Antonio Hernandez

UH ID # 1168567

Project AHc1-1

2D Poisson’s Equation

Please take note that I had Helmholtz but was given the opportunity to set and therefore solve for Poisson’s 2D equation.

**Abstract:**

The 2D Poisson’s Equation was solved using the Gauss-Seidel and SOR methods. This project included three Direlecht conditions and one Neuman condition. The Gauss-Seidel method is a method in which the unknown values of the Poisson Equation are solved using an iterative method. To be more specific once the Direlecht and Neumann boundary conditions have been set the unknown values are what lie within these boundary conditions. By giving an initial value to the unknowns or u’s (for example u =0) the Gauss-Seidel method will calculate the values of u with the implementation of the values from the boundaries and are updated for every calculation. This leads to the explanation of why it is an iterative method because the calculations will have to be done over and over until there is almost no change from the last two successive calculations of u’s that are within the boundary conditions. SOR is a modification of Gauss-Seidel in which a magnification factor is introduced in this case we call that factor ‘w’. By implementing ‘w’ we can reach the desired steady value of u at a faster rate than Gauss-Seidel. In other words, the amount of times that u will be calculated to reach a small error from the last two successive calculations will be less.

**Discretization of the 2D Poisson equation and the given Neumann condition:**

In the case of Poisson, we get the general discretization;

where

Assuming that

In the case of the Neumann condition;

We begin by discretizing the differential equation and then include it to Eq. 1. Since the range of x is: -Pi < x <PI we evaluate Eq. 1 with at i =N+1 for the right boundary Neumann condition.

Eq. 2→

🡪Eq.2

Eq. 3 →

As a result, Eq. 1 provides the Direlecht iterative equation and Eq. 2 provides the Neumann iterative equation.

**Description of Algorithm:**

The heart of the algorithm is encased in the inner for loop;

for jj = 2:N+1

for ii = 2:N+1

U(ii,jj) = (0.25\*(U(ii-1,jj)+U(ii+1,jj)+U(ii,jj-1)+U(ii,jj+1))-D\*F(ii,jj));

end

U(N+2,jj) = (0.25\*(2\*U(N+1,jj)+U(N+2,jj-1)+U(N+2,jj+1))-D\*F(N+2));

for ii = 2:N+1

if U(ii,jj) ~= 0

U(ii,jj) = w\*U(ii,jj)+(1-w)\*uold(ii,jj);

err(ii,jj) = abs((U(ii,jj) - uold(ii,jj))/U(ii,jj)) \* 100;

end

end

end

The firs U(ii,jj) equation is the description of Eq. 1 for which all the inner unknown values were solved and U(ii,jj) is the value that keeps getting updated for every iteration. Then U(N+2,jj) adds the description of Eq. 3 which solves the Neuman Boundary values at x = Pi for all the values of y form -Pi to Pi. SOR was then implemented with the form of U(ii,jj) = w\*U(ii,jj)+(1-w)\*uold(ii,jj);

In this case when w = 1 it simply runs the Gauss-Seidel function but when we use 1<w<2 we are able to magnify the updated value of U(ii,jj) and run it as SOR.

Another important factor within the code is that in order to have a proper comparison between the previous value of U (or uold in the code) and the most recent value of U is that we need to use the relative error which is err(ii,jj) = abs((U(ii,jj) - uold(ii,jj))/U(ii,jj)) \* 100;

The following is the code that encases the for loops which is shown as a ……………

tol = 10^-6;

it = 8\*N^2;

error = 1+tol;

iterations = 0;

err = zeros(N+1,N);

tic

while error > tol

iterations = iterations+1;

uold = U;

w = 1;

…………………………………………………………………………………………………………………………………………………………….

if iterations > it

disp('Max iteration was reached before tolerance was reached');

break

end

error = max(max(err));

if error <= tol

fprintf('Tolerance reached at %d iterations \n',iterations);

end

end

The main function for this is to make sure that the code does not run forever. Instead the code will run until it reaches a tolerance or difference of 10^-6 from the two most recent calculations of U(ii,jj). In addition, this code lets the user know if the tolerance was not reached by stating

'Max iteration was reached before tolerance was reached'.

And to confirm that SOR is indeed magnifying the change of U(ii,jj) the code will let you know how many iterations occurred as you change w from 1 to 2. And ultimately when the tolerance of 10^-6 is met the code will stop.

**Technical Computer Specifications:**

HP Pavilion x360 m3 Convertible :

Processor: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.71 GHz

Installed RAM: 8.00 GB(7.88 GB usable)

# of Cores: 2

# of Threads: 4

Processor Base Frequency: 2.50 GHz

Max Turbo Frequency: 3.10 GHz

Cache: 3 MB SmartCache

Max Memory Size 32 GB

Max # Memory Bandwidth: 34.1 GB/s

**Results:**

The following figures show how the graph consistently stays the same with the changing of N segments.

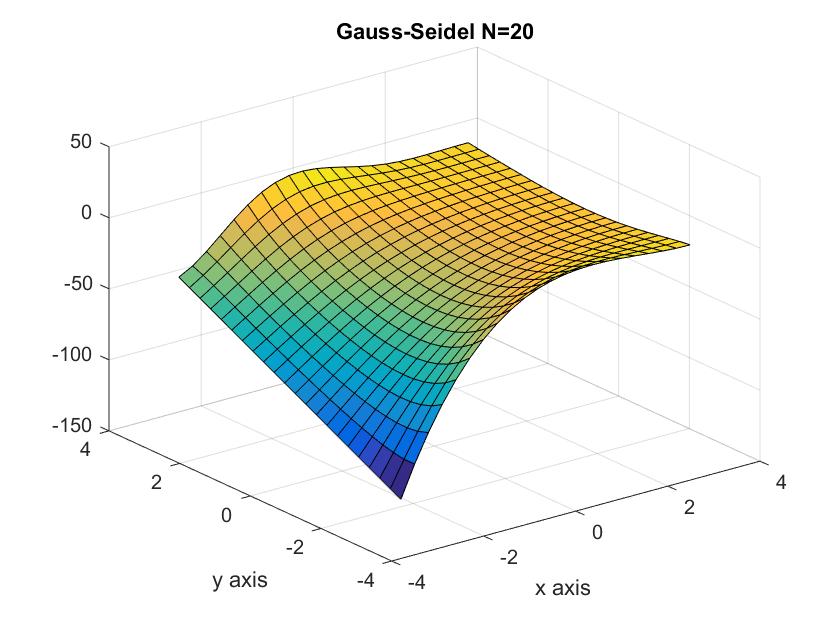


Figure : Gauss-Seidel N=20

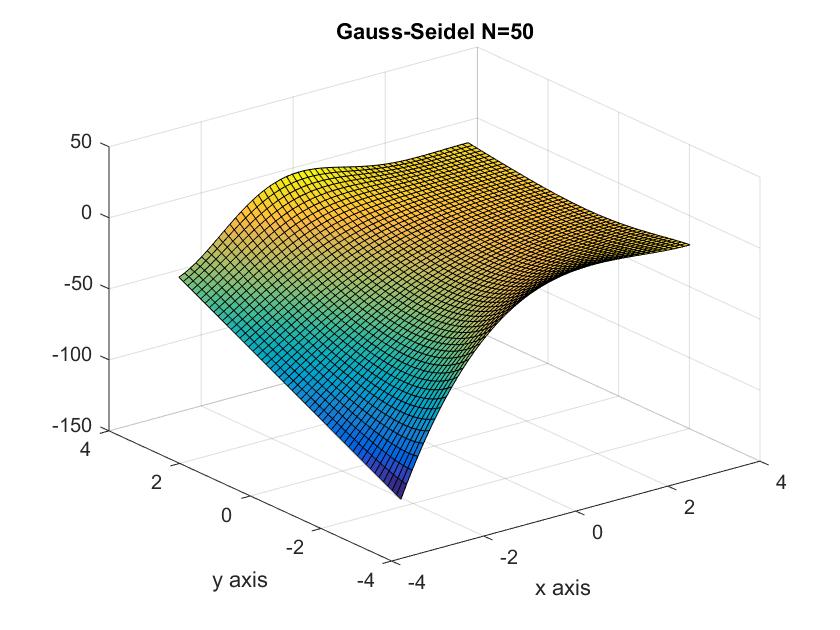


Figure : Gauss-Seidel N=50

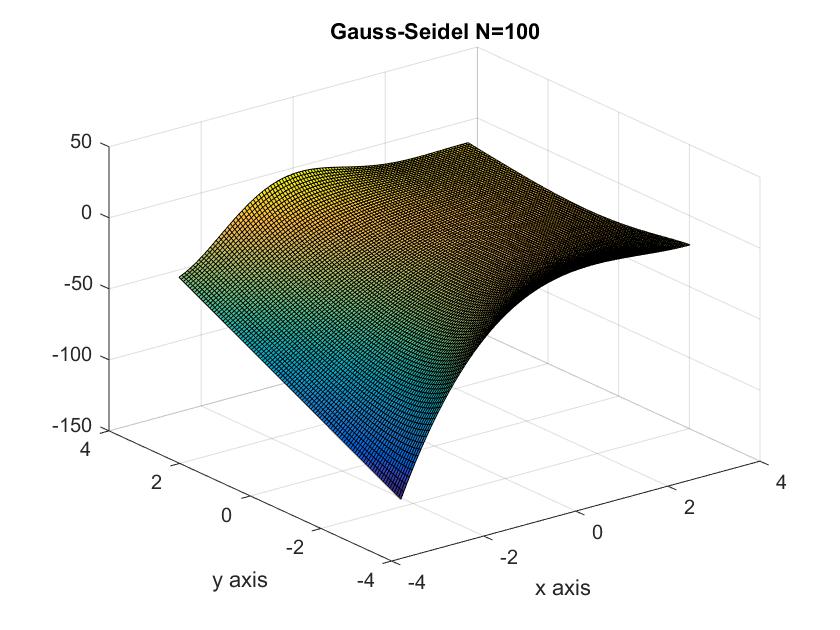


Figure :Gauss-Seidel N=100

Since SOR should reach the tolerance of 10^-6 at fewer iterations than Gauss-Seidel it was important to compare the amount of iterations with varying values of w. It is important to note that in the case where w=1 it is running Gauss-Seidel. The comparison is shown in Figure 4 where we see that SOR definitely lowers the amount of iterations compared to Gauss-Seidel.

Figure : Gauss-Seidel vs SOR

Since SOR requires w to be within the range 1< w < 2 it was important to find the best value of w to run the code with the least amount of iterations. To get an accurate comparison, the iterations were evaluated with N = 50 at different values of w. And ultimately a value of w = 1.76 was the best value with the lowest amount of iterations of 2803 at N= 50. As you can see in Figure 5 the amount of iterations begins to increase after w = 1.76.

Figure :Incresing w decreases iterations up to w = 1.76

In addition we can see that the value of w that you choose is a very specific and delicate value. We can see in Figure 6 that at w = 1.79 the magnitude at which U(ii,jj) is changing becomes too great and leads to a magnification of error which gives another reason as to why w is within the range of 1< w < 1.

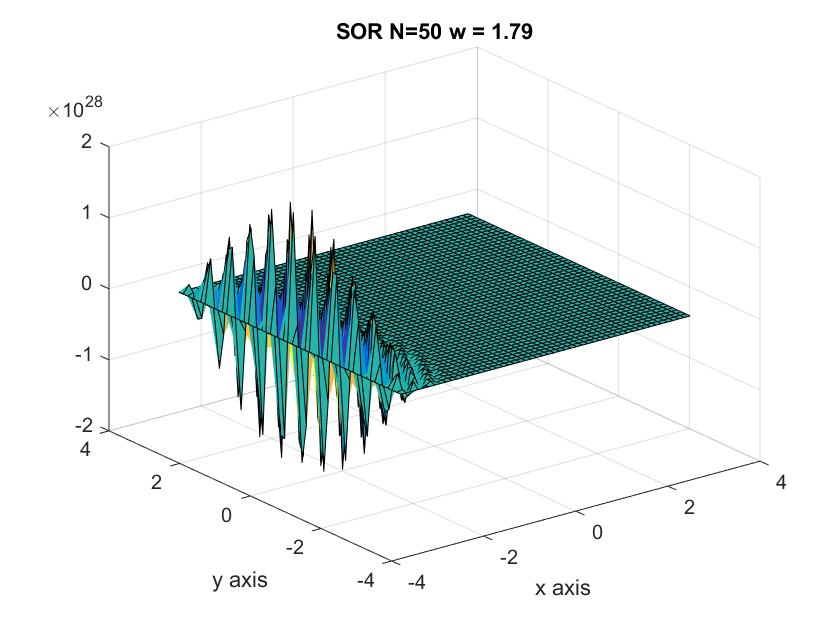


Figure :Error increase at w =1.79