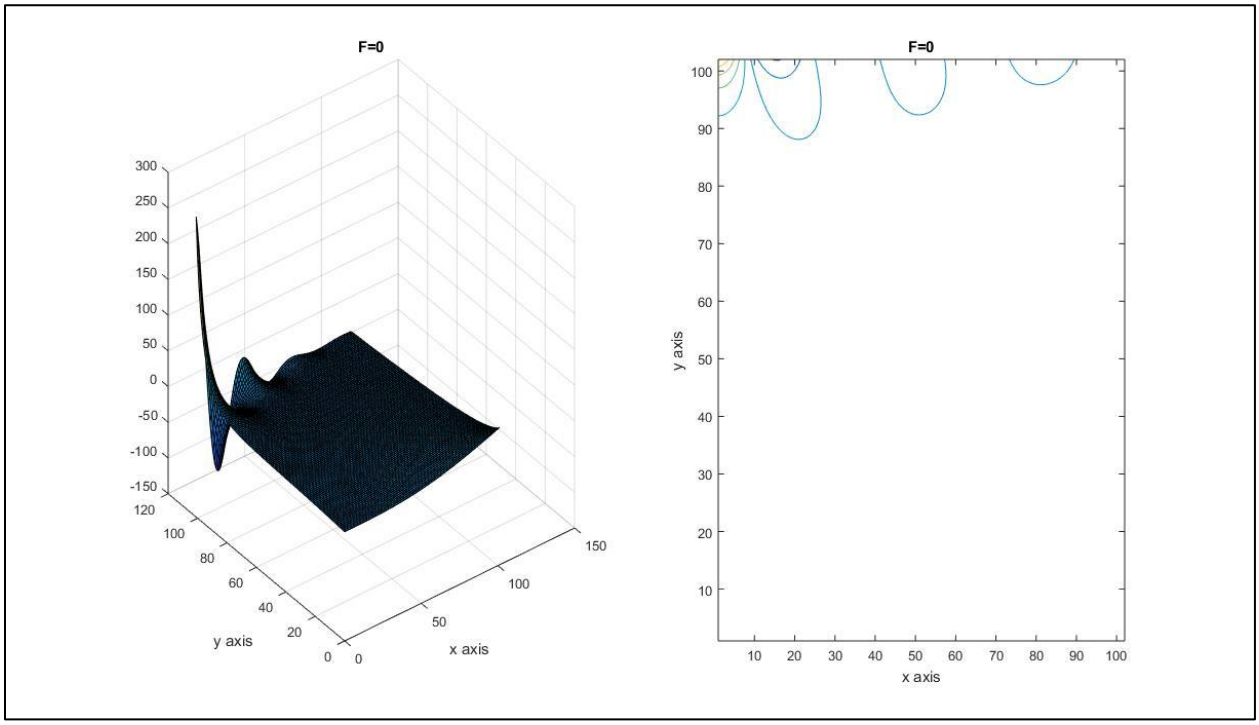


Figure 15: Gauss Seidel 100 x 100 Simulation Case 3.

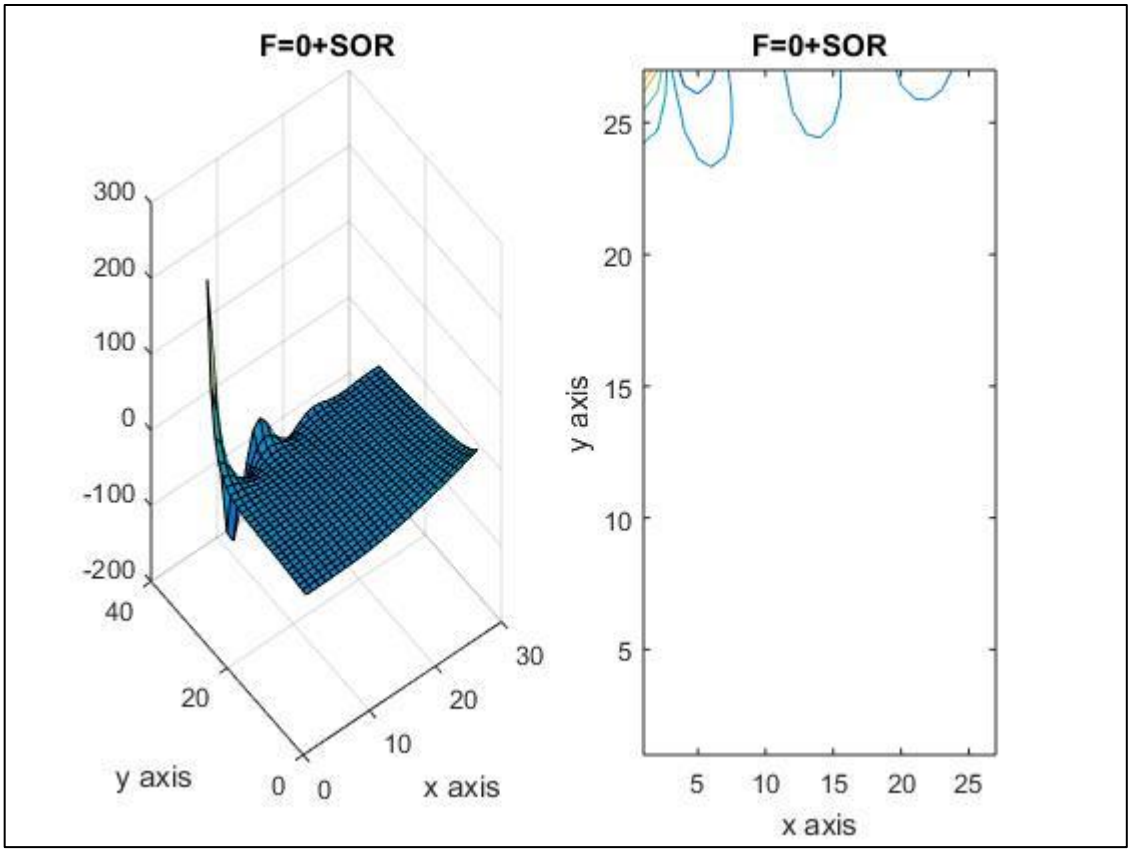


Figure 16: Gauss Seidel SOR 25 x 25 Simulation Case 3

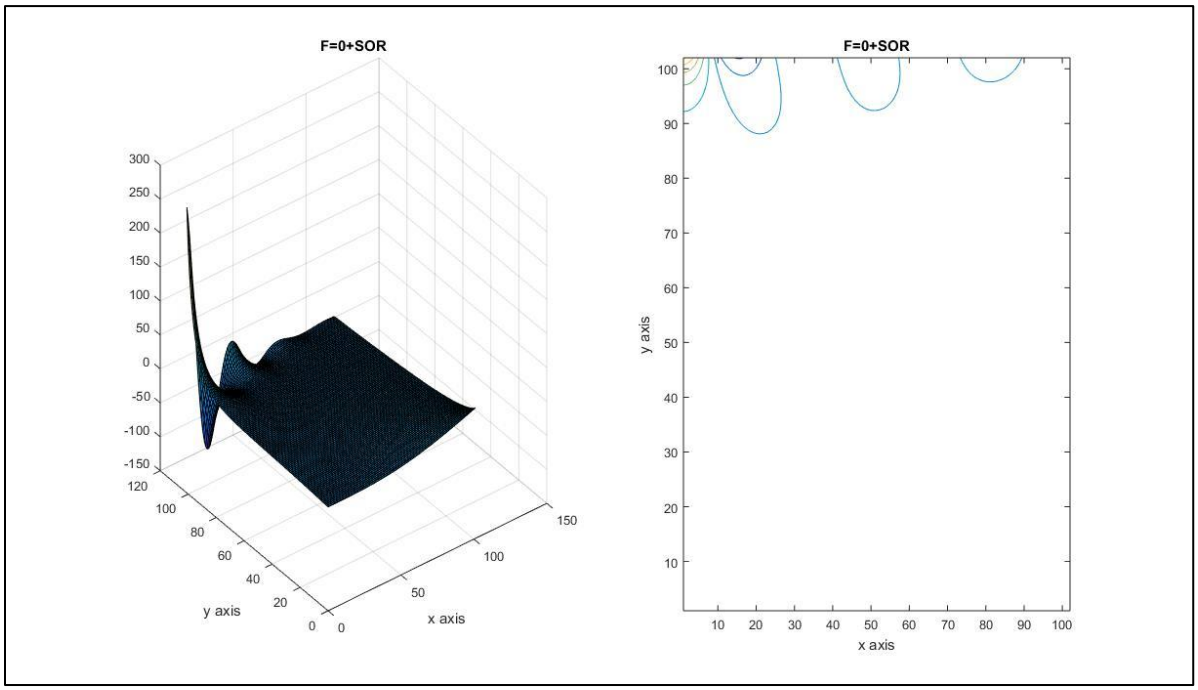


Figure 17: Gauss Seidel SOR 100 x 100 Simulation Case 3

The effect of the number of points made a dramatic difference in producing a finer mesh but takes up computing speed and iterations in the process. Successive Over Relaxation method could reduce the time in half. The following table below contains all the results of the cases mentioned before for three different size of points. The table includes the following iterations and time elapsed for each code and demonstrates the SOR method indeed cuts the time in half compared to the Gauss Seidel Method.

Table 2: Results of all methods with applied Gauss Seidel & Gauss Seidel SOR for following M, N sizes.

Results				
Gauss Seidel	**M,N**	**25x25**	**50x50**	**100x100**
F=Cos(x)Sin(y)	Iterations	2771	10311	47247
	Time Elapsed (sec)	0.843599	8.04209	80.552836
Gauss Seidel (SOR) λ =1.5	**M,N**	**25x25**	**50x50**	**100x100**
F=Cos(x)Sin(y)	Iterations	1033	3748	16809
	Time Elapsed (sec)	0.504753	4.732682	57.266763
Gauss Seidel	**M,N**	**25x25**	**50x50**	**100x100**
F=0	Iterations	2958	11398	41077
	Time Elapsed (sec)	0.753464	7.498831	70.445804
Gauss Seidel (SOR) λ =1.5	**M,N**	**25x25**	**50x50**	**100x100**
F=0	Iterations	1100	4121	14710
	Time Elapsed (sec)	0.52904	4.988216	49.240264
Gauss Seidel	**M,N**	**25x25**	**50x50**	**100x100**
Manufacturing Method	Iterations	2811	10944	40654
	Time Elapsed (sec)	0.733203	7.247998	69.350637
Gauss Seidel (SOR) λ =1.5	**M,N**	**25x25**	**50x50**	**100x100**
Manufacturing Method	Iterations	1044	3962	14554
	Time Elapsed (sec)	0.511304	4.81971	49.03218

The following table includes average U values for N points for both Gauss Seidel and Gauss Seidel Successive Over Relaxation method for the given F equation to determine grid convergence. The values were plotted in Figure(s) 18 & 19. The two curves show as the N value of points increase will tend to converge to a value approximately around the 200's. This study proves the grid independence test was a success. Gauss Seidel method takes a lot more iterations than the Successive Over Relaxation method. This just shows how much more optimized the SOR method is compared to the Gauss-Seidel.

N	Ugrid	Iterations	Ugrid SOR	Iterations
10	744.2196	453	744.2196	182
20	394.8126	1806	394.8126	684
50	259.648	10311	259.648	3748
75	237.1925	23472	237.1925	8436
100	227.0175	47247	227.0175	16808
150	217.5361	93300	217.5361	33159
200	213.0513	164457	213.0513	58307
250	210.441	257514	210.441	91127

Figure 18: Gauss Seidel Independent Study

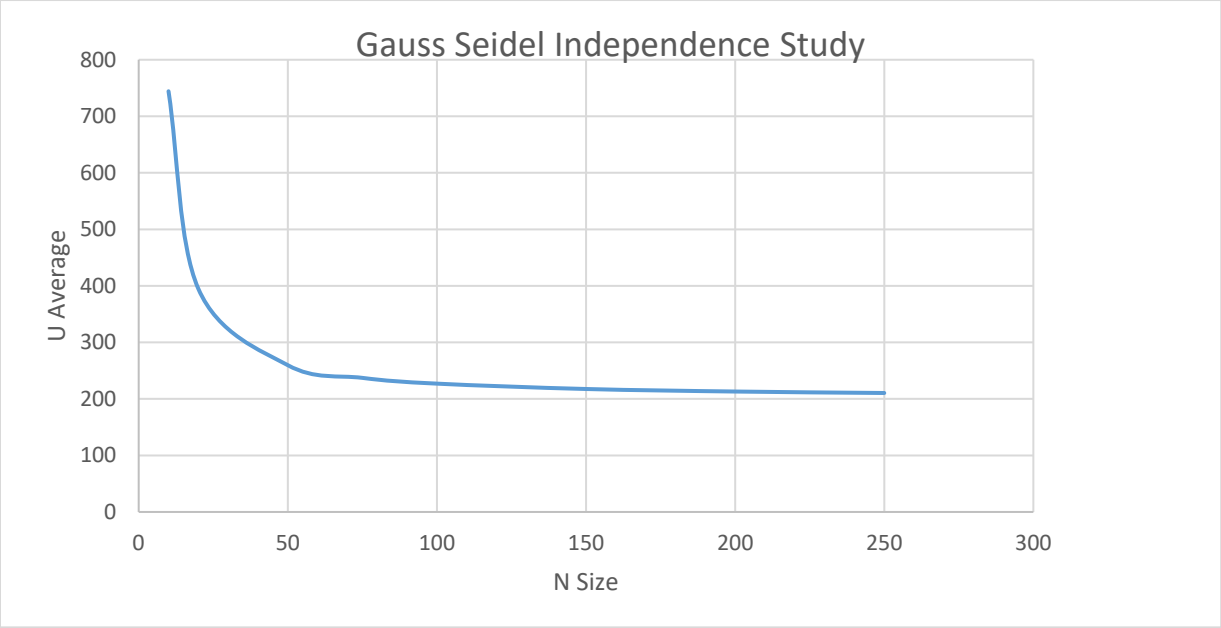
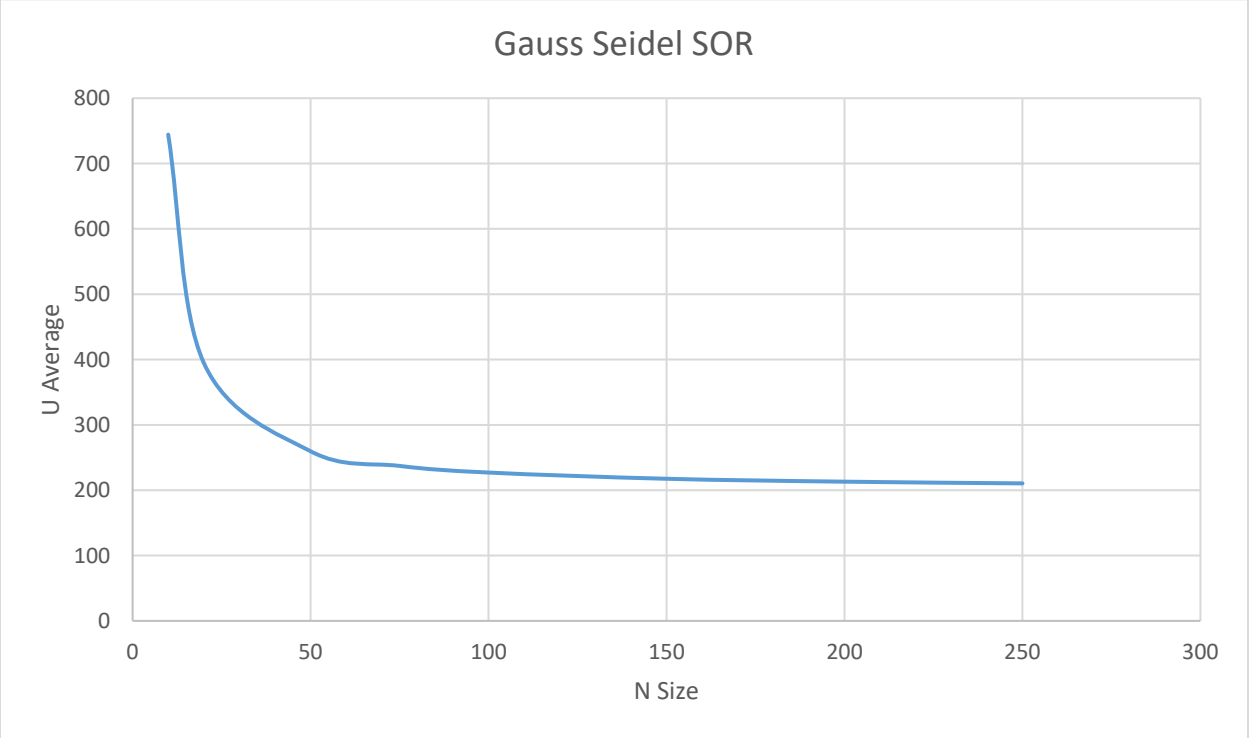


Figure 19: Gauss Seidel SOR Independent Study



Summary and Conclusion

Increasing the value of N proves that the SOR method was the fastest method for solving a 2D Poisson equation. The Gauss-Seidel method took more iterations and time to solve the equation but did meet the grid independence as the Success Over Relaxation Method. The test reveals the methods started converging when N =150-200 which essentially satisfy grid independence. The comparison between all the Cases demonstrate the Manufacturing, given F equation, and F=0 contain some differences but do not change results of the project.

Code used to calculate values (https://github.com/jmchave8/Project)

```
%Matlab Code to solve Poisson's equation with Gauss Seidel Method with the
following conditions in the problem statement.
% Jose Chavez  1161146
clear all; clc;

%% Given Conditions
ax = -pi;
ay = -pi;
bx = pi;
by = pi;

N=input('Value of X Intenal Nodes='); % Number of points on the internal
nodes for N and M%
M=input('Value of Y Internal Nodes=');
Time=tic; %Count Begins %
Me=M+2; %Number of points including exterior boundary points for Ne and Me%
Ne=N+2;
% this generates the x and y values that will be used to calculate
x = linspace(-pi,pi,Ne);
y = linspace(-pi,pi,Me);
%%

U = ones(Ne,Me); %U initial guess %

% For loop solving for right hand side with F equation with i,j indices%
for i=1:length(x);
    for j=1:length(y);
F(i,j) = cos ( (0.5*pi)* (2*((x(i)-ax) / (bx - ax))+1 )).*sin( pi*((y(j)-ay)
/ (by -ay)));
    end
end

%% Boundary Conditions for "top" and "bottom"
```

```
% Bottom boundary values
phi = ((x - ax).^2 ) .* sin( (pi *(x- ax)) / (2*(bx-ax)) ) ;

% Top boundary values
psy = cos (pi*(x-ax)).*cosh(bx-x);

% place these known values in the solution grid
U(1,:) = phi;

U(N+2,:) = psy;
%% Left and Right Boundary points
% Using the given neumann condition yields special cases of the Gauss-siedel
iteration that can be used along entire "side" boundaries.
% F and U is computed in solution grid
% Multipliers that are used in the iterations.
dx = 2*pi/(N+1);
B = 1/dx.^2;
dy = 2*pi/(M+1);
C = 1/dy.^2;
den= -2*(B+C);

% Normalize Multipliers%
B = B/den;
C = C/den;
F = F/den;
den = 1;
error=10;
error_iterations=0;
% check for diagonal dominance of elements
abs(den) >= abs(2*B+2*C)
Time_Count=0;
save('variables.mat')
%%
load('variables.mat')
while error>10^-10;
    T_loop=tic;
    if Time_Count >=.5
        Time_Count=0;
        save ('variables.mat')
    end
    W=U;
for i = 2:N+1;
    % Left boundary
    U(i,1) = den*(  F(i,1) - (2*B)*U(i,2) - C*U(i-1,1) - C*U(i+1,1) );

    % Right Boundary
    U(i,N+2) = den*(  F(i,N+2) - (2*B)*U(i,N+1) - C*U(i-1,N+2) - C*U(i+1,N+2)
);

end

%% Gauss-Siedel iterating the general U equation%

for i = 2:N+1;
```

```
    for j = 2:M+1;
        U(i,j) = den*(  F(i,j) - C*U(i+1,j) - C*U(i-1,j)- B*U(i,j+1) -
B*U(i,j-1) );
    end
end
error=abs(max(max(((W-U)./W))));
error_iterations=error_iterations+1;
P= toc(T_loop);
Time_Count=Time_Count + P;
end

toc(Time)
save('variables.mat')
%%
load('variables.mat')
error_iterations
grid_con=mean(mean(U.^2))
figure
subplot(1,2,1),surf(U),xlabel('x axis'),ylabel('y
axis'),title('F=cos(x)sin(y)');

subplot(1,2,2),contour(U),xlabel('x axis'),ylabel('y
axis'),title('F=cos(x)sin(y)');
```