Final Course Project: 2D Helmholtz Equation

MECE 5397: Scientific Computing for Mechanical Engineers

Instructors
Dr. Amritkar
Dr. Prosperetti

Student: John Carmichael Jimenez
PSID: 1253921

May 05, 2017

GitHub Link:
https://github.com/JohnCJimenez/SciCompProject

Abstract

The document Final Course Project: 2D Helmholtz Equation records the findings made by the author, John Jimenez, when he completed the final course project for the Scientific Computing for Mechanical Engineers course. The author of this report solved the 2D Helmholtz Equation through the creation of a computer code in MATLAB using both the Gauss-Seidel and Successive Over Relaxation methods covered in the course. A brief overview of the pseudocode as well as a description of the methods used by the author (for a 1D case) is provided in the report. The report provides 3D figures showing that both methods provide similar results with the same grid mesh. The report concludes with a summary of the author's grid convergence study, which implies that the solution converges with a grid mesh of approximately 500.

Introduction

The Helmholtz Equation is one of the numerous fundamental equations that model several phenomena in nature. The Helmholtz Equation can be applied to many situations including but not limited to electromagnetic radiation, seismology, and acoustics. Therefore, it is crucial to know how to obtain the solution of the Helmholtz Equation. Methods that produce exact solutions using a computer program such as MATLAB are either time-consuming or inaccurate due to processing limitations. Therefore, the most suitable alternatives to obtain meaningful solutions to the Helmholtz Equation are iterative methods such as the Gauss-Seidel Method, the Successive Over-Relaxation Method (SOR), etc. The methods discussed in this report are the Gauss-Seidel Method and the SOR Method. The results of the simulations will be provided as well as the project statement of the final course project. Grid Convergence of the solutions provided on the methods used is discussed briefly.

Problem Statement

For the final course project of the Scientific Computing for Mechanical Engineering Course, the author was required to solve the two-dimensional Helmholtz Equation using a computer code. The specific form of the Helmholtz Equation that the author had to solve takes the form of:

$$\frac{\partial^2 u}{\partial x^2} + \frac{d^2 u}{dy^2} + \Lambda u = F(x, y) \tag{1}$$

The domain of interest of equation (1) is the rectangle:

$$a_x < x < b_x \qquad\qquad a_y < y < b_y$$

The boundary conditions of interest in both x and y are:

$$u(x = a_x, y) = f_b(y) \qquad\qquad u(x = b_x, y) = g_b(y)$$

$$u(x, y = a_y) = f_b(a_y) + \frac{x - a_x}{b_x - a_x}[g_b(a_y) - f_b(a_y)]$$

$$\left.\frac{\partial u}{\partial y}\right|_{y=b_y} = 0$$

The functions $f_b(y)$ and $g_b(y)$ are expressed as:

$$f_b(y) = y(b_y - y)^2 \qquad g_b(y) = (b_y - y)^2 \cos\left(\frac{\pi y}{b_y}\right)$$

F(x,y) in equation (1) is expressed as:

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

For the specific problem statement, $a_x = a_y = -\pi$, $b_x = b_y = \pi$, and $\Lambda = \frac{1}{2}$. A final simulation must be carried out with *F=0* and $\Lambda = 0$.

Discretization of the 2D Helmholtz Equation

For any numerical methods that solve partial differential equations, a consistent and accurate discretization is necessary. For the specific form of the Helmholtz Equation given by equation (1), the discretization, using a centered difference method on the 2nd order derivatives, can be expressed as:

$$\frac{u(x_{a-1}, y_b) - 2u(x_a, y_b) + u(x_{a+1}, y_b)}{\Delta x^2} + \frac{u(x_a, y_{b-1}) - 2(x_a, y_b) + u(x_a, y_{b+1})}{\Delta y^2} + \Lambda u(x_a, y_b) = F(x_a, y_b)$$

with a and b representing the index of the value of x and y respectively. However, should $\Delta x^2 = \Delta y^2 = h^2$, the discretization can be simplified to

$$u(x_{a-1}, y_b) + u(x_{a+1}, y_b) + (\Lambda h^2 - 4)u(x_a, y_b) + u(x_a, y_{b-1}) + u(x_a, y_{b+1}) = F(x_a, y_b) \qquad (2)$$

A discretization like equation (2) creates a pentadiagonal coefficient matrix.

Explanation of the Numerical Method and Pseudocode

As stated previously, the methods used by the author to solve the 2D Helmholtz Equation are the Gauss-Seidel Method and the SOR Method. Both methods are iterative in nature with SOR converging faster than the Gauss-Seidel Method. Below are the pseudocodes in MATLAB syntax for solving the 2D Helmholtz Equation through both methods used by the author.

Gauss-Seidel Method

```
clc,clearvars
%% Dimensions of the Square Domain and Some Important Parameters
% Note: the domain of interest is the square:
% -pi<x<pi and -pi<y<pi

% Start and End points for domain in x

    ax = -pi;    % Start point of domain in x
    bx =  pi;    % End point of domain in x

% Start and End points for domain in y are the the same as that in x for Project Code.

    ay = -pi;  % Start point of domain in y
    by =  pi;   % End point of domain in y

% A value for Lambda can be assigned directly but is left up to user input for greater
versatility

    Lambda = input('Please input a value for Lambda:'); Lambda = -Lambda;


% Assigning an acceptable limit of error for convergence

    Es = 10^-10;

%% Boundary Conditions Setup and Relevant Parameters

% Defines the Interior nodes for both x and y through user input. However,these do not
% include exterior boundary points.

IntNodesX = input('Please input a value for the no. of nodes for the  x-coordinate:');
IntNodesY = IntNodesX; % Ensures a square matrix, better for operational compatibility
N=IntNodesY;


% Generating a range of linearly spaced vectors for x and y through the variables IntNodesX
% and IntNodesY
% IMPORTANT: THESE ARE THE INPUT VALUES FOR EVERY FUNCTION THAT NEEDS
% THEM!

    xval = linspace(ax,bx,IntNodesX); % Start and End nodes for x can be redefined through
changing ax and bx above.
    yval = linspace(ay,by,IntNodesY); % Start and End nodes for y can be redefined through
changing ay and by above.

% Value of spacing for Each Node

    dx = (bx-ax)/IntNodesX;
    dy = dx;
```

```
% Defining some coefficients that remain the same for all matrix
% operations. Refer to report for details concerning the discretization

    A = Lambda - (2 * (dx^2)) - (2 * (dy^2));
    B = (dx^2) * (dy^2);
    U = zeros(N,N); % Preallocates an initial guess

% The following functions below are needed for the evaluation of the
% boundary conditions as well as the function F(x,y)

    fb  =  y*(by-yval).^2;
    gb  =  (by-yval).^2*(cos((pi*yval)/by));
    uw  =  ((ay*(by-ay).^2))+(((xval-ax)/(bx-ax))*((cos((pi*ay)/by)*(by - ay).^2) - ((ay*(by-
ay).^2))));
    %F = zeros(N,N); % This is for the Final Simulation

    F = sin(pi*(X-ax)/(bx-ax)).*cos((0.5*pi)*(2*((Y-ay)/(by-ay))+1));

     % Dirichlet on x-axis
     U(2:N-1,1) = Y(2:N-1,1).*(by-Y(2:N-1,1)).^2;
     U(2:N-1,N) = ((by-Y(2:N-1,1)).^2).*cos((pi*Y(2:N-1,1)/by));

     % Dirichlet on y-axis
     U(N,:) = ((ay*(by-ay).^2))+(((X(N,:)-ax)/(bx-ax))*((cos((pi*ay)/by)*(by - ay).^2) -
((ay*(by-ay).^2))));

     e = 1; %Preallocated error guess

%% The Gauss-Seidel Method
while e > Es
    U1 = U;
    for a = 2:IntNodesX-1
        for b = 2:IntNodesY-1
            if a == 2
                U(1,b) = (F(1,b)*dx^2) - (2*U(2,b)) - U(1,b+1) - U(1,b-1);
            else
                U(a,b) = ((F(a,b)*(dx^2)) - U(a-1,b) - U(a,b-1) - U(a+1,b) -
U(a,b+1))/((Lambda*dx^2)-4);
            end
        end
    end

    e = max(max(abs((U-U1)./U)));
end

Value = mean(mean(U(2:IntNodesX-1,2:IntNodesX-1).^2)) %Needed for Grid Convergence Study
%% Plotting Results
    figure(1) % VECTOR FIELD U
    set(gcf,'units','normalized','position',[0.02 0.52 0.3 0.32]);
    %surf(X,Y,U);
    mesh(X,Y,U);
    rotate3d
    xlabel('x'); ylabel('y'); zlabel('U');
    title('Solution using Gauss-Seidel Method','fontweight','normal'),colorbar;
```

Successive Over-Relaxation Method

```
%% Dimensions of the Square Domain and Some Important Parameters
%  Note: the domain of interest is the square:
% -pi<x<pi and -pi<y<pi

% Start and End points for domain in x

    ax = -pi;    % Start point of domain in x
    bx =  pi;    % End point of domain in x

% Start and End points for domain in y are the the same as that in x for Project Code.
```

```
    ay = -pi;   % Start point of domain in y
    by =  pi;   % End point of domain in y

% A value for Lambda can be assigned directly but is left up to user input for greater
versatility

    Lambda = input('Please input a value for Lambda:'); Lambda = -Lambda;
    G   = 1.75

    if G > 2
        disp('Please try again. Your selected value for G is incompatible for SOR.')
        disp('Restart process through typing in SORMethodv2.')
        return
    end


% Assigning an acceptable limit of error for convergence

    Es = 10^-10;

%% Boundary Conditions Setup and Relevant Parameters

% Defines the Interior nodes for both x and y through user input. However,these do not
% include exterior boundary points.

    IntNodesX = input('Please input a value for the no. of nodes for the  x-coordinate:');
    IntNodesY = IntNodesX; % Ensures a square matrix, better for operational compatibility
    N=IntNodesY;


% Generating a range of linearly spaced vectors for x and y through the variables IntNodesX
% and IntNodesY
% IMPORTANT: THESE ARE THE INPUT VALUES FOR EVERY FUNCTION THAT NEEDS
% THEM!

    xval = linspace(ax,bx,IntNodesX); % Start and End nodes for x can be redefined through
changing ax and bx above.
    yval = linspace(ay,by,IntNodesY); % Start and End nodes for y can be redefined through
changing ay and by above.
    [X,Y]= meshgrid(xval,yval);
    Y = flipud(Y);

% Value of spacing for Each Node

    dx = (bx-ax)/IntNodesX;
    dy = dx;


% Defining some coefficients that remain the same for all matrix
% operations. Refer to report for details concerning the discretization

    A = Lambda - (2 * (dx^2)) - (2 * (dy^2));
    B = (dx^2) * (dy^2);
    U = zeros(N,N);

% The following functions below are needed for the evaluation of the
% boundary conditions as well as the function F(x,y)

    fb  =  Y*(by-Y).^2;
    gb  =  (by-Y).^2*(cos((pi*Y)/by));
    uw  =  ((ay*(by-ay).^2))+(((X-ax)/(bx-ax))*((cos((pi*ay)/by)*(by - ay).^2) - ((ay*(by-
ay).^2))));
    % F = zeros(N,N); % This is for the Final Simulation

    F =  sin(pi*(X-ax)/(bx-ax)).*cos((0.5*pi)*(2*((Y-ay)/(by-ay))+1));

     % Dirichlet on x-axis
     U(2:N-1,1) = Y(2:N-1,1).*(by-Y(2:N-1,1)).^2;
     U(2:N-1,N) = ((by-Y(2:N-1,1)).^2).*cos((pi*Y(2:N-1,1)/by));

     % Dirichlet on y-axis
```

```
     U(N,:) = ((ay*(by-ay).^2))+(((X(N,:)-ax)/(bx-ax))*((cos((pi*ay)/by)*(by - ay).^2) -
((ay*(by-ay).^2))));


    w = 2/(1+sin(pi*dx/(2*pi)));
    e = 1; % Preallocated Error

%% The Successive Over Relaxation (SOR) Method
while e > Es

    U1 = U;
    for a = 2:IntNodesX-1
        for b = 2:IntNodesY-1
            W(a,b) = U(a,b);
            if a == 2
                U(1,b) = (F(1,b)*dx^2) - (2*U(2,b)) - U(1,b+1) - U(1,b-1);
                U(1,b) = (G * U(1,b)) + ((1-G) * W(1,b));
            else
                U(a,b) = (((F(a,b)*(dx^2)) - U(a-1,b) - U(a,b-1) - U(a+1,b) -
U(a,b+1))/((Lambda*dx^2)-4));
                U(a,b) = (G * U(a,b)) + ((1-G) * W(a,b));
            end
        end
    end

    e = max(max(abs((U-U1)./U)));

end

Value = mean(mean(U(2:IntNodesX-1,2:IntNodesX-1).^2)) % Needed for Grid Convergence Study

%% Plotting Results
    figure(1) % VECTOR FIELD U
    set(gcf,'units','normalized','position',[0.02 0.52 0.3 0.32]);
    %surf(X,Y,U);
    mesh(X,Y,U);
    rotate3d
    xlabel('x'); ylabel('y'); zlabel('U');
    title('Solution using SOR Method','fontweight','normal'),colorbar;
```

As described in the textbook Numerical Methods for Engineers 7th Edition, the Gauss-Seidel and the SOR methods take the form of the following (1D is shown for both methods)

Gauss-Seidel Method

For a 3x3 matrix taking the form of [A]{u} = {F}**:**

$$u_1 = \frac{f_1 - a_{12}u_2 - a_{13}u_3}{a_{11}}$$

$$u_2 = \frac{f_2 - a_{21}u_1 - a_{23}u_3}{a_{22}}$$

$$u_3 = \frac{f_3 - a_{31}x_1 - a_{32}x_2}{a_{33}}$$

the method is done iteratively until the error threshold is reached.

Successive-Over Relaxation Method

For the same 3x3 system: $u_i^{new} = \lambda u_i^{new} + (1 - \lambda) u_i^{old}$

Technical Specifications of the Computer Used

The following information about the technical specifications of the computer used was taken directly from Apple's Support Page. The computer used by the author to execute the code is a MacBook Pro 13 inch, mid 2012. The operating system used by the author is macOS Sierra version 10.12.4. Table 1 below displays the technical specifications.

Table 1: Technical Specifications of Author's Computer

Author's Technical Specifications
Processor ▪ 2.5GHz dual-core Intel Core i5 processor (Turbo Boost up to 3.1GHz) with 3MB L3 cache ▪ 2.9GHz dual-core Intel Core i7 processor (Turbo Boost up to 3.6GHz) with 4MB L3 cache **Memory** ▪ **2.5GHz** 4GB of 1600MHz DDR3 memory *Configurable to 8GB* ▪ **2.9GHz** 8GB of 1600MHz DDR3 memory **Storage** ▪ **2.5GHz** 500GB 5400-rpm hard drive *Configurable options:* o *750GB 5400-rpm hard drive* o *128GB solid-state drive* o *256GB solid-state drive* o *512GB solid-state drive* ▪ **2.9GHz** 750GB 5400-rpm hard drive *Configurable options:* o 1TB 5400-rpm hard drive o 128GB solid-state drive o 256GB solid-state drive

- 512GB solid-state drive

Results

Figures 1 through 4 are a 3D representation of the solution of the 2D Helmholtz Equation with the given boundary conditions. Figures 1 through 4 were generated using the Gauss-Seidel method. Figure 1 uses 10 nodes, Figure 2 uses 50 nodes, and Figures 3 and 4 use 100 and 250 nodes respectively.

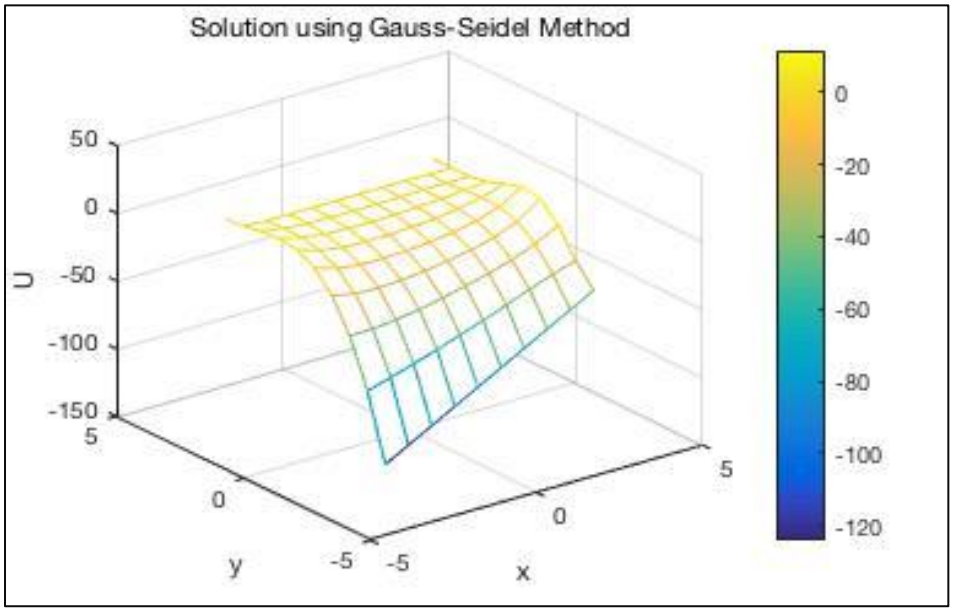

Figure 1: 2D Helmholtz Solution (nodes=10)

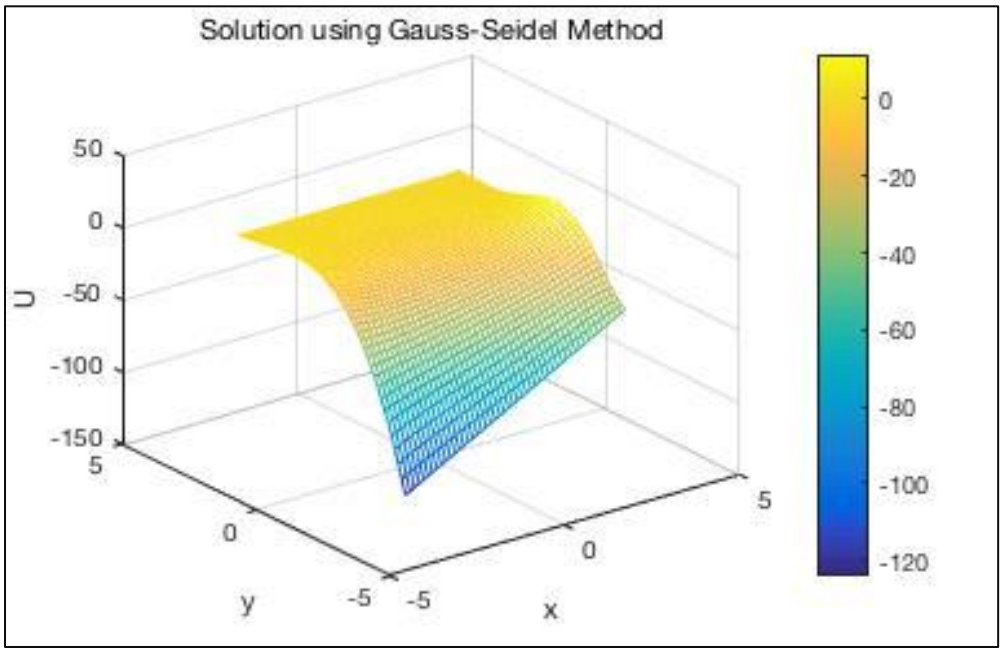

Figure 2: 2D Helmholtz Solution (nodes=50)

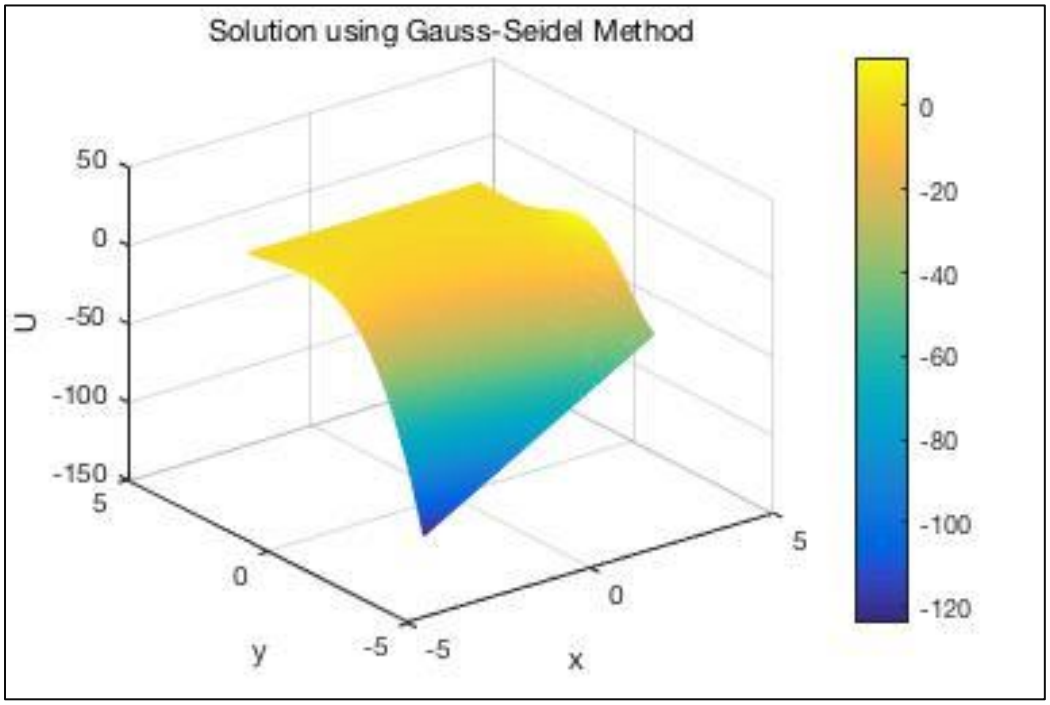

Figure 3: 2D Helmholtz Solution (nodes=100)

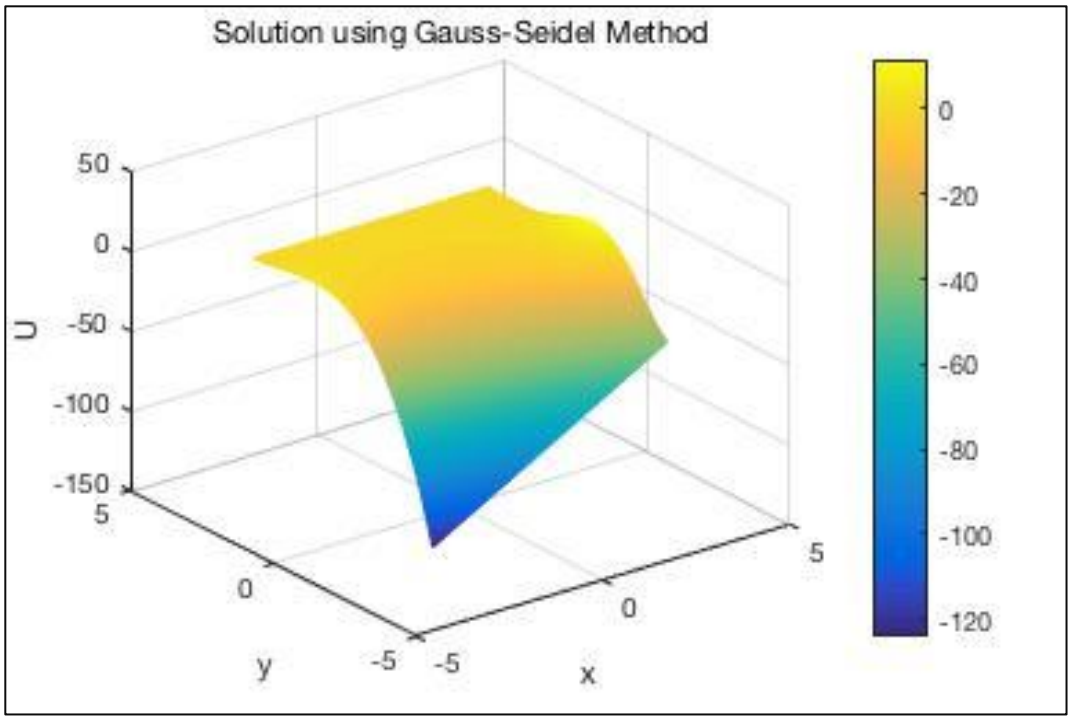

Figure 4: 2D Helmholtz Solution (nodes=250)

Figures 5 through 8 are a 3D representation of the solution of the 2D Helmholtz Equation with the given boundary conditions. Figures 5 through 8 were generated using the SOR Method. The number of nodes for each figure are given as well.

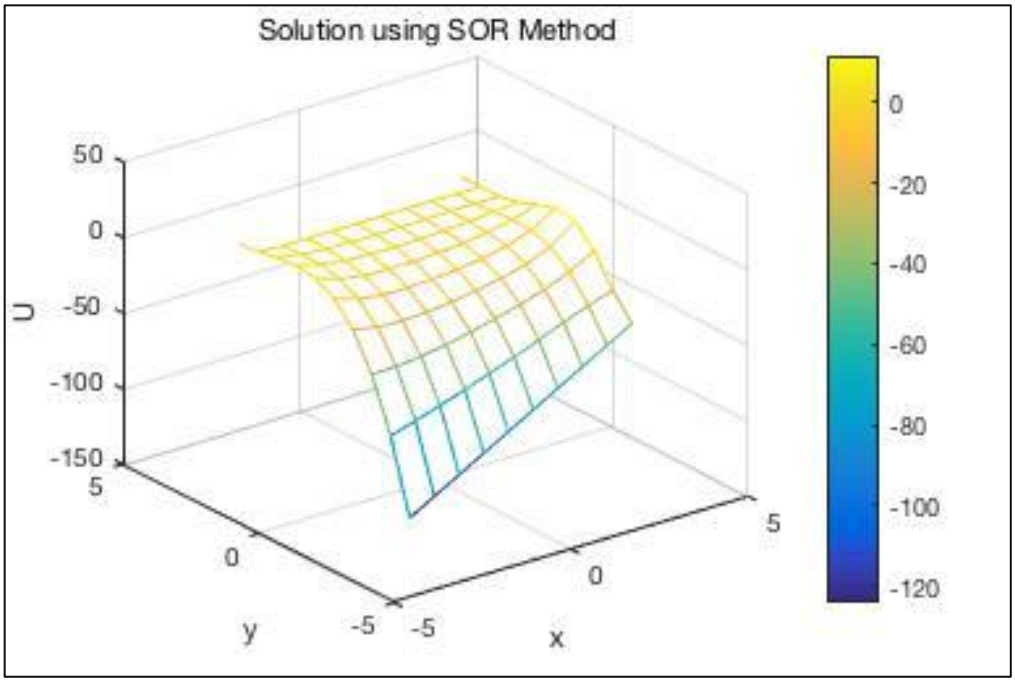

Figure 5: 2D Helmholtz Solution through SOR (nodes=10)

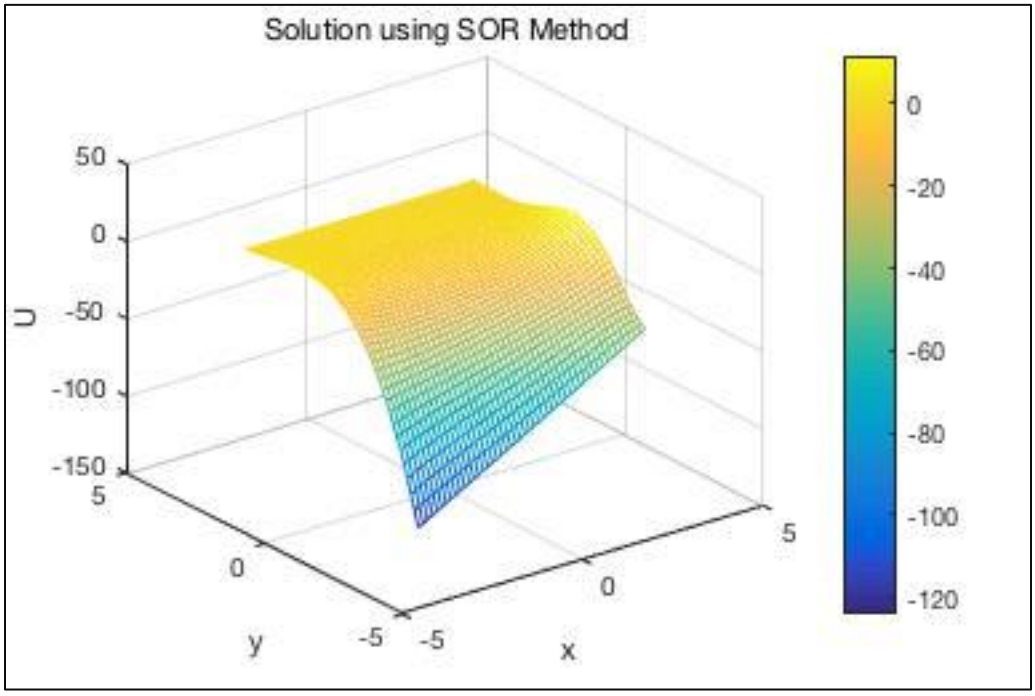

Figure 6: 2D Helmholtz Solution through SOR (nodes=50)

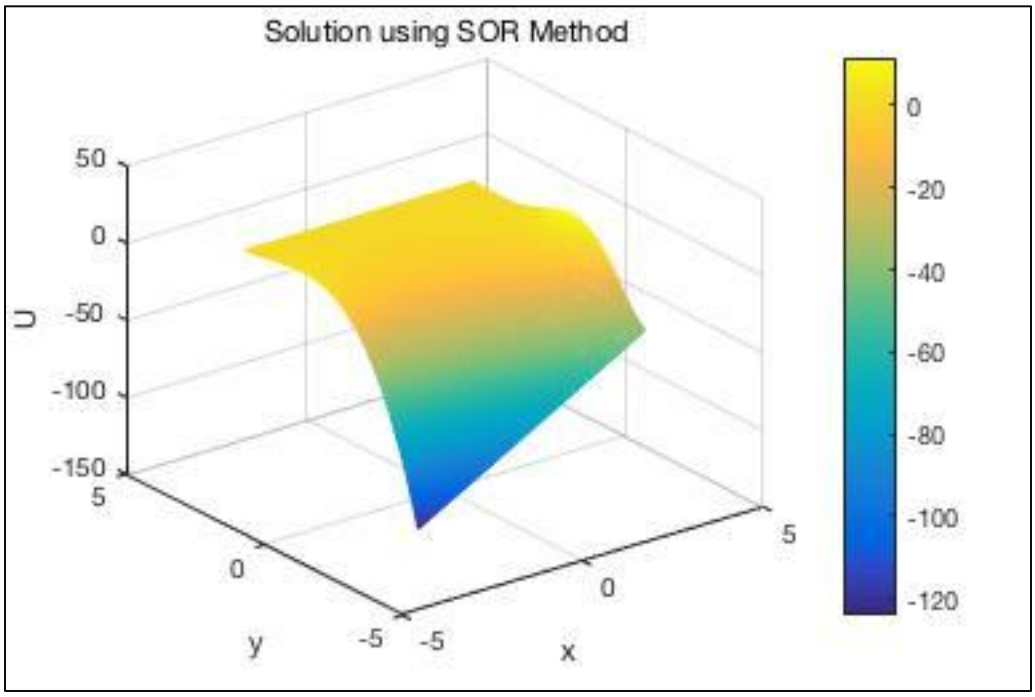

Figure 7: 2D Helmholtz Solution through SOR (nodes=100)

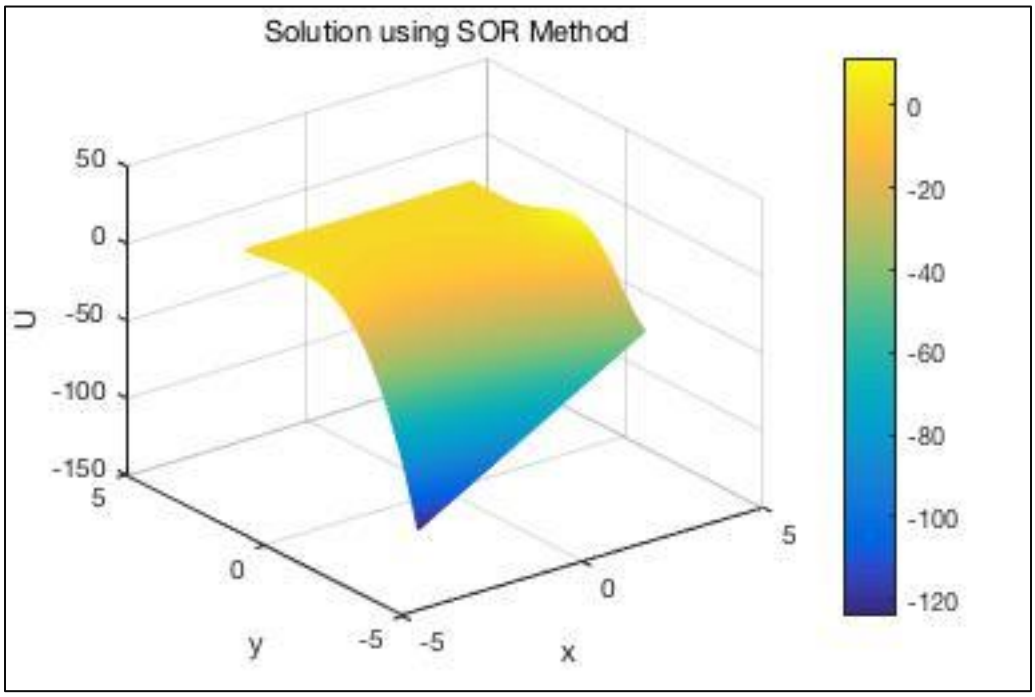

Figure 8: 2D Helmholtz Solution through SOR (nodes=250)

As shown by Figures 1 through 8, a finer discretization leads to smoother curves. However, a larger number of nodes (e.g. 400) leads to longer runtimes for both the Gauss-Seidel method and the SOR method.

Figures 9 and 10 are 3D plots of the 2D Helmholtz Equation when F(x,y) = 0 and $\Lambda = 0$ Therefore, the codes essentially solve Laplace's equation within the domain stated in the Problem Statement section through the Gauss-Seidel Method and the SOR Method.

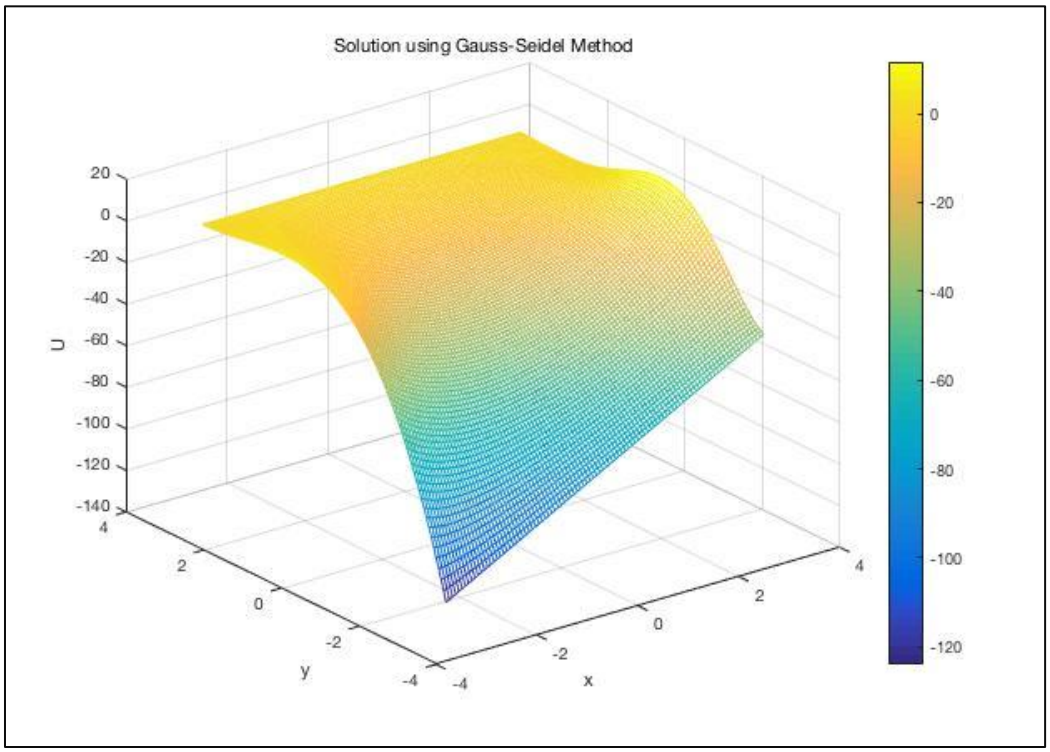

Figure 9: 2D Laplace Equation through Gauss-Seidel (nodes=100)

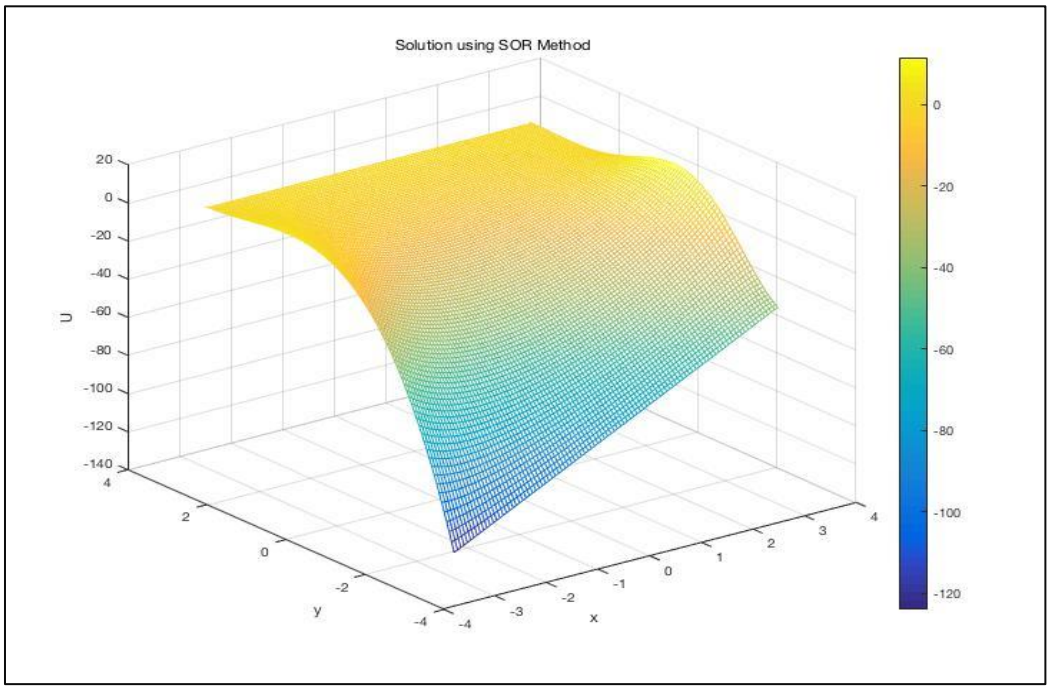

Figure 10: 2D Laplace Equation Solution through SOR (nodes=100)

As shown by Figures 9 and 10, a finer discretization leads to smoother curves. However, a larger number of nodes (e.g. 400) leads to longer runtimes for both the Gauss-Seidel method and the SOR method.

For all simulations conducted, the acceptable error limit is 10^{-10} and a value of 1.75 was used for SOR.

Table 2 below are the results of the observations of grid convergence when the number of nodes were increased to ensure a fine mesh. The results of both the Gauss-Seidel and SOR are presented in the table.

Table 2: Grid Convergence Study Data

Gauss-Seidel Method		SOR Method	
Number of Grid Points	Result	Number of Grid Points	Result
10	521.0384	10	521.0384
25	657.9515	25	657.9515
50	706.8465	50	706.8465
100	731.8835	100	731.8835
150	740.3126	150	740.3126
200	744.5425	200	744.5425
250	747.0852	250	747.0852
300	748.7823	300	748.7823
350	749.9955	350	749.9955
400	750.906	400	750.906

As shown by Table 2, the data is the same for both cases since the only difference between SOR and the Gauss-Seidel Method is a correction factor.

Figure 11 below is a plot of the data shown on Table 2. As shown by Figure 11 and Table 2, the actual result will converge around the 750-760. However, due to the technical specifications of the computer that the author used to run the simulations, a simulation with an extremely fine mesh (e.g. 450) is expensive.

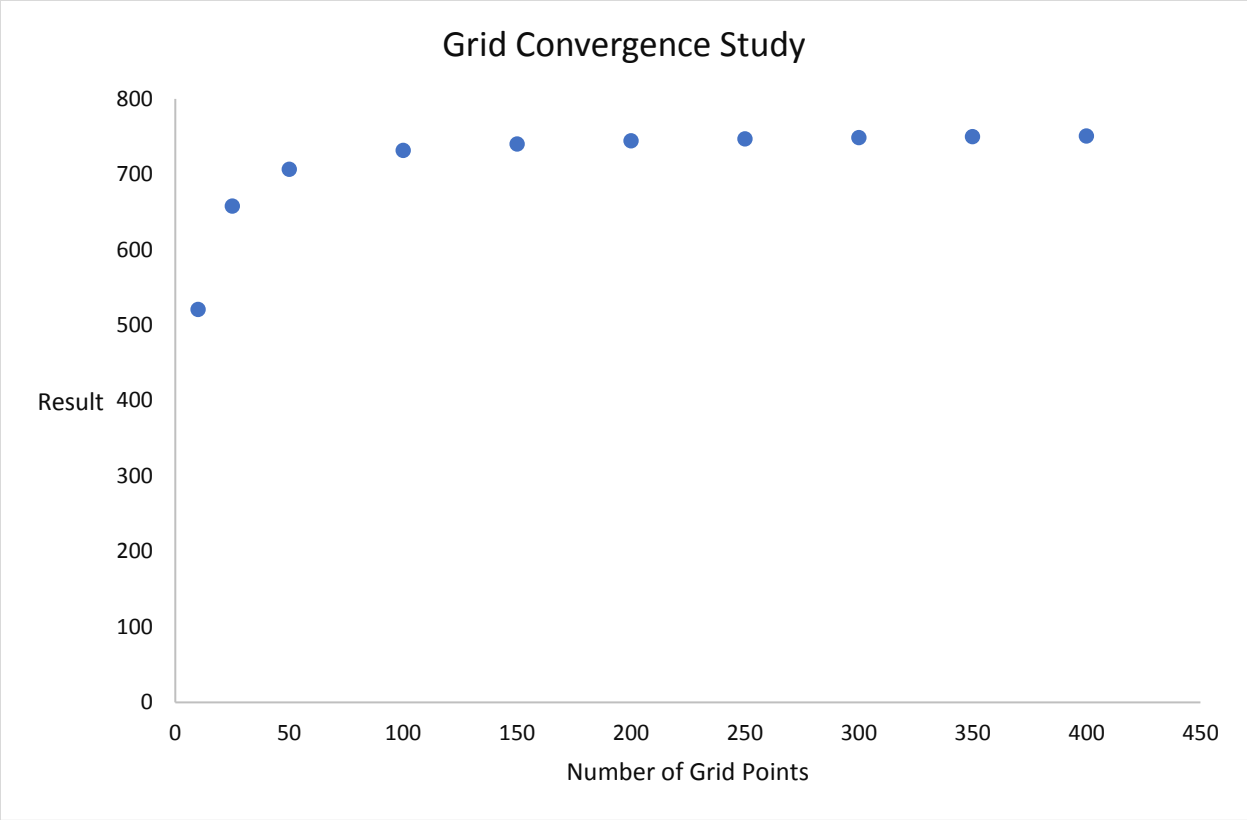

Figure 11: Grid Convergence Study Plot

Conclusion

For the final project of the Scientific Computing for Mechanical Engineers course, students were required to solve a partial differential equation (PDE) of their choice using a computer code implementing the numerical methods covered in the course. The author of this report chose the 2D Helmholtz Equation as his PDE and solved it using the Gauss-Seidel method and the Successive Over Relaxation Method. Both methods yielded similar results with the given boundary conditions and additional parameters that acted as constraints to the solution. A grid convergence study led the author to conclude that the solution will converge at a grid size of around 500.