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). The results presented in this report clearly indicate the SOR method converges much faster than the Gauss-Seidel method because of the relaxation factor.

# **Table of Contents**

Contents

[**Abstract** i](#_Toc513551287)

[**Table of Contents** ii](#_Toc513551288)

[**List of Tables and Figures** iii](#_Toc513551289)

[**Introduction** 1](#_Toc513551290)

[**Discretization** 1](#_Toc513551291)

[**Gauss-Seidel** 2](#_Toc513551292)

[**Successive over Relaxation (SOR)** 3](#_Toc513551293)

[**Numerical Method Description** 3](#_Toc513551294)

[**Gauss-Seidel Method** 3](#_Toc513551295)

[**SOR Method** 4](#_Toc513551296)

[**Technical Specifications** 4](#_Toc513551297)

[Figure 1. System Summary for computer used 6](#_Toc513551298)

[**Two-dimensional Poisson’s equation** 6](#_Toc513551299)

[Figure 2. Graphical Representation of 2D Problem 7](#_Toc513551300)

[**References** 8](#_Toc513551301)

[**MATLAB Code** 9](#_Toc513551302)

# **List of Tables and Figures**

Contents

[**Abstract** i](#_Toc513551303)

[**Table of Contents** ii](#_Toc513551304)

[**List of Tables and Figures** iii](#_Toc513551305)

[**Introduction** 1](#_Toc513551306)

[**Discretization** 1](#_Toc513551307)

[**Gauss-Seidel** 2](#_Toc513551308)

[**Successive over Relaxation (SOR)** 3](#_Toc513551309)

[**Numerical Method Description** 3](#_Toc513551310)

[**Gauss-Seidel Method** 3](#_Toc513551311)

[**SOR Method** 4](#_Toc513551312)

[**Technical Specifications** 4](#_Toc513551313)

[Figure 1. System Summary for computer used 6](#_Toc513551314)

[**Two-dimensional Poisson’s equation** 6](#_Toc513551315)

[Figure 2. Graphical Representation of 2D Problem 7](#_Toc513551316)

[**References** 8](#_Toc513551317)

[**MATLAB Code** 9](#_Toc513551318)

# **Introduction**

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

(1)

and its special case, Laplace’s equation

(2)

where is the Laplacian operator

(3)

is the unknown function and is some given parameter. In a two-space dimension, Poisson’s equation and Laplace’s equation then become

, (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 in 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; Laplace’s equation would just follow a similar method with .

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

(5)

(6)

For simplicity, let us assume = .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 referring to the original iteration and the next iteration

(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 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

(8)

In this case the parameter ω is usually chosen in the range of 1 to 2, i.e. 1 < ω < 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 + aijxj(k)*

*end*

*for j = i+1,…,n*

*σ = σ + aijxj(k-1)*

*end*

*xi(k) = (bi – σ)/aij*

*end*

*check convergence; continue if necessary*

*end*

## **SOR Method**

The successive over relaxation method 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 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. 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 + aijxj(k)*

*for j = i+1,…,n do*

*σ = σ + aijxj(k)*

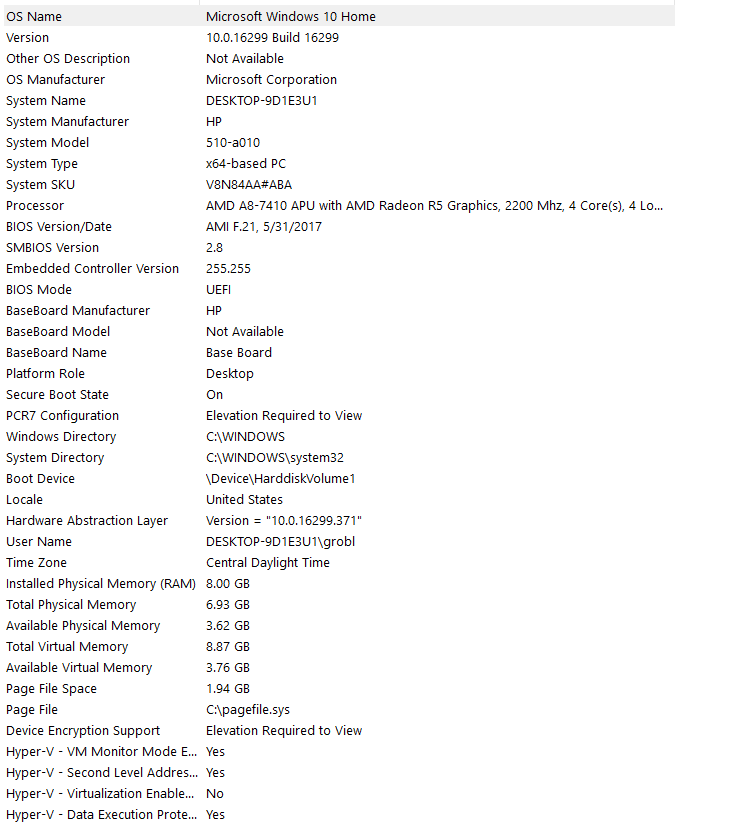
*σ = (bi­ – σ) / aii*

*xi(k) = xi(k-1) + ω(σ - xi(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.



# Figure 1. System Summary for computer used

# **Two-dimensional Poisson’s equation**

The problem proposed for this project is in the form

(11)

with the domain of interest in the rectangle

, (12)

where

, (13)

and with boundary conditions

(14)

, (15)

where

(16)

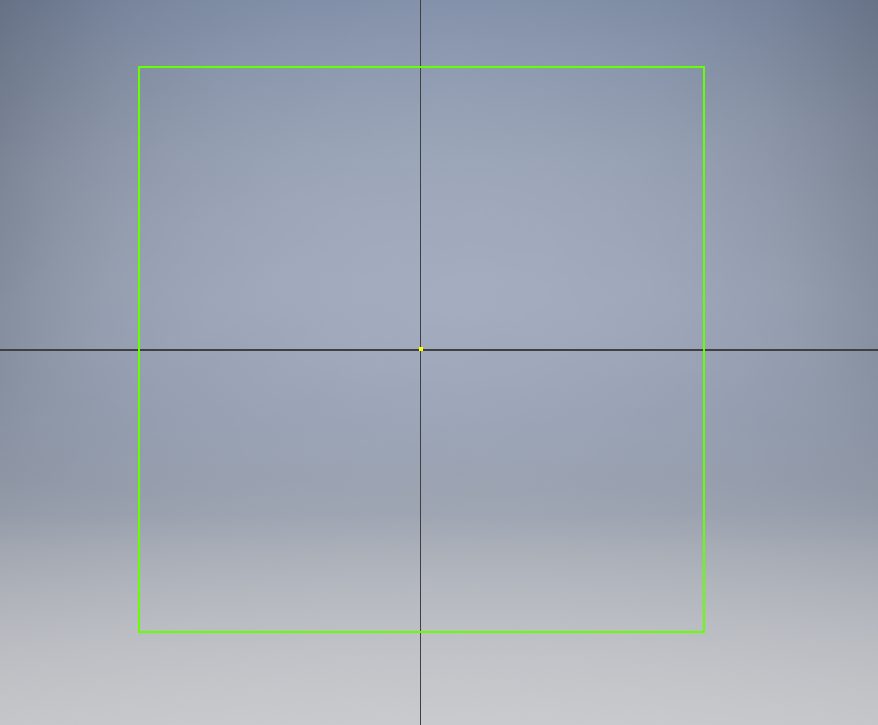
The right side of the equation, the known parameter, for Poisson’s equation is

(17)

while for Laplace’s equation is simply

(18)

Figure 2 shows a graphical representation for this problem.



u(x,π) = fa(x)

π

u(π,y) = eq(15)

u(x,-π) = ga(x)

π

-π

-π

# 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; 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, this is not the purpose of this report.

# **Results**

# **References**

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

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

# **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 ')