MECE 5397: Final Project Report

Project A – Poisson Equation (AP02-4)

Benjamin Rose

1184962

May 5, 2017

The equation discretized was the two-dimensional Poisson equation. The equation, shown in figure 1 below, covered the square domain (x,y) with both x and y varying from 0 to 2\*pi. The three boundary conditions are shown in figure 2. The forcing term F(x,y) shown in figure 2 appears in the right hand side of the equation of figure 1. Three of the boundary conditions were of the Dirichlet type, while the other was Neumann, which are shown in figure 3.

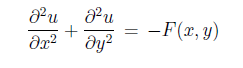


Figure 1: Poisson Equation



Figure 2: Forcing Term

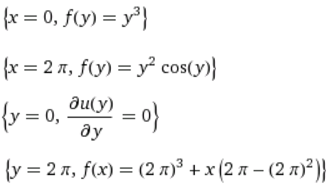


Figure 3: Boundary Conditions

To discretize the equation, the Gauss-Seidel method was used with the 5-point stencil approximation for the left hand side of the equation, shown below in figure 4. This method uses the points directly above, below, left and right of the desired point u(x,y) to approximate the solution to the Poisson Equation. For this method to work, the boundary conditions must first be set up, and an initial guess for values of u must be given, and the equation iterated many times until the values of u converge to the exact solution.

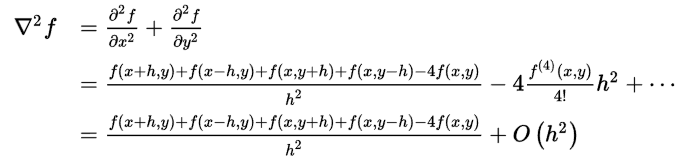


Figure 4: 5-Point Stencil Discetization

Setting up the Dirichlet boundary conditions is straightforward, as they are explicitly given as functions of x and y. The Neumann condition specifies a derivative with respect to y of 0 at y=0. To accomplish this, within the main loop of the function where x is being varied from 1 to N+1, y is fixed at 1 (corresponding to the real value 0) and the main equation is repeated but with the points u(x,y+h) and u(x,y-h) assumed to be the same, resulting in du/dy=0. The main loop of the function has x varied from 1 to N+1, but y is only varied from 2 to N+1, as the Neumann B.C. takes care of the value of y at node 1.

To ensure the program carries out enough iterations to converge, error needs to be quantified in some manner. This code uses the L-infinity norm in its definition for error, taking the difference between the L-infinity norm between the current and previous iterations and dividing that value by the previous norm. This causes several problems, as the boundaries cannot be included since they are constant at the Dirichlet boundaries and will result in only 2 iterations. Secondly, for very fine grids, the edges of the domain converge much faster than the interior, resulting in the program terminating early. To combat these issues, error is defined as very small (around 10^-9) to ensure enough iterations are carried out for the solution to converge, and the boundaries are excluded from calculation of the norm.

Whenever the number of iterations carried out is a multiple of variable Q, the code saves the values of u(x,y) and iter to a backup file, backup.mat. If the code fails, the backup can be loaded, and the code continued from where it left off by disabling the lines of code that set up the original values of u and the iteration number.

The solutions generated by the code seem to be grid independent until the grid becomes very fine (on the order of 1000x1000). For test grids 5x5, 10x10, 30x30 and 100x100, the surface plots look almost identical. These results are shown below in figures 5 and 6 illustrate the grid independence of the solution. The main factor contributing to the accuracy of the solution seems to be the number of iterations carried out. For sufficiently low values of error, the solution converges as illustrated in figures 7 and 8, where error is varied on a 25x25 grid. For values of error below 10^-8, no noticeable difference in successive surface plots occurs. For the 25x25 grid, an error of 10^-8 corresponds to 781 iterations.

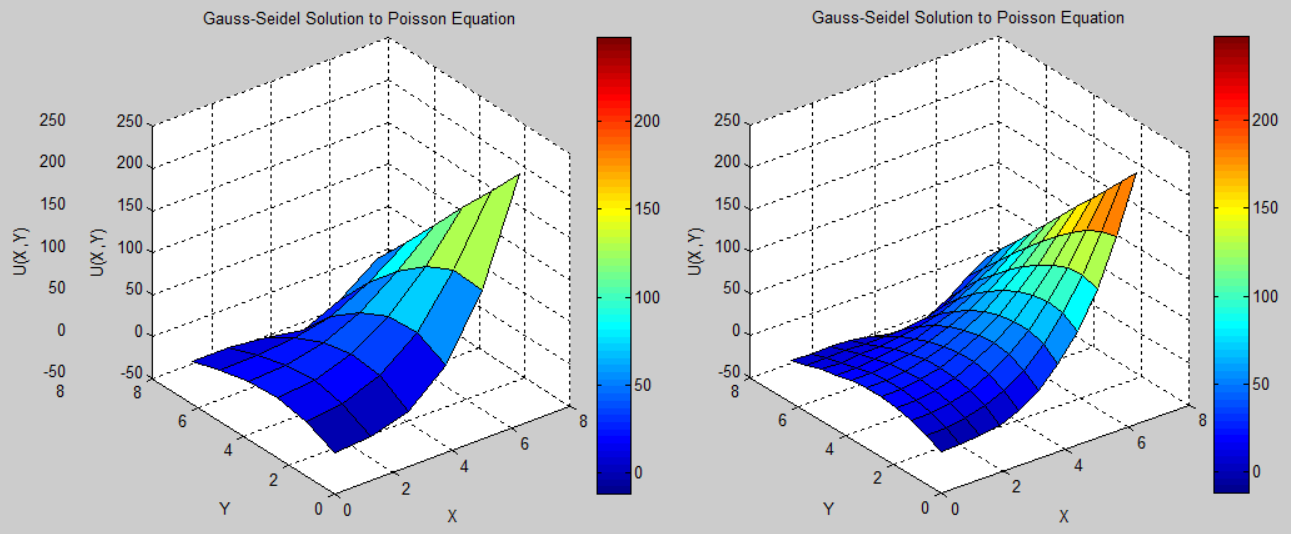


Figure 5: Left (5x5 grid), Right (10x10 grid)

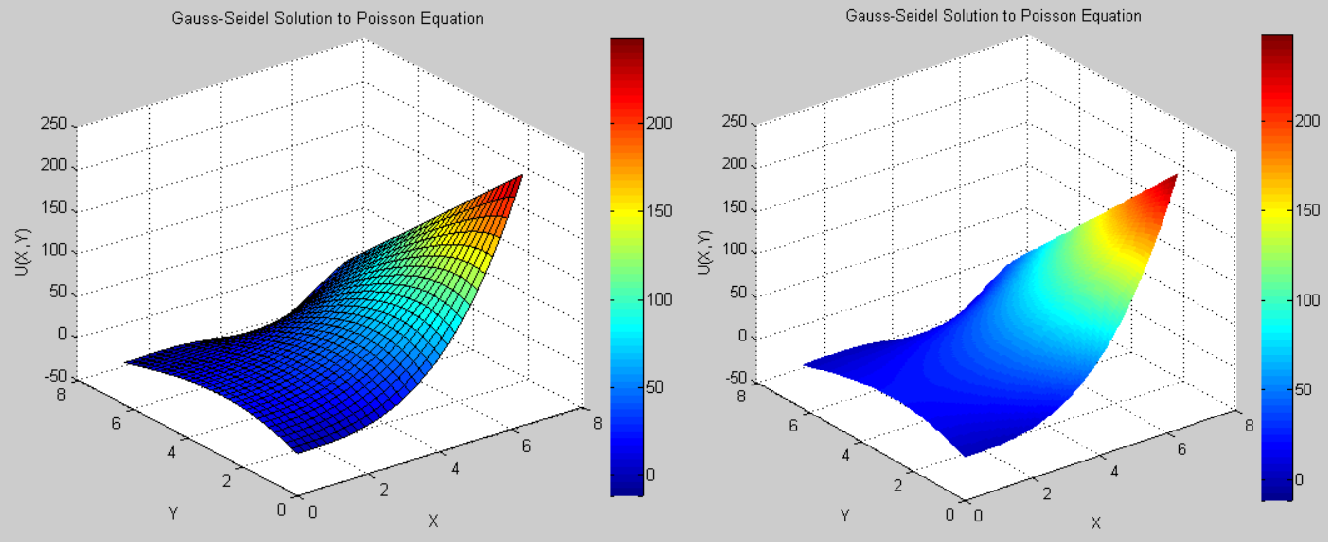


Figure 6: Left (30x30 grid), Right (100x100 grid)

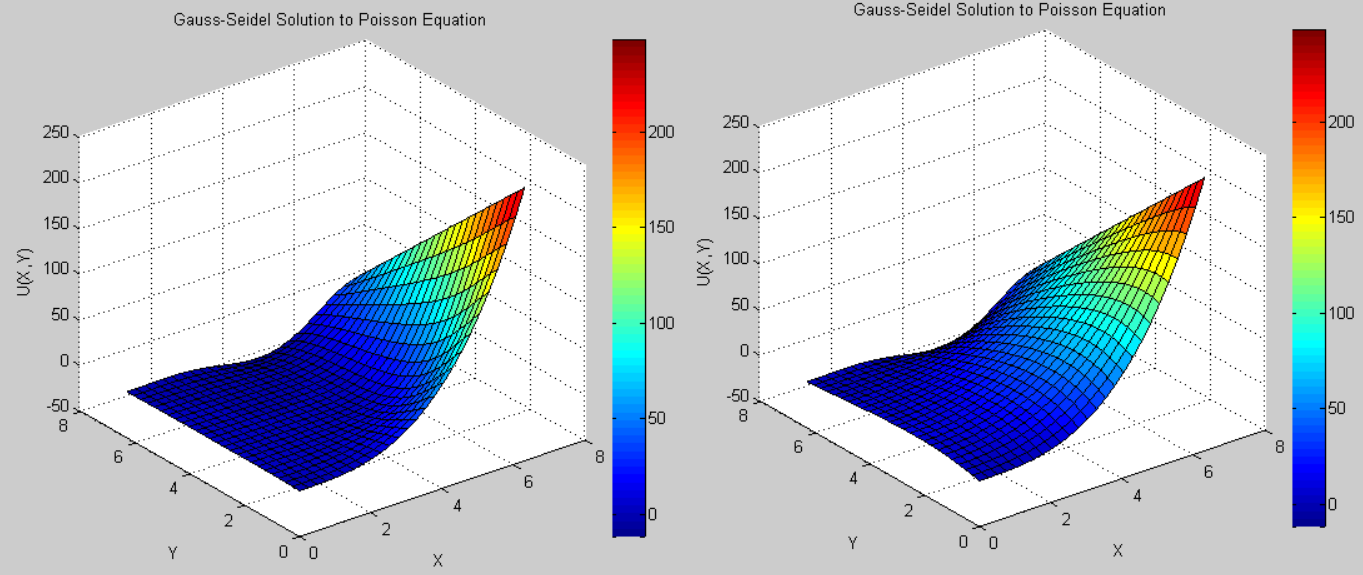


Figure 7: Left (Error=10^-3), Right (Error=10^-5)

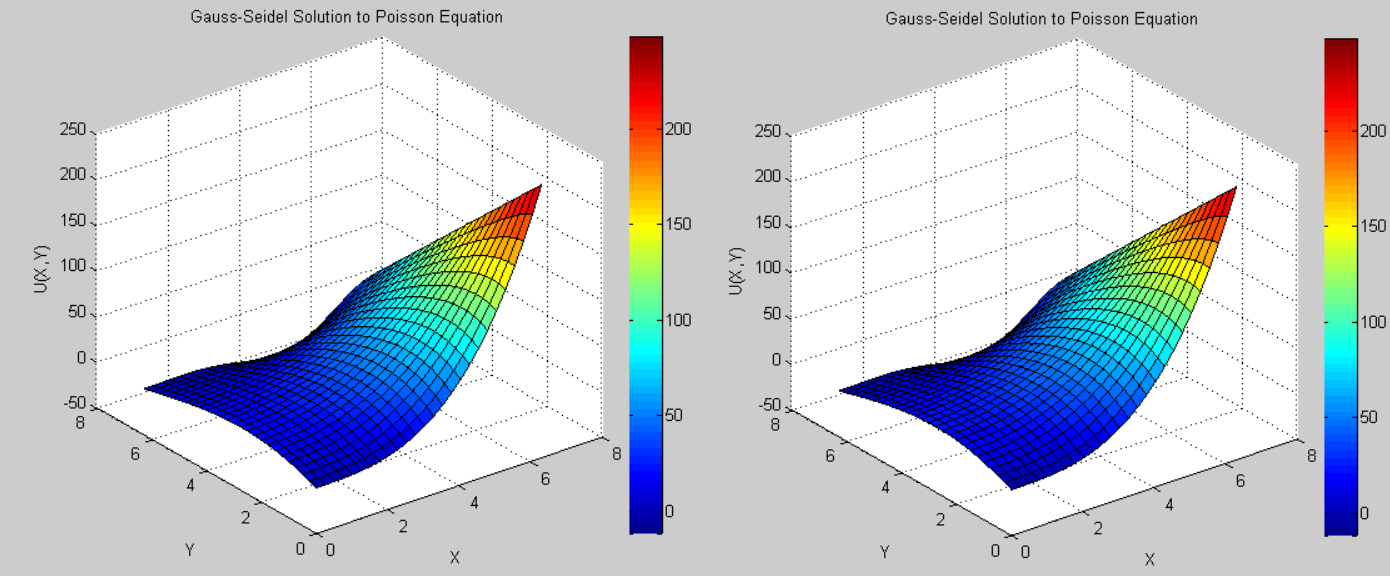


Figure 8: Left (Error=10^-6), Right (Error=10^-8)

The alternative method used to solve the equation was the Jacobi method. This method differs from Gauss-Seidel in that the program only uses values of u from the previous iteration when calculating the next, whereas Gauss-Seidel uses the updated values of u as soon as they are available. Each value of u(x,y) requires the values of the four points above, below, left and right of it to calculate the next iteration. Since the code traverses along rows, this means for Gauss-Seidel the points above and to the left of u(x,y) are from the current iteration, whereas for Jacobi all points are from the previous iteration. The result is that the Jacobi method takes about twice as many iterations for the same error cutoff, however the solution is identical. For example, for error cutoff value of 10^-9, Gauss-Seidel takes 7045 iterations and Jacobi takes 11869 iterations.

When F(x,y) is set to zero with grid size 100x100, the resulting surface plot appears nearly identical to the original case of F(x,y). When running the two side-by-side, a subtle difference can be seen, namely when F(x,y)=0 the surface plot is slightly more elevated. For example, for the original F(x,y), node u(50,50)=48.6886, whereas for F(x,y)=0, u(50,50)=51.0974. Both cases were tested for the error cutoff value of 10^-9. The two cases can be seen in figures 9 and 10.

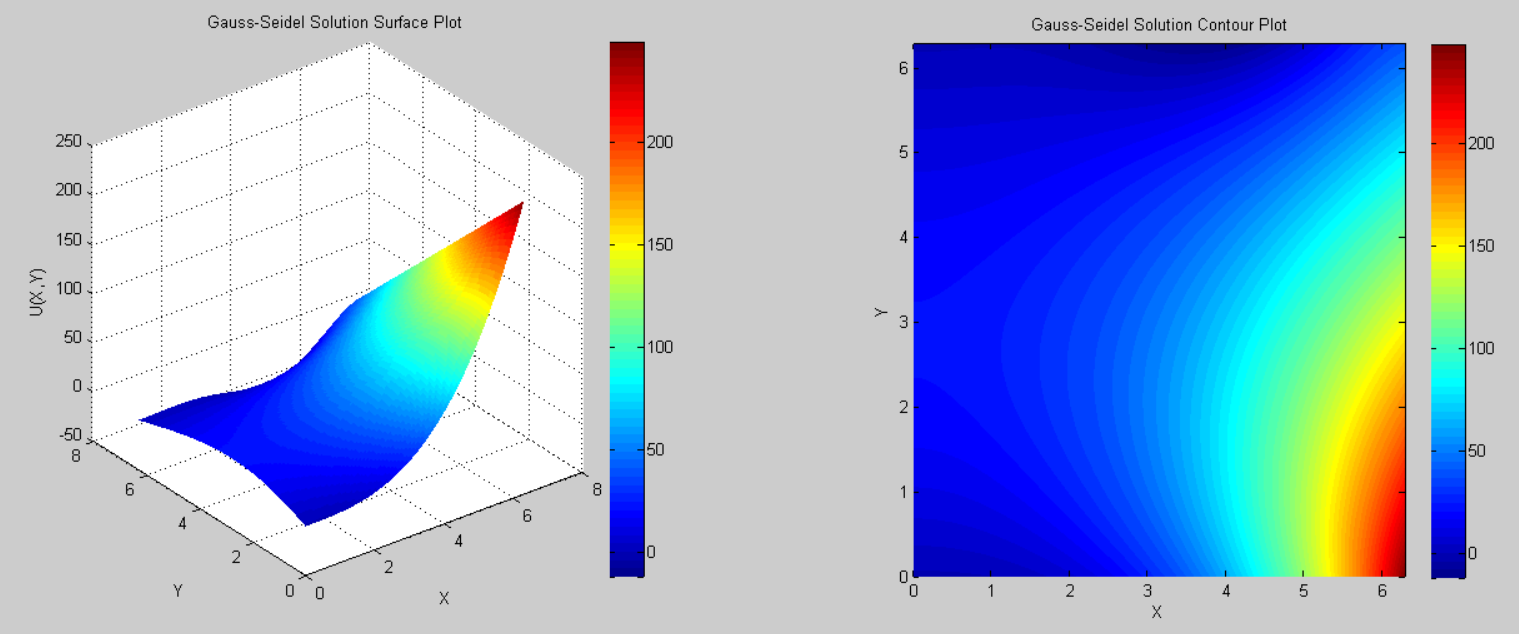


Figure 9: Gauss-Seidel Solution With Original F(X,Y)

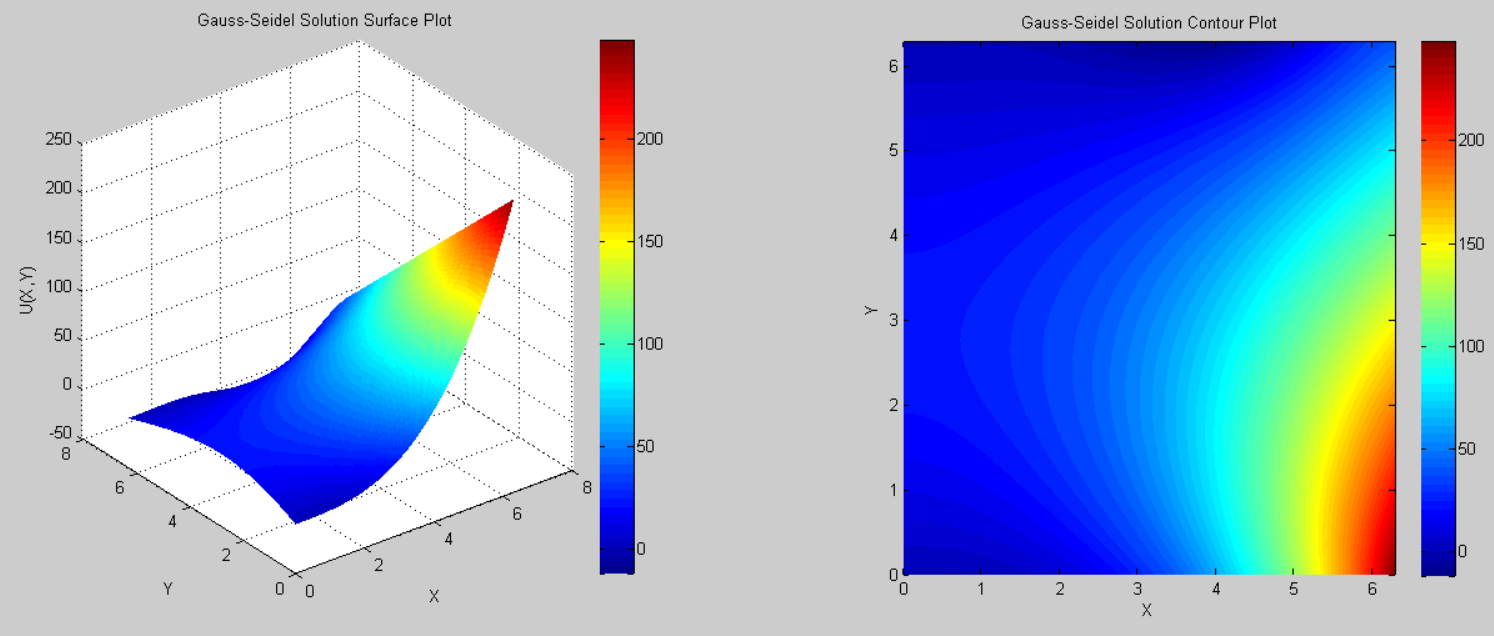


Figure : Gauss-Seidel Solution with F(X,Y)=0

The Following is the code for the basic Gauss-Seidel Method. The Jacobi code can be found in the /src/ folder.

%N=number of grid sections, there are N+1 nodes

%h is the spacing between nodes

N=30; h=(2\*pi)/N;

x=0:h:2\*pi;

y=0:h:2\*pi;

%If you are loading backup values of u and iter, place % before line 7 and 25 to continue

u=ones(N+1,N+1);

%Values for F(x,y)

F=zeros(N+1,N+1);

for i=1:N+1

for j=1:N+1

F(i,j)=cos(x(i)/2+pi/2)\*sin(y(j)/2);

end

end

%Dirichlet Boundary Conditions

for i=1:1:N+1

u(1,i)=(y(i))^3;

u(N+1,i)=((y(i))^2)\*cos(y(i));

u(i,N+1)=((2\*pi)^3)+x(i)\*(2\*pi-(2\*pi)^2);

end

%5-Point Stencil Method (Gauss-Seidel)

%establishing starting values for error and # of iterations

h2=h^2;

err = 1;

iter=1;

%Error desired to be very small because of inaccuracy of L-infinity Norm

%While error is above threshold, program continues to iterate

while err>0.000000000001

umaxprev=max(max(u(2:N,2:N)));

%Saves u and iter every Q iterations

Q=1000;

if mod(iter,Q)==0;

save('backup.mat','u','iter');

end

iter=iter+1;

%x varies from 2 to N as values at 1 and N+1 are fixed

for i=2:N

for j=2:N

u(i,j)=(-F(i,j)\*h2-(u(i+1,j)+u(i-1,j)+u(i,j+1)+u(i,j-1)))/-4;

end

%Neumann B.C.

%To acheive du/dy=0, ghost node at node -1 is the same value as

%node 2

j=1;

u(i,j)=(-F(i,j)\*h2-(u(i+1,j)+u(i-1,j)+2\*u(i,j+1)))/-4;

end

%Error specified with L-infinity Norm

err = (max(max(u(2:N,2:N))-umaxprev))/umaxprev;

end

%Plotting the Results

surf(x,y,u)

xlabel('X'),ylabel('Y'),zlabel('U(X,Y)'), title('Gauss-Seidel Solution')