MECE 5397 Scientific Computing for Mechanical Engineers

Project A – Poisson Equation

Version: AP02-5

Abraham Blanco

1223970

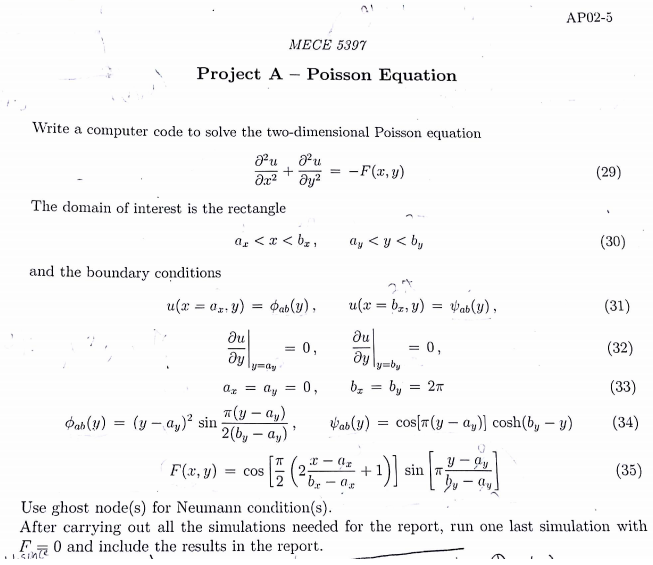
Class section: Mon, Wed 1- 2:30 pm

May 5th, 2017

# Abstract

This report discusses the creation of a Matlab code that is able to solve a given F(x) equation using a two dimensional Poisson equation. The Poisson equation is used for numerous applications such as gravity and heat flow. Using methods in order to solve complex system of equations, two methods known as the Gauss-Seidel and the Successive over Relaxation (SOR) methods are used to solve a given Poisson Equation and compared with one another. For this project, the assigned question included two Dirichlet boundary conditions along the X-axis and two Neumann conditions along the y-axis. The function U(x,y) was bounded over a range from 0<x<2, and 0<y<2. This report includes the discretization along with visuals provided. The Matlab code written for Gauss- seidel method uses source control GitHub and is available for public access. . Data provided within this report proves that the SOR method converges faster and requires less iterations. The main purpose of this report is to learn to write a code to solve the Poisson equation using two methods of solution, employ checkpoints and use data visualization to compare the results while providing a clean, optimized code.

# Mathematical Statement of Problem



# Discretized version of the Equations

Below in **Figure 2**, is the discretization using the Centered difference formula of the given problem 2D Poisson Equation using the two given Neumann boundary conditions.

|  |
| --- |
|  |
| **Figure 2**: Discretization of the 2D Poisson Equation |

From the discretization one is able to solve for three equation: General equation (eq. 1), and both Neumann boundary equations (eq. 2 and 3) (**Figure 3**).

|  |
| --- |
|  |
| **Figure 3**: Discretized Neumann equation and general equation |

# Description of the numerical method (pseudo code)

**Numerical methods** are designed for the constructive solution of mathematical problems requiring particular numerical results, usually on a computer. A numerical method is a complete and unambiguous set of procedures for the solution of a problem, together with computable error estimates. The study and implementation of such methods is the province of numerical analysis.

* Set Variables
* Set stopwatch in matlab using “tic”
* Set to include “ghost” nodes
* Set x and y values to be equally spaced
* Set a matrix (U) to provide Matlab with an initial guess
* Evaluate for an F matrix with given equation
* Define both Dirchelet Boundary conditions and use them to set U matrix boundary
* Redefine coefficients for constants
* Use a while loop driven with an error estimation that will reevaluate solutions for U until it converges onto a solution that satisfies a set error parameter
* Set an iteration counter to take note and take the time taken to complete convergence with each method and display visuals for comparison and grid convergence study
* Provide checkpointing within the code and optimize it.

# Computer Specifications

For this project I used the computer within the University of Houston Library Commons, to run calculations using Matlab. The specifications are as followed:

|  |
| --- |
|  |

# Results

## Specs of parameters used in simulations

Parameters for the calculation included the use of a changing delta X and Y. Three inputs for the F(x,y) equation were used for calculations and verification.

3 simulation cases

1. Method of Manufactured solutions

## Evaluate the effect of number of points used for discretization

**Figures 4-16** show my findings calculating the values for the U matrix when using different M and N values. One can note that all figures appear to be similar but by observing **Table 1** it is easy to note that SOR proved to be the faster method as it significantly took less time and iterations to compute the same results as the Gauss Seidel Method.

|  |
| --- |
|  |
| **Figure 4**: M=N=20 nodes for simulation case 1 (Gauss\_Seidel) |

|  |
| --- |
|  |
| **Figure 5**: M=50, N=50 nodes for simulation case 1 (Gauss\_Seidel) |

|  |
| --- |
|  |
| **Figure 6**: M=N=100 nodes for simulation case 1 (Gauss\_Seidel) |

|  |
| --- |
|  |
| ***Figure 7****: 20x20 Successive Over Relaxation Method Case 1* |

|  |
| --- |
|  |
| **Figure 8:** 100x100 Successive Over Relaxation Method Case 1 |

|  |
| --- |
|  |
| **Figure 9:** 20x20 Gauss\_Seidel Method Case 2 |

|  |
| --- |
|  |
| **Figure 10:** 100x100 Gauss\_Seidel Method Case 2 |

|  |
| --- |
|  |
| **Figure 11:** 20x20 Successive Over Relaxation Method Case 2 |

|  |
| --- |
|  |
| **Figure 12:** 100x100 Successive Over Relaxation Method Case 2 |

|  |
| --- |
|  |
| **Figure 13**: 20x20 Gauss\_Seidel Method Case 3 |

|  |
| --- |
|  |
| **Figure 14:** 100x100 Gauss\_Seidel Method Case 3 |

|  |
| --- |
|  |
| **Figure 15:** 20x20 Successive Over Relaxation Method Case 3 |

|  |
| --- |
|  |
| **Figure 16:** 100x100 Successive Over Relaxation Method Case 3 |

**Table 1** below compares both the Gauss-Seidel and the Successive over Relaxation Methods for each of the Cases for which the function F is changed based on the amount of internal nodes (M and N) selected for calculation. It is apparent that the amount of time and the iterations for Matlab to compute is much less when using the SOR Method. One can input any combination of M and N node values into the code provided.

**Table 1:** Computed Results using Gauss Seidel and SOR for each Case

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| F=cos(X)Sin(Y) | | Gauss Seidel | | SOR (λ =1.5) | |
| M(# of x nodes) | N (# of y nodes | Time (T) | Iterations | Time (T) | Iterations |
| 20 | 20 | 0.16891 | 1806 | 0.070803 | 732 |
| 50 | 50 | 3.9344 | 10311 | 1.46273 | 3748 |
| 100 | 100 | 68.127103 | 47248 | 25.07156 | 16808 |
| 40 | 60 | 3.56464 | 10918 | 1.376737 | 3943 |
| F=0 | | Gauss Seidel | | SOR (λ =1.5) | |
| M(# of x nodes) | N (# of y nodes | Time (T) | Iterations | Time (T) | Iterations |
| 20 | 20 | 0.174107 | 1946 | 0.070739 | 732 |
| 50 | 50 | 4.124128 | 11398 | 1.587339 | 4122 |
| 100 | 100 | 58.925272 | 41077 | 22.46702 | 14710 |
| 40 | 60 | 3.636822 | 11045 | 1.4315 | 3983 |
| F=1+X^2+2Y^2 (Manufactured Eq) | | Gauss Seidel | | SOR (λ =1.5) | |
| M(# of x nodes) | N (# of y nodes | Time (T) | Iterations | Time (T) | Iterations |
| 20 | 20 | 0.161453 | 1762 | 0.080161 | 686 |
| 50 | 50 | 3.456946 | 9387 | 1.460665 | 3489 |
| 100 | 100 | 54.642703 | 37756 | 22.86809 | 13704 |
| 40 | 60 | 3.086373 | 9396 | 1.350026 | 3483 |

# Perform grid convergence study

**Figure 17** and **Table 2** below are the calculated results when observing grid independence. One can see as the number of inputted nodes increases the average value of U^2 converges to a uniform value.

|  |
| --- |
|  |
| **Figure 17**: Grid Independence Study Plot for Gauss Seidel and SOR Methods |

**Table 2:** Computed Results for Grid Independence Study

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Gauss Seidel | | | SOR (λ=1.5) | | |
| N Value | Average of U^2 | Time | iterations | Average of U^2 | Time | Iterations |
| 10 | 744.2196 | 0.051045 | 453 | 744.2196 | 0.046243 | 182 |
| 20 | 394.8126 | 0.168869 | 1806 | 394.8126 | 0.083672 | 684 |
| 50 | 259.648 | 3.677012 | 10311 | 259.648 | 1.379319 | 3748 |
| 100 | 227.0175 | 67.989636 | 47248 | 227.0175 | 24.439437 | 16808 |
| 200 | 213.0513 | 1009.22673 | 164458 | 213.0513 | 357.290871 | 58303 |
| 250 | 210.441 | 2539.783064 | 257514 | 210.441 | 933.872342 | 91128 |

To conclude my findings, expected to out perform as increasing values of M and N were inputed, SOR Method proved to be the faster of the two methods for solving this problem. While the Gauss-Seidel method took thousands of iterations to solve the equation, the SOR method was able to reduce the time and iterations need to compute the same solution. Grid independence testing was used to varify the solutions computed from both methods. I would like to thank the instructors for allowing me to witness the significance of optimization of computing.

Below is the provided link to my Github username: abeblanco

<https://github.com/abeblanco/Project-Main>

Matlab Code for Gauss Seidel Method:

%Code Method Gauss\_Seidel METHOD

% Abraham Blanco 1223970

clear all; clc;

%% Parameters

ax = 0;

ay = 0;

bx = 2\*pi;

by = 2\*pi;

% Define the number of points on the interior (this does not include the exterior boundary points)

%lamda=input('Value of lamda=');

M=input('Value of X Intenal Nodes=');

N=input('Value of Y Internal Nodes=');

Time=tic' %time begins here after inputs are set

M1=M+2;

N1=N+2;

% this generates the x and y values that will be used to calculate the F matrix

xvalues = linspace(0,2\*pi,M+2);

yvalues = linspace(0,2\*pi,N+2);

%%

%U matrix ( initial guess)

U = ones(M+2,N+2); %preallocation (optmization)

%solving for right hand side (F equation)

for i=1:length(xvalues);

for j=1:length(yvalues);

F(i,j) = cos ( (0.5\*pi)\* (2\*((xvalues(i)-ax) / (bx - ax))+1 )).\*sin( pi\*((yvalues(j)-ay) / (by -ay)));

%F(i,j)= 0; %last simulation with F=0

end

end

%% Boundary Conditions for "Top" and "Bottom" side of Matrix

% Top boundary values (Dirchelet Condition)

U(1,:) = ((yvalues - ay).^2 ) .\* sin( pi \*(yvalues - ay) / (2\*(by-ay)) ) ;

% Bottom boundary values (Dirchelet Condition)

U(end,:) = cos (pi\*(yvalues-ay)).\*cosh(by-yvalues);

% place these known values in the solution grid

%solveing for Left and Right Boundaries

W(1,:)=U(1,:);

W(end,:)=U(end,:);

%%

DX = 2\*pi/(M+1);

A = 1/DX.^2;

DY = 2\*pi/(N+1);

B = 1/DY.^2;

R = -2\*(A+B);

% normalize elements

A = A/R;

B = B/R;

F = F/R;

R = 1;

error=10;

error\_iterations=0;

% check for diagonal dominance of elements

abs(R) >= abs(2\*A+2\*B);

Time\_count=0;

save('variables.mat')

%%

load('variables.mat')

while error>10^-10;

T\_loop=tic;

if Time\_count>=.5

Time\_count=0;

save('variables.mat')

end

W=U;

for j = 2:M+1;

% Left boundary

W(j,1) = U(j,1);

U(j,1) = ( F(j,1) - (2\*B)\*U(j,2) - A\*U(j-1,1) - A\*U(j+1,1) );

error(j,1) = abs((U(j,1) - W(j,1)) / U(j,1));

% Right Boundary

W(j,end)= U(j,end);

U(j,end) = ( F(j,end) - (2\*B)\*U(j,end-1) - A\*U(j-1,end) - A\*U(j+1,end) );

error(j,M+2) = abs((U(j,M+2) - W(j,M+2)) / U(j,M+2));

end

%% Main Sweep of Gauss-Siedel

for j= 2:M+1;

for k = 2:N+1;

W(j,k)=U(j,k);

U(j,k) = ( F(j,k) - B\*U(j,k-1) - B\*U(j,k+1)- A\*U(j-1,k) - A\*U(j+1,k) );

%U(j,k) = lamda\*U(j,k)+(1-lamda)\*W(j,k); %for SOR portion of Gauss\_siedel

error(j,k)= abs((U(j,k) - W(j,k)) / U(j,k));

end

end

error=abs(max(max(((W-U)./W))));

error\_iterations=error\_iterations+1;

P= toc(T\_loop);

Time\_count=Time\_count + P;

end

toc(Time)

save('variables.mat')

%%

load('variables.mat')

Grid=mean(mean(U.^2))

error\_iterations

figure

subplot(1,2,1),surf(U), xlabel('Y axis'), ylabel('X axis'), zlabel('Z axis'), title('F=cosx\*siny')

subplot(1,2,2),contour(U), xlabel('Y axis'), ylabel('X axis'), title('F=cosx\*siny')