Final Project

HELMHOLTZ EQUATION EVALUATED WITH GAUSS-SEIDEL AND SUCCESSIVE-OVER-RELAXATION

Franks, Travis D
PSID: 1372696 | MECE 5397

Travis Franks
PSID: 1372696
Project Code: AHc2-6

Abstract

The Helmholtz partial differential equation was discretized and evaluated using two different iterative approximation methods. These approaches were inclusive of the Gauss-Seidel and Successive-Over-Relaxation (SOR) methods of approximation. To do this, the Helmholtz partial differential equation and its associated Neumann Boundary Conditions were first discretized. Then, an algorithm was produced for each in MATLAB to iteratively evaluate for convergent solutions to the Helmholtz equation. Next, the SOR method was optimized by processing the performance time for 0.05 increasing increments in the SOR relaxation coefficient (λ) in which the value of 1.93 was determined to be the most universally optimal λ value. After optimizing the SOR λ value, the Method of Manufactured Solutions was applied to verify the accuracy of both the Gauss-Seidel and SOR approximations. Both methods were deemed acceptable approximations to the value of U, the solution matrix, as the solutions converged to the same values in the primary region of the solution, with only minor variances in the gradients leading from the edges. Following the verification of the approximations, a grid independence study was performed to prove that the solution would remain independent of the grid size for both approximations. The approximations were determined to have achieved grid independence since the mean(U.^2) values quickly converged to a zero slope value. Once grid independence was established, a study was done analyzing how varying the lambda coefficient for the Helmholtz equation would impact the approximated, discretized solution of the Helmholtz equation. It was determined that the value provided in the problem statement, 1.5, would produce divergent results and that decreasing to a value of -1.5, 0, or 0.2 would produce convergent plots of varying behaviors. Finally, a speed performance test was performed comparing the SOR and Gauss-Seidel methods. The SOR method, with a λ value of 1.93, was determined to be significantly faster than the Gauss-Seidel method, achieving a time reduction up to 6.8% of the time it took for the Gauss-Seidel method at a 300X300 grid size.

Travis Franks
PSID: 1372696
Project Code: AHc2-6

Helmholtz Problem Statement

AHc2-6

MECE 5397

Project A – Helmholtz Equation

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

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \Lambda u = F(x,y) \tag{41}$$

$$\Lambda = \frac{3}{2} \tag{42}$$

The domain of interest is the rectangle

$$a_x < x < b_x\,, \qquad a_y < y < b_y \tag{43}$$

and the boundary conditions

$$u(x, y = a_y) = \phi_{ab}(x)\,, \qquad u(x, y = b_y) = \psi_{ab}(x)\,, \tag{44}$$

$$\left.\frac{\partial u}{\partial x}\right|_{x=a_x} = 0\,, \qquad \left.\frac{\partial u}{\partial x}\right|_{x=b_x} = 0\,, \tag{45}$$

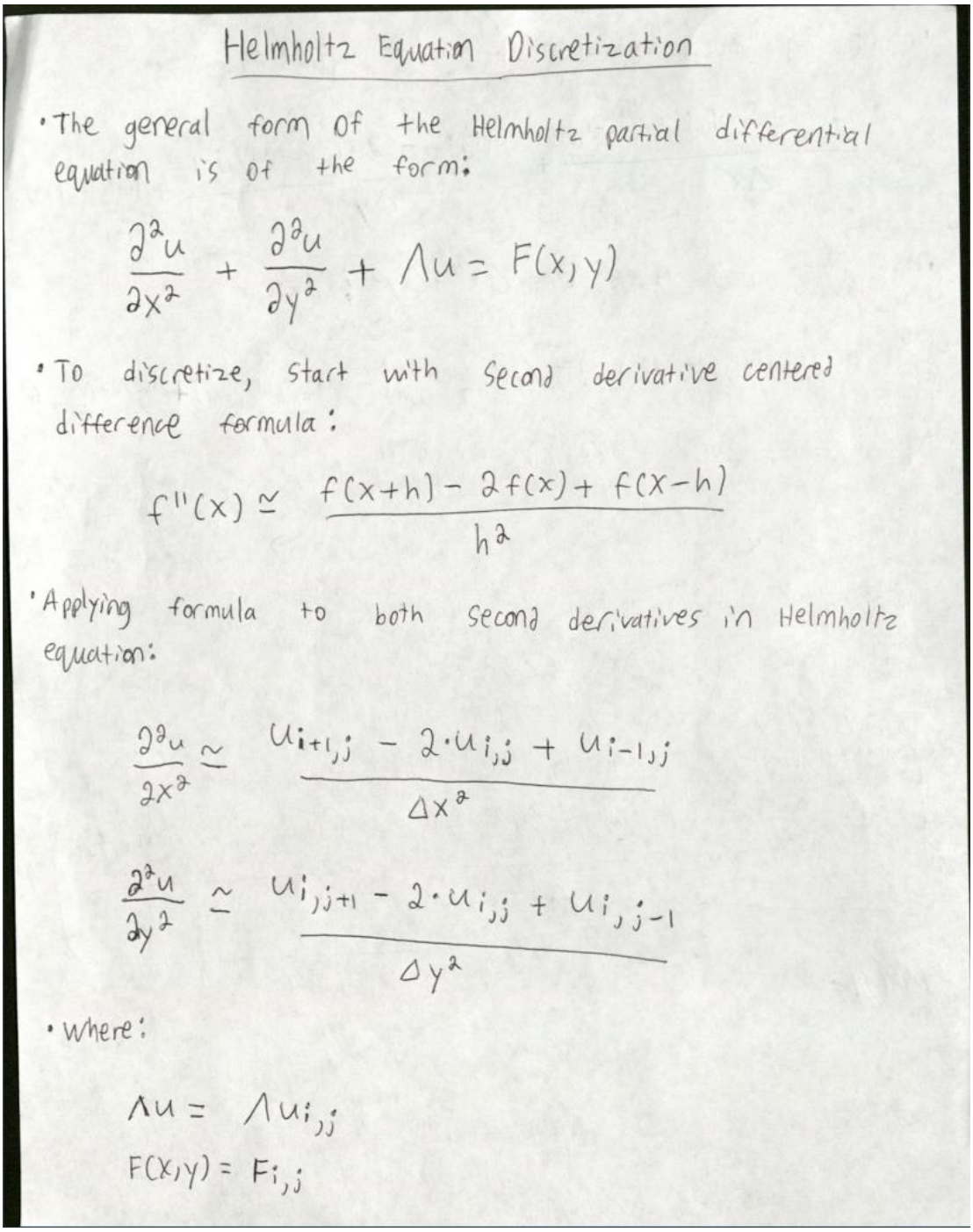
$$a_x = a_y = -\pi\,, \qquad b_x = b_y = \pi \tag{46}$$

$$\phi_{ab}(x) = \cos[\pi(x - a_x)]\,\cosh(b_x - x)\,, \qquad \psi_{ab}(x) = (x - a_x)^2 \sin\frac{\pi(x - a_x)}{2(b_x - a_x)} \tag{47}$$

$$F(x,y) = \cos\left[\frac{\pi}{2}\left(2\frac{x - a_x}{b_x - a_x} + 1\right)\right] \sin\left[\pi\frac{y - a_y}{b_y - a_y}\right] \tag{48}$$

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

Discretization



Helmholtz Equation Discretization

- The general form of the Helmholtz partial differential equation is of the form:

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

- To discretize, start with second derivative centered difference formula:

$$f''(x) \simeq \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

- Applying formula to both second derivatives in Helmholtz equation:

$$\frac{\partial^2 u}{\partial x^2} \simeq \frac{u_{i+1,j} - 2 \cdot u_{i,j} + u_{i-1,j}}{\Delta x^2}$$

$$\frac{\partial^2 u}{\partial y^2} \simeq \frac{u_{i,j+1} - 2 \cdot u_{i,j} + u_{i,j-1}}{\Delta y^2}$$

- where:

$$\lambda u = \lambda u_{i,j}$$

$$F(x,y) = F_{i,j}$$

Figure 1 – Page 1 of Discretization Process

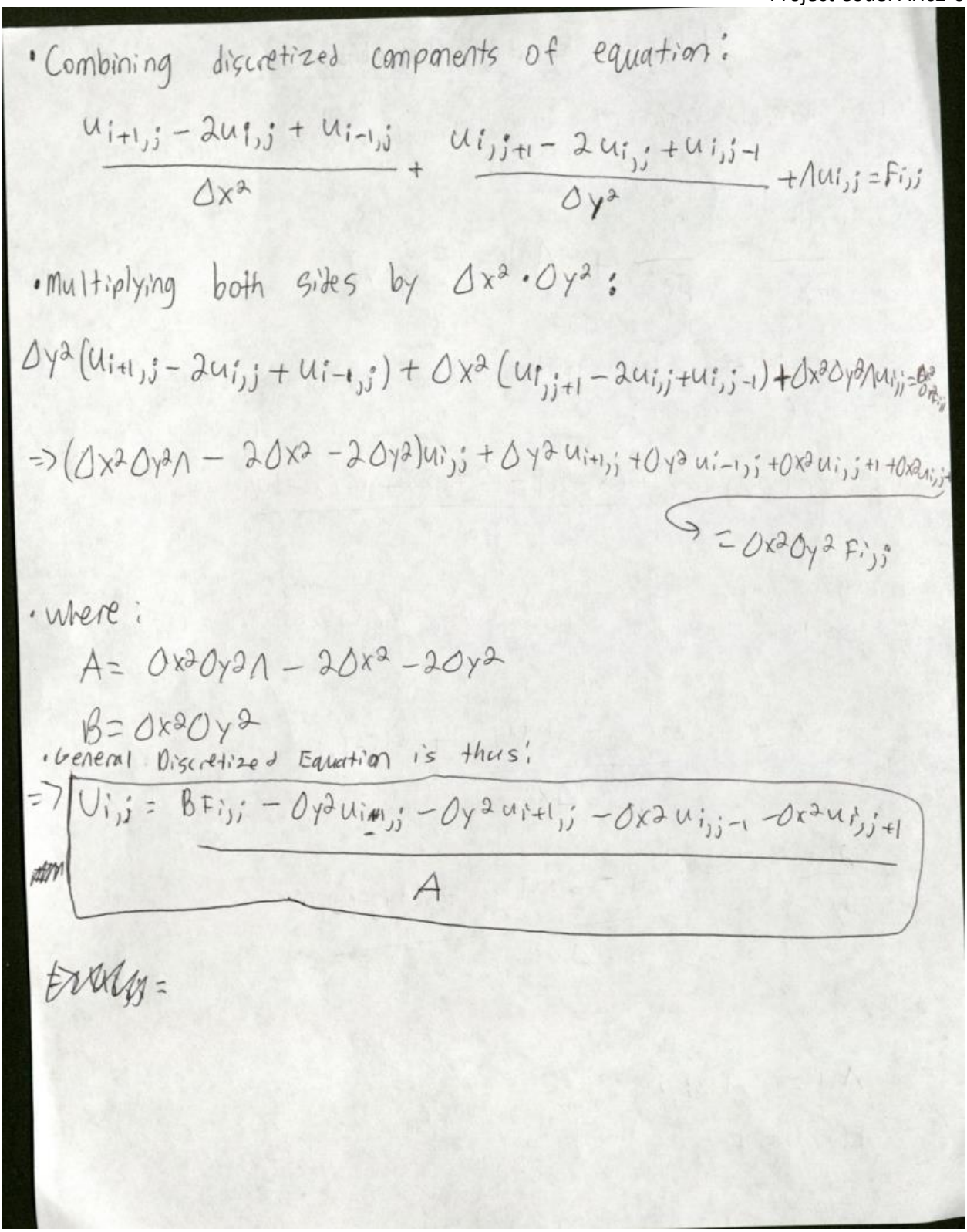


Figure 2 – Page 2 of Discretization Process

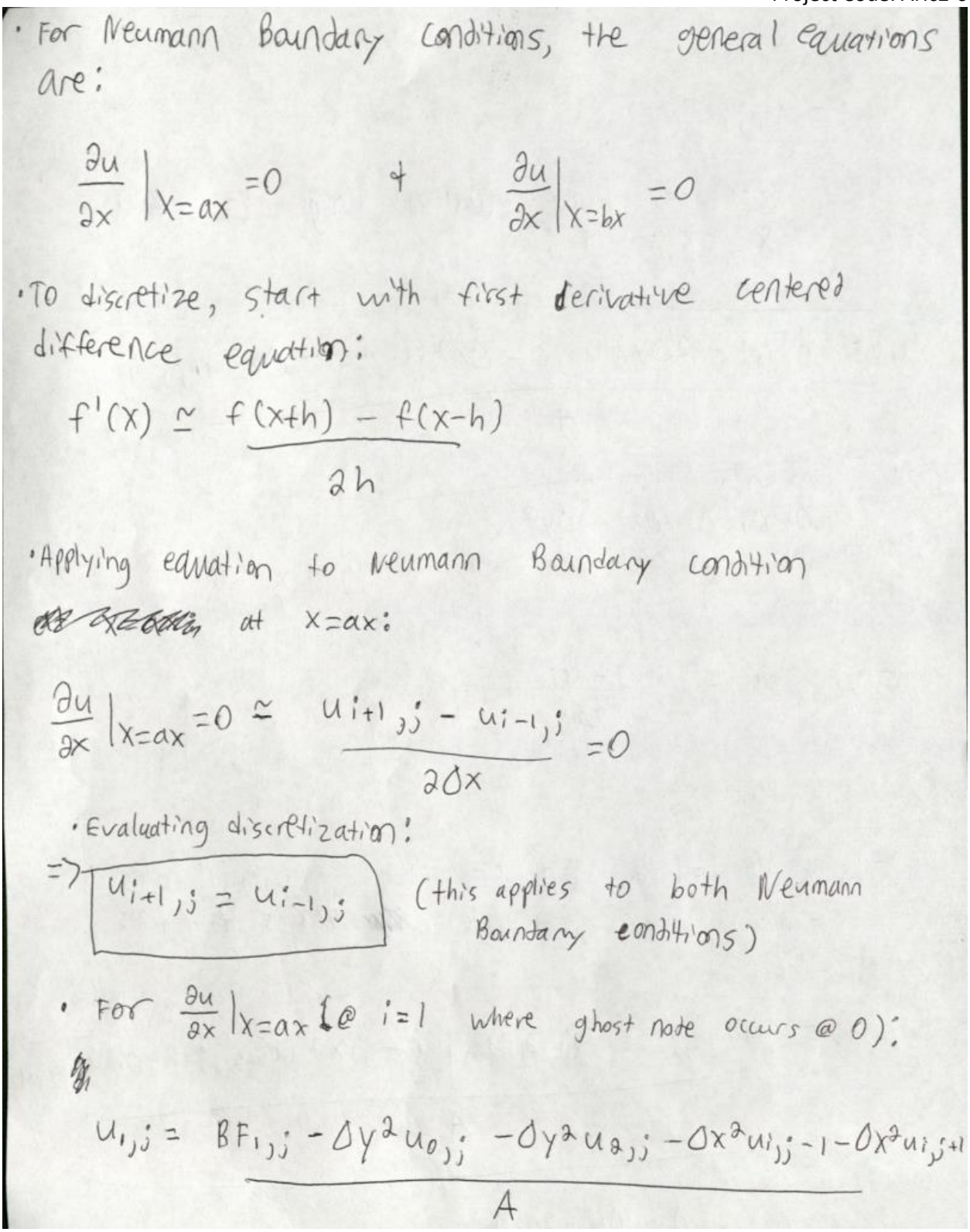


Figure 3 – Page 3 of Discretization Process

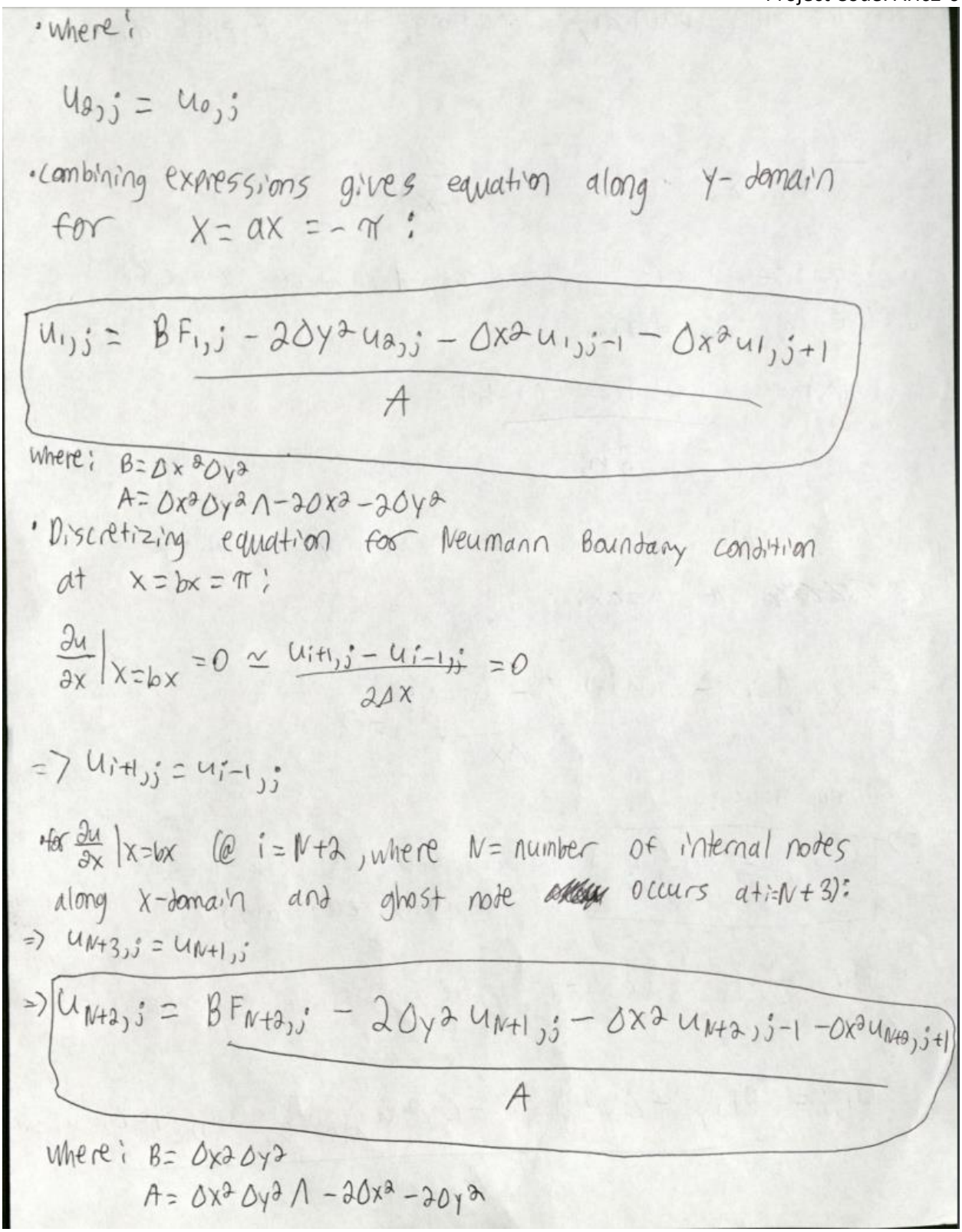


Figure 4 – Page 4 of Discretization Process

Description of Algorithm

Gauss-Seidel

A Gauss-Seidel approximation for a 2-dimensional system was performed for the Helmholtz partial differential equation with certain boundary conditions. To do this, the Helmholtz partial differential equation was first discretized, as mentioned in Figures 1, 2, 3, and 4 above, by applying centered difference derivative approximations. The centered difference approximation equations are direct consequences of the application of the Taylor Series. This discretization process then led to the equations to be evaluated via Gauss-Seidel iterations. The general discretized equation is shown below in Equation 1, with the Neumann boundary condition discretized equations respectively shown below in Equations 2 and 3:

- Equation 1 – General Discretized Helmholtz Equation:

$$U(i,j) = \frac{[\Delta X^2*\Delta Y^2*F(i,j) - \Delta Y^2*U(i-1,j) - \Delta Y^2*U(i+1,j) - \Delta X^2*U(i,j-1) - \Delta X^2*U(i,j+1)]}{[\Delta X^2*\Delta Y^2*\Lambda - 2*\Delta X^2 - 2*\Delta Y^2]}$$

- Equation 2 – Neumann Boundary Condition at X = ax = -π:

$$U(1,j) = \frac{[\Delta X^2*\Delta Y^2*F(1,j) - 2*\Delta Y^2*U(2,j) - \Delta X^2*U(1,j-1) - \Delta X^2*U(1,j+1)]}{[\Delta X^2*\Delta Y^2*\Lambda - 2*\Delta X^2 - 2*\Delta Y^2]}$$

- Equation 3 – Neumann Boundary Condition at X = bx = π:

$$U(N+2,j) = \frac{[\Delta X^2*\Delta Y^2*F(N+2,j) - 2*\Delta Y^2*U(N+1,j) - \Delta X^2*U(N+2,j-1) - \Delta X^2*U(N+2,j+1)]}{[\Delta X^2*\Delta Y^2*\Lambda - 2*\Delta X^2 - 2*\Delta Y^2]}$$

Once equations are discretized, values along the boundaries defined by the Dirichlet Boundary Conditions are obtained. Once determined, these values will not be adjusted by Gauss-Seidel loop iterations but will impact the values obtained for the remainder of the solution matrix. Next, initial guesses of zero for all unknown values are made for the solution matrix. This initial guess along with the evaluation of the Dirichlet values are shown below in the MATLAB code in Figure 5.

```
%Performing setup for Gauss-Seidel Approximation that will solve for U values
%(unknown solution values over X and Y domains):
U = zeros(N,M);
W = zeros(N,M);
Z = 0; %Z functions as a counter for number of iterations performed during Gauss-Seidel
Error = zeros(N,M-2); %(M-2) rather than just M because the Dirichlet Boundary Conditions cause two rows to have constant values
Ea = 100; %Provides initial value of Ea, or the relative iterative error, for the Gauss_Seidel Approximation
%Solving for U values defined by Dirichlet Boundary Conditions that will
%remain constant as Gauss Seidel iterations are performed:
for i = 1:N
    U(i,1) = cos(pi() * DX * (i-1)) * cosh((2 * pi()) - (DX * (i-1))); %Dirichlet BC at Y = -pi
    U(i,M) = ((i-1) * DX)^2 * sin(((i-1) * DX) / 4); %Dirichlet BC at Y = pi
end
```

Figure 5 – Gauss-Seidel initial guesses with Dirichlet values evaluated

Using the initial guesses, the discretized equations are evaluated solving for U(i,j), the value at the point being examined. With each U(i,j) determined, the values in the initial guess matrix are substituted with the new U values and used for each successive calculation. This process continues until the maximum error of the full system iteration reaches a setpoint error value that is predetermined for adequacy and system convergence. The convergence criterion equation is shown below in Equation 4:

- Equation 4 – Gauss-Seidel Convergence Criterion Equation

$$|\varepsilon_{a,i}| = \left| \frac{X_i^j - X_i^{j-1}}{X_i^j} \right| 100\% < \varepsilon_s$$

The error is implemented in MATLAB by taking the U values for each iteration and subtracting from them the U values from the previous iteration, then dividing the resulting value by the U value from the current iteration, and finally taking the absolute value of the quotient to be compared to a selected setpoint error value. This implementation of the Gauss-Seidel approximation with the convergence criterion implemented is shown below in Figure 6.

```
%Setting the value of Es, the acceptable limit of error for system convergence:
Es = 10^-9;
Ea = 100; %Provides initial value of Ea, or the relative iterative error, for the Gauss_Seidel Approximation
%Performing Gauss Seidel Approximation to solve for U values:
if (A ~= 0)%Set condition for just in case the variable coefficient is equal to zero from a poor choice in nodes along X and Y domains, as it will function as a denominator
    while (Ea > Es)
        %Evaluating for general expression (internal nodes):
        for j = 2:MM
            for i = 2:NN
                W(i,j) = U(i,j); %W saves value of U for error calculation
                U(i,j) = (B * (cos((pi() / 2) * ((((i-1) * DX) / pi()) + 1)) * sin(((j - 1) * DY) / 2)) - (DY^2) * U(i-1,j) - (DY^2) * U(i+1,j) - (DX^2) * U(i,j-1) - (DX^2) * U(i,j+1)) / A;
                Error(i,j) = abs((U(i,j) - W(i,j)) / U(i,j)); %Computes relative error for this calculation inside this iteration
            end
            %Evaluating for U (solution) values defined by Neumann Boundary
            %Conditions that are evaluated by Gauss-Seidel Method:
            W(1,j) = U(1,j); %W saves value of U for error calculation
            U(1,j) = (- 2 * (DY^2) * U(2,j) - (DX^2) * U(1,j-1) - (DX^2) * U(1,j+1)) / A;  %cos((pi() / 2) * (0 + 1)) = 0 so Fi,j = 0, Neumann condition for X = -pi
            Error(1,j) = abs((U(1,j) - W(1,j)) / U(1,j)); %Computes relative error for this calculation inside this iteration

            W(N,j) = U(N,j); %W saves value of U for error calculation
            U(N,j) = (- 2 * (DY^2) * U(NN,j) - (DX^2) * U(N,j-1) - (DX^2) * U(N,j+1)) / A; %cos(3 * pi() / 2) = 0 so Fi,j = 0, Neumann condition for X = pi
            Error(N,j) = abs((U(N,j) - W(N,j)) / U(N,j)); %Computes relative error for this calculation inside this iteration
        end
        Ea = max(max(Error));
        Z = Z + 1; %Counts the number of loop iterations
        end
else
    disp('Select a different number of nodes for X or Y domain or change the value of C, the given constant for capital lambda.')
end
```

Figure 6 – Implementation of Gauss-Seidel Approximation with Convergence Criterion

Successive-Over-Relaxation (SOR)

After performing the steps required to implement the Gauss-Seidel method of approximation, the Successive-Over-Relaxation (SOR) was applied. The SOR method of approximation, once an ideal relaxation coefficient is selected between the values of 1 and 2, is a quicker approximation method for discretized systems, as it will converge to the approximated solutions faster than the unrelaxed system defined by the basic Gauss-Seidel method. To implement the SOR method of approximation, the Helmholtz partial differential equation was first discretized as done in the Gauss-Seidel method shown in Equations 1, 2, and 3. Next, initial guesses of zero were made for the first iteration of the SOR approximation and the Dirichlet Boundary Conditions were applied as shown in the Gauss-Seidel code sample in Figure 5. Once the initial and constant values of the U solution matrix were set, an error limit defined by the convergence criterion was then defined by Equation 4, as seen in the Gauss-Seidel approximation, and given a setpoint value of 10^-9. Finally, the portion that truly differentiates the SOR method from the unrelaxed Gauss-Seidel method is that within every iteration solving for every variable value in the U solution matrix, the U values are modified by a weighted average of the results of the previous and the current iterations. This application of weighted over-relaxation is observed with the general relaxation equation shown below in Equation 5, with a relaxation coefficient value set between 1 and 2, whichever produces the quickest convergence.

- Equation 5 – SOR Weighted Average Equation, Assuming Relaxation Coefficient (λ) between values 1 and 2:

$$x_i^{\text{new}} = \lambda x_i^{\text{new}} + (1 - \lambda) x_i^{\text{old}}$$

Utilizing the SOR weighted average equation for all iterations performed with the iterative approximation being applied gives quicker convergence and, by association, quicker computations. The SOR method with the SOR weighted average equation is integrated into MATLAB as shown below in Figure 7.

```
%Performing SOR Approximation to solve for U values:
if (A ~= 0)%Set condition for just in case the variable coefficient is equal to zero from a poor choice in nodes
          %along X and Y domains, as it will function as a denominator
    while (Ea > Es)
        %Evaluating for general expression (internal nodes):
        for j = 2:MM
            for i = 2:NN
                W(i,j) = U(i,j); %W saves value of U for error calculation

                %Evaluates starting from bottom left corner of domain:
                U(i,j) = (B * F(i,j) - (DY^2) * U(i-1,j) - (DY^2) * U(i+1,j) - (DX^2) * U(i,j-1) - (DX^2) * U(i,j+1)) / A;

                U(i,j) = (G * U(i,j)) + ((1-G) * W(i,j)); %Applying SOR iteration

                %Computes relative error for this calculation inside this iteration
                Error(i,j) = abs((U(i,j) - W(i,j)) / U(i,j));

                end

            %Evaluating for U (solution) values defined by Neumann Boundary
            %Conditions that are evaluated by SOR Method
            W(1,j) = U(1,j); %W saves value of U for error calculation
            U(1,j) = (B * F(1,j) - 2 * (DY^2) * U(2,j) - (DX^2) * U(1,j-1) - (DX^2) * U(1,j+1)) / A;  %Neumann condition for X = -pi
            U(1,j) = (G * U(1,j)) + ((1-G) * W(1,j)); %Applying SOR iteration
            Error(1,j) = abs((U(1,j) - W(1,j)) / U(1,j)); %Computes relative error for this calculation inside this iteration

            W(N,j) = U(N,j); %W saves value of U for error calculation
            U(N,j) = (B * F(N,j) - 2 * (DY^2) * U(NN,j) - (DX^2) * U(N,j-1) - (DX^2) * U(N,j+1)) / A; %Neumann condition for X = pi

            U(N,j) = (G * U(N,j)) + ((1-G) * W(N,j)); %Applying SOR iteration
            Error(N,j) = abs((U(N,j) - W(N,j)) / U(N,j)); %Computes relative error for this calculation inside this iteration
        end
    end
end
```

Figure 7 – Implementation of SOR Approximation in MATLAB

After implementing the basic structure for the SOR code, the full range of relaxation coefficients from 1 to 2 were tested for quickness of convergence. This was done by generating a for-loop that inserted values scaling up from 1 by increments of 0.05. The results for a 100X100 mesh grid are displayed below in Table 1, in which the ideal value for the relaxation coefficient with this mesh size, 1.9, is highlighted in green.

Table 1 - Test for Ideal SOR Coefficient with 100X100 Mesh Size

Relaxation Coefficient	Time of Approximation
1	9.2222136
1.05	8.40446972
1.1	7.64724856
1.15	6.9212541
1.2	6.28950086
1.25	5.70094489
1.3	5.14577175
1.35	4.64369938
1.4	4.13969877
1.45	3.68409033
1.5	3.2714742
1.55	2.85030597
1.6	2.50112787
1.65	2.10736639
1.7	1.76731814
1.75	1.46104753
1.8	1.13408159
1.85	0.81031408
1.9	0.47300212
1.95	0.83218
2	No convergence in time allowed

The results for the different values of the relaxation coefficient from 1 to 2 for a 200X200 mesh size are shown below in Table 2.

Table 2 - Test for Ideal SOR Coefficient with 200X200 Mesh Size

Relaxation Coefficient	Time of Approximation
1	132.309592
1.05	120.849307
1.1	109.012133
1.15	98.7280522

1.2	89.5213704
1.25	80.6954423
1.3	74.5548862
1.35	65.7620497
1.4	58.6176469
1.45	51.5626606
1.5	45.5860608
1.55	40.0306645
1.6	34.673606
1.65	29.6670682
1.7	24.8935006
1.75	20.4083843
1.8	16.0617459
1.85	11.8213283
1.9	7.62401163
1.95	3.46041754
2	No convergence in time allowed

Comparing the results from the two relaxation coefficient tests with 100X100 mesh and 200X200 mesh, it was determined that the ideal relaxation coefficient value would lie between 1.9 and 1.95 for this system. Changing the relaxation coefficient value from 1.9 to 1.95 in the case with the 200X200 mesh size produced a speed performance improvement of 54.6%. Changing the relaxation coefficient value from 1.9 to 1.95 in the case with the 100X100 mesh size produced a speed performance detriment of 43.4%. As such, a relaxation coefficient value of 1.93 was selected to be the ideal choice, as the speed performance leaned slightly in favor of the higher relaxation coefficient value.

Verification

To verify the validity of the Gauss-Seidel and SOR discretized approximations of the Helmholtz equation, the Method of Manufactured Solutions was utilized. An assumed value of U was used as the initial guess input for both of the approximations. This value of U was also inserted into the Helmholtz partial differential equation and used to solve for F(X,Y). This

expression for F(X,Y) was then included in both of the approximations. With both the U and F(X,Y) expressions substituted as inputs for the approximations, the MATLAB codes were run and compared to the original solutions with a lambda (Λ) coefficient of -1.5. The setup described is shown below in Figure 8.

Method of Manufactured Solutions

- For the Helmholtz partial differential equation:

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

- A guess value for u is made:

$$u = 1 + x^2 + 2y^2$$

- Solving for $F(x,y)$ yields:

$$F(x,y) = 2 + 4 + \lambda(1 + x^2 + 2y^2)$$

$$\Rightarrow F(x,y) = 6 + \lambda(1 + x^2 + 2y^2)$$

- The expressions for u and $F(x,y)$ are then substituted into the Gauss-Seidel and SOR approximation codes and compared to the original approximations to test for validity.

Figure 8 – Method of Manufactured Solutions Setup

The MATLAB code utilized to implement the Method of Manufactured Solutions for both the Gauss-Seidel and SOR approximation methods is shown below in Figure 9.

```
%Performing setup for Gauss-Seidel Approximation that will solve for U values
%(unknown solution values over X and Y domains):
U = zeros(N,M); %Provides initial guesses and preallocates for optimization
for i = 1:N
    for j = 1:M
        U(i,j) = 1 + X(i,j)^2 + 2 * Y(i,j)^2; %To be used only for method of manufactured solutions
    end
end
W = zeros(N,M);
Z = 0; %Z functions as a counter for number of iterations performed during Gauss-Seidel
Error = zeros(N,M-2); %(M-2) rather than just M because the Dirichlet Boundary Conditions cause two rows to have
%constant values and therefore will have an error of 0 per iteration (optimal to exclude unnecessary repeated calculations
Ea = 100; %Provides initial value of Ea, or the relative iterative error, for the Gauss_Seidel Approximation

%Evaluating for F values:
F = zeros(N,M);
for i = 1:N
    for j = 1:M
        F(i,j) = 6 + C * (1 + X(i,j)^2 + 2 * Y(i,j)^2); %To be used only for Method of Manufactured Solutions
        %F(i,j) = 0; %To be commented out unless simulating Laplace or debugging
        %F(i,j) = cos((pi/2) * (2 * ((X(i,j) - ax) / (bx - ax)) + 1)) * sin(pi * ((Y(i,j) - ay) / (by - ay)));
    end
end
```

Figure 9 – MATLAB code for Method of Manufactured Solutions

The results of the Gauss-Seidel approximation with a 64X64 mesh with lambda equal to -1.5 is shown below with the original inputs of U and F(X,Y) described by the problem statement in Figure 10.

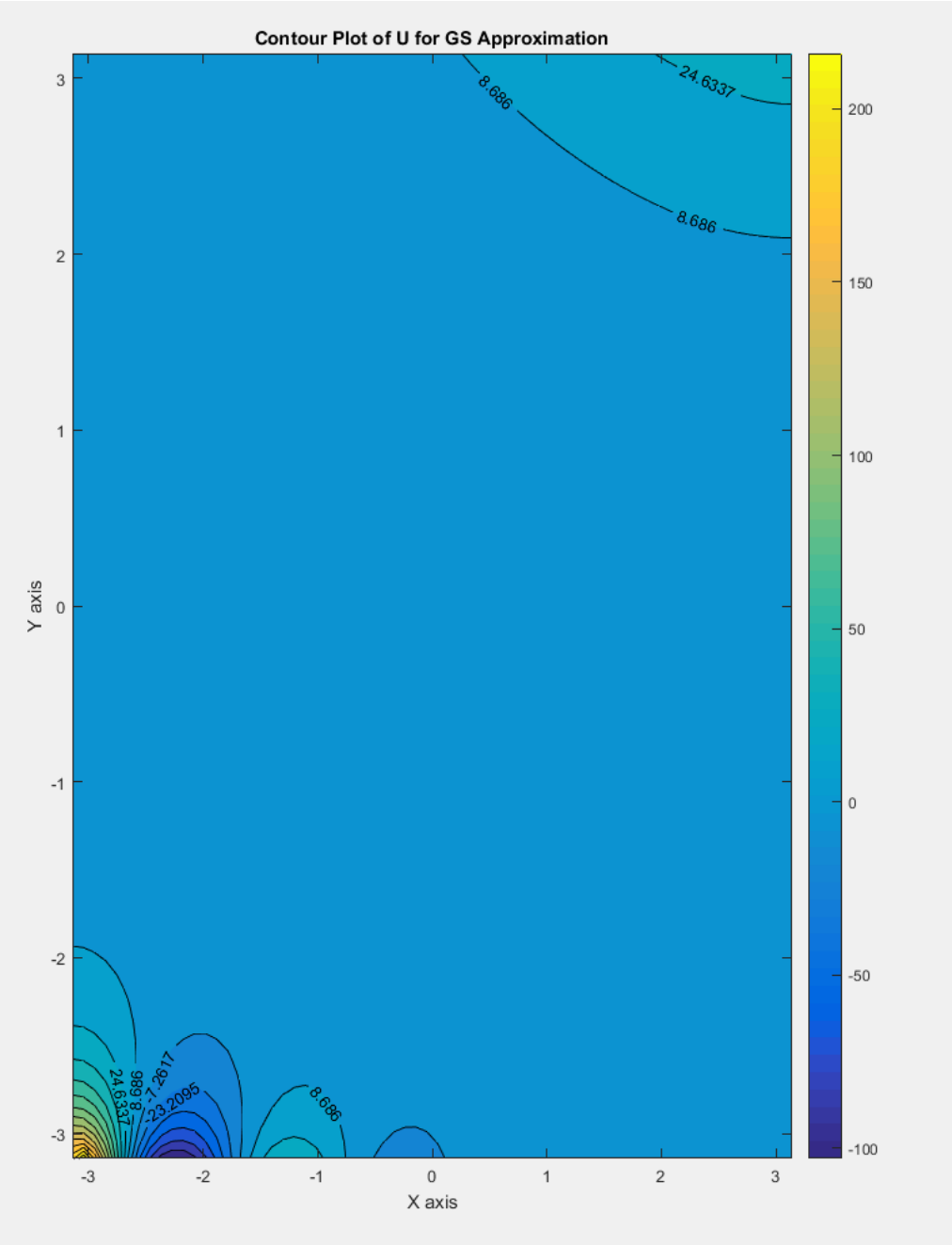


Figure 10 – Contour Plot of Original Gauss-Seidel Approximation

The results of the Gauss-Seidel approximation with the Method of Manufactured Solutions expressions included as system inputs is shown below in Figure 11 with a lambda value of -1.5 and a 64X64 mesh size.

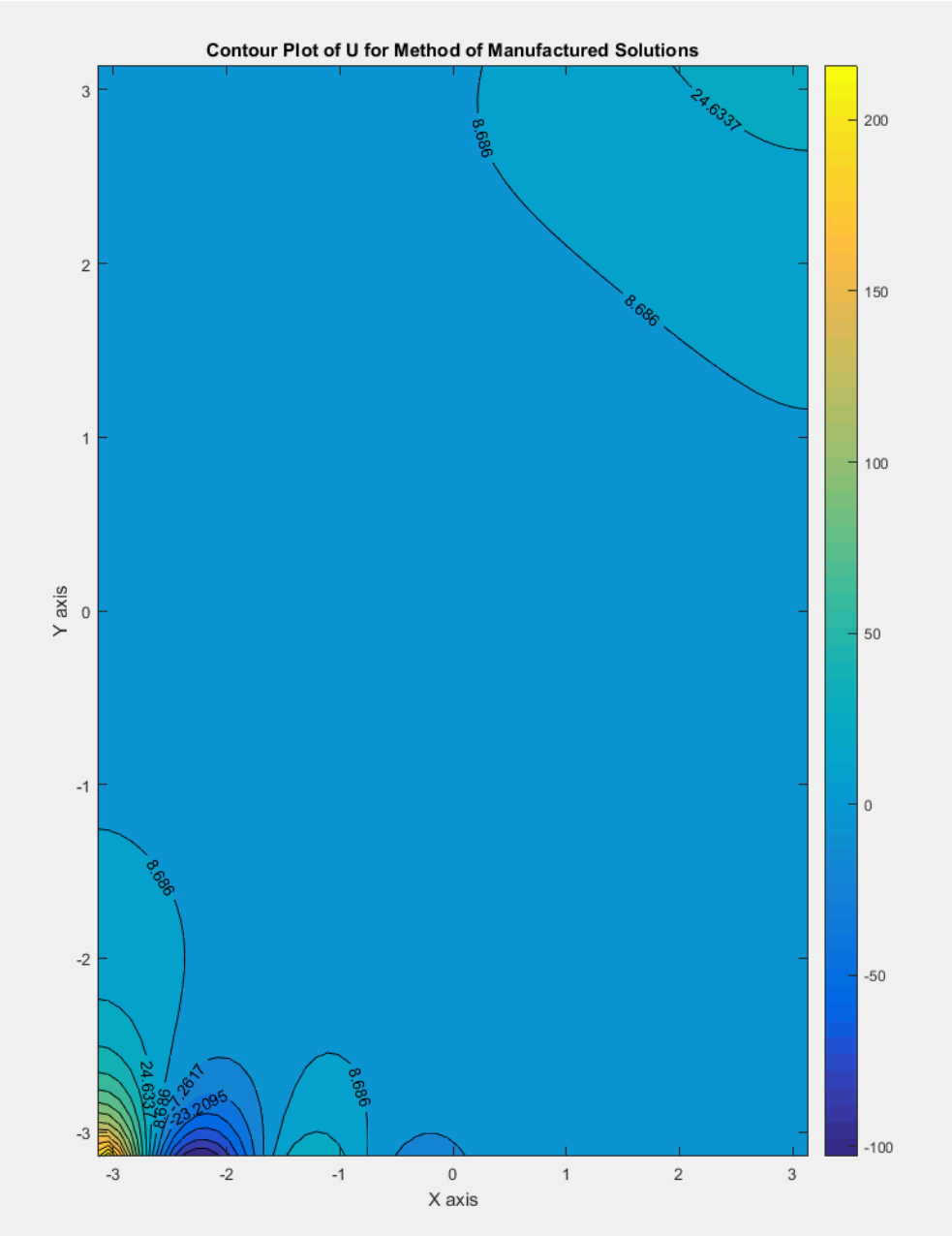


Figure 11 – Contour Plot of Method of Manufactured Solutions Gauss-Seidel Approximation

Comparing the results from both the original Gauss-Seidel approximation and the Method of Manufactured Solutions Gauss-Seidel approximation verifies that the Gauss-Seidel discretized solution to the given Helmholtz equation is a good approximation for the solution to the Helmholtz partial differential equation, as they both exhibit identical behavior and values. The results of the SOR approximation with a 64X64 mesh with lambda equal to -1.5 is displayed below with the original guesses for U and the values of F(X,Y) given by the problem statement in Figure 12.

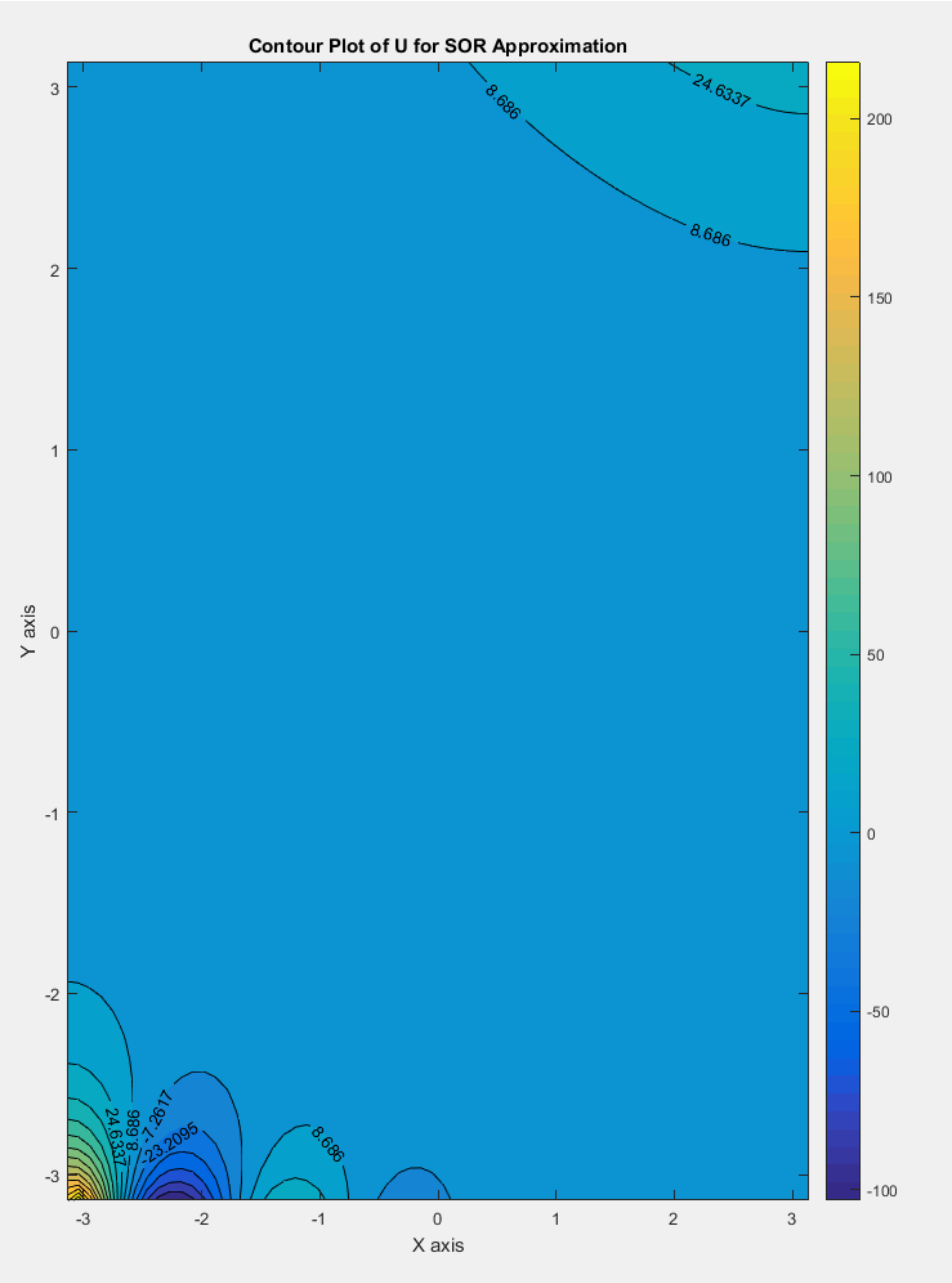


Figure 12 – Contour Plot of Original SOR Approximation

The results of the SOR approximation with the Method of Manufactured Solutions expressions substituted into the system as inputs is represented below in Figure 13 with a value of lambda set to -1.5 and a 64X64 mesh grid size.

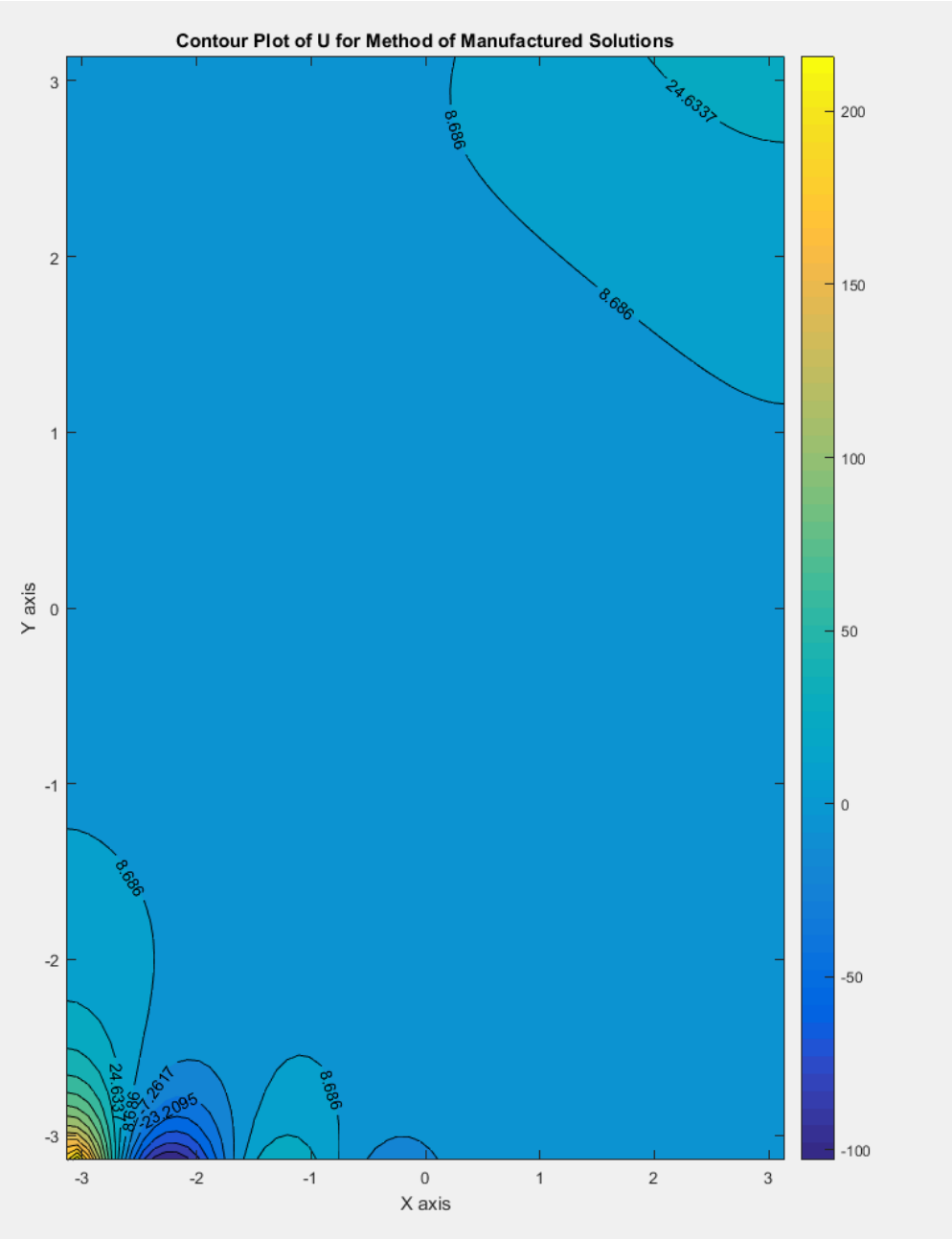


Figure 13 – Contour Plot of Method of Manufactured Solutions SOR Approximation

Comparing the results from both the original SOR approximation and the Method of Manufactured Solutions SOR approximation with relaxation coefficients of 1.93, it was determined that the SOR approximation was satisfactory as the values in the primary region of the U solution matrix were in accordance with each other. The SOR approximation was further validated as the plots of the U solution matrices for both the Gauss-Seidel and SOR approximations showed the same values in the main converging regions, and the Gauss-Seidel had already been verified to be an appropriate approximation to the Helmholtz equation being studied.

Grid Independence

A grid independence study was performed to determine that neither increasing nor decreasing the mesh size would impact the system's solution results in a significant way. This was done by a standard statistical approach, taking the mean of the mean of the solution matrix squared element by element, and then comparing the values for various grid sizes. The sizes selected were 32X32, doubled iteratively until a mesh size of 256X256, and 100X100, doubled iteratively until a mesh size of 300X300. The results from the grid independence study are shown below in Table 3, with a Helmholtz equation lambda coefficient selected to be -1.5.

Table 3 – Grid Independence Study with Lambda = -1.5

Grid Independence Study with Lambda = -1.5			
Gauss-Seidel		Successive-Over-Relaxation	
Internal Nodes for Square Mesh	Mean of U^2	Internal Nodes for Square Mesh	Mean of U^2
32	133.5140749	32	133.5140749
64	125.0878146	64	125.0878146
100	116.9279953	100	116.9279953
128	113.0461685	128	113.0461685
200	107.5285689	200	107.5285689
256	105.2383769	256	105.2383769
300	104.0097181	300	104.0097181

The data presented in Table 3, above, was then plotted to check if the system values would converge to a certain value, indicating that the grid size was sufficiently independent of the solution. The grid convergence plot is shown below in Figure 14.

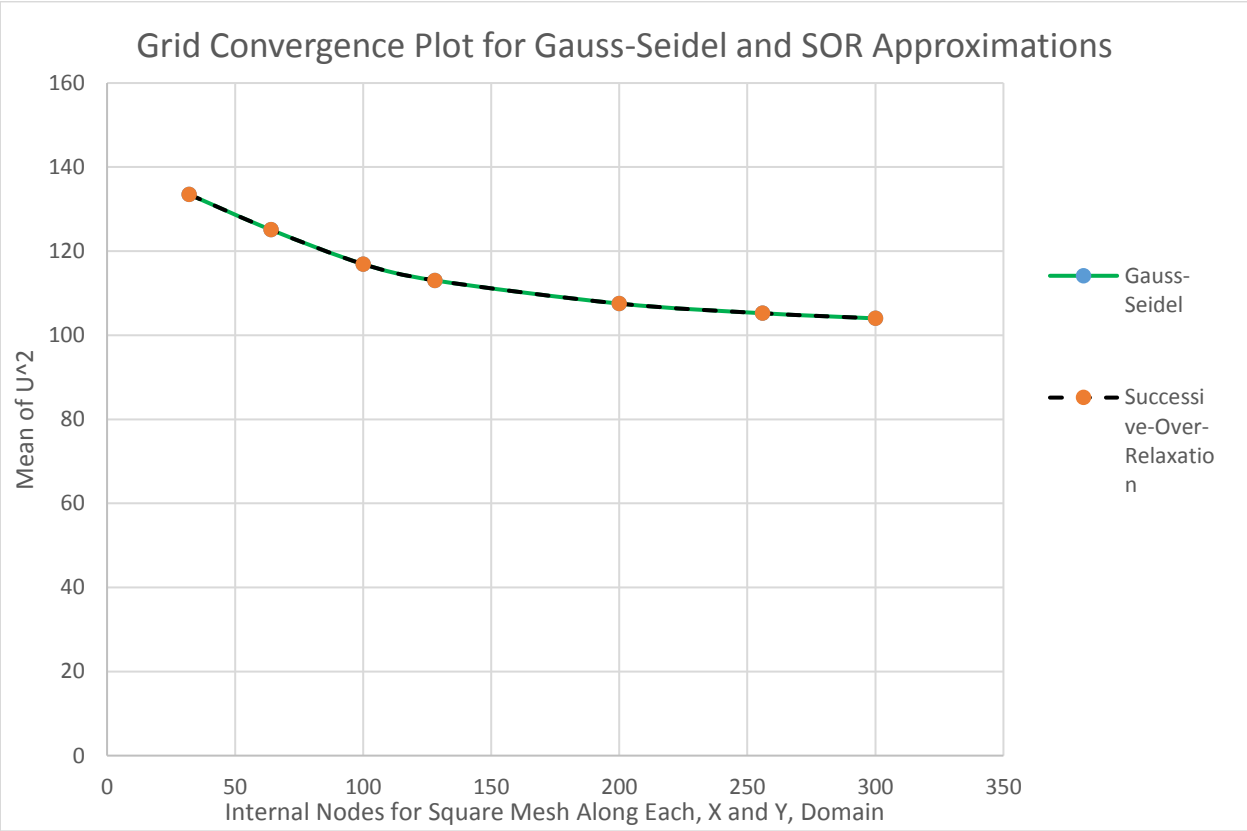


Figure 14 – Grid Convergence Plot for Gauss-Seidel and SOR Approximation Methods

The grid size converged just above the value of 100, with nearly no slope on the convergence plot. Therefore, the solution was deemed independent of the grid size and was classified as acceptable for approximations.

Visualizations and Discussion of Results

Technical Specifications

Using different computer hardware can have a significant effect on the performance speed of running memory intensive programs such as the Gauss-Seidel and SOR approximations. As such, the computer specifications, as were available using the computer at the public library where command prompt is disabled, are shown below in Figure 6.

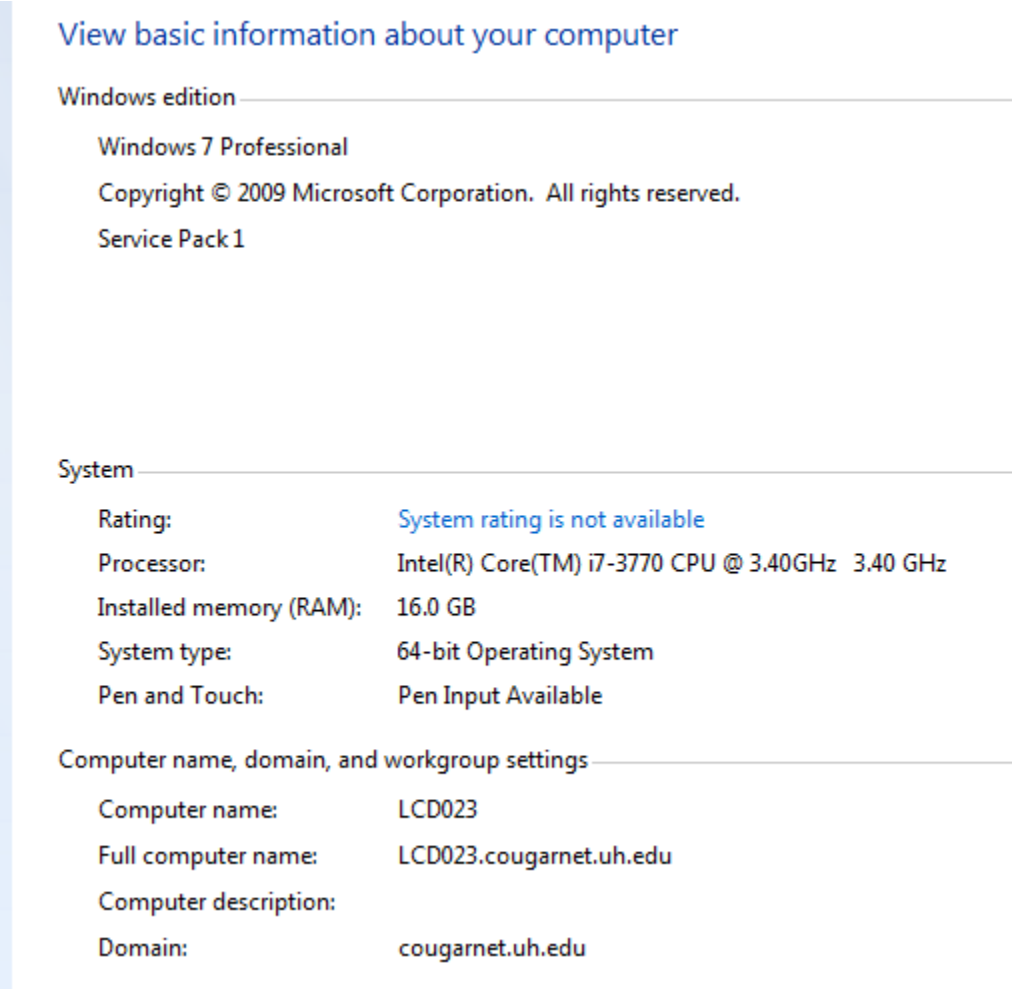


Figure 6 – Computer Technical Specifications

Gauss-Seidel

The Helmholtz equation with lambda equal to 1.5 was evaluated via Gauss-Seidel approximation. However, using this value of lambda, the system proved divergent and values across the entire solution domain approached infinity. As such, after debugging code, multi-checking with peer review, isolating boundary conditions, F matrix values, setting lambda equal to zero, and running the same Gauss-Seidel code for the problem statement conditions for the Poisson's equation, APc2-1, to see if the results generated the correct plot, it was determined that the lambda value being altered was the only way to produce convergent and correct solutions. As such, the results of different lambda values being tested with are discussed in this section.

For Lambda = 1.5:

Travis Franks
PSID: 1372696
Project Code: AHc2-6

Using the original problem statement for the Helmholtz equation (AHc2-6) provided a value of 1.5 for the lambda coefficient. However, shown using a simple 20X20 node mesh in Table 1 below, it is apparent that the system is divergent with approximated values of U approaching infinity.

Table 1 – U matrix for Lambda Equal to 1.5

		U Matrix for Lambda = 1.5																					
		Y axis																					
		-3.14159	-2.84239	-2.54319	-2.24399	-1.9448	-1.6456	-1.3464	-1.0472	-0.748	-0.4488	-0.1496	0.1496	0.448799	0.747998	1.047198	1.346397	1.645596	1.944795	2.243995	2.543194	2.842393	3.141593
X axis	-3.14159	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	1.29E+307
	-2.84239	117.0855	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	0.00669
	-2.54319	-44.7761	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	0.053369
	-2.24399	-103.523	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	0.179281
	-1.9448	-65.9289	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	0.422185
	-1.6456	-0.75445	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	0.817635
	-1.3464	35.58387	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	1.398289
	-1.0472	31.53953	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	2.193245
	-0.748	8.023708	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	3.227426
	-0.4488	-10.3263	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	4.521011
	-0.1496	-13.4518	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	6.088921
	0.1496	-6.09197	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	7.940379
	0.448799	2.07896	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	10.07852
	0.747998	5.193179	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	12.5001
	1.047198	3.417871	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	15.19525
	1.346397	0.116692	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	18.14736
	1.645596	-1.83929	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	21.333
	1.944795	-1.73973	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	24.72195
	2.243995	-0.50322	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	28.27735
	2.543194	0.649625	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	31.95585
	2.842393	1.043773	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	35.70796
	3.141593	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf

The divergence of the Helmholtz equation system can be further observed through the extreme magnitude (infinite) of the values of U plotted on the shaded contour map of U shown below in Figure 7.

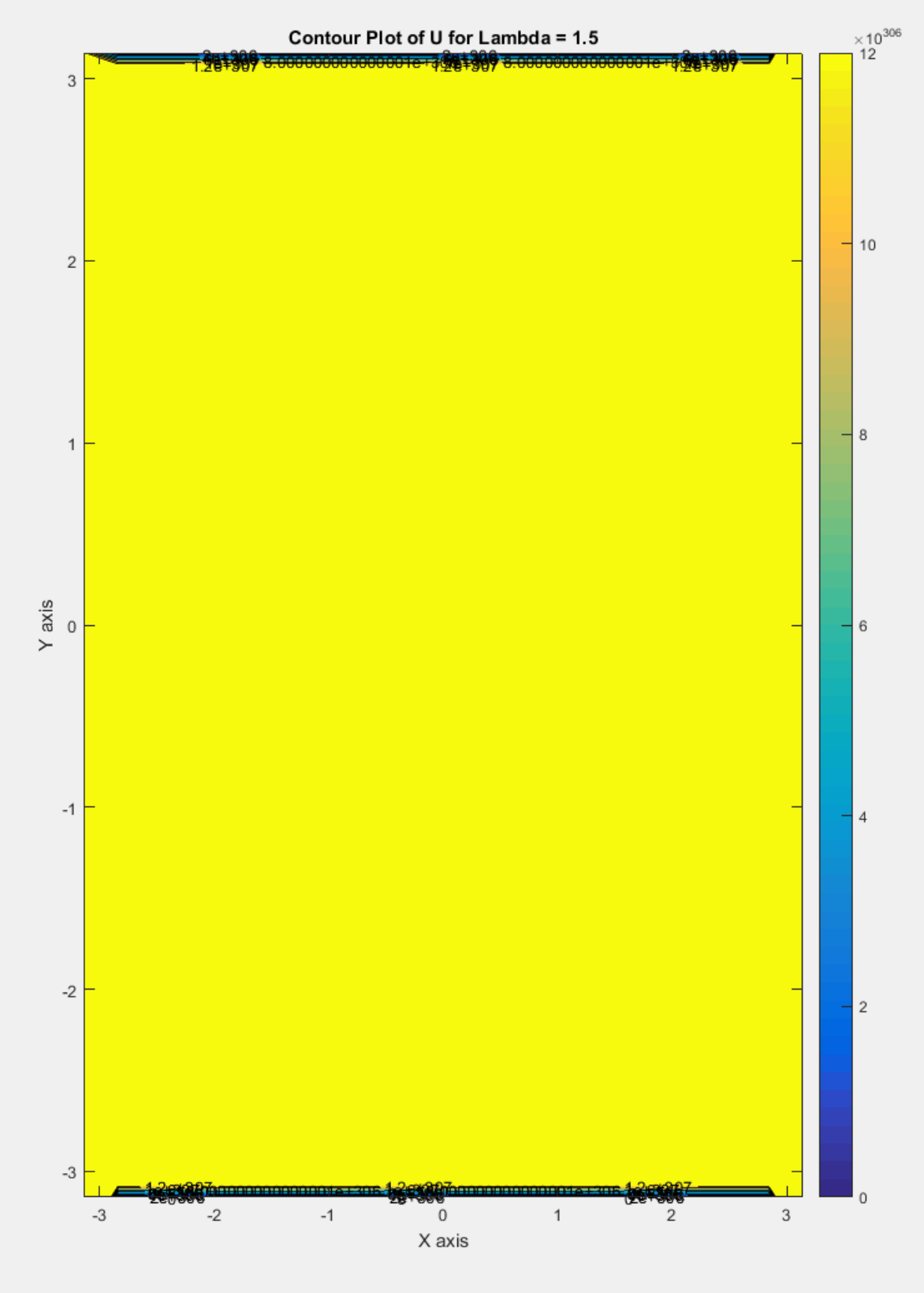


Figure 7 – Contour Plot of U for lambda = 1.5

For Lambda = 0:

In order to verify that the code was functional but that the lambda value was an issue of concern, the lambda value was set equal to zero, forcing it to behave as a Poisson's equation. The results of a 64X64 mesh are shown below in Figures 8 and 9 with a shaded contour plot of the U solution and a surface plot of the U solution respectively.

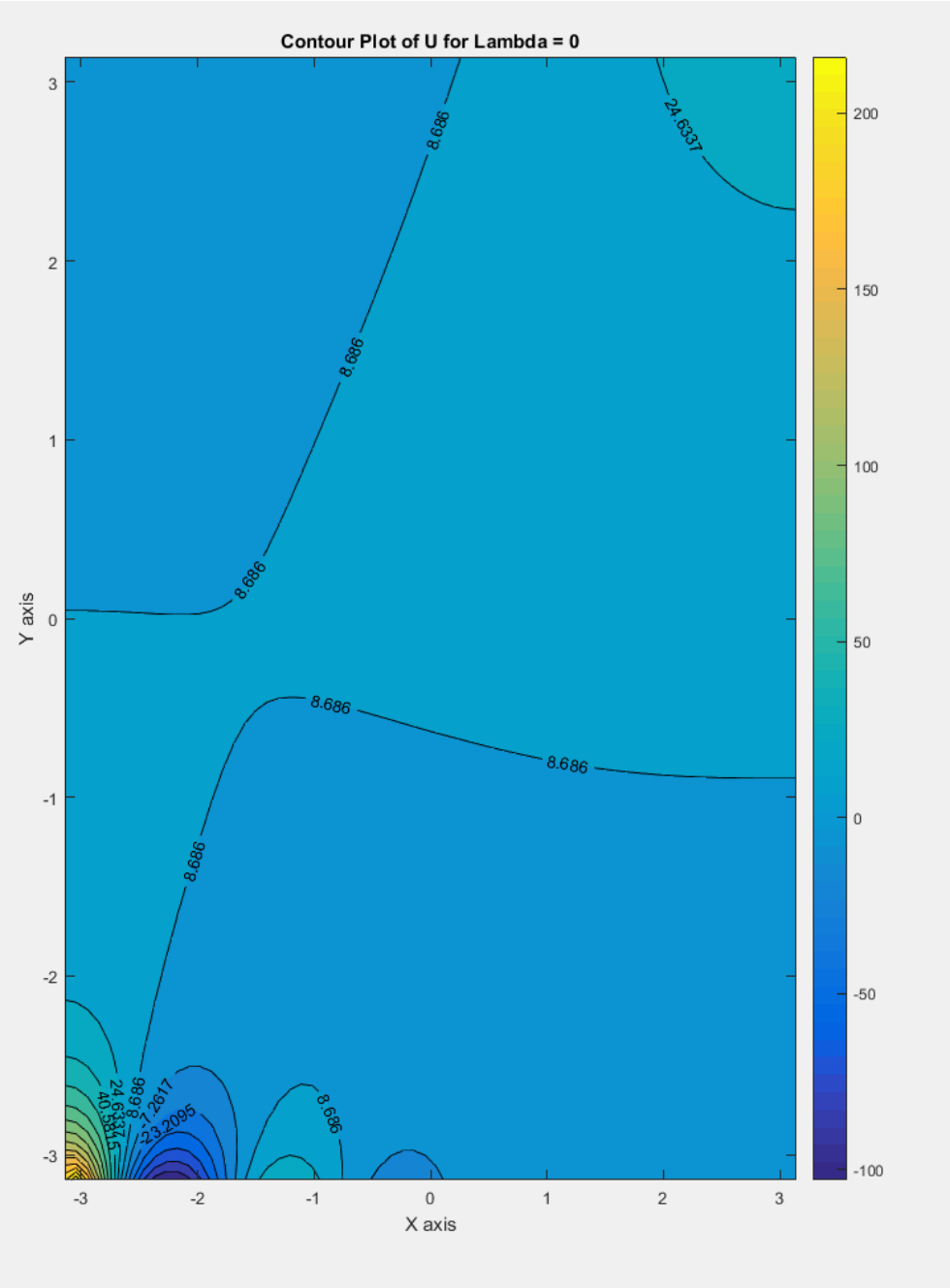


Figure 8 – Contour Plot of U for Lambda = 0

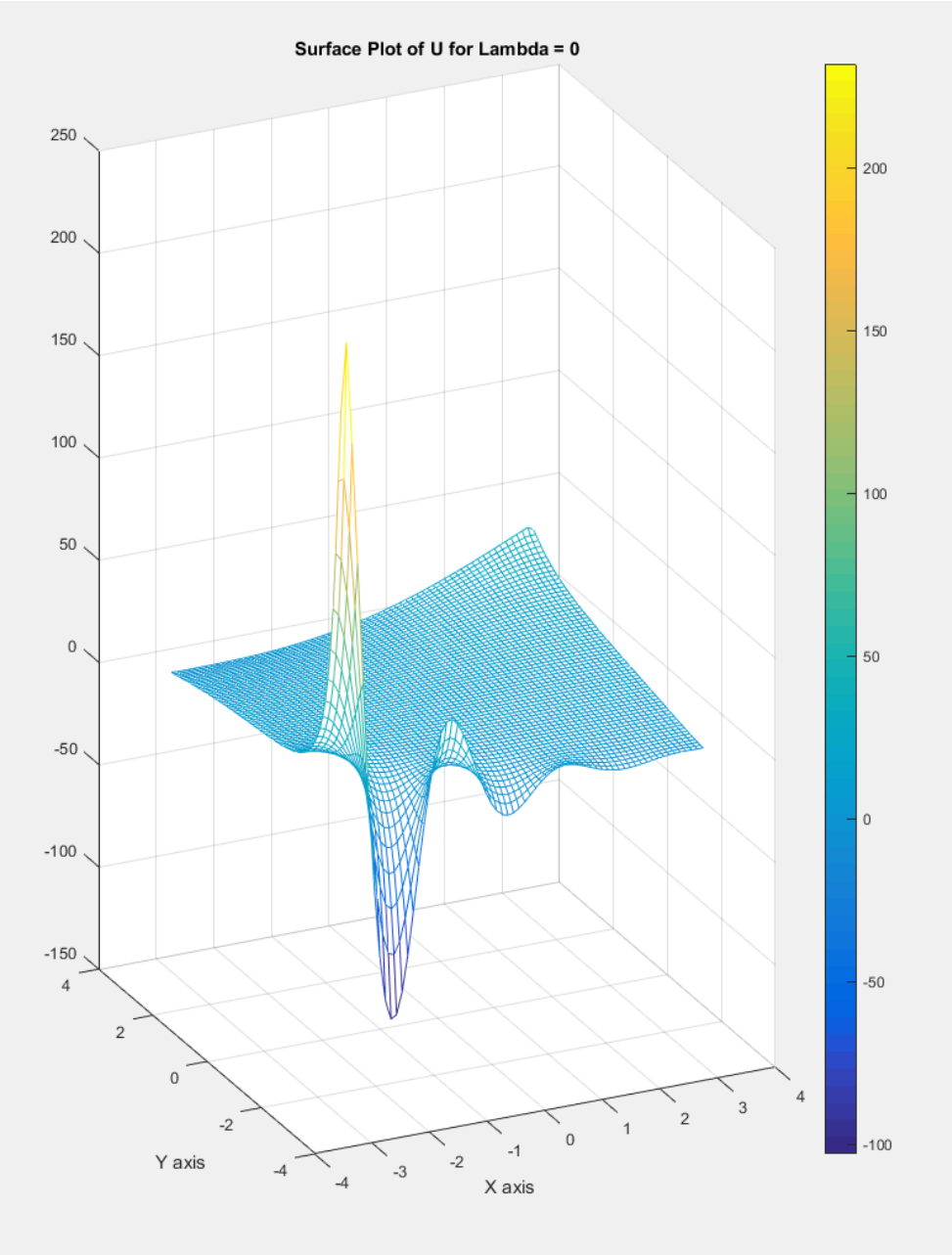


Figure 9 – Surface Plot of U for Lambda = 0

These plots of the U values for lambda set equal to zero show a convergent solution to the partial differential equation being evaluated by the Gauss-Seidel approximation method. This suggests that the lambda value plays a significant role in the possibility of convergence for the discretized Helmholtz equation when applying the Gauss-Seidel method of approximation.

For Lambda = -1.5:

To observe the behavior of the discretized Helmholtz equation system after altering the sign of the lambda coefficient, the value of the lambda coefficient was then altered to -1.5. This

produced a convergent solution similar in behavior to that of lambda set equal to zero, but with smoother gradients along the entire domain and a flatter surface in the main region of the plots. The results of setting the lambda value equivalent to -1.5 with a 64X64 mesh are shown below in Figures 10 and 11 below.

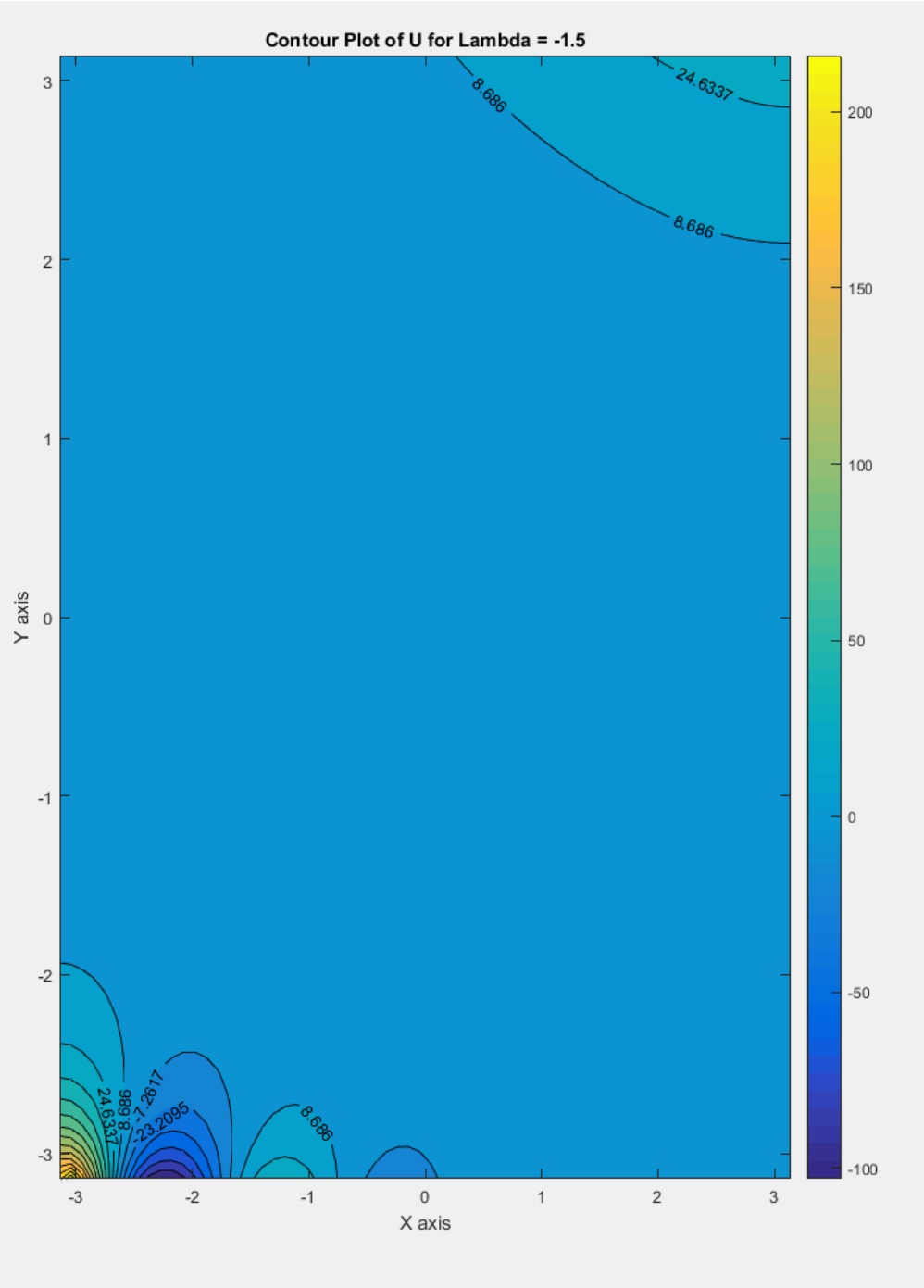


Figure 10 – Contour Plot of U for Lambda = -1.5

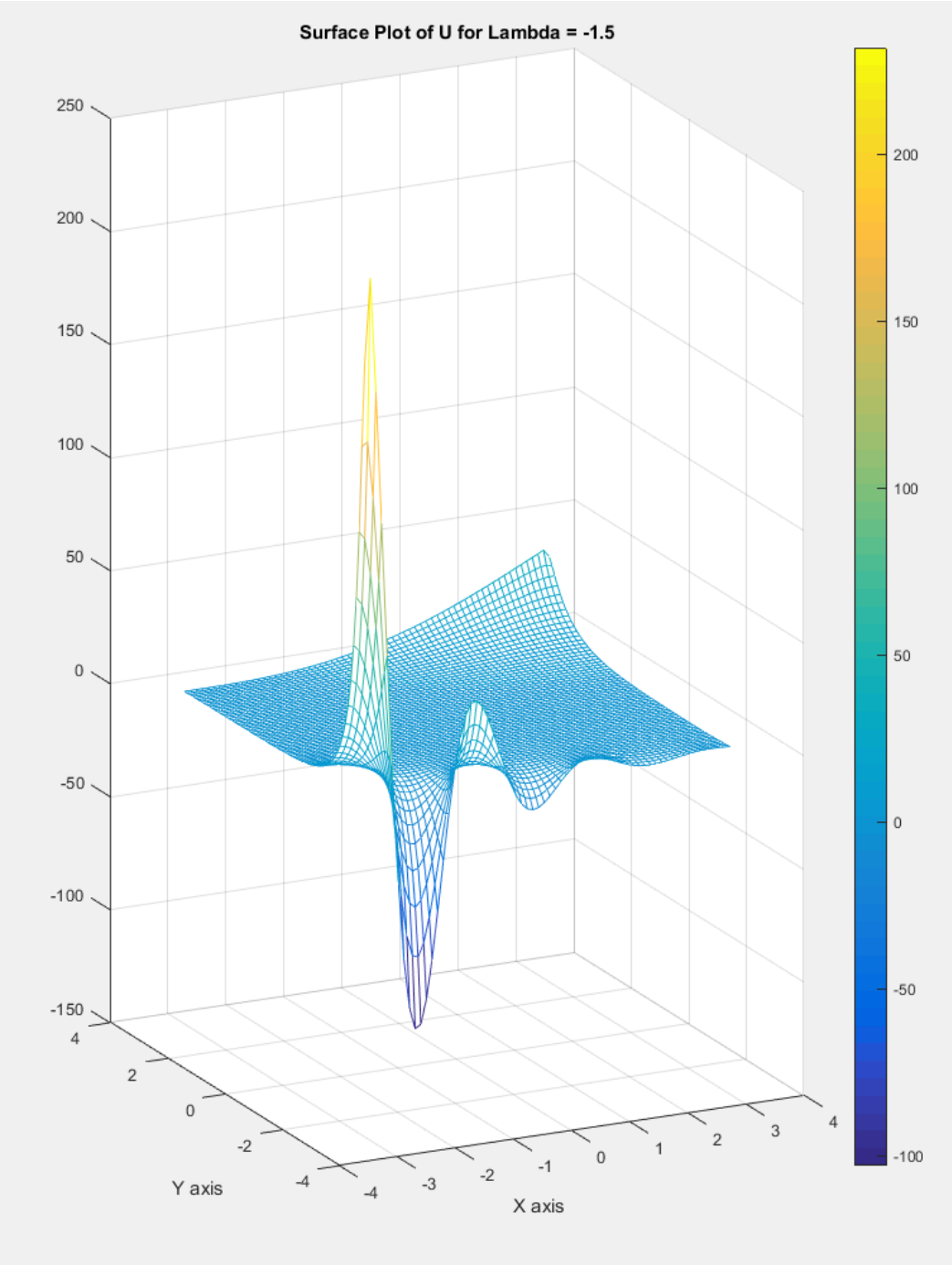


Figure 11 – Surface Plot of U for Lambda = -1.5

For Lambda = 0.2:

To further determine the effects of altering the lambda value of the Helmholtz equation, several evaluations were performed by increasing the value of lambda from zero until the system was divergent. Then, a value of lambda close to where the system diverges was selected and was found to be 0.2. The solution matrix of this system with lambda set equal to 0.2 rapidly

increased in value towards the center of the domain from the boundaries defined by the Dirichlet Boundary Conditions defined along the X axis at both ends of the Y axis and exhibited a smooth curve in doing so. The resulting plots for mesh grids of 64X64 are shown below in Figures 12 and 13.

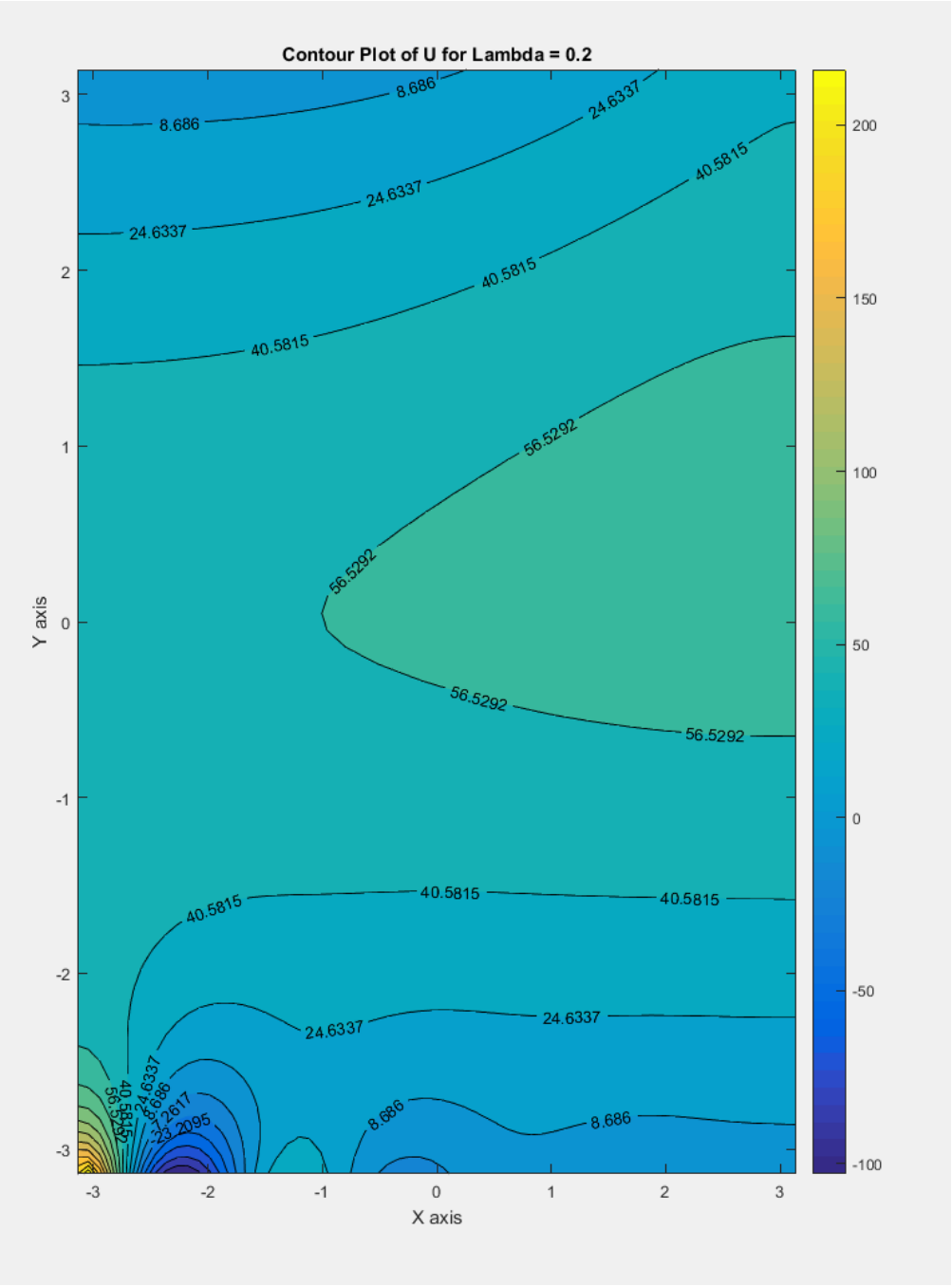


Figure 12 – Contour Plot of U for Lambda = 0.2

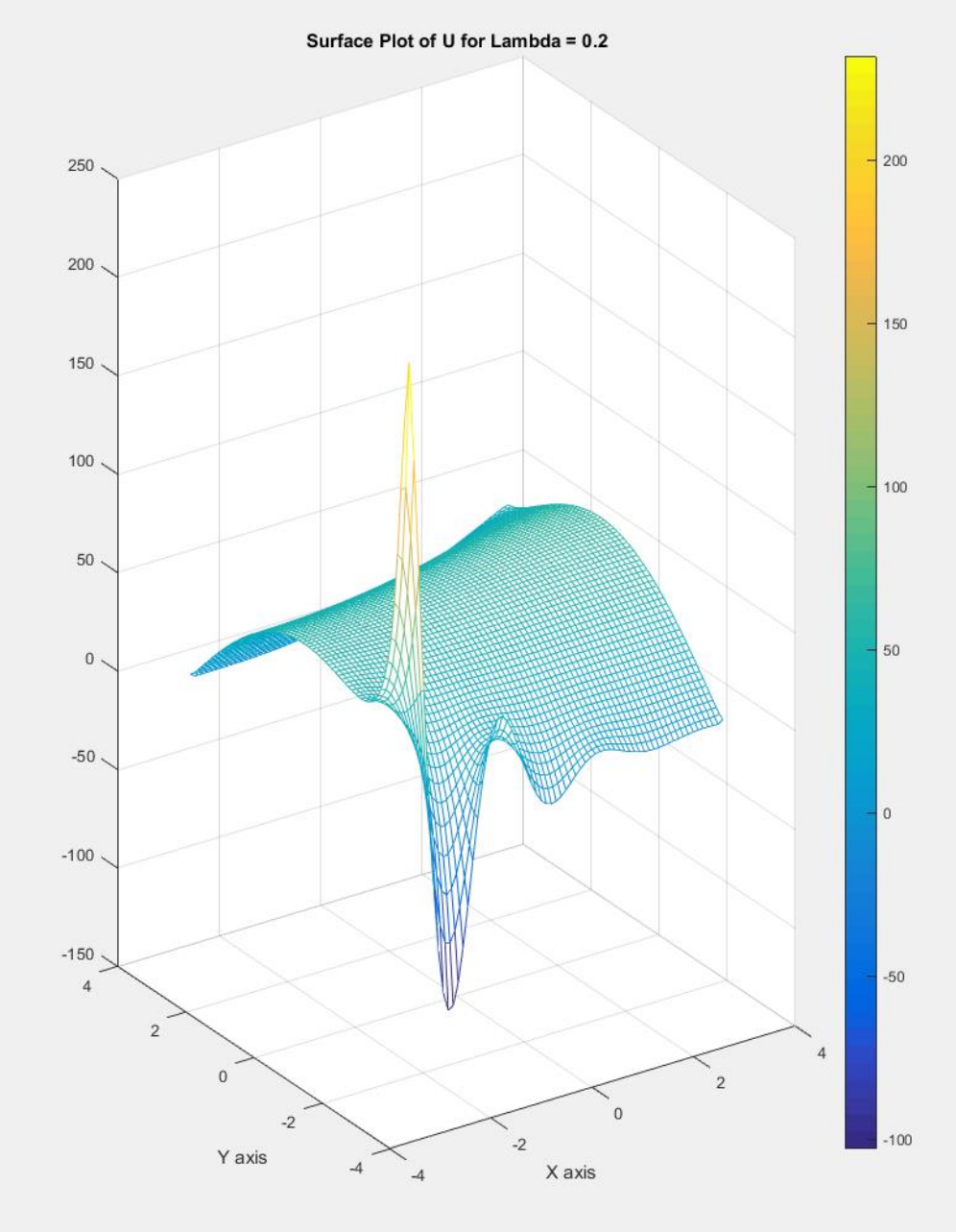


Figure 13 – Surface Plot of U for Lambda = 0.2

For Special Test Case with F = 0 and lambda (Λ) = 0:

A control test case for both F and lambda being set equal to zero was performed. This simulates a Laplace equation in functionality and provides a basis of comparison for previous test results for a more thorough investigation of results. The results are nearly identical in behavior to that of lambda set equal to zero with a non-zero F value but have corner slopes

with less steep gradients. The plots of a 64X64 mesh size are displayed below in Figures 14 and 15.

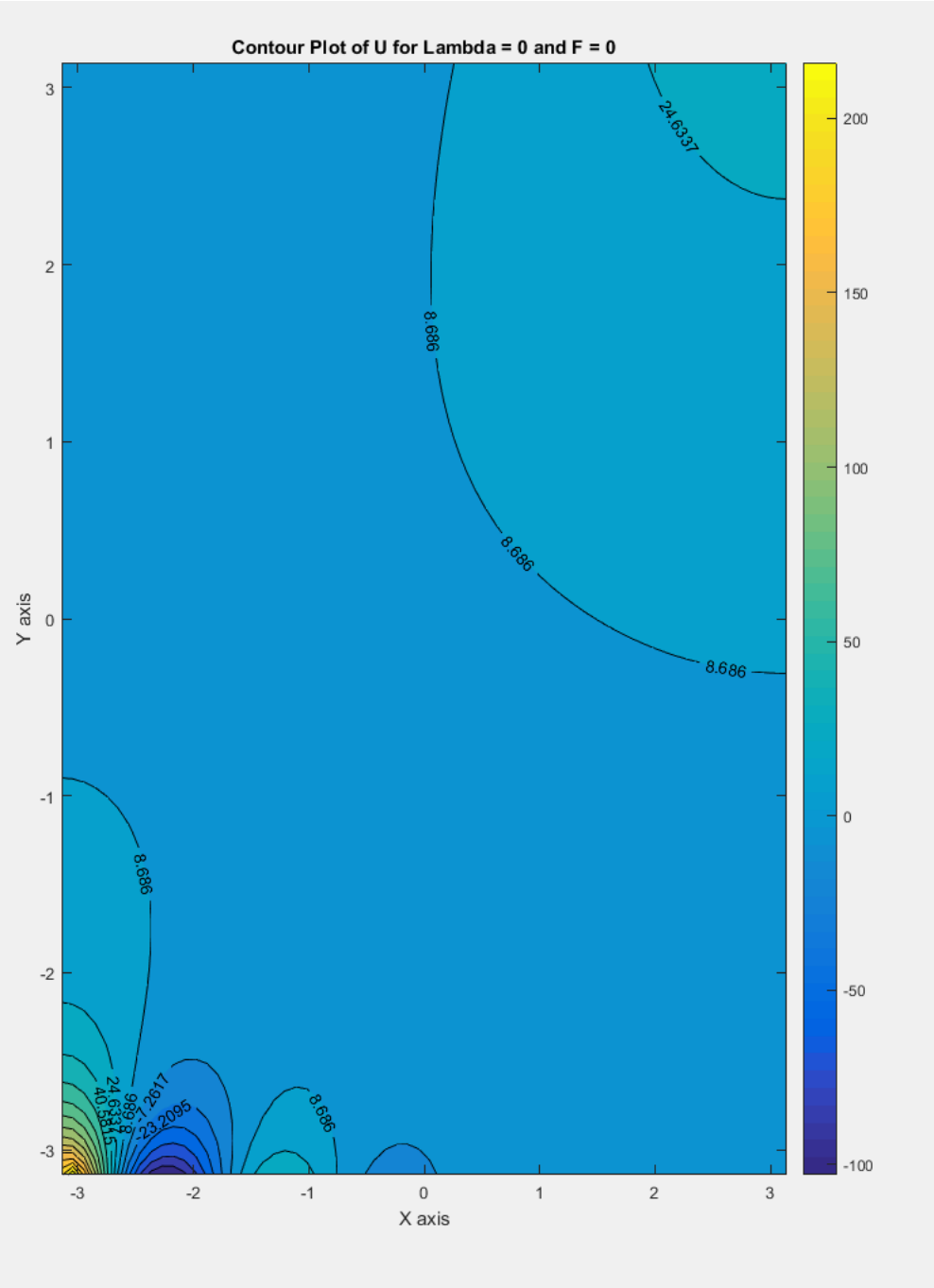


Figure 14 – Contour Plot of U for Lambda = 0 and F = 0

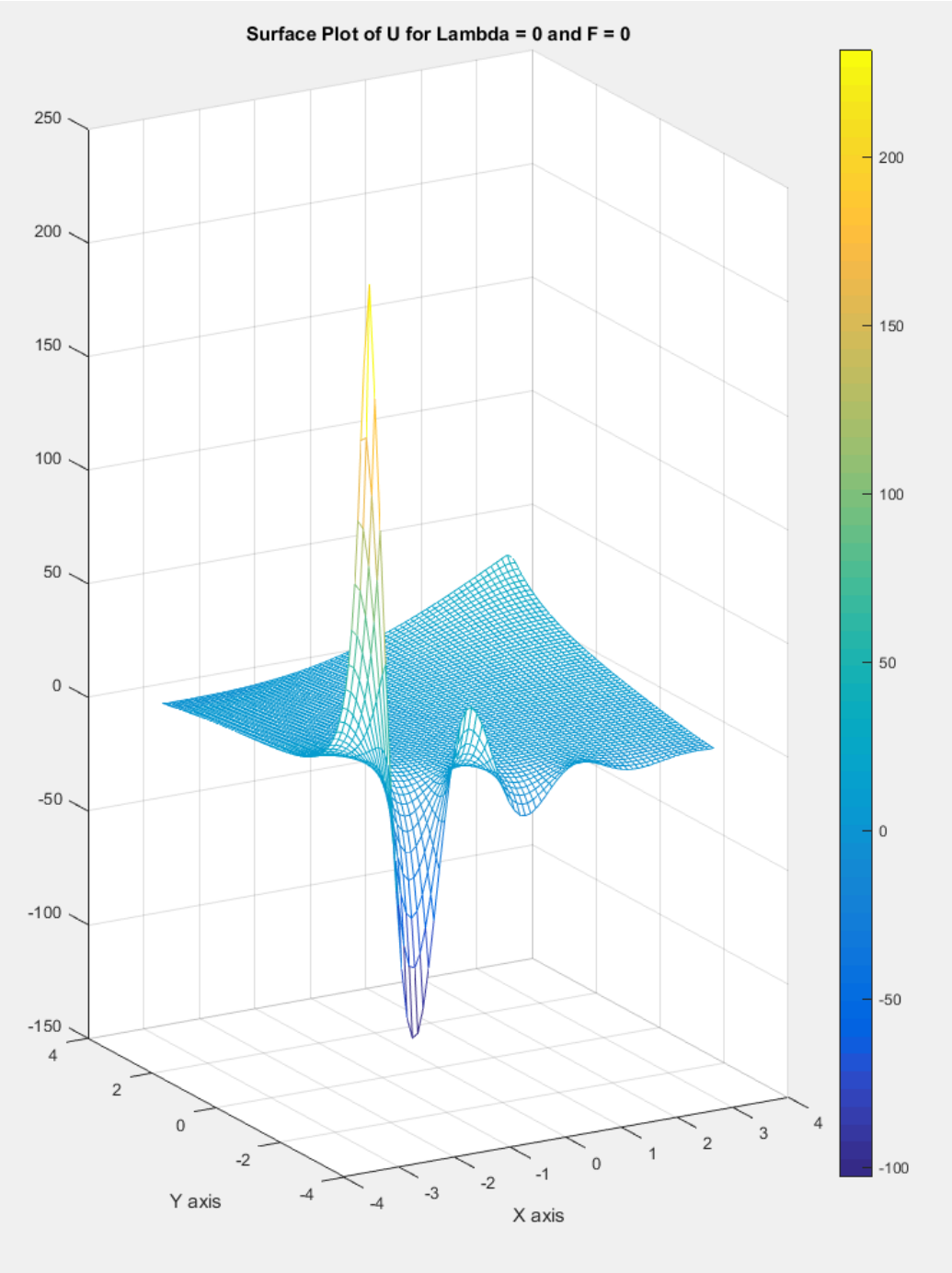


Figure 15 – Surface Plot of U for Lambda = 0 and F = 0

Successive-Over-Relaxation (SOR)

The Helmholtz equation with a lambda coefficient of value equal to -1.5 was then evaluated using the Successive-Over-Relaxation (SOR) iterative approximation method. The purpose of applying the SOR method of approximation versus the Gauss-Seidel method of approximation us that the SOR method converges significantly faster than the Gauss-Seidel

method. The study of how much faster the SOR method converges to a good solution than the Gauss-Seidel method will be discussed in the next section titled, “Performance Comparison.” Relating the plots for two different mesh sizes for the SOR method of approximation will allow for visual comparison to the Gauss-Seidel method results to aid in verifying that they both converge at similar solution values. The solution plots of a 64X64 mesh size with the Helmholtz equation lambda coefficient set equal to -1.5 and the SOR relaxation coefficient set equal to the generalized optimal value of 1.93 are shown below in Figures 6 and 7 below.

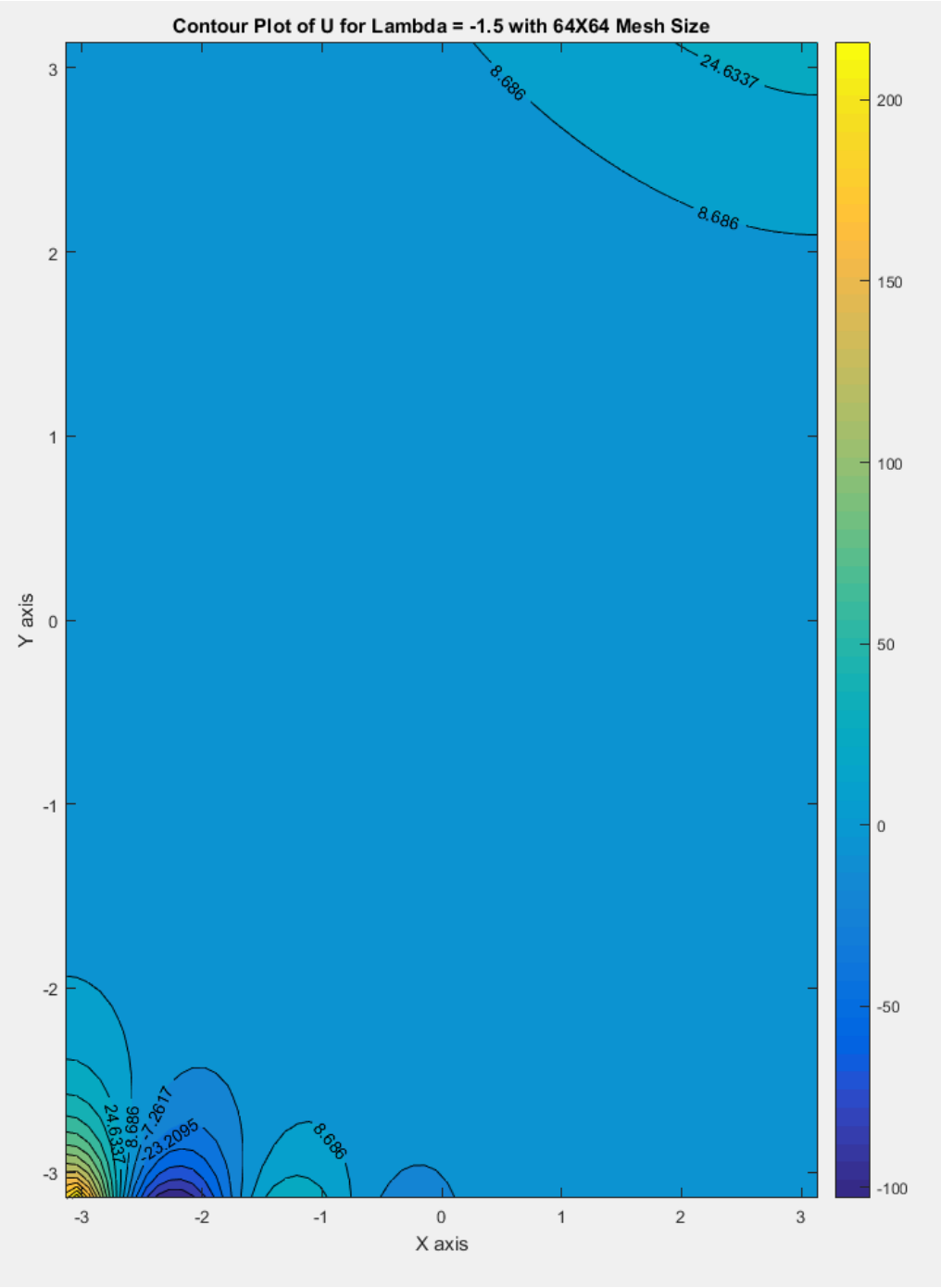


Figure 6 – Contour Plot of U for Lambda = -1.5 with 64X64 Mesh Size

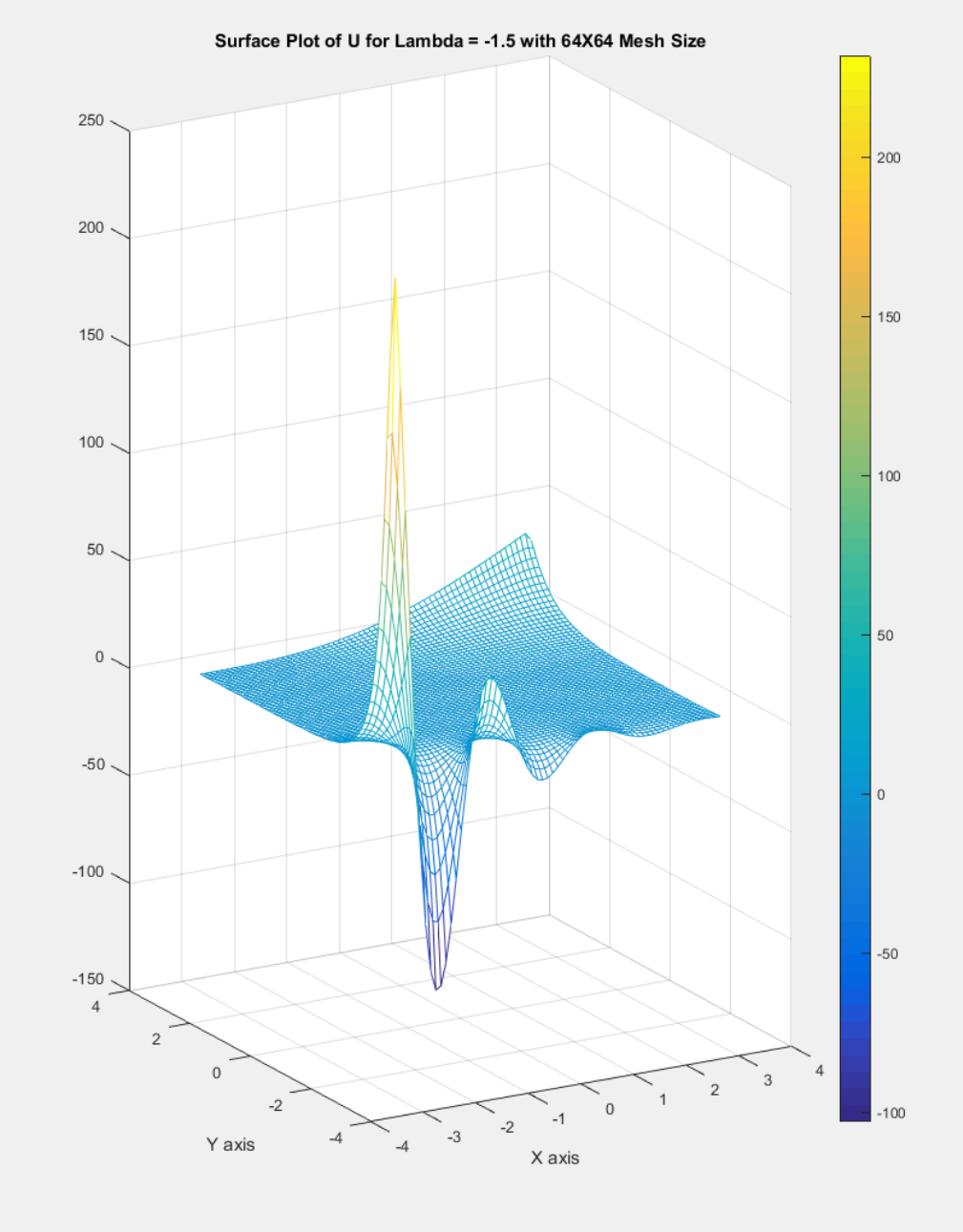


Figure 7 – Surface Plot of U for Lambda = -1.5 with 64X64 Mesh Size

The solution plots of the SOR method of approximation for a 256X256 mesh size with the Helmholtz equation lambda coefficient set equal to -1.5 and the SOR relaxation coefficient set to 1.93 are displayed below in Figures 6 and 7 below.

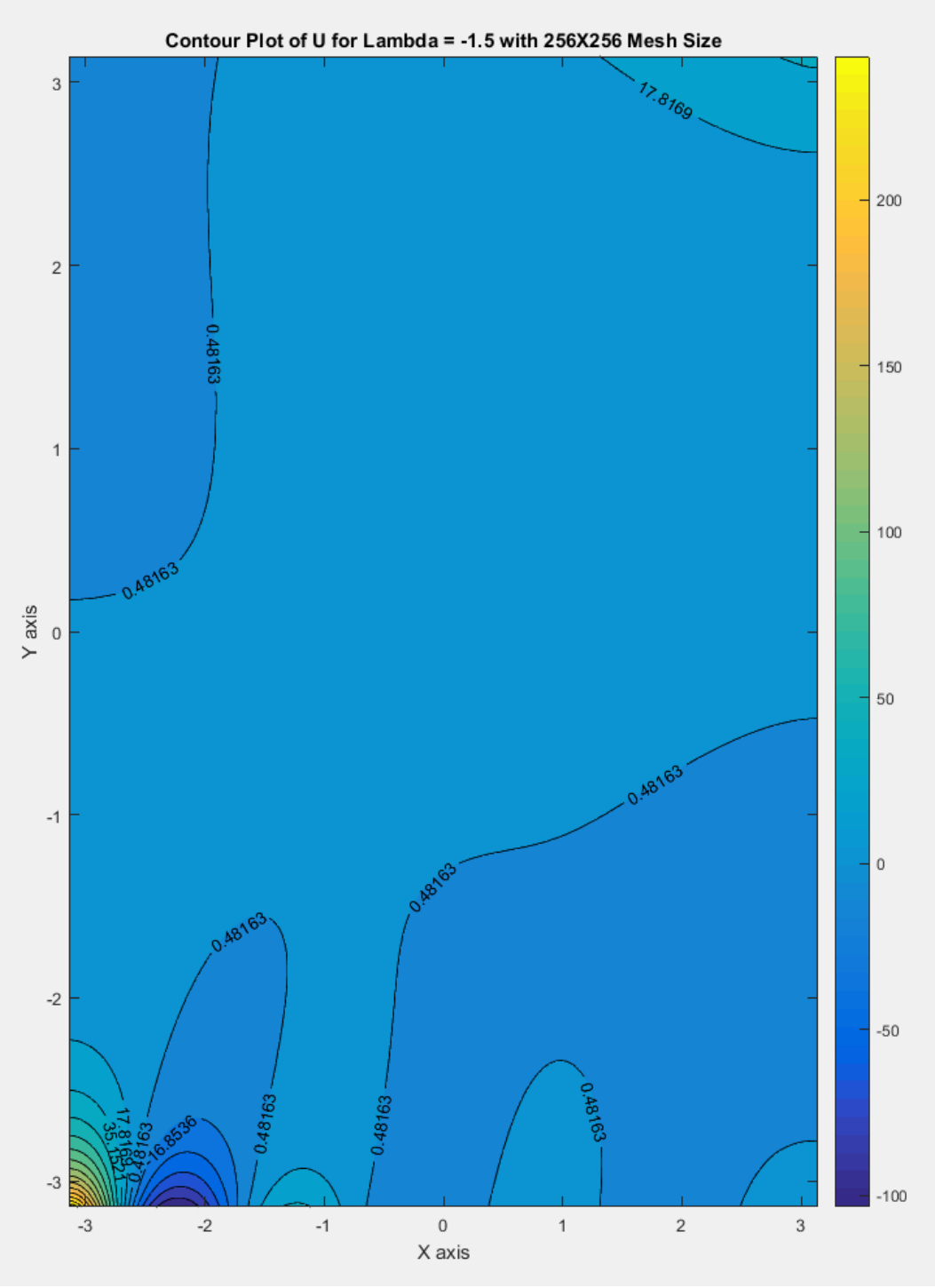


Figure 6 – Contour Plot of U for Lambda = -1.5 with 256X256 Mesh Size

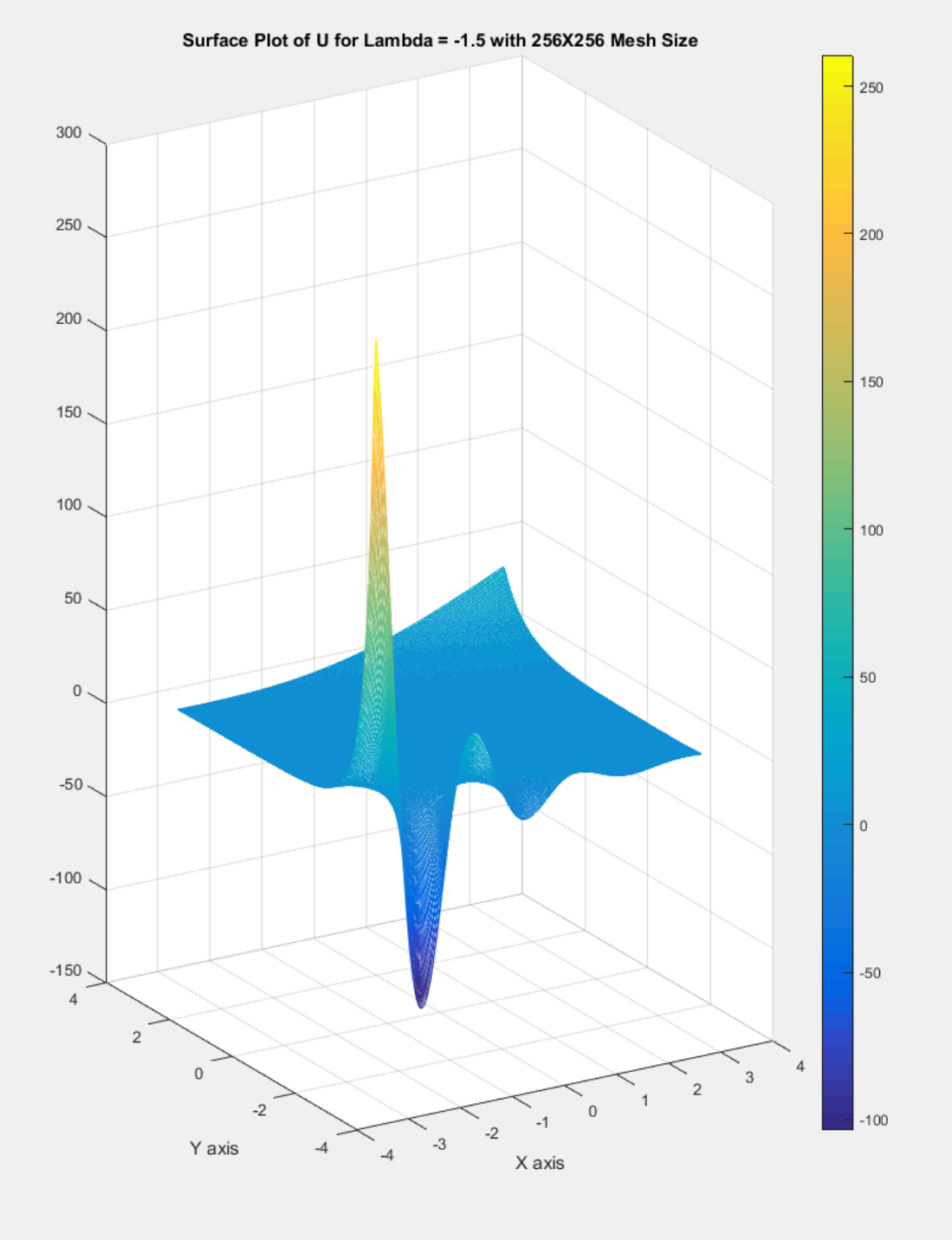


Figure 7 – Surface Plot of U for Lambda = -1.5 with 256X256 Mesh Size

Performance Comparison

As mentioned in previous sections of this report, the main advantage of using the SOR method of approximation over the Gauss-Seidel method is that it converges to the solution much quicker than that of the Gauss-Seidel method, without a loss of accuracy or precision. To

prove that the SOR method has a much higher speed performance than the Gauss-Seidel method, a speed test was run comparing the amount of time each method took to converge with equal mesh sizes. The results of this time comparison study are shown below in Table 4, in which the Helmholtz equation lambda coefficient was set equal to -1.5 and the SOR relaxation coefficient set to a value of 1.93, as was determined to be most universally optimal.

Table 4 – Time Comparison Study with Lambda = -1.5

Time Comparison Study with Lambda = -1.5			
Gauss-Seidel		Successive-Over-Relaxation (λ = 1.93)	
Mesh Size	Run Time (Seconds)	Mesh Size	Run Time (Seconds)
32X32	0.083024064	32X32	0.082388371
64X64	1.057973044	64X64	0.266342763
100X100	5.505424069	100X100	0.59630978
128X128	12.76152843	128X128	1.037336258
200X200	79.19716842	200X200	5.099671619
256X256	241.605559	256X256	16.1887734
300X300	404.1742841	300X300	27.62786141

As is observed in the time comparison study shown above in Table 4, the SOR method achieved much faster performance times than the Gauss-Seidel method. These improvements to the performance time were much more significant and useful at larger mesh sizes with a difference at 300X300 grid size of 376.54 seconds, which was 6.8% of the time it took the Gauss-Seidel method to fully converge to the solution. These results are readily apparent in the plot shown below in Figure 6.

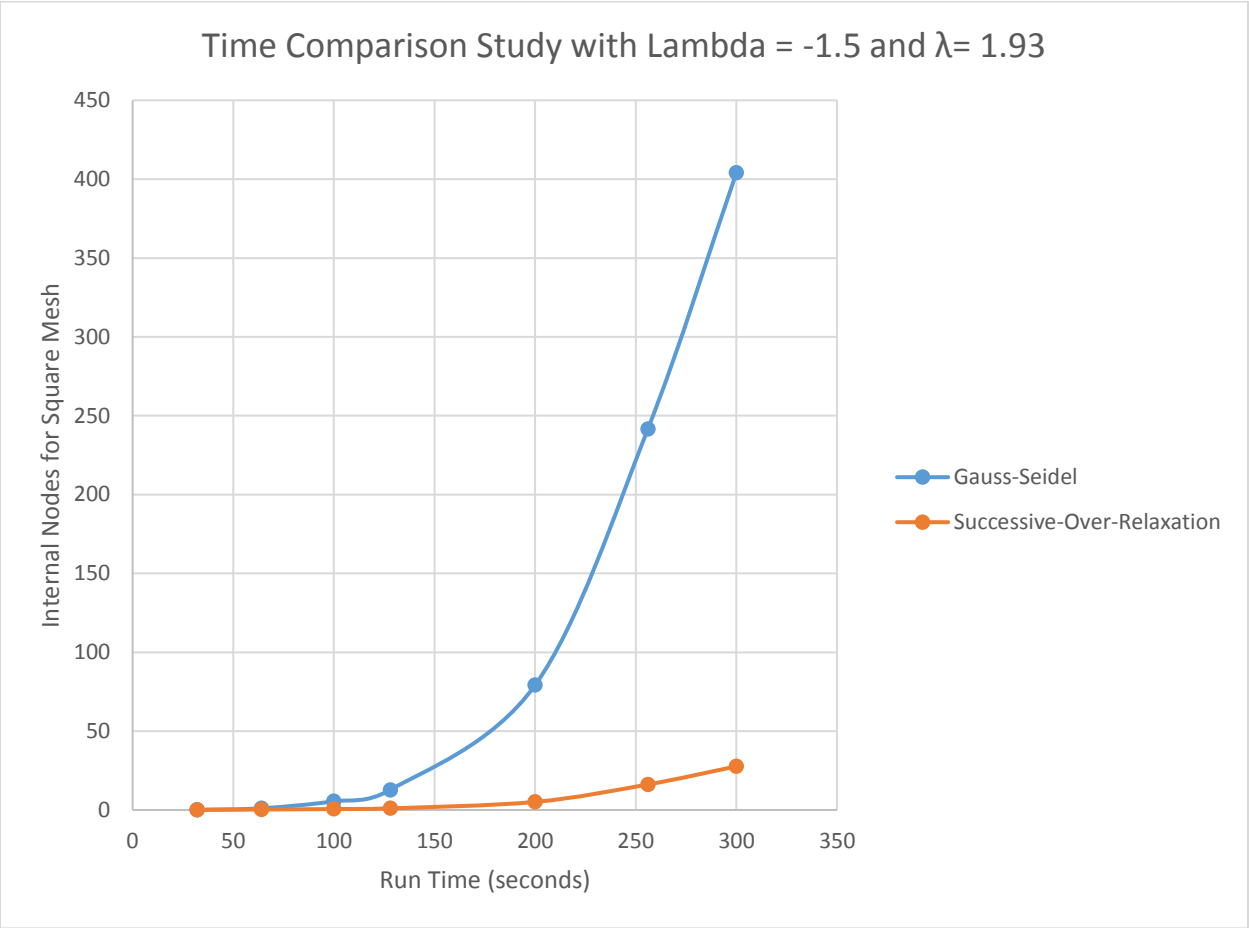


Figure 6 – Time Comparison Study with Lambda = -1.5 and λ = 1.93