SCIENTIFIC COMPUTING FOR MECHANICAL ENGINEERS

HELMHOLTZ/POISSON 2-D EQUATION

AH01-3

# Abstract

The Helmholtz equation, also known as the time harmonic wave equation, is related to many steady-state oscillation problems, such as how violin bows effect the strings when being played, sound waves, Helmholtz resonators, heat waves; it is a useful stepping stone in solving many wave equations. Using numerical methods, one can find a close approximation to the real solution of these Helmholtz equations. While numerical methods may not one hundred percent accurate (as they always have some discretization error and computer based error involved), they are still a useful tool for engineers, and various other professions, for their close accuracy. The numerical methods one can use to solve a 2-D Helmholtz equation includes Gauss elimination and iterative methods, such as Jacobi, Gauss-Seidel, and Successive Over Relaxation. Each numerical method has its own unique criteria that it must meet and may provide solutions with different accuracies. In this report, the and Gauss-Seidel and Successive-Over Relaxtion methods are used to solve a 2-D Helmholtz Equation with 3 Dirichlet and 1 Neumann boundary condition.

Table of Contents

[Abstract 2](#_Toc513662484)

[List of Tables 3](#_Toc513662485)

[List of Figures 3](#_Toc513662486)

[Mathematical Statement of Problem 5](#_Toc513662487)

[Discretization 6](#_Toc513662488)

[Numerical Methods 7](#_Toc513662489)

[Technical Specifications of the Computer 11](#_Toc513662490)

[Results 11](#_Toc513662491)

# List of Tables

[Table 1. Table for Grid Independence of Manufactured Solutions using Gauss-Seidel. 25](#_Toc513662492)

[Table 2. Table for Grid Independence of Manufactured Solutions using Successive-Over Relaxation. 26](#_Toc513662493)

[Table 3. Table for Grid Independence of actual code using Gauss-Seidel. 28](#_Toc513662494)

[Table 4. Table for Grid Independence of actual code using Successive-Over Relaxation. 30](#_Toc513662495)

# List of Figures

[Figure 1. 2-D Helmholtz Equation 5](#_Toc513662496)

[Figure 2. Specific problem solved in this report with boundary conditions and the Right Hand Side given by F(x,y). 5](#_Toc513662497)

[Figure 3. Showing what the boundary condition on each side look like for the specific problem solved in this report. 6](#_Toc513662498)

[Figure 4. Approximation to a 1-D second order derivative. 6](#_Toc513662499)

[Figure 5. Discretized 2-D Helmholtz Equation 7](#_Toc513662500)

[Figure 6. The middle point is the one being solved for (uij) and the 4 points highlighted around it are the points that are being used to solve for uij. 7](#_Toc513662501)

[Figure 7. Solving the discretization of the 2-D Helmholtz equation for uijk+1 for Gauss-Seidel. 8](#_Toc513662502)

[Figure 8. Solving the discretization of the 2-D Helmholtz equation for uijk+1 for Gauss-Seidel when Δx = Δy. 8](#_Toc513662503)

[Figure 9. The Guass-Seidel Method Pseudo Code 9](#_Toc513662504)

[Figure 10. Pseudo code for Gauss-Siedel from NCSU.edu 10](#_Toc513662505)

[Figure 11. Solving the discretization of the 2-D Helmholtz equation for uijk+1 for SOR. 10](#_Toc513662506)

[Figure 12. Solving the discretization of the 2-D Helmholtz equation for uijk+1 for SOR when Δx = Δy. 10](#_Toc513662507)

[Figure 13. Pseudo code from NCSU.edu for the SOR method. 11](#_Toc513662508)

[Figure 14. Parameters for the specific problem as shown in Figure 2, with the exception that λ=0 in the code. 12](#_Toc513662509)

[Figure 15. Surface Plot for given F when using Gauss-Seidel. 13](#_Toc513662510)

[Figure 16. Contour Plot for given F when using Gauss-Seidel. 13](#_Toc513662511)

[Figure 17. Surface Plot for given F when using Successive-Over Relaxation. 14](#_Toc513662512)

[Figure 18. Contour Plot for given F when using Successive-Over Relaxation. 14](#_Toc513662513)

[Figure 19. Surface Plot for F=0 when using Gauss-Seidel. 15](#_Toc513662514)

[Figure 20. Surface Plot for F=0 when using Gauss-Seidel. 16](#_Toc513662515)

[Figure 21. Surface Plot for F=0 when using Successive-Over Relaxation. 17](#_Toc513662516)

[Figure 22. Contour Plot for F=0 when using Successive-Over Relaxation. 17](#_Toc513662517)

[Figure 23. Surface plot for Gauss-Seidel Nx=Ny=60 and 1st Order Neumann at top boundary (y=2\*pi). 19](#_Toc513662518)

[Figure 24. Contour plot for Gauss-Seidel Nx=Ny=60 and 1st Order Neumann at top boundary (y=2\*pi). 19](#_Toc513662519)

[Figure 25. Surface plot for Successive-Over Relaxation Nx=Ny=60 and 1st Order Neumann at top boundary (y=2\*pi). 20](#_Toc513662520)

[Figure 26. Contour plot for Successive-Over Relaxation Nx=Ny=60 and 1st Order Neumann at top boundary (y=2\*pi). 20](#_Toc513662521)

[Figure 27. Solving the Manufactured Solution to enter into the code. 21](#_Toc513662522)

[Figure 28. Boundary conditions for Manufactured Solutions. 21](#_Toc513662523)

[Figure 29. Plot of Manufactured Solution calculated U using the Gauss-Seidel method 22](#_Toc513662524)

[Figure 30. Plot of Manufactured Solution exact U using the Gauss-Seidel method. 22](#_Toc513662525)

[Figure 31. Plot of error2 |U(I,j)-Uexact(I,j)|2 using the Gauss-Seidel method. 23](#_Toc513662526)

[Figure 32. Plot of Manufactured Solution calculated U using the Successive-Over Relaxation method. 23](#_Toc513662527)

[Figure 33. Plot of Manufactured Solution exact U using the Successive-Over Relaxation method. 24](#_Toc513662528)

[Figure 34. Plot of error2 |U(I,j)-Uexact(I,j)|2 using the Successive-Over Relaxation method. 24](#_Toc513662529)

[Figure 35. Uaverage vs Total Nodes for Manufactured Solutions using Gauss-Seidel. 25](#_Toc513662530)

[Figure 36. L2Error vs Total Nodes for Manufactured Solutions using Gauss-Seidel. 26](#_Toc513662531)

[Figure 37. Uaverage vs Total Nodes for Manufactured Solutions using Successive-Over Relaxation. 27](#_Toc513662532)

[Figure 38. L2Error vs Total Nodes for Manufactured Solutions using Successive-Over Relaxation. 27](#_Toc513662533)

[Figure 39. Uaverage vs Total Nodes for actual code using Gauss-Seidel. 28](#_Toc513662534)

[Figure 40. L2Error vs Total Nodes for actual code using Gauss-Seidel. 29](#_Toc513662535)

[Figure 41. Uaverage vs Total Nodes for actual code using Successive-Over Relaxation. 30](#_Toc513662536)

[Figure 42. L2 Error vs Total Nodes for actual code using Successive-Over Relaxation. 31](#_Toc513662537)

[Figure 43. log(L2 Error) vs log(Δx) for GS. The slope is approximately 2 as expected which shows that the approximation is indeed O(h2). 32](#_Toc513662538)

[Figure 44. log(L2 Error) vs log(Δx) for SOR. The slope is approximately 2 as expected which shows that the approximation is indeed O(h2). 32](#_Toc513662539)

[Figure 45. log(L1 Error) vs log(Δx) for GS actual code. The slope is approximately 2 as expected which shows that the approximation is indeed O(h2). 33](#_Toc513662540)

[Figure 46. log(L1 Error) vs log(Δx) for SOR actual code. The slope is approximately 2 as expected which shows that the approximation is indeed O(h2). 33](#_Toc513662541)

# Mathematical Statement of Problem

The Helhmoltz equation is a partial differential equation that is often used to help study physical problems involving PDEs in both space and time. It can be used to study problems such as how violin bows effect the strings when being played, phenomenon of air resonance in cavities (like blowing across the top of an empty bottle), and it a useful stepping stone in solving the wave equation as it is a time-independent form of the wave equation that can result using the separation of variables method to solve the wave equation.

One expects the 2-D Helmhotz equation to have a form like the one shown below and in order to solve 2-D elliptical problems the boundary conditions must be specified over each boundary and over the entire boundary.

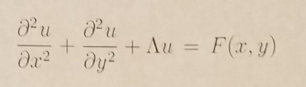


Figure 1. 2-D Helmholtz Equation

For the specific problem solved in this report, the domain of interest is from ax = ay = 0 to bx = by = 2π in both the x and y directions. The specific boundary conditions are u(x=ax , y) = (by-y)2\*cos(π\*y/ by) and u(x= bx , y) = y\* (by – y)2 and u(x , y= ay)=fb(ay) + (x-ax)/( bx – ax)\*(gb(ay)-fb(ay)) and du/dy @ y= by = 0. F(x,y) = cos(π/2\*(2\*(x- ax)/( bx - ax)+1))\*sin(π\*(y - ax)/( by – ay). Also, λ =1/2. This is also shown in the figure below.

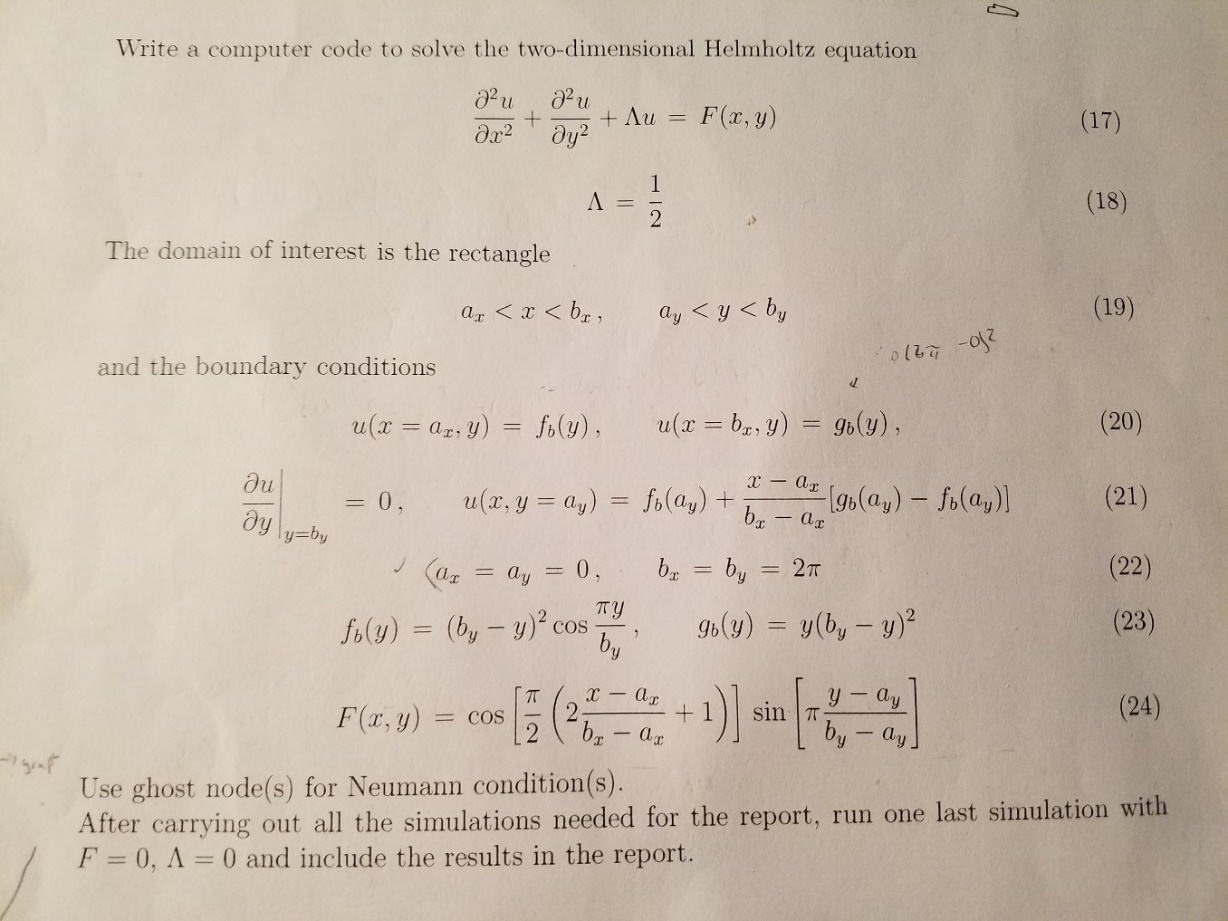


Figure 2. Specific problem solved in this report with boundary conditions and the Right Hand Side given by F(x,y).

As one can see, there are 3 direlecht conditions given at u(x=ax,y) ; u(x=bx,y) ; and u(x,y=ay) and 1 Neumann condition given at u(x,y=by). The figure shown below shows how this would look on a x and y grid. Then after the problem is solved for Helmholtz, one must solve it again when λ = 0, which results in the Poisson equation.

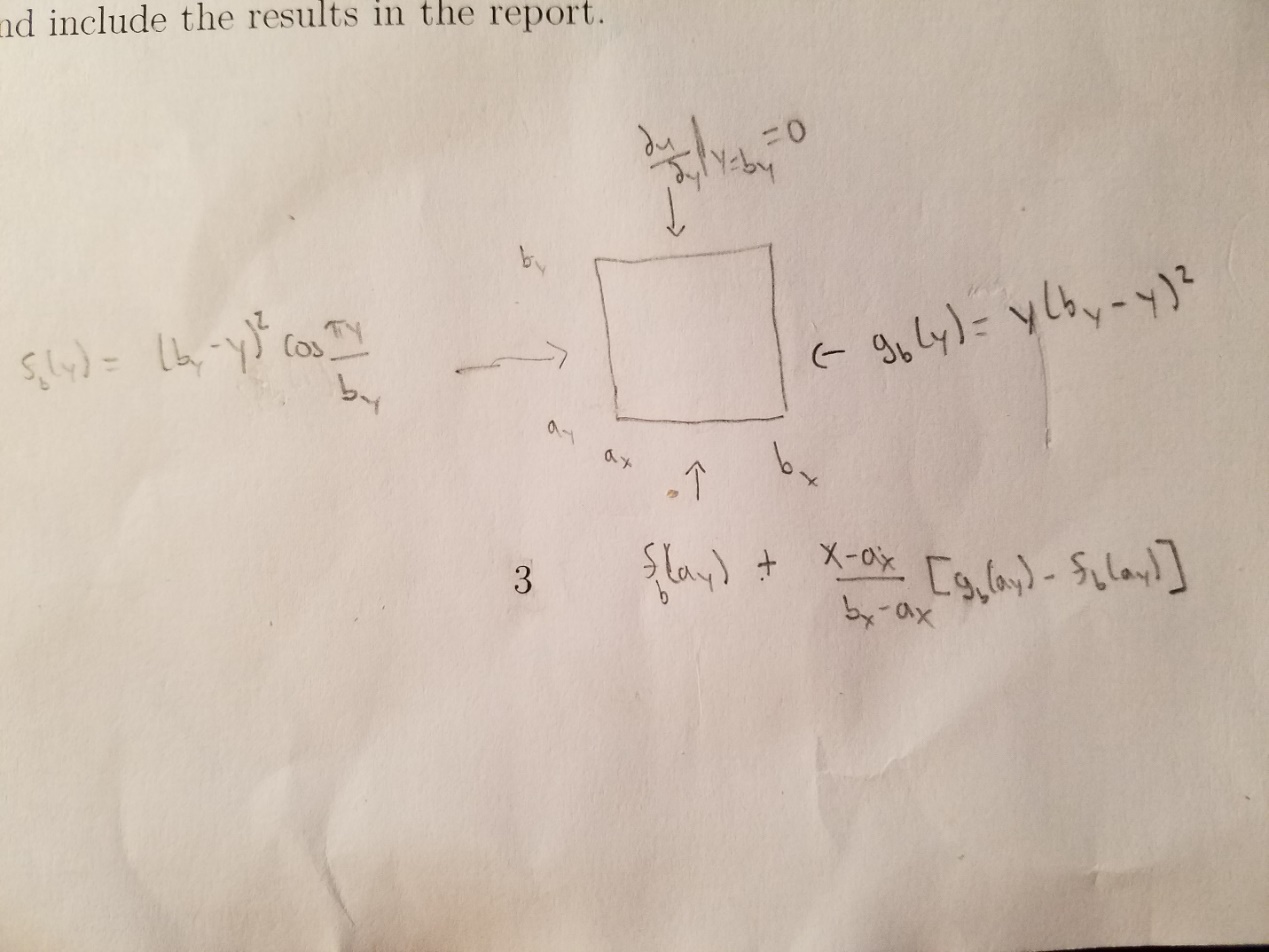


Figure 3. Showing what the boundary condition on each side look like for the specific problem solved in this report.

# Discretization

In order to solve the Helmholtz equation using numerical methods, one must discretize the equation in order for computers to “understand” the problem. Discretization of the Helmholtz equation shown in Figure 1 can be started by dividing the x axis into M+1 equal segments with length Δx, where Δx = Lx / (M+1) and indexing the points with i so that xi = i\* Δx. Similarly, divide the y axis into N+1 equal segments with length Δy where Δy = Lx /(N+1) and indexing the points with j so that yj = j\* Δy. For simplicity write, uij = u(xi,yj). Then realizing that d2u/dx2 and d2u/dy2 can be discretized using derivative approximations with an error of O(h^2). Such as shown in the figure shown below:

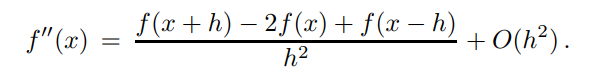


Figure 4. Approximation to a 1-D second order derivative.

If doing a partial differential with respect to x, one would switch the h to Δx, same as the Δx shown above. Similarly, for a partial differential with respect to y, one would switch the h to Δy, same as the Δy shown above. If one is doing the derivative of u, one simply switches the f to a u. This results in the discretized form of Figure 1 as shown below.

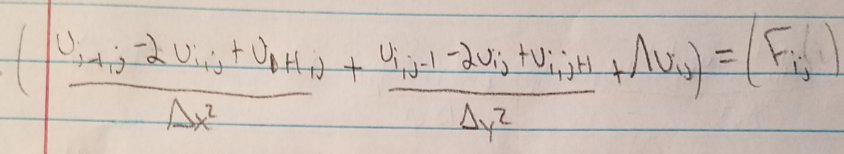


Figure 5. Discretized 2-D Helmholtz Equation

# Numerical Methods

The three methods that were allowed for the 2-D Helmholtz equation were Gauss elimination method, Gauss-Seidel method, and the Successive Over Relaxation (SOR) method. The 2 methods chosen for this report are the Gauss-Seidel and the Successive Over Relaxation method. Both methods involve using the 4 points surrounding the point of interest (uij) as shown in the figure below. These methods take what can be considered as an average of the 4 surrounding points in order to find the approximate values for uij. They are iterative as for each time one runs through the code the values surrounding uij (except at the boundaries) will be updated and thus so will the new value for uij. Since both of these methods are iterative, they should be unconditionally stable.

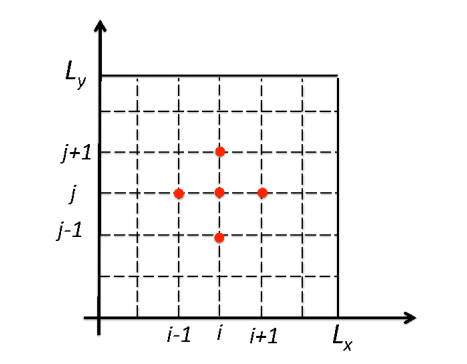


Figure 6. The middle point is the one being solved for (uij) and the 4 points highlighted around it are the points that are being used to solve for uij.

Gauss-Seidel Method

The Gauss-Seidel Method is an iterative method that is useful for solving linear equations, such as the discretized 2-D Helmholtz equation. Starting with the discretization given in Figure 4 one can solve it for uijk+1 as shown in the figure below. Where the k+1 represents the uij for the next iteration of Gauss-Seidel, the next iteration of ui-1,j and ui,j-1 is used and is represented by k+1 as the next iterations have already been found for uij at the previous i and previous j values. If this was Jacobi, everywhere would use the k, or non-updated, iteration.

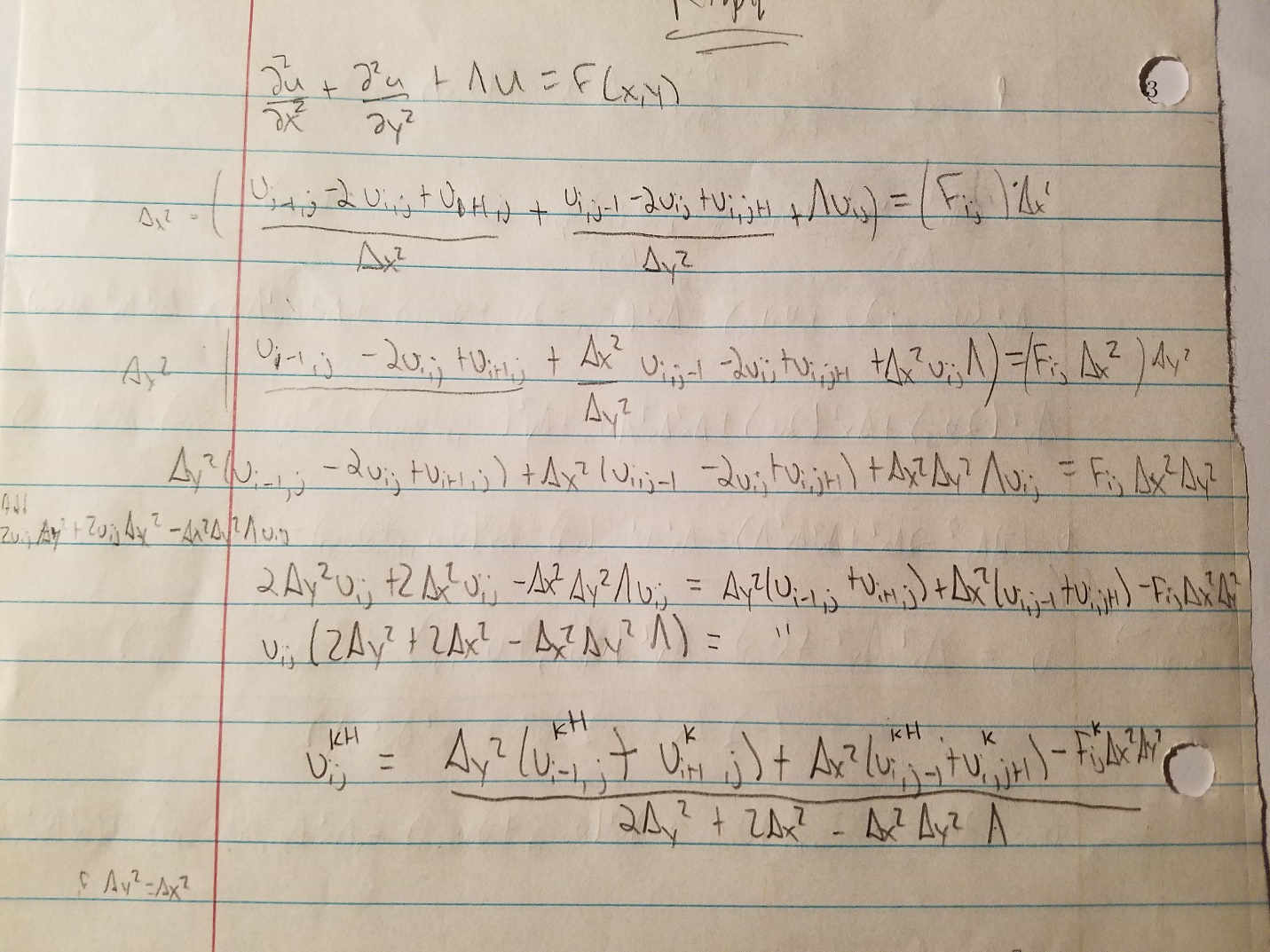


Figure 7. Solving the discretization of the 2-D Helmholtz equation for uijk+1 for Gauss-Seidel.

If one considers the simplest case of Δx= Δy, then one would find uijk is equal to the equation shown in the figure below.

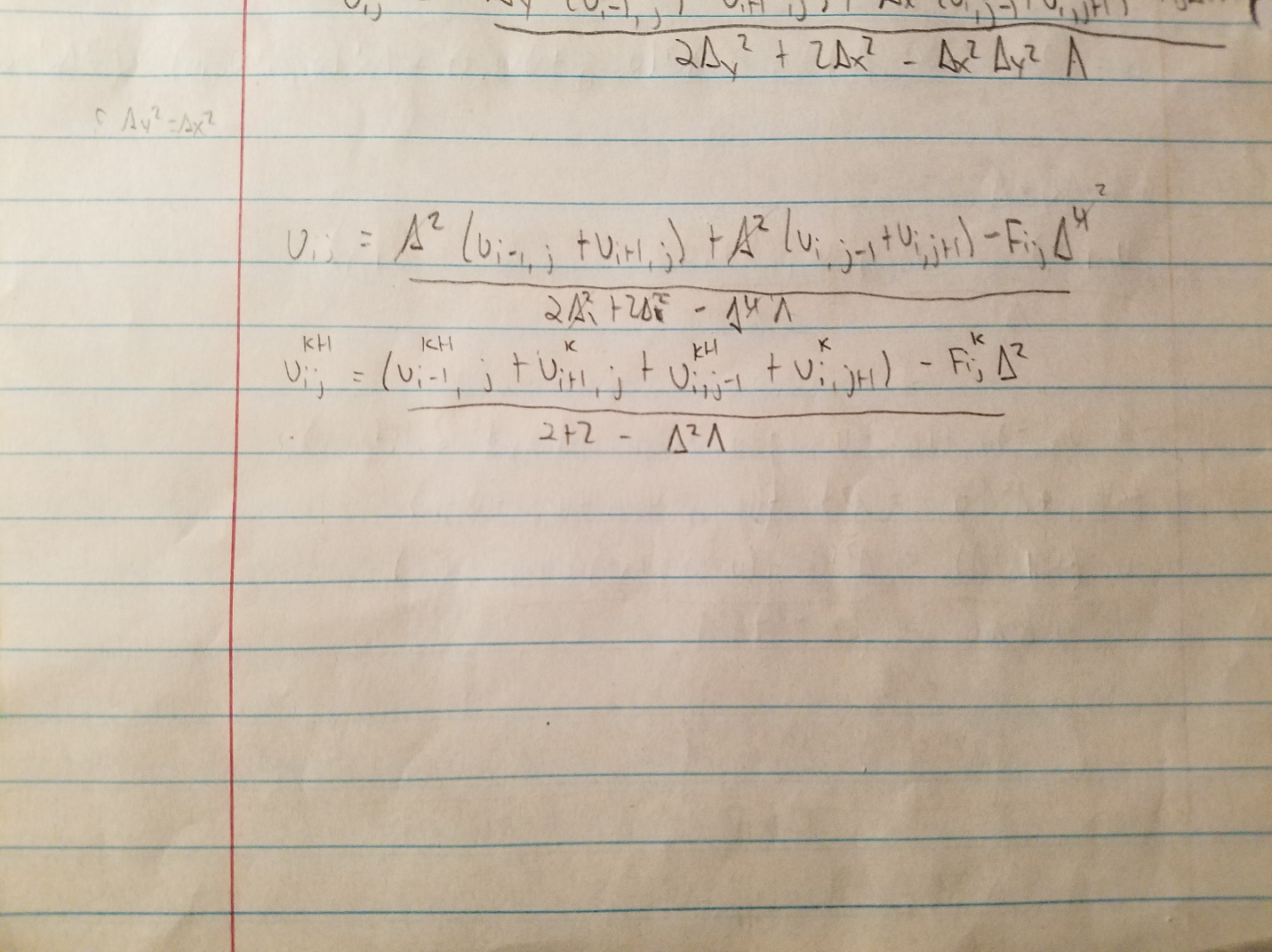


Figure 8. Solving the discretization of the 2-D Helmholtz equation for uijk+1 for Gauss-Seidel when Δx = Δy.

One form of pseudo code for the Gauss Seidel method looks like the code shown in the figure below.

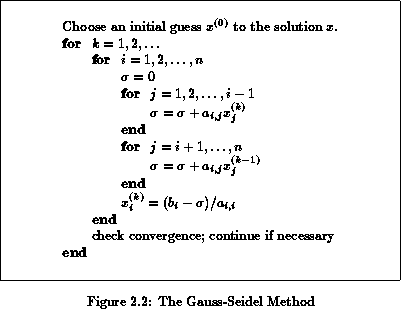


Figure 9. The Guass-Seidel Method Pseudo Code

For Matlab a simple pseudo code looks like:

F=zeros(Nx+2,Ny+2);

For i=1:Nx+2

For j=1:Ny+2

F= given function

End

End

u=zeroes(Nx+2,Ny+2)

u(1,:) = boundary condition

u(Nx+2,:) = boundary condition

u(:,1) = boundary condition

u(:,Ny+2)=boundary condition

For z=1:iterate

For i=2:Nx+1

For j=2:Ny+1

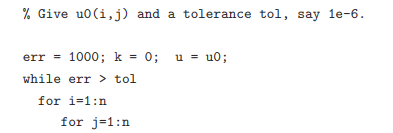
ui,j k+1=((ui-1,j + ui+1,j + ui,j-1k+1 + ui,j+1k ) – fij \*Δ2)/(4-λΔ2)

end

end

end

And one other pseudo code looks like:



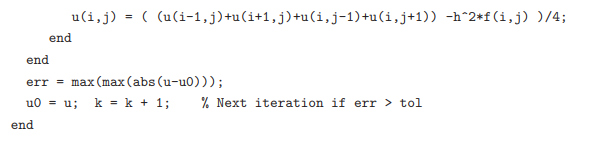


Figure 10. Pseudo code for Gauss-Siedel from NCSU.edu

Successive-Over Relaxation

Successive-over relaxation (SOR) method is very similar to the Gauss-Seidel method in that it is an iterative method and uses the result from Gauss-Seidel to approximate the value of uij. If one considers that uGSk+1 is the result one gets for uijk+1 after running the Gauss-Seidel method, then the SOR method is shown below, where the ω is the relaxation parameter. If ω is less than 1, the method shown below is considered an interpolation. If ω is greater than 1 it is an extrapolation, also known as over-relaxation. If ω equals 1, then one is left with the Gauss-Seidel method. For elliptical problems, such as the 2-D Helmholtz equation, one usually chooses 1 ≤ ω ≤ 2. According to North Carolina State University, the optimal choice for ω is 2/(1+sin(π/Nx+1)) where Nx=Ny and Δx = Δy = 1/(Nx+1)

uijk+1= (1-ω)\*uijk+ ω\*uGSk+1

So, this method depends on the 4 points surrounding uij as well. uijk+1 for SOR is shown in the figure below.

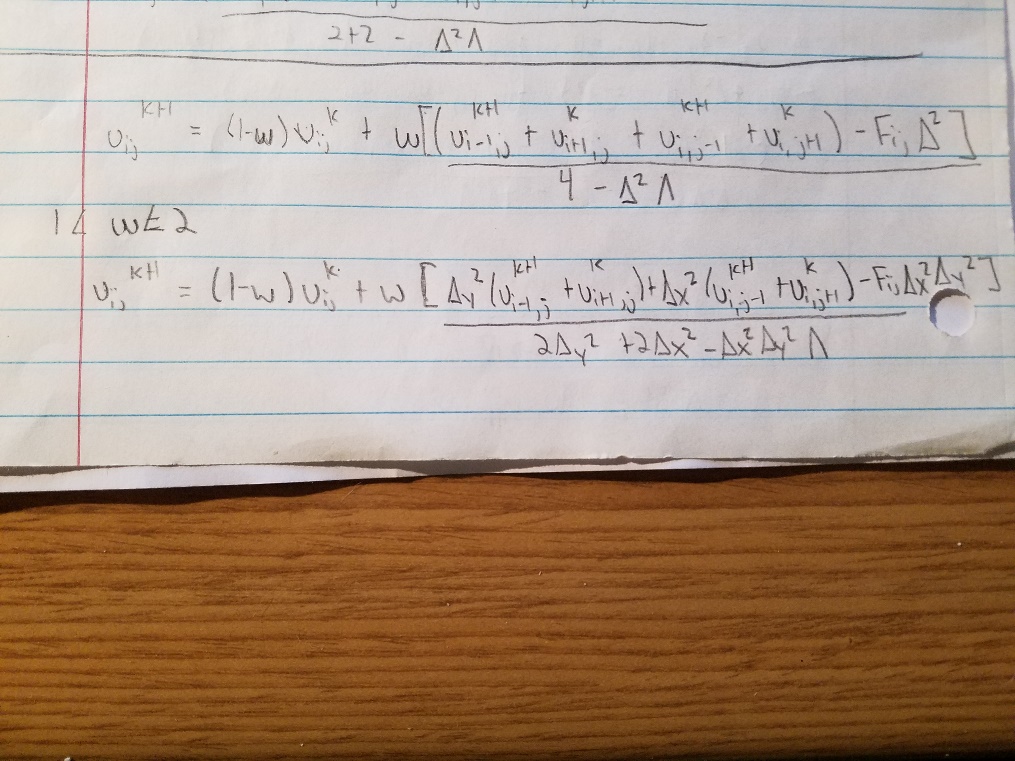


Figure 11. Solving the discretization of the 2-D Helmholtz equation for uijk+1 for SOR.

If one considers the simplest case when Δx = Δy, then the figure below shows the equation for uijk+1.

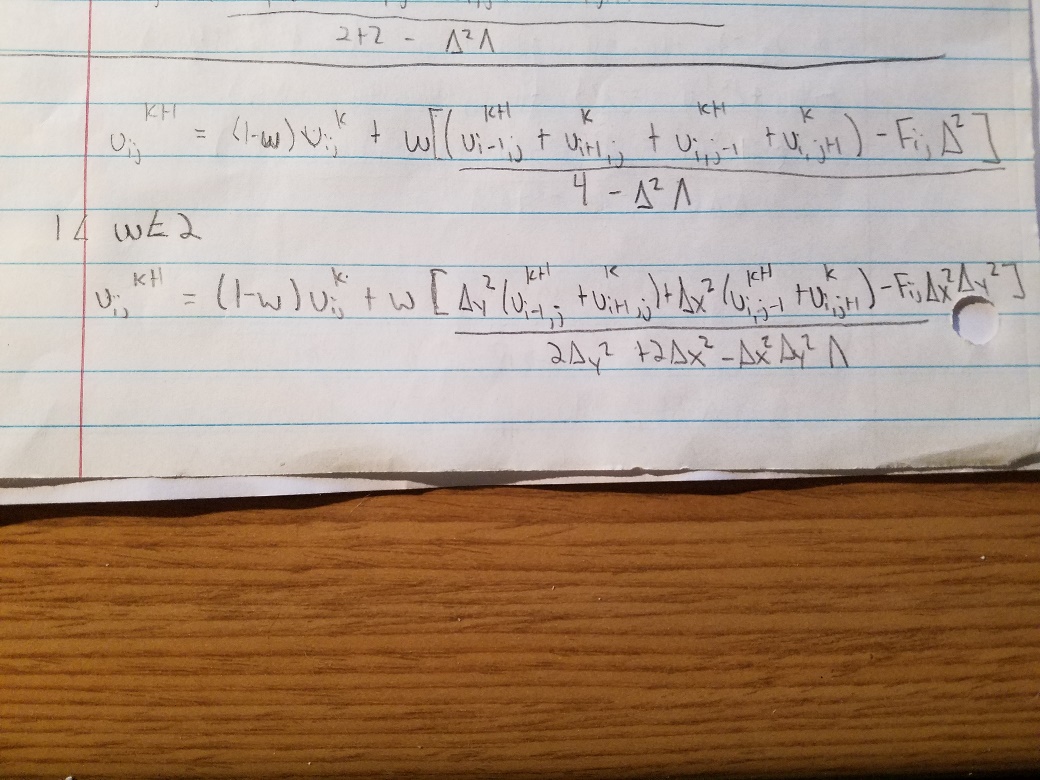
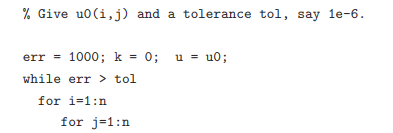


Figure 12. Solving the discretization of the 2-D Helmholtz equation for uijk+1 for SOR when Δx = Δy.

A pseudo code for SOR is given in the figure below.





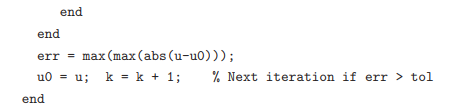


Figure 13. Pseudo code from NCSU.edu for the SOR method.

# Technical Specifications of the Computer

The computer that was used to run the simulation has an Intel i7-6700k CPU that has a semiconductor size of 14 nm, 4 cores, 8 threads, base frequency of 4.00 GHz, a maximum turbo frequency of 4.20 GHz, an 8 MB SmartCache, a Bus Speed of 8 GT/s DMI3, a Thermal Design Power of 91 Watts, and a Maximum Memory Bandwidth of 34.1 GB/s. The L1 Cahce size is 4 x 32KB 8-way set associative instruction caches and 4 x 32KB 8-way set associative data caches (1 L1 cache for each core), the L2 Cache size is 4 x 256KB 4-way set associative caches (1 L2 cache for each core), and the L3 cache size is 8MB 16-way set associative shared cache (1 L3 cache shared among all the cores in a socket, which is 4). The GPU used is the Hydro GFX GTX 1080 liquid cooled graphics card, which has a Boost Base Core Clock of 1847 MHz, a 256-bit Memory Bus, a Memory Size of 8192MB, a Memory Clock of 10108 MHz, the Memory is GDDR5x, 2560 CUDA cores, has a PCI Express x16 3.0 port, and uses the NVIDA GeForce GTX 1080 GPU. There are 2 x (8 GB) for a total of 16 GB of DDR4 RAM installed. The motherboard used is the Asus ROG Maximus VII Hero.

# Results

**\*NOTE: The code that was ran for this report solves a Poisson equation, which is the Helmholtz equation where λ=0. This is due to an error with unknown cause that effected the code that was run and permission from the professor. The numerical methods shown above still hold true for Poisson, with the only change being λ is set equal to 0.**

**Parameter Specifications**

The parameters for the problem solved in this report are given above in Figure 2, with the exception that λ=0. But for reference I have included the figure below.

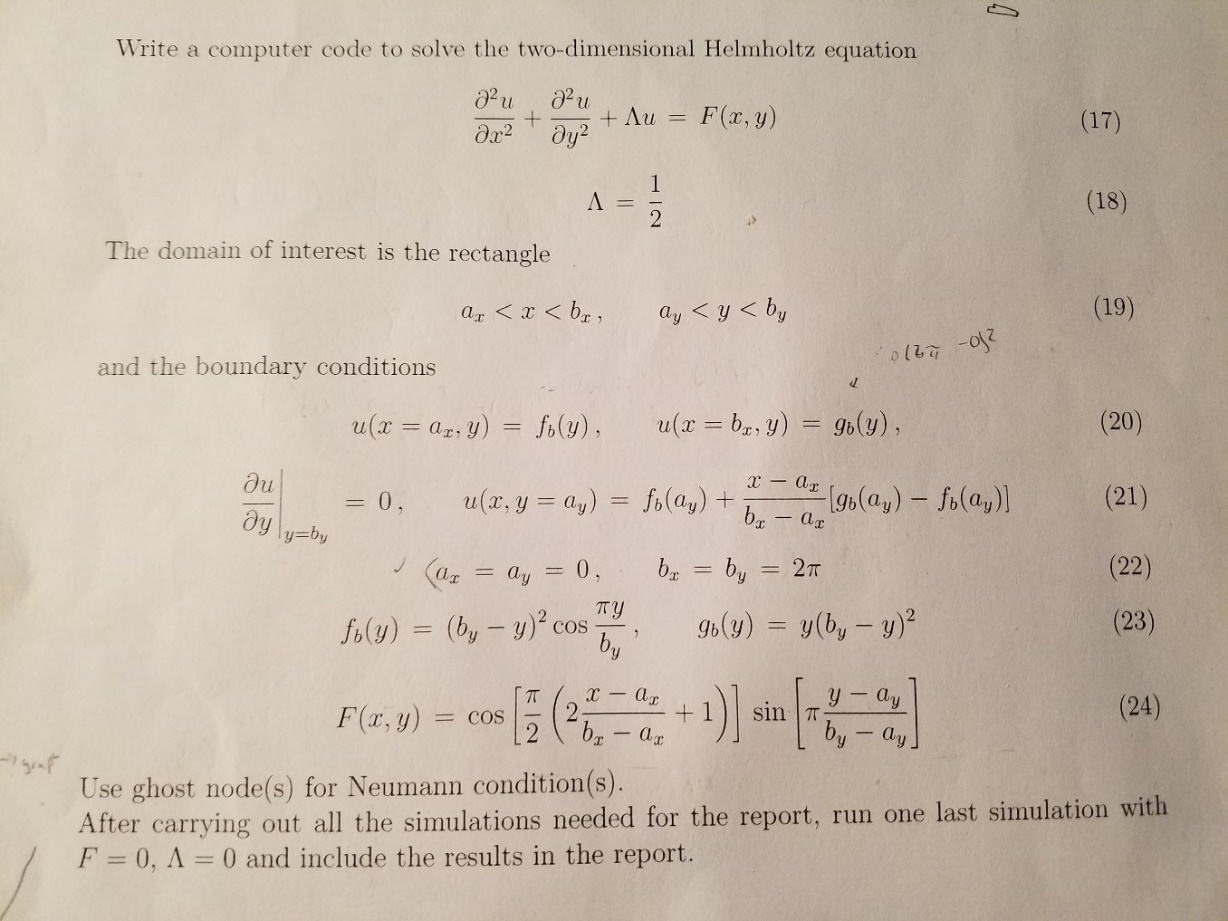


Figure 14. Parameters for the specific problem as shown in Figure 2, with the exception that λ=0 in the code.

**Results**

When running the Gauss-Seidel code with a Nx=Ny=60 and a tolerance of 10-8, one finds that it takes 9759 iterations until the code converges (i.e. the maximum of (current iteration of U - the previous iteration of U)/ the previous iteration of U is less than 10^-8). Uaverage is the average value of all the points in the matrix U and is a useful variable to help describe the matrix convergence. It is a representative value of the plot. For the describe Gauss-Seidel code the Uaverage=15.2971. And gives the plots shown below. The surface plot helps to visualize the how the given boundary conditions look when graphed and then also shows how the inner nodes slope towards the boundaries. The contour plot helps to visualize the value of U at a given point.

\*Note: The colored contour plots are used because the grayscale contour plot made the bottom left and middle right outermost lines disappear.

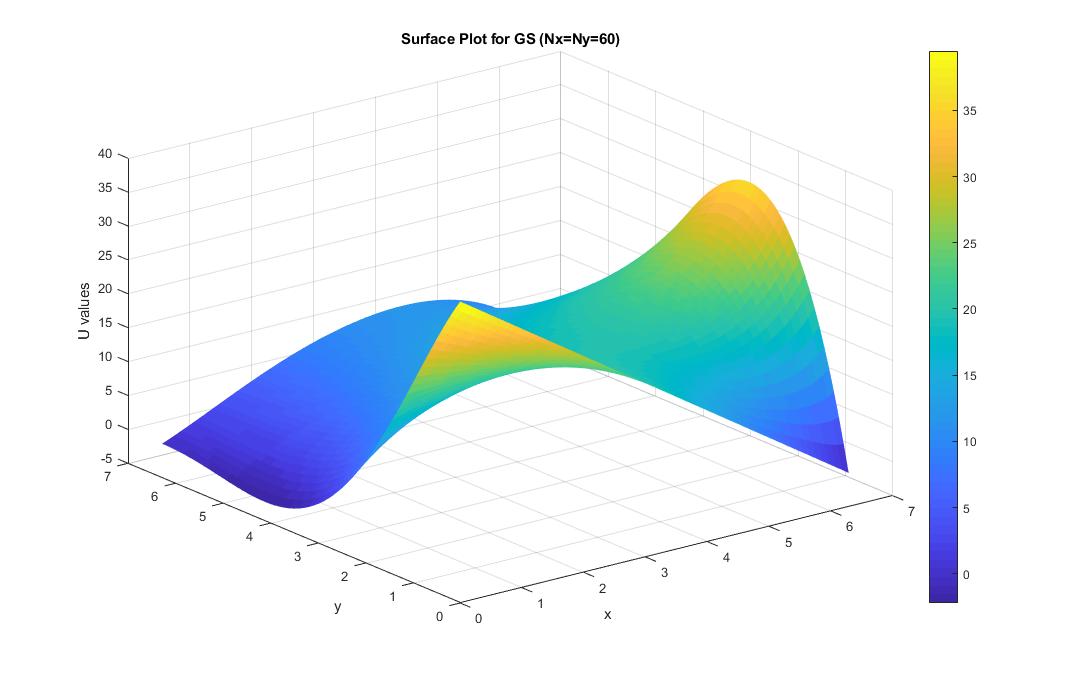


Figure 15. Surface Plot for given F when using Gauss-Seidel.

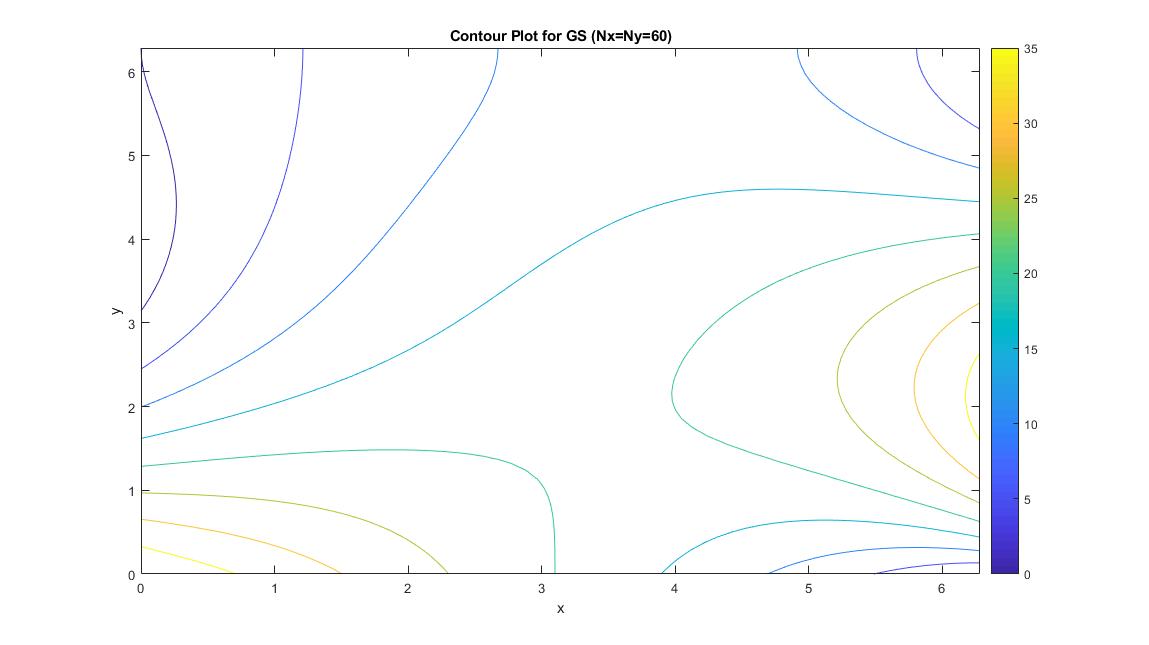


Figure 16. Contour Plot for given F when using Gauss-Seidel.

When running the Successive-Over Relaxation code with a Nx=Ny=60 and a tolerance of 10-8, one finds that it takes 521 iterations until the code converges and the Uaverage = 15.9271. And gives the plots shown below.

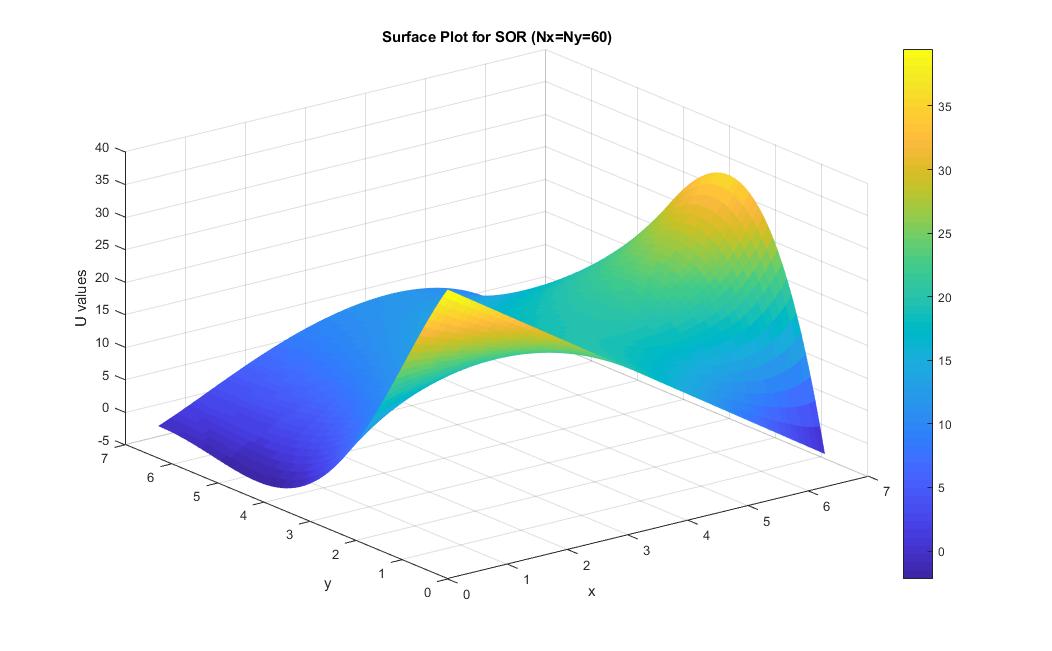


Figure 17. Surface Plot for given F when using Successive-Over Relaxation.

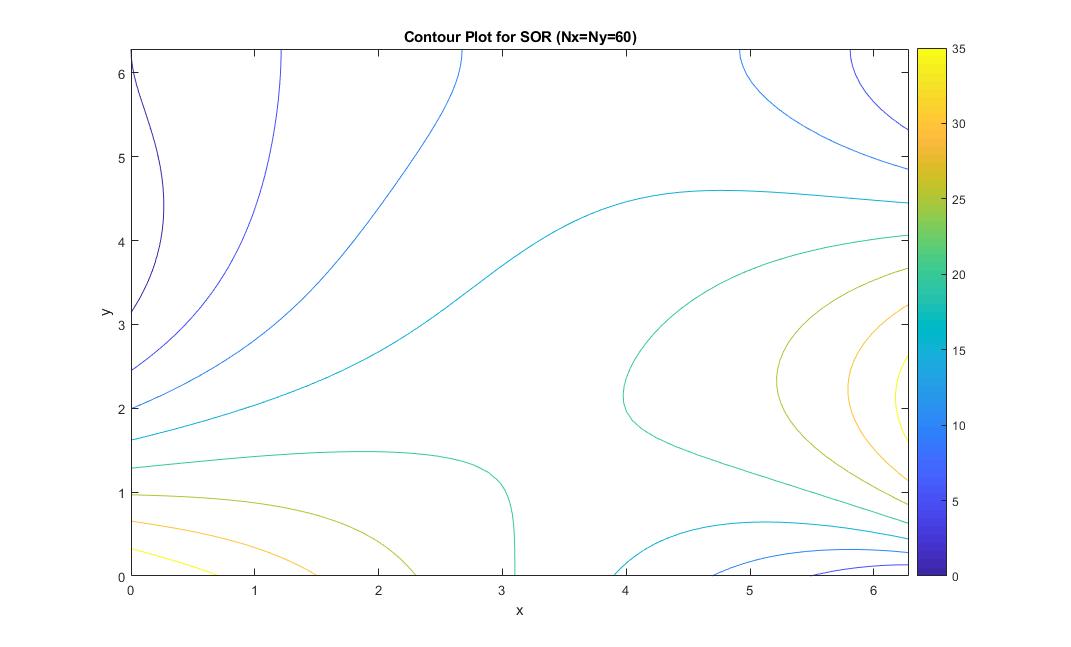


Figure 18. Contour Plot for given F when using Successive-Over Relaxation.

When running the Gauss-Seidel code with Nx=Ny=60, a tolerance of 10-8, and F=0, one finds that it takes 10775 iterations until the code converges and Uaverage = 14.1442. And gives the plots shown below.

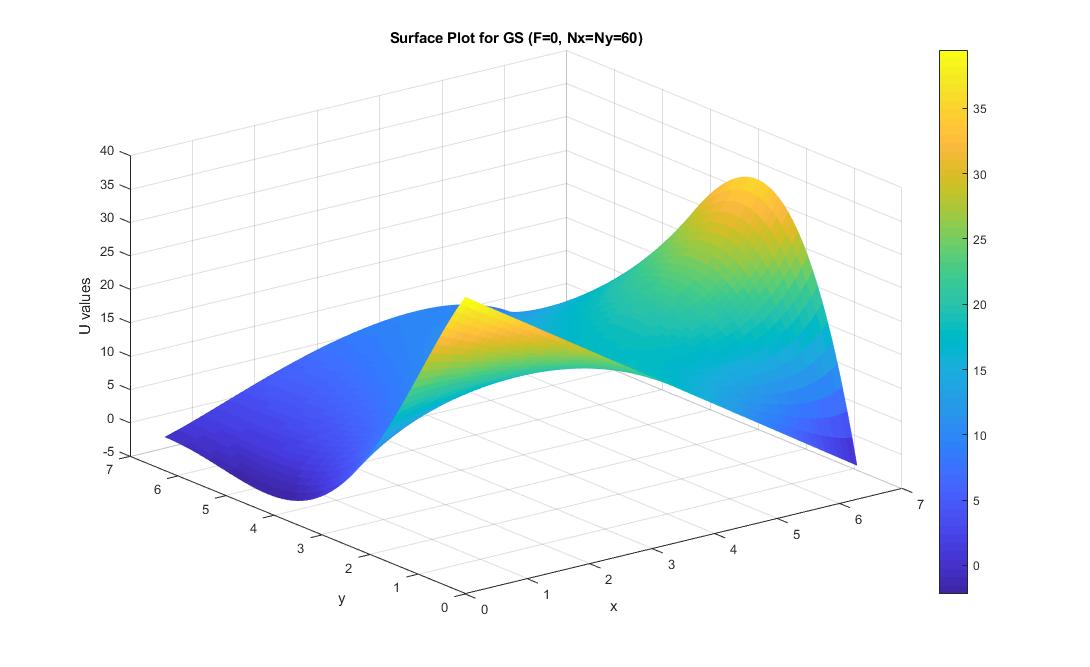


Figure 19. Surface Plot for F=0 when using Gauss-Seidel.

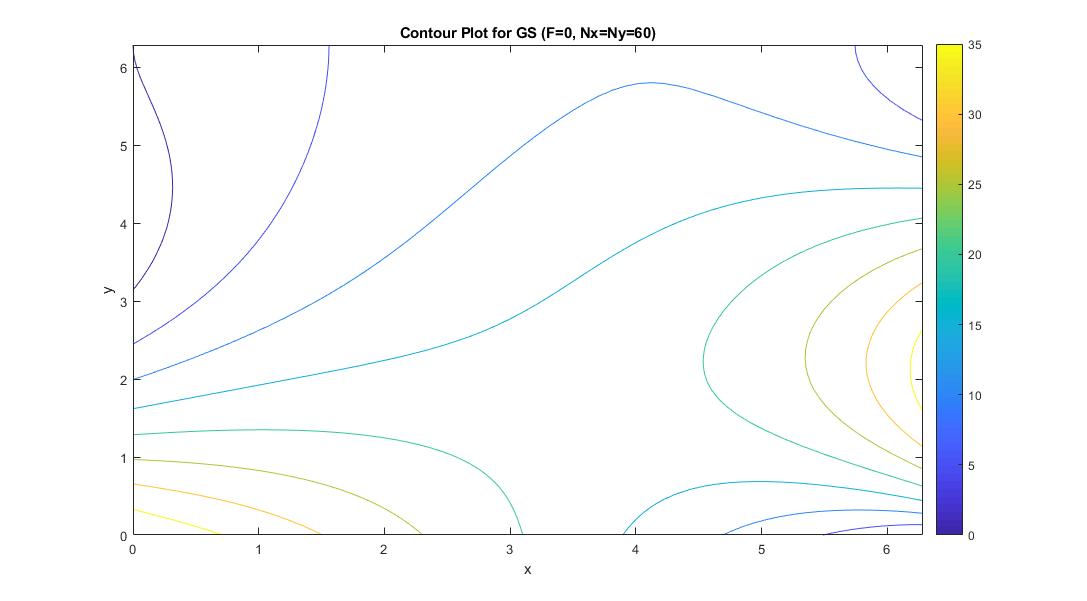


Figure 20. Surface Plot for F=0 when using Gauss-Seidel.

When running the Successive-Over Relxation code with Nx=Ny=60, a tolerance of 10-8, and F=0, one finds that it takes 560 iterations until the code converges and the Uaverage = 14.1442. And gives the plots shown below.

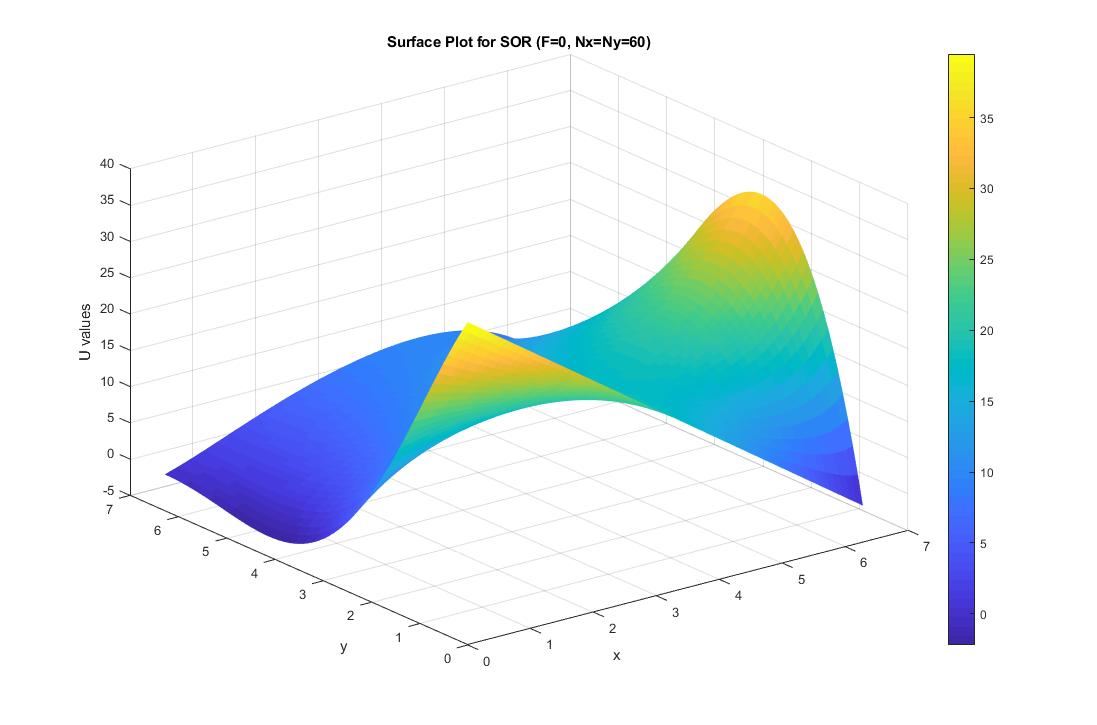


Figure 21. Surface Plot for F=0 when using Successive-Over Relaxation.

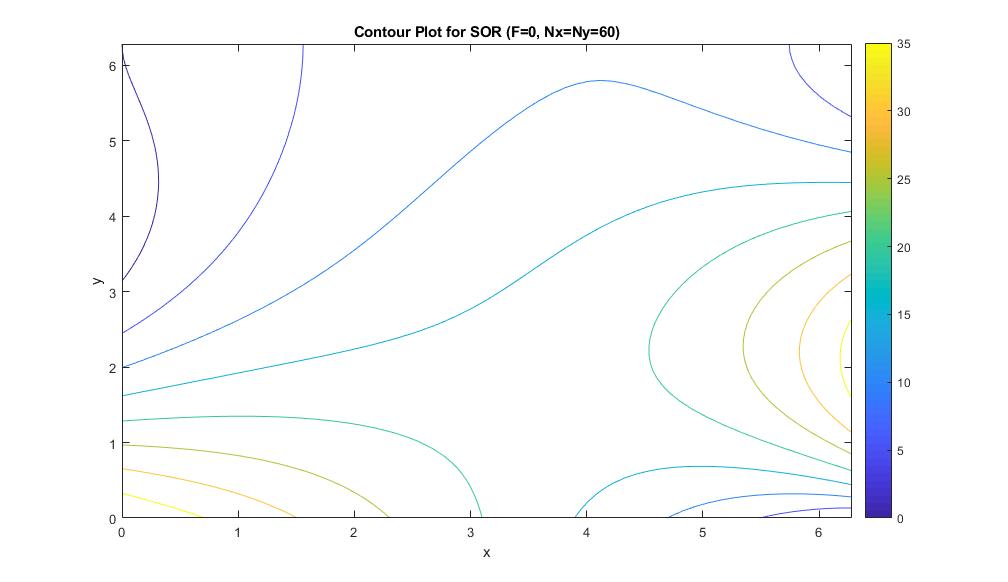


Figure 22. Contour Plot for F=0 when using Successive-Over Relaxation.

As one can see from the plots shown above, the GS and SOR methods produce almost identical surface and contour plots, just as they should since they are both iterative methods that are converging to the same boundary conditions. Additionally, one can see from the iteration numbers, that SOR converges significantly faster that Gauss-Seidel, taking only 521 iterations vs 9759 iterations. Additionally, one can see that the Uaverage for GS and SOR are the same for a given F, i.e. for F=0 the Uaverage value for both GS and SOR is 14.1442 and when F≠0 the Uaverage value for both GS and SOR is 15.2971.

**Effect of Number of points used for discretization**

To figure out the effect of the number of points used for discretization, one should change the order of accuracy of the discretization used for either the code or a boundary condition, because if there is one part of the algorithm that has a worse order of accuracy then the whole algorithm has that worse order of accuracy. I.e. if everything in one’s code has error of Order (h2) except a boundary condition that has Order (h), then the entire code has an accuracy of Order (h).

To test this in the code used for the project, the easiest thing to do is convert the Neumann boundary condition to an Order (h) approximation instead of an Order (h^2) approximation (as used above). Doing this for the same Nx=Ny=60 as used in the Results subsection of the Results, results in a different Uaverage for Gauss-Seidel and SOR. Previously, when the code was run for Gauss-Seidel and SOR for the same conditions, the plots appeared almost identical. However, when the Neumann boundary condition is Order(h), one can see that along the Neumann condition (top of the plot) there is a more pronounced difference between the GS and SOR plots. The difference between the plots is more easily identified in the contour plots than in the surface plots.

Another interesting point is that the Uaverage is not the same for Gauss-Seidel and Successive-Over Relaxation methods, as it was for the discretization with second order accuracy.

The plots below show a surface plot and contour plot for Gauss-Seidel with NX=Ny=60. The Uaverage is 15.2971 for this and it takes 9759 iterations to converge.

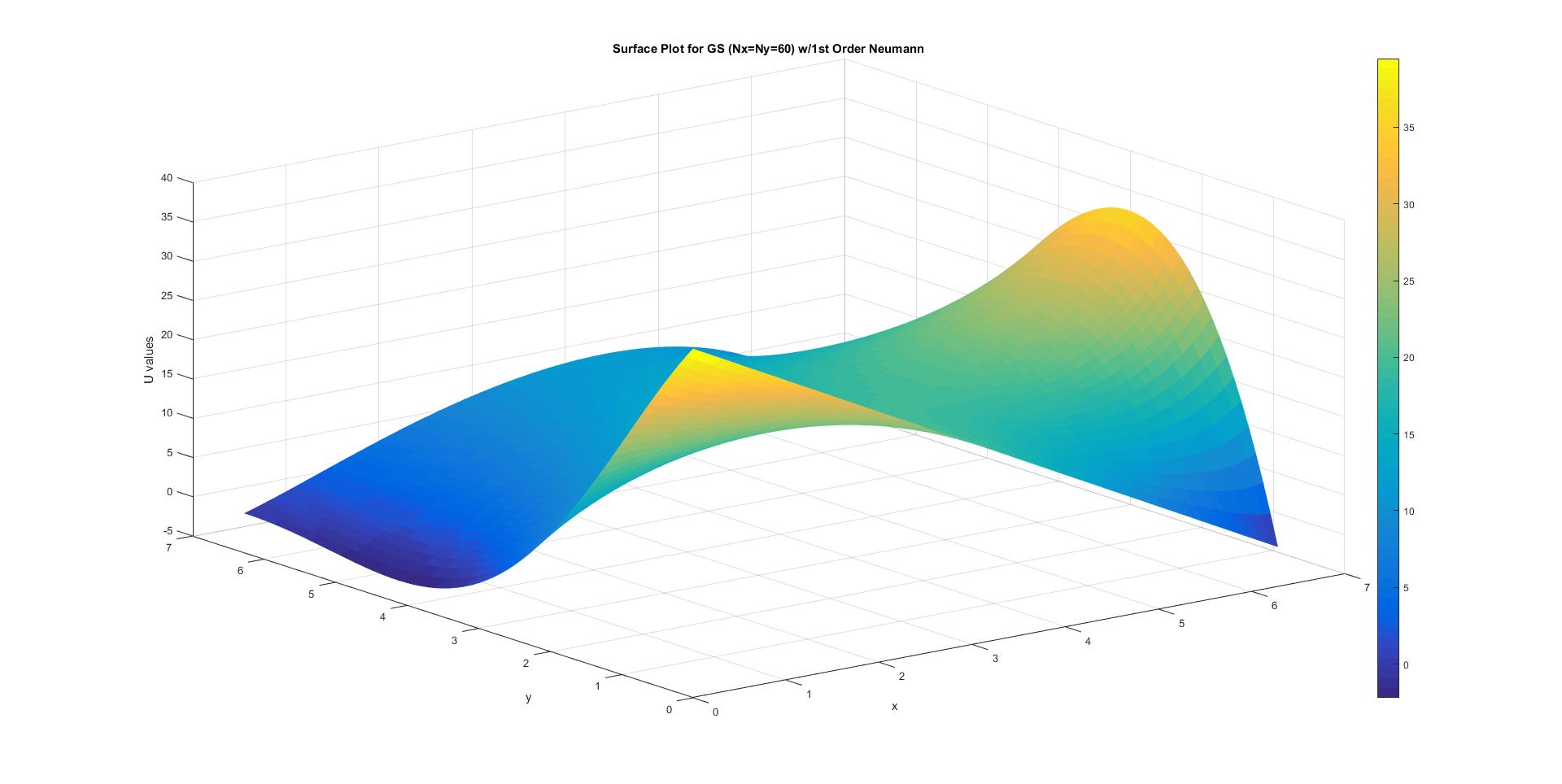


Figure 23. Surface plot for Gauss-Seidel Nx=Ny=60 and 1st Order Neumann at top boundary (y=2\*pi).

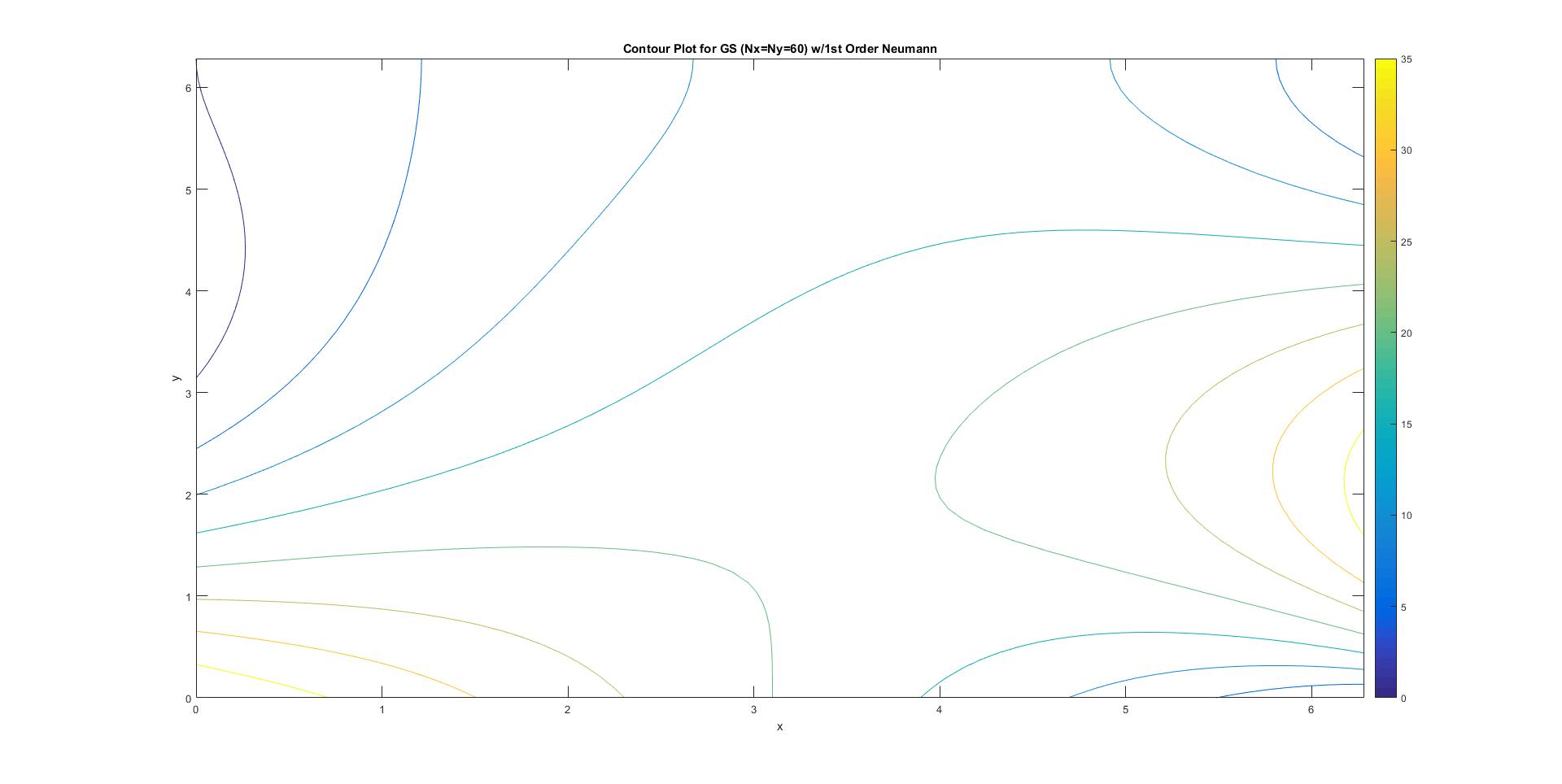


Figure 24. Contour plot for Gauss-Seidel Nx=Ny=60 and 1st Order Neumann at top boundary (y=2\*pi).

The plots below show a surface plot and contour plot for Successive-Over Relaxation with NX=Ny=60. The Uaverage is 15.2423 for this and it takes 644 iterations to converge.

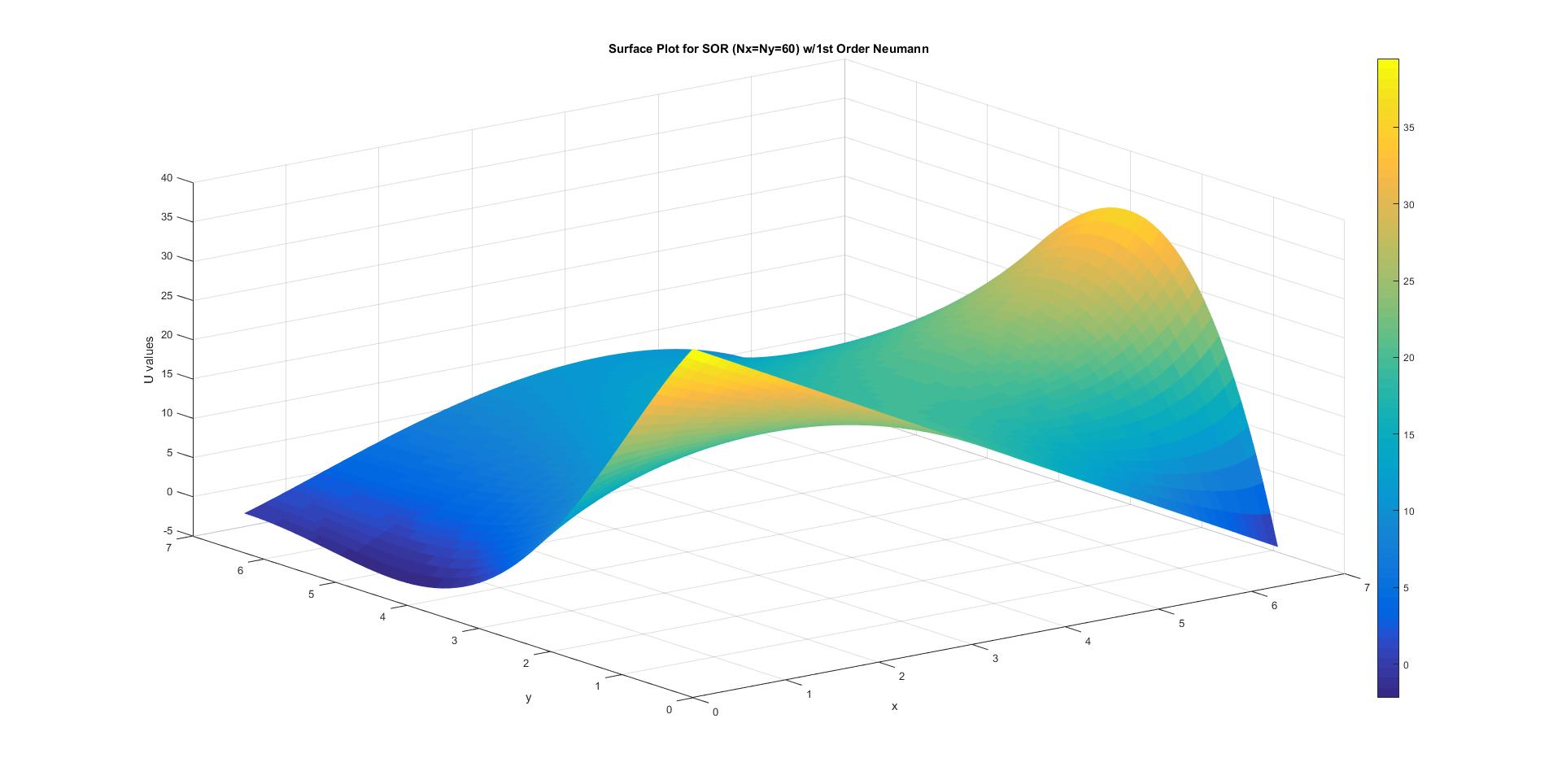
****

Figure 25. Surface plot for Successive-Over Relaxation Nx=Ny=60 and 1st Order Neumann at top boundary (y=2\*pi).

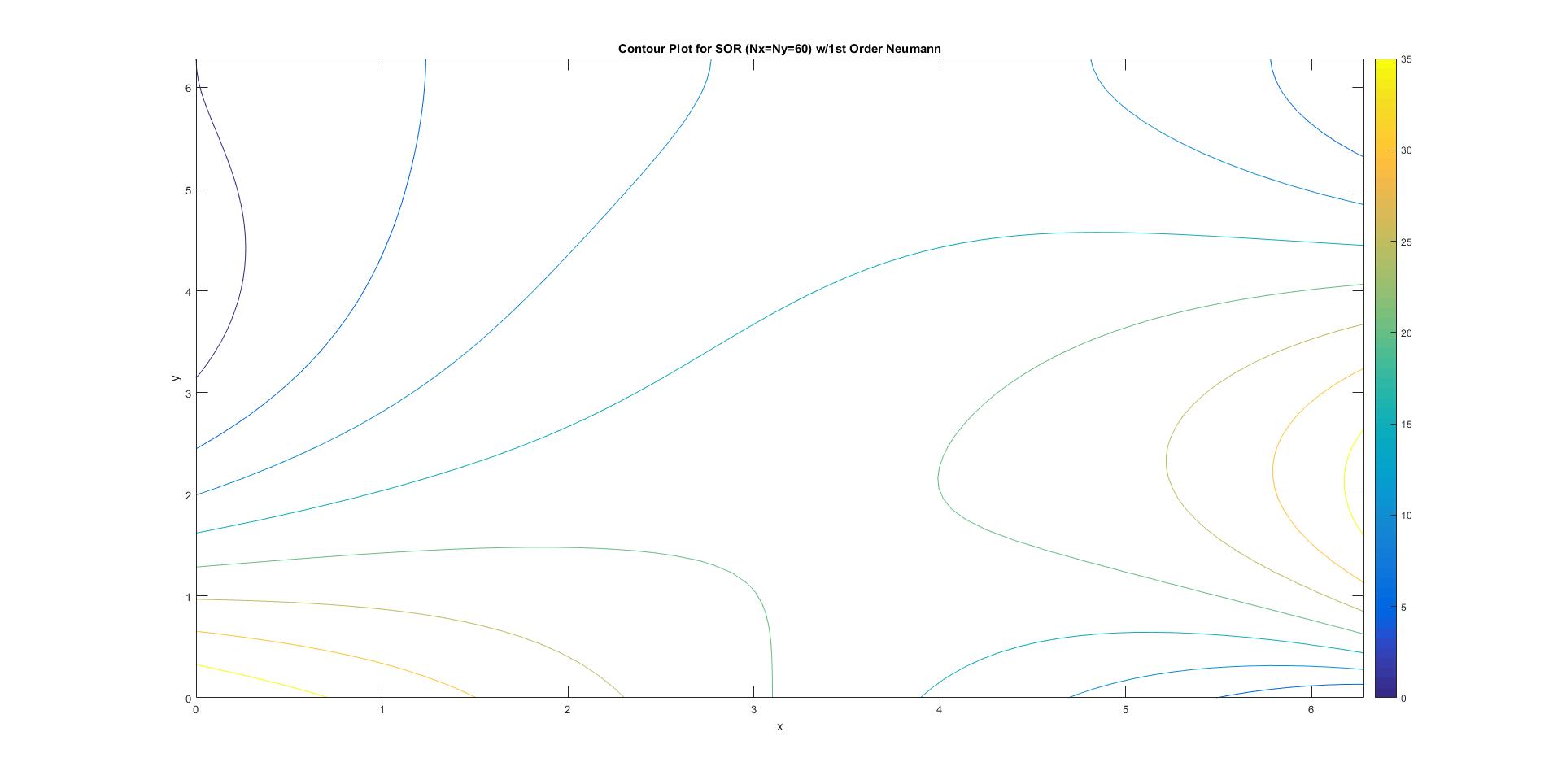


Figure 26. Contour plot for Successive-Over Relaxation Nx=Ny=60 and 1st Order Neumann at top boundary (y=2\*pi).

**Effect of Diffusive CFL**

This is not applicable since this report does not solve the diffusion equation.

**Results vs Theoretical Behavior**

One method to compare the results to theoretical behavior is the Method of Manufactured Solutions. This is where one takes their code and applies an equation with a known solution to see if it can solve it. In this case, v was set equal to v(x,y)=cos(k\*x)\*cos(h\*y) then it was substituted into the given equation . The following figures show the mathematical steps taken to solve and the boundary conditions set at each point.

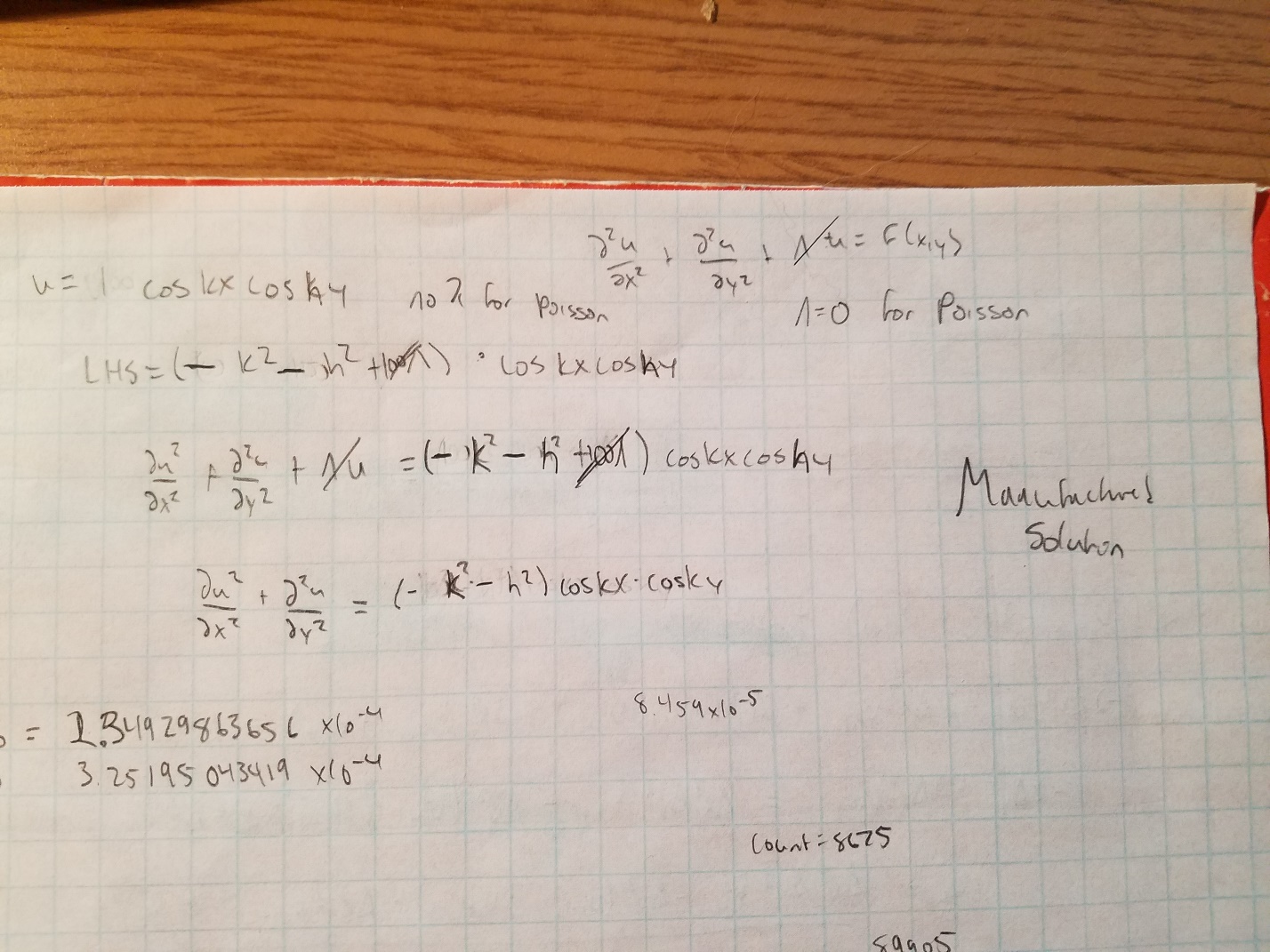


Figure 27. Solving the Manufactured Solution to enter into the code.

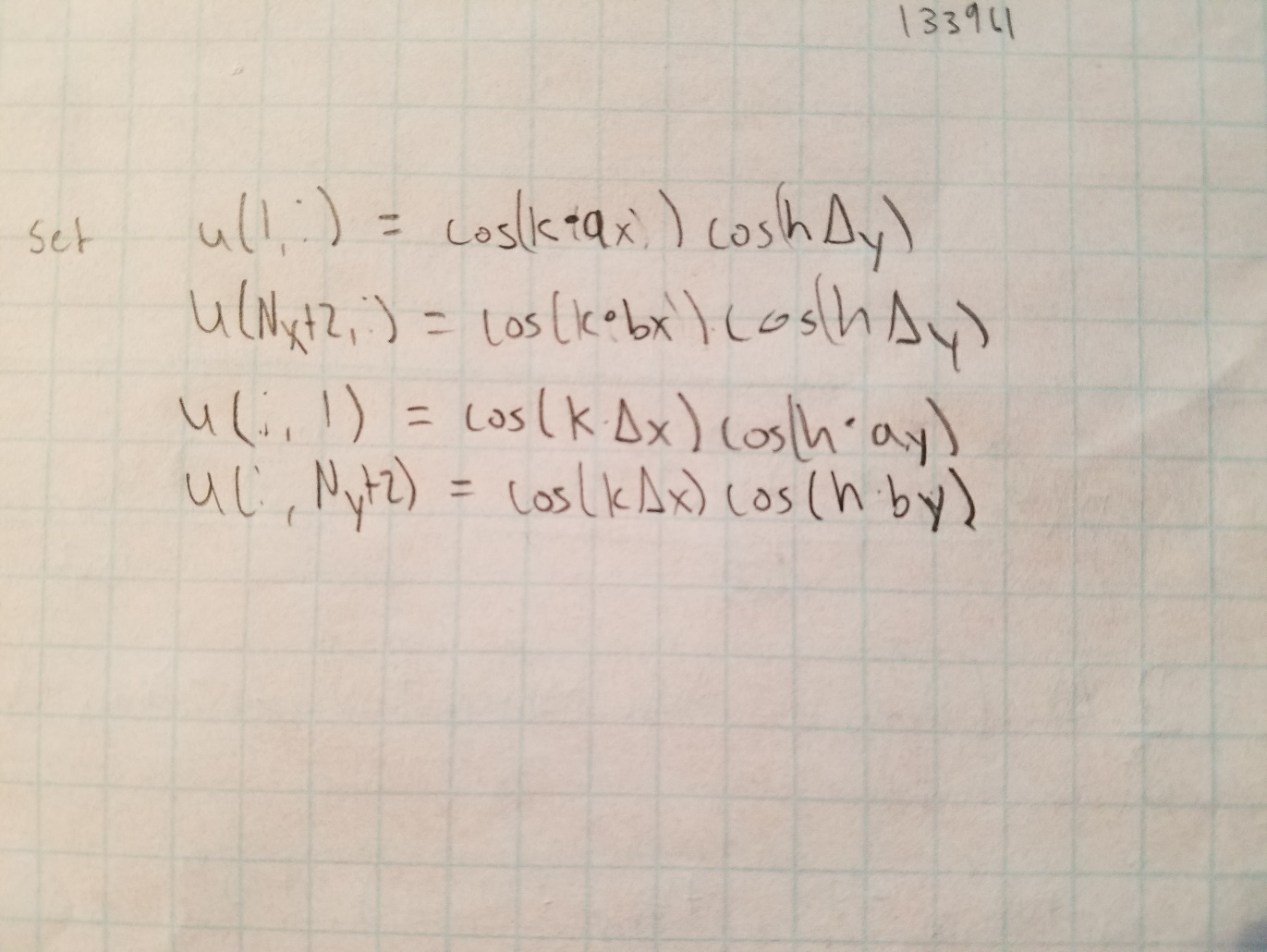


Figure 28. Boundary conditions for Manufactured Solutions.

Then after creating this, one simply inserts the code they previously used to find U (ie the Gauss-Seidel or Successive-Over Relaxation algorithim), and see if it their code can solve this as for the manufactured solution one knows Uexact. There were 2 manufactured codes that were run for this project, one to test the Gauss Seidel algorithm and one to test the Successive-Over Relaxation algorithm. The boundary conditions, value for F, and Uexact were the same. It took 5482 iterations for the Gauss Seidel to converge with a Gauss Seidel loop error of 9.9747\*10^-9 (calculated by taking the maximum value of the |Uprev – U|/|Uprev| matrix), a L1 error of 0.0012 and a L2 error of 0.0016. For Successive Over Relaxation it took 254 iterations to converge with the relaxation parameter set to 2/(1+sin(π/average(Nx &Ny)), a SOR loop error of 8.528\*10^-9 calculated the same as the GS loop error, a L1 error of 0.0012, and a L2 error of 0.0016. Since the algorithm was able to correctly solve the Manufactured Solution, it shows that the algorithm used is correct. Since the algorithm is correct, it further proves that as long as no coding errors were made in defining the boundary conditions, variables, constants, or other parameters, then the information that was shown above in the Results section should also be correct. It is highly unlikely that there are any errors in the boundary conditions, as all Dirichlet conditions were checked by computing the value at each point along the x-axis with an online solver, and comparing to what the code produced and all values matched.

The 3 GS plots below show that the graphs of U and Uexact (U=cos(k\*x)\*cos(h\*y)) are approximately the same and also show the graph of (U(i,j)-Uexact(i,j))2.

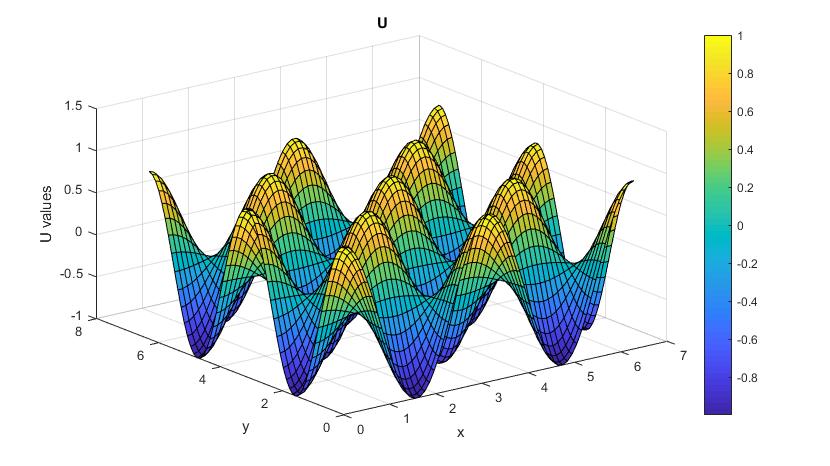


Figure 29. Plot of Manufactured Solution calculated U using the Gauss-Seidel method

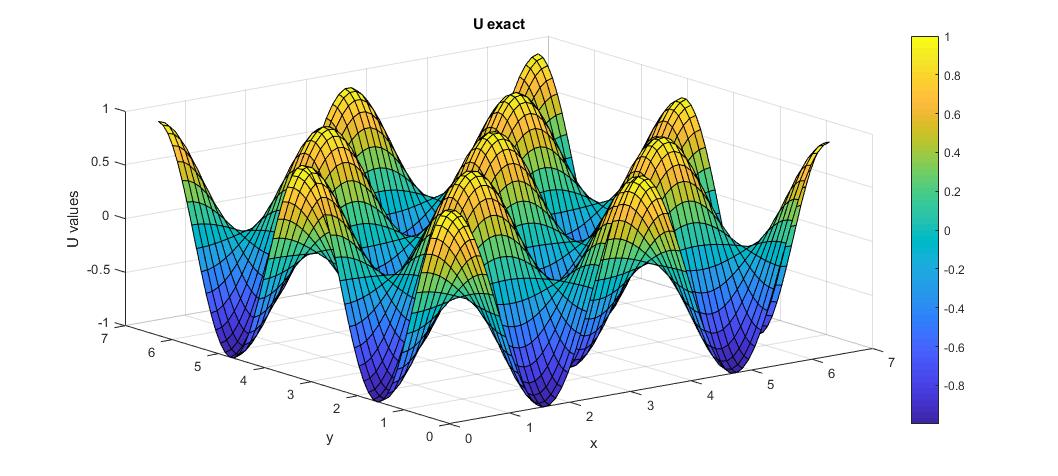


Figure 30. Plot of Manufactured Solution exact U using the Gauss-Seidel method.

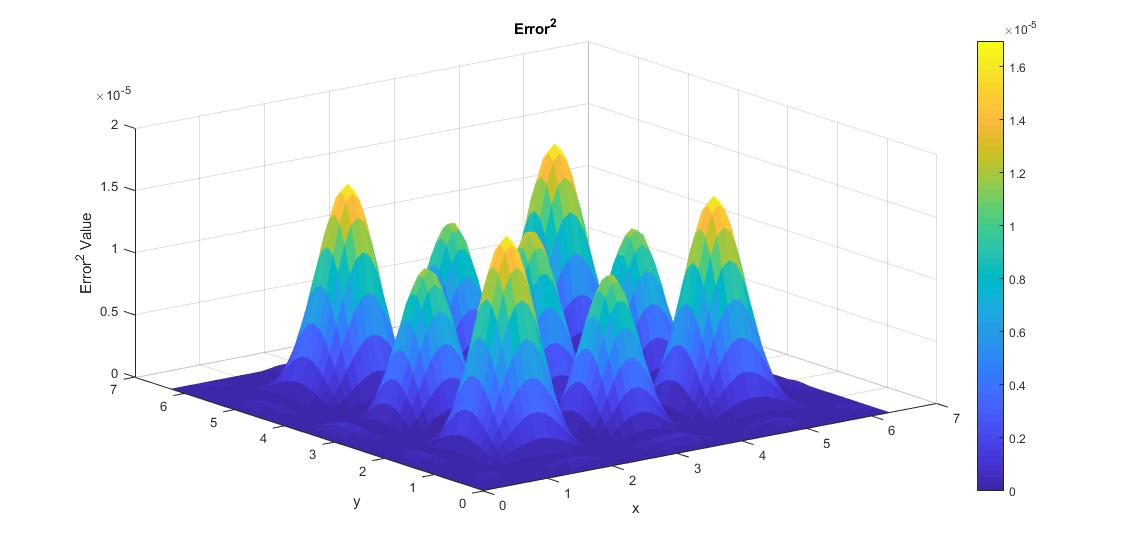


Figure 31. Plot of error2 |U(I,j)-Uexact(I,j)|2 using the Gauss-Seidel method.

The 3 SOR plots below show that the graphs of U and Uexact (U=cos(k\*x)\*cos(h\*y)) are approximately the same and also shows the graph of (U(i,j)-Uexact(i,j))2, which is the error.

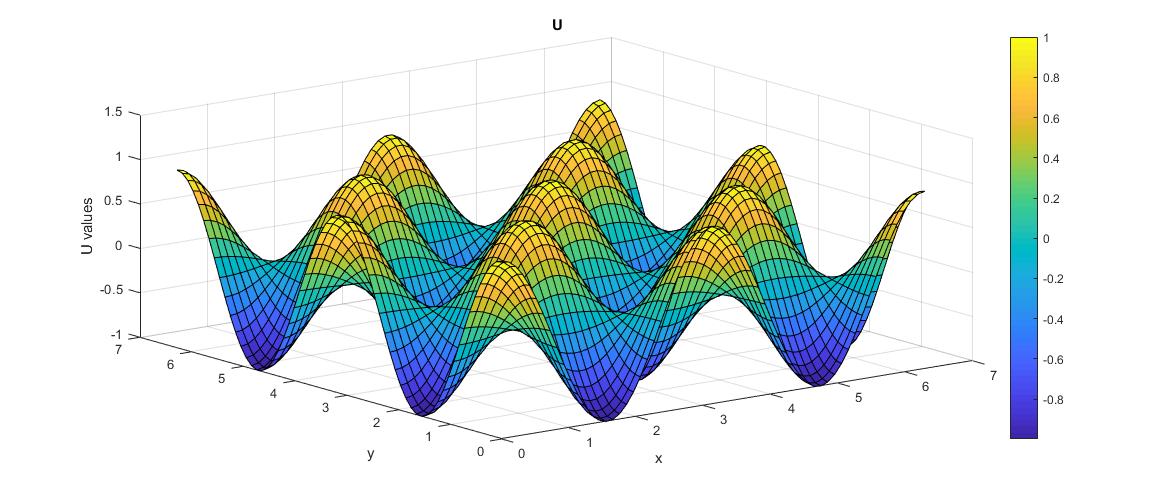


Figure 32. Plot of Manufactured Solution calculated U using the Successive-Over Relaxation method.

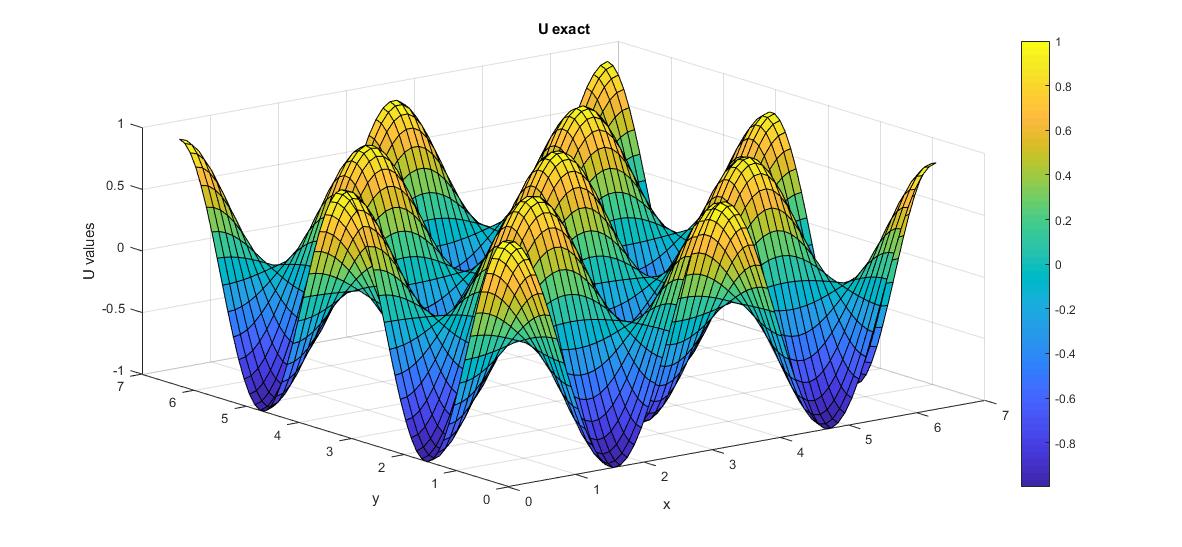


Figure 33. Plot of Manufactured Solution exact U using the Successive-Over Relaxation method.

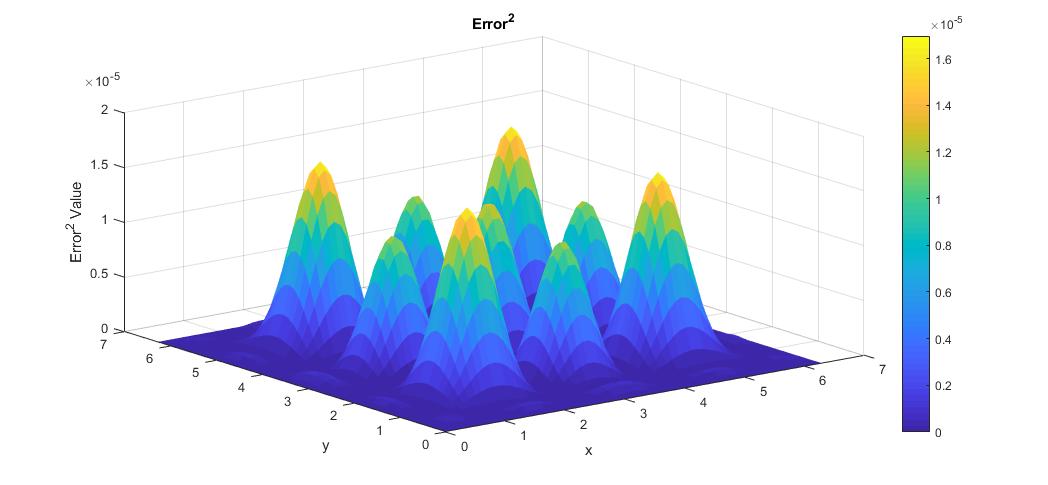


Figure 34. Plot of error2 |U(I,j)-Uexact(I,j)|2 using the Successive-Over Relaxation method.

**Grid Convergence Study**

The grid convergence study was performed on both the Manufactured Solution discussed in the Manufactured Solutions section and the regular code. For grid convergence, one doubles the nodes that are being used to discretize the equation and compares it to the previous nodes, i.e. compares 2\*N and N to see if the error has become negligible and they are the same.

First, the grid study for the manufactured solutions. For Gauss-Seidel, the code was ran for 2\*Nx, 2^2\*Nx, 2^3\*Nx, 2^4\*Nx, and 2^5\*Nx. 2^6 Nx took too long to run and converge. The table below shows the results for Gauss-Seidel: Number of Nodes, Iterations to solve GS, GS Loop Error, L2 Error for GS Loop U vs U exact, log (L2 Error), log(Δx), and Uaverage. The Uaverage appears to converge to a Uaverage=0. This makes sense as the plot of cos(k\*x)\*cos(h\*y) oscillates between -1 and 1, which means the average value of all the points should be 0.

Table 1. Table for Grid Independence of Manufactured Solutions using Gauss-Seidel.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k | Nodes in X 2^k\*Nx | Nodes in X 2^k\*Ny | Iterations to  solve GS | GS Loop Error | L2 Error for GS Loop U  vs U Exact | log(L2 Error) | log(Δx) | Uaverage sum (U(i,j))/((Nx+2)\*(Ny+2)) |
| 1 | 10 | 10 | 191 | 9.51E-09 | 0.056304 | -1.24946 | -0.2432 | 0.015677078 |
| 2 | 20 | 20 | 667 | 9.96E-09 | 0.013898 | -1.85706 | -0.5240 | 0.004348074 |
| 3 | 40 | 40 | 2218 | 9.96E-09 | 0.003499 | -2.45610 | -0.8146 | 0.001172597 |
| 4 | 80 | 80 | 8569 | 1.00E-08 | 0.000881 | -3.05484 | -1.1103 | 0.000306301 |
| 5 | 160 | 160 | 33741 | 1.00E-08 | 0.000221 | -3.65473 | -1.4086 | 7.84E-05 |
| 6 | 320 | 320 | 133961 | 1.00E-08 | 0.000056 | -4.25557 | -1.7083 | 1.98E-05 |

This table can be used to produce the 2 following plots, one that shows the Uaverage converging to 0 and L2Error converging to 0.

Figure 35. Uaverage vs Total Nodes for Manufactured Solutions using Gauss-Seidel.

Figure 36. L2Error vs Total Nodes for Manufactured Solutions using Gauss-Seidel.

For Successive-Over Relaxation Manufactured Solutions, the code was ran for 2\*Nx, 2^2\*Nx, 2^3\*Nx, 2^4\*Nx, and 2^5\*Nx, 2^6\*Nx, and 2^7\*Nx. 2^8 Nx took too long to run and converge. The table below shows the results for Gauss-Seidel: Number of Nodes, Iterations to solve SOR, SOR Loop Error, L2 Error for SOR Loop U vs U exact, log (L2 Error), log(Δx), and Uaverage. The Uaverage appears to converge to a Uaverage=0. This makes sense as the plot of cos(k\*x)\*cos(h\*y) oscillates between -1 and 1, which means the average value of all the points should be 0.

Table 2. Table for Grid Independence of Manufactured Solutions using Successive-Over Relaxation.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k | Nodes in X 2^k\*Nx | Nodes in Y 2^k\*Ny | Iterations to  solve SOR | SOR Loop Error | L2 Error for SOR Loop U  vs U Exact | log(L2 Error) | log(Δx) | Uaverage sum (U(i,j))/((Nx+2)\*(Ny+2)) |
| 1 | 10 | 10 | 52 | 8.82E-09 | 0.056304 | -1.24946 | -0.2432 | 0.015677079 |
| 2 | 20 | 20 | 88 | 8.23E-09 | 0.013898 | -1.85706 | -0.5240 | 0.004348075 |
| 3 | 40 | 40 | 168 | 9.36E-09 | 0.003499 | -2.45610 | -0.8146 | 0.001172604 |
| 4 | 80 | 80 | 340 | 5.03E-09 | 0.000881 | -3.05483 | -1.1103 | 0.000306308 |
| 5 | 160 | 160 | 677 | 9.16E-09 | 0.000221 | -3.65472 | -1.4086 | 7.84E-05 |
| 6 | 320 | 320 | 1422 | 9.58E-09 | 0.000056 | -4.25556 | -1.7083 | 1.98E-05 |
| 7 | 640 | 640 | 2899 | 9.66E-09 | 0.000014 | -4.85698 | -2.0087 | 4.99E-06 |
| 8 | 1280 | 1280 | 5796 | 8.06E-09 | 3.48E-06 | -5.458709787 | -2.30937 | 1.25E-06 |

This table can be used to produce the 2 following plots, one that shows the Uaverage converging to 0 and L2Error converging to 0.

Figure 37. Uaverage vs Total Nodes for Manufactured Solutions using Successive-Over Relaxation.

Figure 38. L2Error vs Total Nodes for Manufactured Solutions using Successive-Over Relaxation.

Now, the grid study for the actual code. For Gauss-Seidel, the code was ran for 2\*Nx, 2^2\*Nx, 2^3\*Nx, 2^4\*Nx, and 2^5\*Nx. 2^6 Nx took too long to run and converge. The table below shows the results for Gauss-Seidel: Number of Nodes, Iterations to solve GS, GS Loop Error, L1 Error for GS Loop U vs U exact, log (L1 Error), log(Δx), and Uaverage. The Uaverage appears to converge to a Uaverage=15.33. This makes sense as the boundary conditions vary from 39.5 to -2.2, which is close to 15. L1 error is calculated using U that is found from the final iteration of the SOR loop and the U that is found from the final-1 iteration of the SOR loop.

Table 3. Table for Grid Independence of actual code using Gauss-Seidel.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k | Nodes in X 2^k\*Nx | Nodes in Y 2^k\*Ny | Iterations to  solve GS | GS Loop Error | L1 Error for GS Loop U  vs U Exact | log(L2 Error) | log(Δx) | Uaverage sum (U(i,j))/((Nx+2)\*(Ny+2)) |
| 1 | 10 | 10 | 324 | 9.54E-09 | 3.58E-08 | -7.4465 | -0.2432 | 15.08736204 |
| 2 | 20 | 20 | 1189 | 9.97E-09 | 8.34E-09 | -8.0789 | -0.5240 | 15.21183923 |
| 3 | 40 | 40 | 5047 | 9.99E-09 | 3.17E-10 | -9.4988 | -0.8146 | 15.27540327 |
| 4 | 80 | 80 | 19300 | 1.00E-08 | 1.15E-10 | -9.9382 | -1.1103 | 15.30802261 |
| 5 | 160 | 160 | 75064 | 1.00E-08 | 3.82E-11 | -10.4183 | -1.4086 | 15.32462461 |
| 6 | 320 | 320 | 278783 | 1.00E-08 | 3.08E-11 | -10.5110 | -1.7083 | 15.33300972 |

This table can be used to produce the 2 following plots, one that shows the Uaverage converging to 15.33 and L2Error converging to 0.

Figure 39. Uaverage vs Total Nodes for actual code using Gauss-Seidel.

Figure 40. L2Error vs Total Nodes for actual code using Gauss-Seidel.

For Successive-Over Relaxation Manufactured Solutions, the code was ran for 2\*Nx, 2^2\*Nx, 2^3\*Nx, 2^4\*Nx, and 2^5\*Nx, 2^6\*Nx, and 2^7\*Nx. 2^8 Nx took too long to run and converge. The table below shows the results for Gauss-Seidel: Number of Nodes, Iterations to solve GS, GS Loop Error, L1 Error for SOR Loop U vs U exact, log (L1 Error), log(Δx), and Uaverage. The Uaverage appears to converge to a Uaverage=15.33. This makes sense as the boundary conditions vary from 39.5 to -2.2, which is close to 15. L1 error is calculated using U that is found from the final iteration of the GS loop and the U that is found from the final-1 iteration of the GS loop.

Table 4. Table for Grid Independence of actual code using Successive-Over Relaxation.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k | Nodes in X 2^k\*Nx | Nodes in Y 2^k\*Ny | Iterations to  solve SOR | SOR Loop Error | L1 Error for SOR Loop U  vs U Exact | log(L1 Error) | log(Δx) | Uaverage sum (U(i,j))/((Nx+2)\*(Ny+2)) |
| 1 | 10 | 10 | 95 | 9.48E-09 | 3.01E-08 | -7.5220 | -0.2432 | 15.0874 |
| 2 | 20 | 20 | 180 | 9.84E-09 | 6.01E-09 | -8.2209 | -0.5240 | 15.2118 |
| 3 | 40 | 40 | 384 | 9.48E-09 | 2.42E-10 | -9.6159 | -0.8146 | 15.2754 |
| 4 | 80 | 80 | 763 | 9.90E-09 | 7.12E-11 | -10.1472 | -1.1103 | 15.3080 |
| 5 | 160 | 160 | 1510 | 9.99E-09 | 3.18E-11 | -10.4981 | -1.4086 | 15.3246 |
| 6 | 320 | 320 | 2940 | 9.97E-09 | 2.43E-11 | -10.6150 | -1.7083 | 15.3330 |
| 7 | 640 | 640 | 7181 | 7.34E-09 | 8.92E-14 | -13.0494 | -2.0087 | 15.3372 |
| 8 | 1280 | 1280 | 12234 | 9.98E-09 | 2.16E-12 | -11.6656 | -2.3094 | 15.3393 |

This table can be used to produce the 2 following plots, one that shows the Uaverage converging to 15.33 and L2Error converging to 0.

Figure 41. Uaverage vs Total Nodes for actual code using Successive-Over Relaxation.

Figure 42. L2 Error vs Total Nodes for actual code using Successive-Over Relaxation.

**Order of Spatial Accuracy of Discretization**

The order of spatial accuracy was calculated by using the Manufactured Solution discussed above. It was found by taking the log(L2 Error) vs log(Δx) and plotting that for each time Nx was doubled. The slope of the line for SOR was found to be approximately 2 and the slope of the line for Gauss-Seidel was found to be approximately 2. This shows that the approximation is indeed O(h^2) as expected based on theoretical data. This result is based on the fact that if the discretization has an order of accuracy N and inner nodes with length Δ, for example, one would expect that for some constant C, L2 error should satisfy the relation:

Or if one takes the logarithm:

This equation shows that the logarithm of the error should be approximately N times the logarithm of the length of the inner nodes, Δ, in a given discretization. So, when an approximation of O(h2) is used, one would expect the slope to be approximately 2.

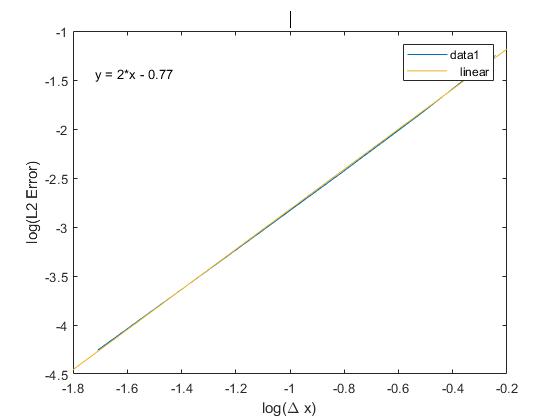


Figure 43. log(L2 Error) vs log(Δx) for GS. The slope is approximately 2 as expected which shows that the approximation is indeed O(h2).

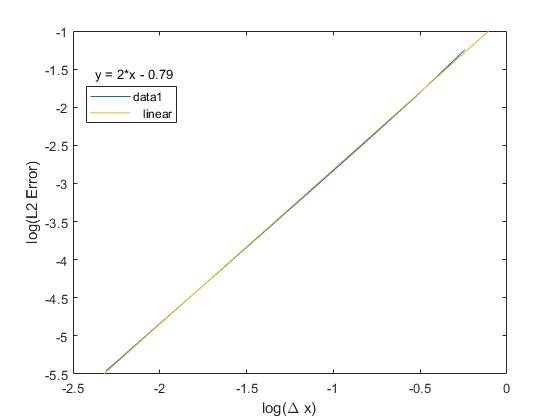


Figure 44. log(L2 Error) vs log(Δx) for SOR. The slope is approximately 2 as expected which shows that the approximation is indeed O(h2).

For the actual code, similar results were found. Using Gauss-Seidel, the slope of the line of log(L1 error) vs log(Δx) was found to be 2.211, which is approximately 2. The L1 Error was found by taking |the final iteration of U from the GS loop – the final iteration-1 of U|/(Nx\*Ny).

Figure 45. log(L1 Error) vs log(Δx) for GS actual code. The slope is approximately 2 as expected which shows that the approximation is indeed O(h2).

Using Successive-Over Relaxation, the slope of the line of log(L1 error) vs log(Δx) was found to be 2.2687, which is approximately 2. The L1 Error was found by taking |the final iteration of U from the GS loop – the final iteration-1 of U|/(Nx\*Ny).

Figure 46. log(L1 Error) vs log(Δx) for SOR actual code. The slope is approximately 2 as expected which shows that the approximation is indeed O(h2).