Project A: 2D Poisson Equation
(APc1-2)

German Robles
UH ID: 1456165

Scientific Computing for Mechanical Engineers
University of Houston
May 9th, 2018

Abstract

Two of the most easily recognizable elliptic partial differential equations that one can encounter in the field of engineering are Poisson's equation and its special case Laplace's Equation. The simplicity and general relevance of these equations make them fundamental equations in most engineering applications. The applications of Poisson's equation are vast; it can be used to model a variety of problems involving the potential of an unknown variable. For example, it can be used to model heat transfer problems where there are sources or sinks of heat, electromagnetic problems with given a potential field, and many more. The purpose of this report is to solve the two-dimensional Poisson's and Laplace's equation using linear approximations and two iterative methods, Gauss-Seidel and Successive over Relaxation (SOR). Although both iterative methods are able to solve the proposed 2D equations, the results presented in this report clearly indicate the SOR method is a better option since it converges much faster than the Gauss-Seidel method due to the relaxation factor introduced.

Table of Contents

Contents

List of Tables and Figures

Contents

Introduction

One of the most famous elliptic partial differential equations is Poisson's equation

$$\nabla^2 u = f(x) \tag{1}$$

and its special case, Laplace's equation

$$\nabla^2 u = 0 \tag{2}$$

where ∇^2 is the Laplacian operator

$$\nabla^2 = \frac{\partial^2 u}{\partial x} + \frac{\partial^2 u}{\partial y} + \frac{\partial^2 u}{\partial x} \tag{3}$$

u is the unknown function and $f(x)$ is some given parameter. In a two-space dimension, Poisson's equation and Laplace's equation then become

$$\frac{\partial^2 u}{\partial x} + \frac{\partial^2 u}{\partial y} = f(x, y), \qquad \frac{\partial^2 u}{\partial x} + \frac{\partial^2 u}{\partial y} = 0 \tag{4}$$

In order to solve these two-dimensional equations, boundary conditions must be specified over the entire boundary, i.e. over the contour of the domain of interest. Typical boundary conditions can either be of the Dirichlet type, where the function itself is assigned on the boundary, the Neumann type, where the normal derivative of the function is assigned on the boundary, or the Robin type, a combination of both. With this information given, Poisson's and Laplace's equation can then be solved for.

Discretization

As discussed in class, a computer cannot understand complex mathematical operations such a derivative; in fact, a computer is only capable to producing the results the user inputs. To overcome this complication, linear approximations of derivatives as well as iterative methods have been extremely helpful to solve complex problems involving higher order derivatives; the main idea behind linear approximations in derivatives is to take a specified number of points as

close as possible to each other to approximate the real value at a point. Inherent in these approximations are the errors caused by such approximations but which most of the time are small enough to even pose a complication to the problem. Since Poisson's equation and Laplace's equation are extremely similar to each other, only Poisson's equation will be the one examined in this paper moving forward; Laplace's equation would just follow a similar method with $f(x, y) = 0$ instead.

The discretized two-dimensional Poisson's equation is obtained using the familiar second derivative formula in the i and j notation, i.e.

$$u_{ij} = u(x_i, y_j) \tag{5}$$

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta x^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta y^2} = f_{ij} \tag{6}$$

For simplicity, let us assume $\Delta x^2 = \Delta y^2 = \Delta^2$. The main advantage of using iterative methods is that complex differential equations can be converted to a much simpler linear system of equations. This report and project focuses on two iterative methods: Gauss-Seidel and Successive over Relaxation (SOR).

Gauss-Seidel

The main idea behind the Gauss-Seidel method is to use the calculated values of a "round" in the next "round" rather than waiting until the entire "round" is finished. In other words, as an iteration is performed, the previously calculated value is used in subsequent iterations rather than using the original value. Using the calculated values as soon as they are available increases the convergence speed, and hence produces a faster result. Rearranging equation (6) yields the Gauss-Seidel method with the superscript k referring to the original iteration and $k + 1$ to the next iteration

$$u_{i,j}^{(k+1)} = \frac{1}{4}\left[u_{i-1,j}^{(k+1)} + u_{i+1,j}^{(k)} + u_{i,j-1}^{(k+1)} + u_{i,j+1}^{(k)}\right] - \frac{\Delta^2}{4} f_{ij} \quad (7)$$

Successive over Relaxation (SOR)

Successive over Relaxation is another such type of iterative method that helps solve Poisson's equation. The main difference between SOR and Gauss-Seidel is that SOR converges much faster since the parameter ω, called the relaxation factor, is introduced. A key point to remember is that for the SOR method, the error often grows with the first few iterations before convergence sets in. The SOR method is defined as

$$u_{i,j}^{(k+1)} = \frac{\omega}{4}\left[u_{i-1,j}^{(k+1)} + u_{i+1,j}^{(k)} + u_{i,j-1}^{(k+1)} + u_{i,j+1}^{(k)}\right] - \frac{\Delta^2}{4} f_{ij} + (1-\omega)u_{i,j}^{(k)} \quad (8)$$

In this case the parameter ω is usually chosen in the range of 1 to 2, i.e. $1 < \omega < 2$. A common value for ω is 1.5 but an optimal ω is key to achieve the fastest convergence rate possible.

Numerical Method Description

Gauss-Seidel Method

As stated above, the Gauss-Seidel method is one of the most universal and widely used iterative methods due to its simplicity and nearly always applicability. This iterative method displays a higher convergence rate than other iterative methods, such as the Jacobi, but a much slower convergence rate than the SOR. A major disadvantage in the Gauss-Seidel method is that since the next iteration depends on the previously solved iteration, the program cannot take advantage of parallelism, hence the long waiting time to obtain a solution. A sample pseudo code that describes this method is given

```
choose an initial guess x(0) to the solution x                    (9)
for k = 1, 2,…
      for i = 1, 2, … , n
            σ = 0
```

```
            for j = 1, 2, …, i-1
                  σ = σ + a_ij x_j^(k)
            end
            for j = i+1,…,n
                  σ = σ + a_ij x_j^(k-1)
            end
            x_i^(k) = (b_i – σ)/a_ij
      end
      check convergence; continue if necessary
end
```

SOR Method

The successive over relaxation method is the second iterative method analyzed in this project. As mentioned earlier, the SOR method converges much faster than the Gauss-Seidel since a relaxation factor ω (usually $1 < \omega < 2$) is introduced to accelerate the rate of convergence. The main complication with this method is that the error often grows with the fist iterations until convergence sets in. An interesting thing to point out is that setting $\omega = 1$ gives the Gauss-Seidel method.; this is why the relaxation factor must be greater than 1. A sample pseudo code that describes this method is given

```
set x^(0) equal to 0                                                    (10)
for k = 1,2,… do
      for I = 1,2,…,n do
            σ = 0
            for j = 1,…,i-1 do
                  σ = σ + a_ij x_j^(k)
            for j = i+1,…,n do
                  σ = σ + a_ij x_j^(k)
                  σ = (b_i - σ) / a_ii
                  x_i^(k) = x_i^(k-1) + ω(σ – x_i^(k-1))
check residual; continue if necessary
```

Technical Specifications

The computer used to complete this assignment is the HP 510-a010. This computer has an AMD A8-7410 APU with AMD Radeon R5 Graphics processor, 2200 MHz, 4 Cores, and 4

Logical Processors. It also has an installed physical memory RAM of 8 GB and available physical memory of 6.93 GB. Figure 1 shows the complete system summary of the computer used while the text file *"System Information"* found in the Project/doc folder shows the complete system information for the computer used.

OS Name	Microsoft Windows 10 Home
Version	10.0.16299 Build 16299
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	DESKTOP-9D1E3U1
System Manufacturer	HP
System Model	510-a010
System Type	x64-based PC
System SKU	V8N84AA#ABA
Processor	AMD A8-7410 APU with AMD Radeon R5 Graphics, 2200 Mhz, 4 Core(s), 4 Lo...
BIOS Version/Date	AMI F.21, 5/31/2017
SMBIOS Version	2.8
Embedded Controller Version	255.255
BIOS Mode	UEFI
BaseBoard Manufacturer	HP
BaseBoard Model	Not Available
BaseBoard Name	Base Board
Platform Role	Desktop
Secure Boot State	On
PCR7 Configuration	Elevation Required to View
Windows Directory	C:\WINDOWS
System Directory	C:\WINDOWS\system32
Boot Device	\Device\HarddiskVolume1
Locale	United States
Hardware Abstraction Layer	Version = "10.0.16299.371"
User Name	DESKTOP-9D1E3U1\grobl
Time Zone	Central Daylight Time
Installed Physical Memory (RAM)	8.00 GB
Total Physical Memory	6.93 GB
Available Physical Memory	3.62 GB
Total Virtual Memory	8.87 GB
Available Virtual Memory	3.76 GB
Page File Space	1.94 GB
Page File	C:\pagefile.sys
Device Encryption Support	Elevation Required to View
Hyper-V - VM Monitor Mode E...	Yes
Hyper-V - Second Level Addres...	Yes
Hyper-V - Virtualization Enable...	No
Hyper-V - Data Execution Prote...	Yes

Figure 1. System Summary for computer used

Two-dimensional Poisson's equation

The problem proposed for this project is in the form

$$\frac{\partial^2 u}{\partial x} + \frac{\partial^2 u}{\partial y} = -F(x,y) \tag{11}$$

with the domain of interest in the square

$$a_x < x < b_x\,, \qquad\qquad a_y < x < b_y \tag{12}$$

where

$$a_x = a_y = -\pi, \qquad\qquad b_x = b_y = \pi \tag{13}$$

and with boundary conditions

$$u\big(x, y = b_y\big) = f_a(x), \qquad\qquad u\big(x, y = a_y\big) = g_a(x) \tag{14}$$

$$\left.\frac{du}{dx}\right|_{x=a_x} = 0, \qquad u(x = b_x, y) = g_a(b_x) + \frac{y-a_y}{b_y-a_y}[\,f_a(b_x) - g_a(b_x)] \tag{15}$$

where

$$f_a(x) = (x-a_x)^2 cos\frac{\pi x}{a_x}, \qquad\qquad g_a(x) = x(x-a_x)^2 \tag{16}$$

The right side of the equation, the known parameter, for Poisson's equation is

$$F(x,y) = \cos\left[\frac{\pi}{2}\left(2\frac{x-a_x}{b_x-a_x} + 1\right)\right] \sin\left[\pi\frac{y-a_y}{b_y-a_y}\right] \tag{17}$$

while for Laplace's equation is simply

$$F(x,y) = 0 \tag{18}$$

Figure 2 shows a graphical representation for this problem.

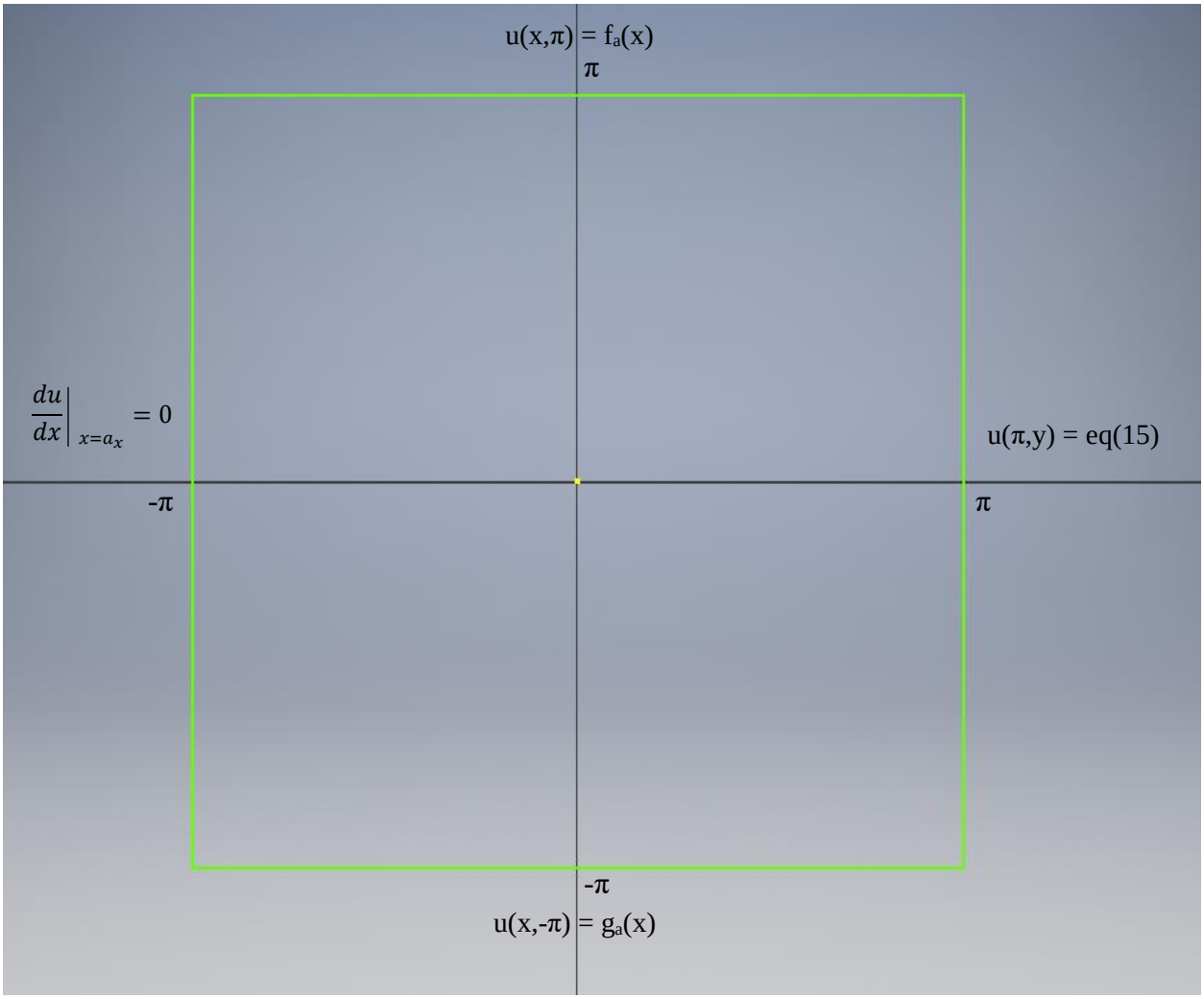

Figure 2. Graphical Representation of 2D Problem

This report will not focus on how the code to solve this problem was written but rather discuss some of the results obtained through the simulation; the code written for this report can be found in the *MATLAB Code* section of this report. One important thing to note is that a ghost node was implemented on the left Neumann boundary condition since we want the derivative to be zero; for a complete step by step walkthrough on how to implement the ghost node, the class notes "*Numerical Solution of Elliptic Equations I*" do an excellent job at it. Like stated previously, it is not the objective of this report to show how the code was written.

Results

This section will discuss the written code and the simulated results by first analyzing the solutions obtained for Poisson's equation using both iterative methods. It will then analyze the results for Laplace's equation. Finally, it will discuss some of the findings and observations made from this project. As can be seen from Figures 3 and 4, which are the graphical solutions to Poisson's Equation, both plots are nearly identical to the human eye. Using the provided information, the code was able to execute to come up with the matrix solution to the problem. As it was expected, both iterative methods are able to come up with the same solution even though the SOR method does it in much fewer iterations. Table 1 shows a comparison between both methods as the number of nodes is changed while Table 2 shows an optimization for the relaxation factor ω in the SOR method. As can be seen from Table 1, the SOR method was able to solve the problem nearly 20 times faster than the Gauss-Seidel method, considering an $\omega = 1.5$. Figure 5 shows a graphical representation of the results in Table 1, which clearly show the SOR method was able to compute the solution in much fewer iterations. An optimization was then performed using a standard 200 nodes and the results are given in Table 2. Initially ω was assumed to be 1.5 but once the optimization was performed, it can be clearly seen from Figure 6 that it is the worst possible value to choose. As the graph shows, an ω near the edges of the limitations ($1 < \omega < 2$) can solve the problem in much fewer iterations. The ideal ω found in this optimization was 1.9. Increasing the relaxation factor from 1.5 to 1.9 decreases the number of iterations needed to solve the problem by half, which would be nearly 40 times faster than the Gauss-Seidel method!

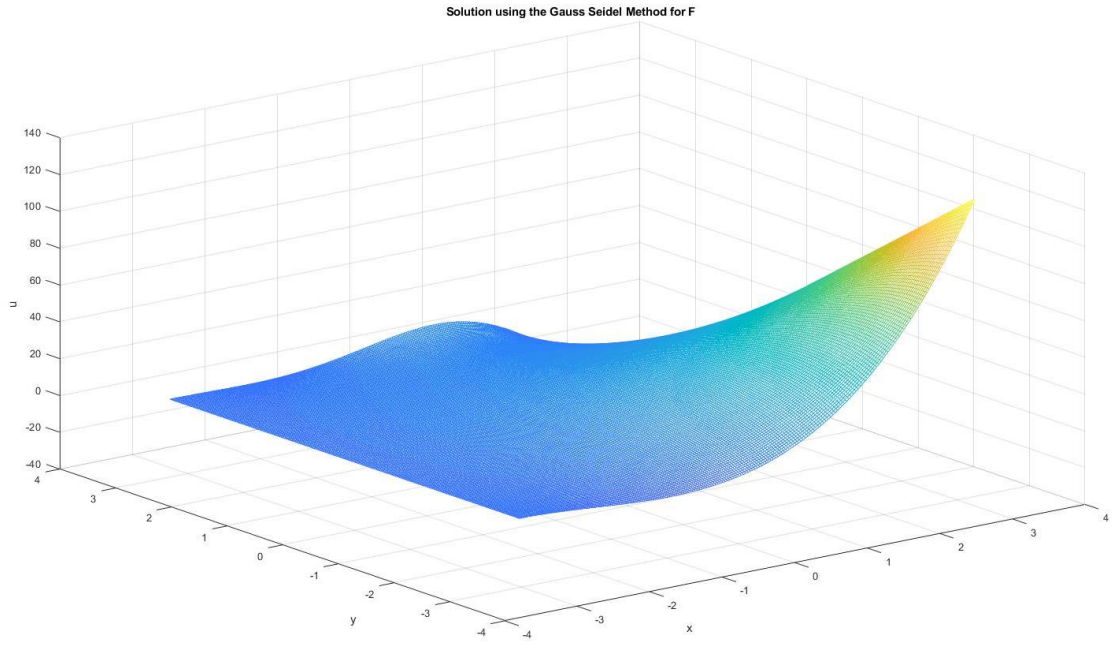

Figure 3. Solution using Gauss-Seidel Method for Poisson's Equation

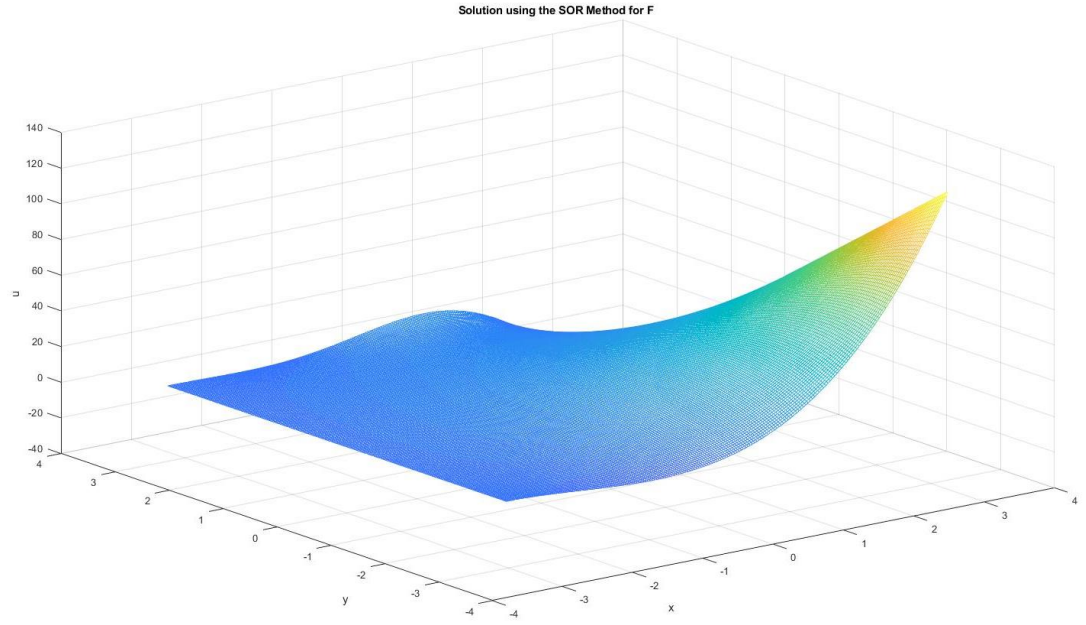

Figure 4. Solution using SOR Method for Poisson's Equation

Table 1. Comparison between both methods

Iterations vs Nodes for both methods		
Nodes	Iterations to Solve GS)	Iterations to Solve (SOR) ($\omega = 1.5$)
20	456	14
50	2572	90
100	9093	365
150	18750	825
200	31115	1470
250	45890	2302
300	62848	3318

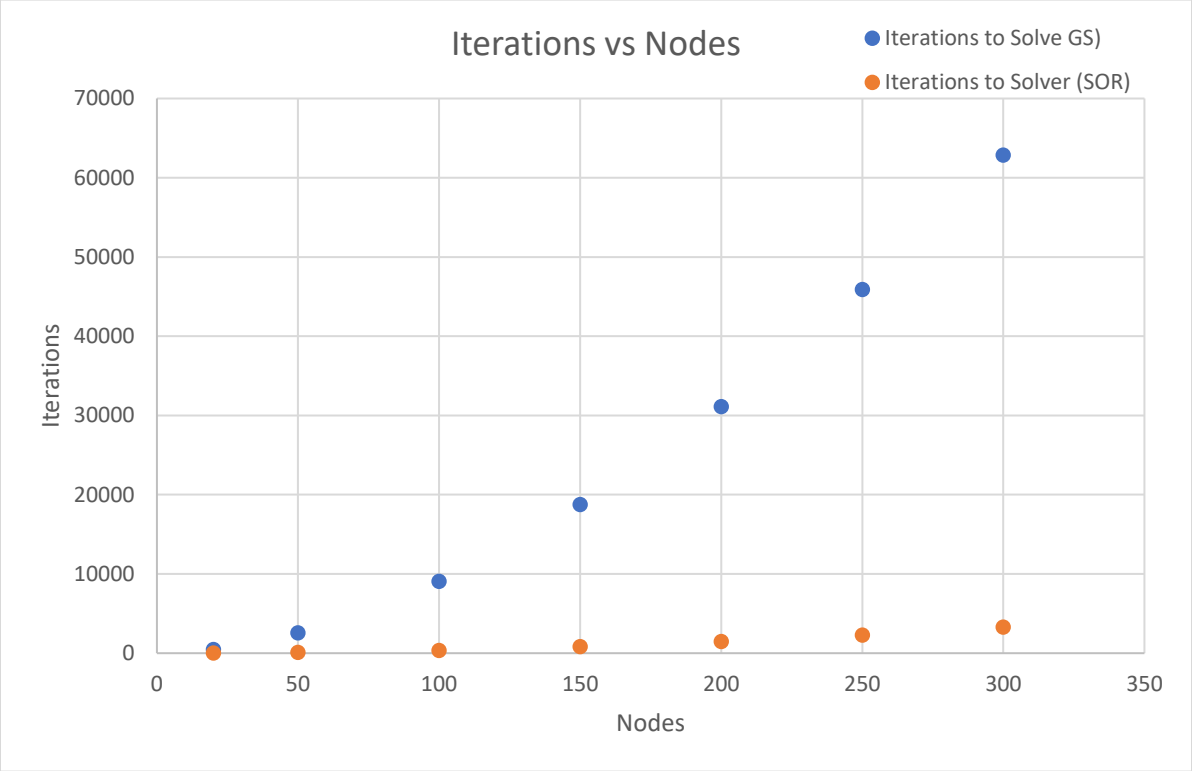

Figure 5. Iterations vs Nodes for both methods

Table 2. Relaxation Factor Optimization

SOR Omega Optimization using 200 Nodes	
Omega (ω)	Iterations to Solve
1.1	659
1.2	1085
1.3	1338
1.4	1458
1.5	1470
1.6	1392
1.7	1230
1.8	981
1.9	621

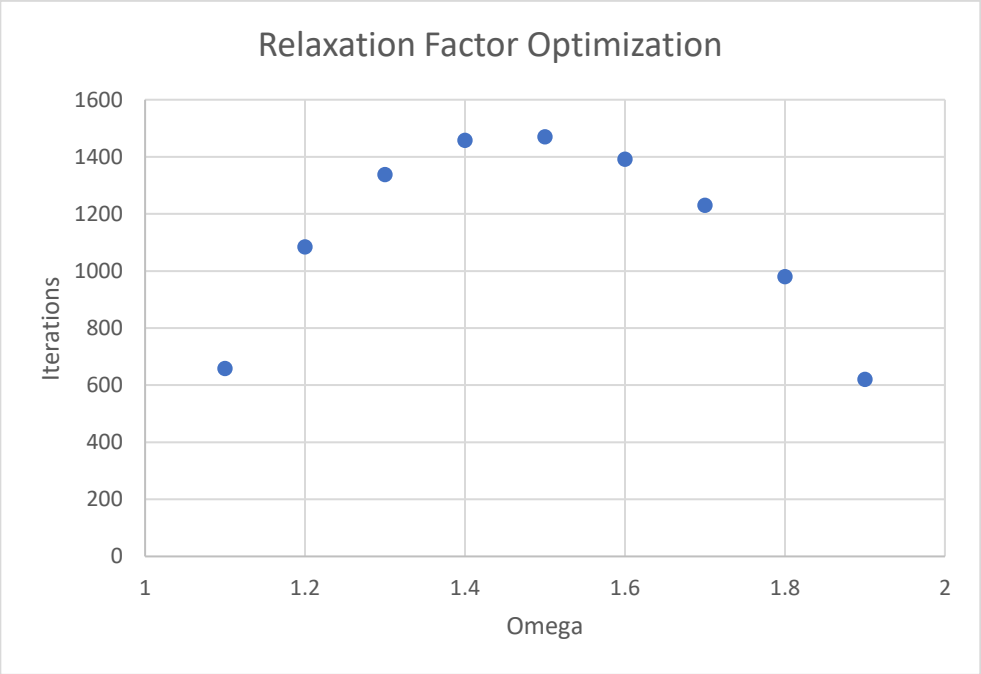

Figure 6. Optimization Graph

The second analysis in this report deals with the solutions to Laplace's equation, i.e., $F(x, y) = 0$. As can be seen from Figures 7 and 8, both are really similar to each other and to the plots of Poisson's equation. The biggest factor is that the right-hand side, the known parameter, does not make a major impact on the final solution for this problem. The $F(x, y)$ values for

Poisson's equation are small enough so that they don't differ much from the Laplace's equation $F(x, y)$ values. Had the $F(x, y)$ values for Poisson's equation been greater, there would have been a distinct difference between both equations and their respective solutions. Some other values would have been desirable so that a much bigger difference could be seen from all the plots.

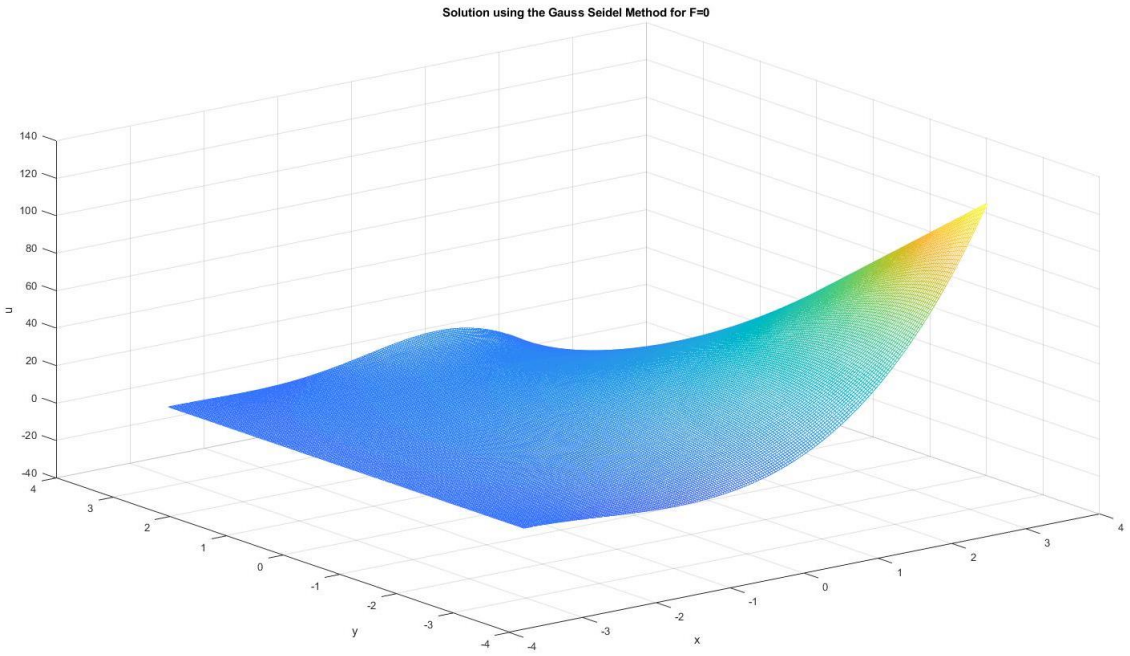

Figure 7. Solution using Gauss-Seidel Method for Laplace's Equation

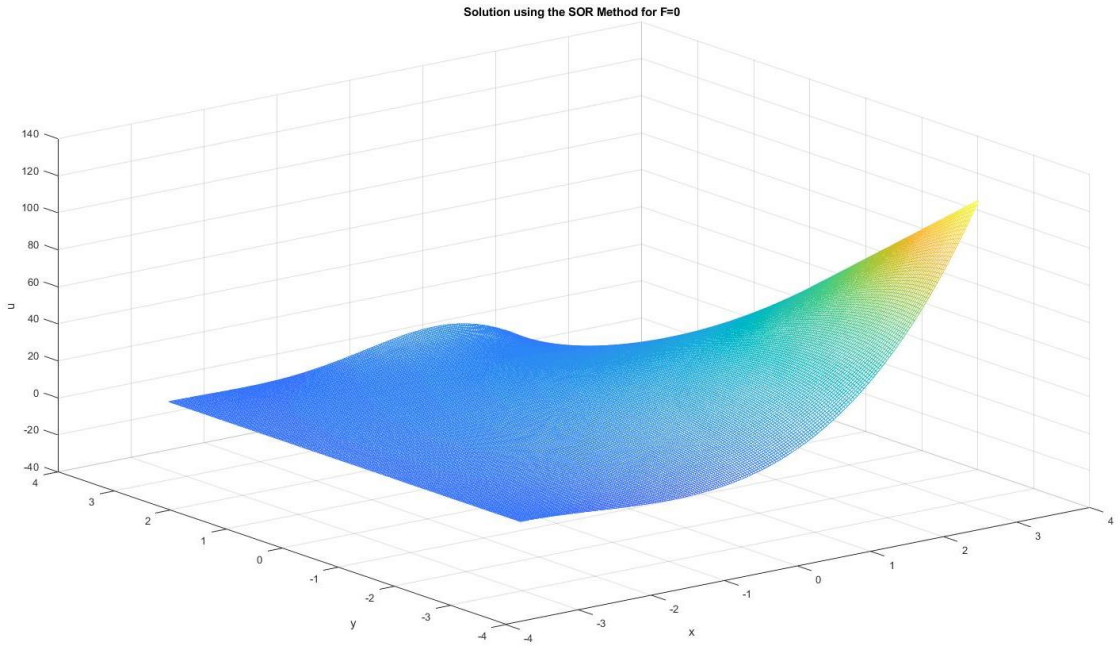

Figure 8. Solution using SOR Method for Laplace's Equation

The final observation to be made from the results of this project is the effect of adding more nodes to the problem. Adding more nodes has two major effects on the problem. The first effect, and probable the reason why more nodes are added in the first place, is the ability to obtain more accurate results. As more nodes in both directions are added, the domain of interest is separated into smaller and smaller intervals so the intersecting nodes in both directions are closer and closer to each other. This, as a result, produces more accurate results. The second effect of adding more nodes is the number of iterations needed, and as a result the time it takes, to compute the solution. Creating a finer and finer mesh is obviously desirable so that errors can be minimized, but at the same time the computation time has to be reasonable. There has to be a compromise between accuracy and efficiency in order to produce the best suitable result.

Conclusion

This project was essentially a whole recap of the material covered in this class. One must understand why linear approximations are important, the different types of iterative methods available, as well as how to apply the already familiar elliptic partial derivative equations such as Poisson's and Laplace's. Through this study, we were able to obtain solutions that otherwise would be extremely hard to solve for. As demonstrated in this report, the SOR method has a much faster convergence speed compared to the Gauss-Seidel method; just because of this fact, the SOR becomes much more desirable and the iterative method to go whenever a problem such as the one stated in this report is given.

References

Chapra, Steven C., and Raymond P. Canale. *Numerical Methods for Mechanical Engineers*. McGraw-Hill Higher Education, 2015.

http://mathworld.wolfram.com/SuccessiveOverrelaxationMethod.html

http://www.netlib.org/linalg/old_html_templates/subsection2.6.2.2.html

https://openi.nlm.nih.gov/detailedresult.php?img=PMC2234413_1743-0003-4-46-16&req=4

MATLAB Code

```
%German Robles
%1456165
%2D Poisson Equation
%APc1-2 version
%May 9th, 2018
%Scientific Computing

clear all
clc
close all

%% variables that will be used in this problem to solve for 2D Poisson and
Laplace equation

%rectangle of interest
ax = -pi;               %left boundary of rectangle of interest
ay = -pi;               %bottom boundary of rectangle of interest
bx = pi;                %right boundary of rectangle of interest
by = pi;                %top boundary of rectangle of interest
Lx = bx - ax;           %Lx=2pi since that's magnitude of the rectangle in x-
                        direction
Ly = by - ay;           %Ly=2pi since that's magnitude of the rectangle in y-
                        direction

%number of nodes analyzed in this problem
disp('Enter the number of nodes to be analyzed:')       %user friendly input
                                                depending on the number of nodes
N = input('Nodes: ');   %choose a desired number of poles

Nx = N;                 %Ny is set equal to the desired number of nodes
Ny = N;                 %Ny is set equal to the desired number of nodes

h2 = (1/(Nx+1))^2;      %since h is the same for delta x and delta y, h^2 is
                        used for the following calculation

hx = linspace(ax,bx,Nx);        %equally spaced vector from -pi to pi using N
                                 nodes in x direction
hy = linspace(ay,by,Ny);        %equally spaced vector from -pi to pi using N
                                 nodes in y direction

[x,y] = meshgrid(hx,hy);        %created matrix from -pi to pi in both
                                 direction to simulate rectangle
y = flipud(y);                  %flipped y vector to have it go from -pi to
                                 pi in vertical direction

%boundary conditions defined
fa = ((x-ax).^2).*cos(pi.*x/ax);          %equation used for the top
                                          boundary condition (Dirichlet)
ga = x.*((x-ax).^2);                      %equation used for the bottom
                                          boundary condition (Dirichlet)
```

```
F = cos((pi/2).*(2.*((x-ax)/(bx-ax))+1)).*sin(pi.*((y-ay)/(by-ay)));
                          %right hand side for Poisson equation (first case)
%F = zeros(Nx,Ny);        %right hand side for Laplace equation
                           (second case)
uby = fa;                        %top BC
uay = ga;                        %bottom BC
ubx = (bx.*((bx-ax).^2))+(((y-ay)/(by-ay)).*((((bx-ax).^2).*cos(pi.*bx/ax))-
(bx.*((bx-ax).^2))));            %Right BC

%% boundary conditions prescribed on u matrix

%solving for outer edges of the u matrix solution
u = zeros(Nx,Ny);             %preallocating the matrix for u matrix solution
u(1,2:Ny-1)=uby(1,2:Ny-1);      %plugging in top BC's defined earlier on the
                                u matrix
u(Nx,2:Ny-1)=uay(Nx,2:Ny-1);    %plugging in bottom BC's defined earlier on
                                the u matrix
u(2:Nx-1,Ny)=ubx(2:Nx-1,Ny);    %plugging in right BC's defined earlier on
                                the u matrix

% Neumann boundary condition on the left side of the u matrix

for i = 2:Nx-1
    u(i,1) = (1/4)*(2*u(i,1)+u(i-1,1)+u(i+1,1)+(h2)*F(i,1));    %algorithm
                      used to calculate left BC since a ghost node is required

end

% to provide better results, the corners will be calculated as averages of
% the neighboring grids

u(1,1) = (u(1,2)+u(2,1))/2;              %average for top left corner
u(1,Ny) = (u(1,Ny-1)+u(2,Ny))/2;         %average for top right corner
u(Nx,1) = (u(Nx-1,1)+u(Nx,2))/2;         %average for bottom left corner
u(Nx,Ny) = (u(Nx,Ny-1)+u(Nx-1,Ny))/2;    %average for bottom right corner

%% first iterative method used to solve Poisson equation: gauss-seidel

error = 1;                %define the error
tole = 1e-6;              %tolerance used when iterating using the
                          gauss-seidel method
gaussit = 0;              %keeps track of the number of iterations to date
ukp1 = u;                 %defines the u (k+1) matrix to the original u matrix

while error > tole
    gaussit = gaussit + 1;      %iterator count keeps growing for each itera.
    for j = 2:Ny-1
        for i = 2:Nx-1
            ukp1(i,j)=.25*(ukp1(i-1,j)+u(i+1,j)+ukp1(i,j-
1)+u(i,j+1)+h2*F(i,j));          %algorithm for the GS method
        end
    end
    error =(1/(Nx*Ny))*sum(sum(abs(ukp1-u)));    %calculated error
                                                 used for the next iteration
    u = ukp1;
```

```
end

%iterations needed to solve
disp('Number of Iterations using the Gauss Seidel method for F =')
disp(gaussit)                    %shows the total number of iteration to
                                  converge

%plots solution
figure(1)                        %plots the u solution on a x, y, and z plane
mesh(x,y,u)
xlabel('x')
ylabel('y')
zlabel('u')
title('Solution using the Gauss Seidel Method for F ')

%% second iterative method used to solve Poisson equation: SOR method

error = 1;
tole = 1e-6;
gaussit = 0;
ukp1 = u;
w = 1.5;                         %relaxation factor used to solve the SOR
method. ideally this value must be within 1 to 2. for this first calculation
I used 1.5 but this value needs to be optimized to produce the best results
possible in the form of the fastest convergence

while error > tole
    gaussit = gaussit + 1;
    for j = 2:Ny-1
        for i = 2:Nx-1
            ukp1(i,j)=((w/4)*(u(i+1,j)+ukp1(i-1,j)+ u(i,j+1)+ ukp1(i,j-
1)+(h2*F(i,j))))+(1-w)*u(i,j);            %algorithm for the SOR method
        end
    end
    error =(1/(Nx*Ny))*sum(sum(abs(ukp1-u)));
    u = ukp1;
end

%iterations needed to solve
disp('Number of Iterations using the SOR method for F =')
disp(gaussit)                  %shows the total number of iteration to converge

%plots solution
figure(2)                      %plots a second figure to display the SOR
                                solution on a x, y, and z plane
mesh(x,y,u)
xlabel('x')
ylabel('y')
zlabel('u')
title('Solution using the SOR Method for F ')
```