Scientific Computing Project Report

**Iterative Numerical Solutions for the Two-Dimensional Poisson Equation:**

**Gauss – Seidel and Successive Over Relaxation Methods**

**(Project: AP02-2)**

Daniel Bosquez

PSID: 1332758

May 5, 2019

MECE5397: Scientific Computing for Mechanical Engineers

Spring 2019

Dr. A. Prosperetti

Dr. A. Amritkar

**Abstract**

**Mathematical Problem Statement**

A MATLAB code was written to generate a numerical solution for the following problem:

Where the domain is a rectangle with bounds

,

,

And the following boundary conditions

, ,

,

**Discretization of 2nd Order 2D Poisson Eqn.**

Gauss – Seidel:

Approximate 2nd order terms using the Taylor Series 2nd Order Centered Difference Formula, where,

And substitute

Assuming , expression simplifies to

Which yields to a 4-point discretized solution with an error of O

*“Gauss-Seidel Discretized Solution”*

Successive Over-Relaxation:

The method of *Successive Over-Relaxation,* or *SOR,* can be viewed a modification of the *Gauss-Seidel* method, where *i = iteration*

*“Successive Over-Relaxation Discretized Solution”*

Where is a *relaxation factor* and ***uGS***represents the discretized *Gauss-Seidel* solution. It is interesting to note that an *SOR* solution with simplifies to the regular *Gauss-Seidel* method.

**Description of Numerical Method**

Algorithm:

It is convenient to note the algorithm for *Gauss-Seidel* and *SOR* are very similar, with just a couple small accommodations. The algorithm for modeling both the *Gauss-Seidel* and *SOR* iterative numerical methods is modeled below in Figure 1. The blue lines represent the process paths used in both methods, while the red lines and blocks are steps which apply to *SOR* only.

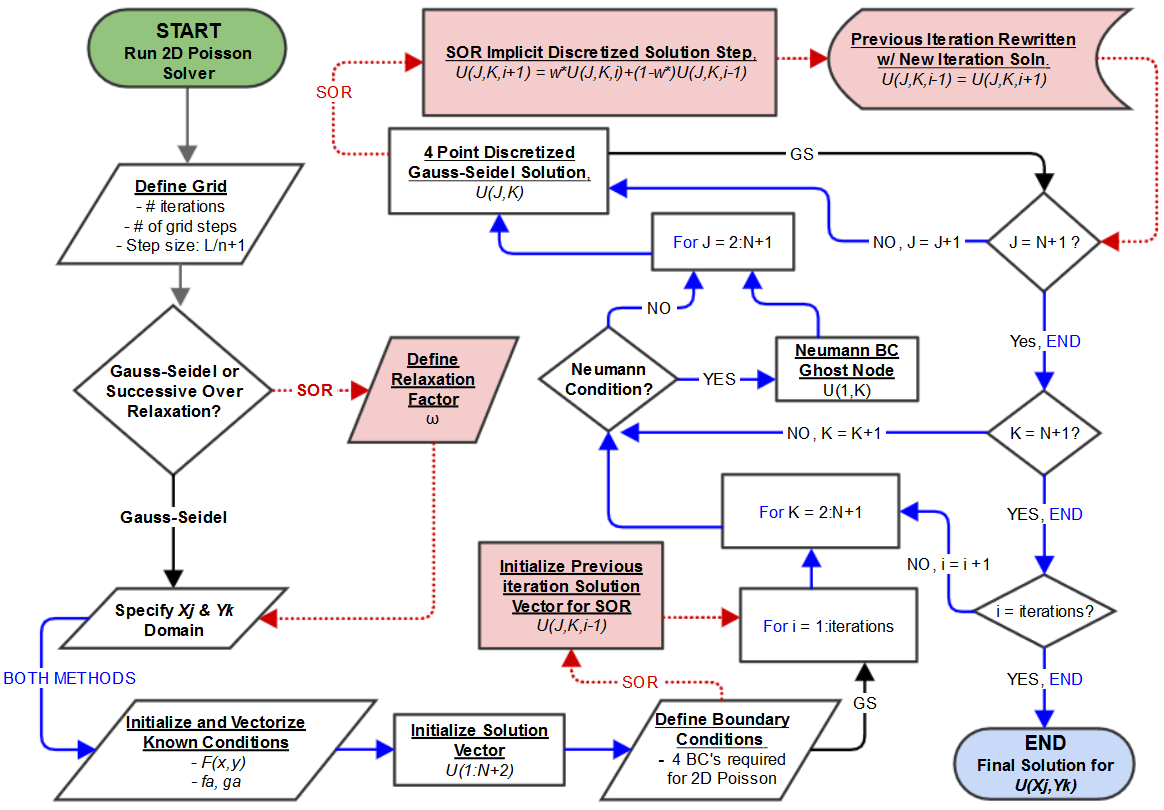


Figure 1: *2D Poisson Iterative Solver Algorithm for Gauss-Seidel & Successive Over-Relaxation*

The *Gauss-Seidel* method is the core structure for both iterative methods. First the grid must be defined by the number and size of the spatial steps in both dimensions as well as specifying the number of iterations necessary for acceptable accuracy. The relaxation factor, **,** must be uniquely defined for the specific grid parameters. Appropriate grid parameters for a given *Gauss-Seidel* solution may not necessarily be the ideal parameters for attempting an *SOR* method solution. With the appropriate , *SOR* can often yield an accurate solution with significantly less iterations than what would be required with *Gauss-Seidel*. Applying the same number of iterative steps for a *Gauss-Seidel* solution to an *SOR* solution would be computationally wasteful. Following grid definitions and/or relaxation factors, the *X* and *Y* domains may be defined and all known conditions at the boundaries respective to the given problem may be initialized. A *U* solution matrix or vector may be initialized as a zero matrix or vector, sized appropriately for the number of grid steps in both dimensions. A zero solution matrix defined prior to the main computational steps is necessary for defining an initial value for every element *u(x,y)* and aids in computational efficiency. Boundary conditions are then defined and a three-level nested for loop is created to process the iterative solutions. The first level is the spans the number of iterations necessary for an accurate solution. The second level cycles through each column entry K for *yk* and provides a location to define a Neumann ghost node condition for *x = 0*. The third and most rapidly alternating level cycles through every row entry J for each *xj* and is the location for processing the 4 Point *Gauss-Seidel* discretized solution as well as the extra implicit steps for the discretized *SOR* method solution. After each for loop bound has been satisfied, the program outputs the final solution for *u(xj,yk)*.

Pseudo Code:

The code structure for both methods are nearly identical. *SOR* steps are highlighted in blue and may be omitted for a pure *Gauss-Seidel* method solver or set = 1.

% Define grid

iter = 6000; % Enter # of iterations for convergence

N = 600; % Enter # of grid intervals (ConvStud: n = 595, w = 1.70524)

w = 1.71; % Enter relaxation coef.(ONLY USED FOR SOR METHOD)

h = (2\*pi)/(N+1); % grid step dx = dy

% Specify Bounds

ax = 0; % x = 0

ay = ax; % y = 0

bx = 2\*pi; % x = L

by = bx; % y = L

len = N+2;

totl = len^2;

endbc = totl-len+1;

% Initialize and vectorize known conditions

F = sin(pi.\*(((h.\*(j-1))-ax)./(bx-ax)))\*cos((pi\*.5).\*((2.\*(((h.\*(k-1))-ay)./(by-ay)))+1)); % F matrix of known F's for all x and y's

F = F(:); % Vectorizes F matrix

fa = (h.\*(j-1)).\*((h.\*(j-1))-ax).^2; % BC equation U(x,y=by)

ga = ((h.\*(j-1))-ax).^2.\*cos((h.\*(j-1)));% BC equation U(x,y=ay)

% Create the U vector then populate with known conditions.

U = zeros(len); % initialize soln. array, zeros are initial vals

U(1:len)= ga; % U(x,y=ay) Boundary Condition

U(endbc:totl)=fa; % U(x,y=by) Boundary Condition

U(len,:) = ga(len)+((((h.\*(k-1))-ay)/(bx-ay))\*(fa(len)-ga(len)));

%U(bx,y) Boundary Condition

preU = U; % SOR init. val for Ujkn-1 (Previous iteration solution)

% Continued on next page

% Commence SOR/Gauss-Seidel Numerical solver

for i=1:iter % loop for every i iter of method until soln convergence

for K = 2:len-1 % Cycling through column entries (Y dimension) U(1,K)=(.25\*(U(2,K)+U(3,K)+U(2,K-1)+U(2,K+1)))+(.25\*h\*h\*F(1+((K-1)\*len))); % "Ghost Node" Neumann condition

for J = 2:len-1 % Cycling through row entries (X dimension)

U(J,K)= (.25\*(U(J-1,K)+U(J+1,K)+U(J,K-1)+U(J,K+1))) +(.25\*h\*h\*F(J+((K-1)\*len)));% Explicit Ujki value for step i

U(J,K)=w\*U(J,K)+(1-w)\*preU(J,K);% SOR Implicit equation soln

preU(J,K) = U(J,K); % Ujki-1 term for next i iteration

end % J

end % K

end % i

% END

**Computer Specifications**

System Info:

System Name: ECC-E-34

Item Value

OS Name Microsoft Windows 10 Enterprise

Version 10.0.16299 Build 16299

OS Manufacturer Microsoft Corporation

System Name ECC-E-34

System Manufacturer Dell Inc.

System Model Precision T5610

System Type x64-based PC

System SKU Precision T5610

Processor Intel(R) Xeon(R) CPU E5-2609 v2 @ 2.50GHz, 2500 Mhz, 4 Core(s), 4 Logical Processor(s)

MaxClockSpeed 2500 MHz

L2 Cache Size 1024 KB

L3 CacheSize 10240 KB

BIOS Version/Date Dell Inc. A17, 7/9/2018

SMBIOS Version 2.7

Embedded Controller Version 255.255

BIOS Mode UEFI

BaseBoard Manufacturer Dell Inc.

Platform Role Desktop

Secure Boot State Off

PCR7 Configuration Binding Not Possible

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 COUGARNET\dzbosque

Time Zone Central Daylight Time

Installed Physical Memory (RAM) 32.0 GB

Total Physical Memory 31.9 GB

Available Physical Memory 27.4 GB

Total Virtual Memory 36.7 GB

Available Virtual Memory 32.1 GB

Page File Space 4.75 GB

Page File C:\pagefile.sys

**Results**

*Gauss-Seidel:*

The Gauss-Seidel method for solving the 2D Poisson equation was a very simple but time intensive way of approximating a numerical solution. The parameters used for defining the grid were:

I = # of iterations, N = # of spatial intervals, Δx = Δy = Δ = Step Size = L/(N+1)

3D contour grids where used to visualize how different parameters affect the numerical solution. The contour lines are 2D functions dependent on x and y for every z-axis level solution U(x,y) defined by x and y. The 3rd dimension is the numerical value for each U(x,y), and the color gradient simply visualizes the increase in magnitude of the numerical values of U.

Figure 2 below displays the solution behavior when the number of iterations are too low for an accurate solution. By not having enough iterations, there are portions of space where the solution is simply not defined while the rest of the solutions have not the chance to mature close to real values.

I = 2000 , N = 600 , Δ = .0105 , Processing Time = 13 s

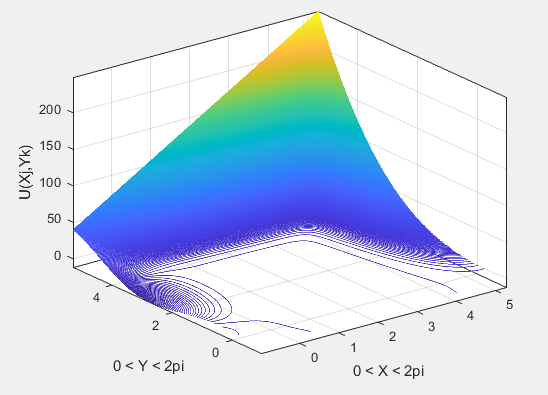


Figure 2: *Gauss-Seidel Numerical Solution Plot for Low # Iterations (2k)*

Next, Figure 3 shows an increase in the spatial area where solutions for U(x,y) are becoming more and more defined as the number of iterations increases. Processing time also increases as the process becomes more computationally intensive.

I = 16000 , N = 600 , Δ = .0105 , Processing Time = 101 s

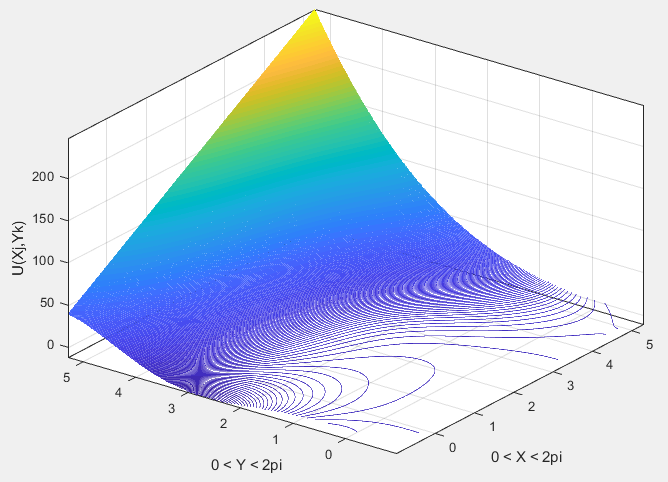


Figure 3: *Gauss-Seidel Numerical Solution Plot for Medium # Iterations (16k)*

Based on the boundary conditions of the problem, it can be assumed more solutions may exist in the empty portions of the plot where X > 0 and Y > 0. The number of iterations must still be increased in order to further define solutions for the rest of the plot.

After tracking the performance of increasing iterations, it was concluded that 32,000 was the most adequate number of iterations necessary for this particular problem. Figure 4 is the extent which 32,000 iterations performed in fully defining the solution set.

I = 32000 , N = 600 , Δ = .0105 , Processing Time = 220 s

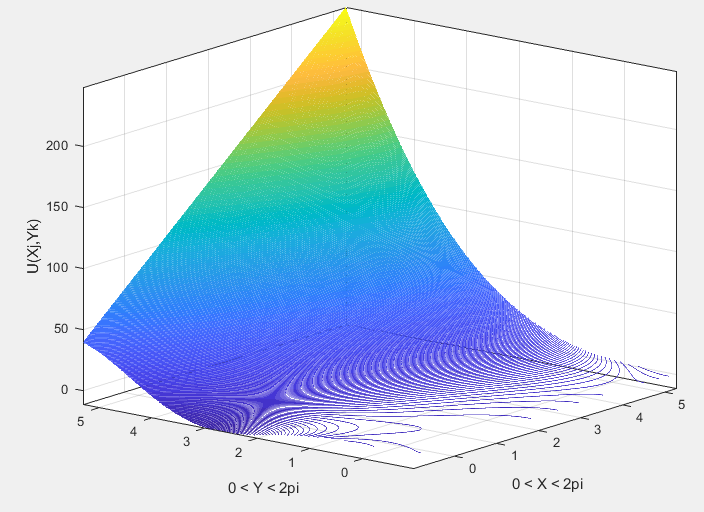


Figure 3: *Gauss-Seidel Numerical Solution Plot for High # Iterations (32k)*

Furthermore, the solution performance was tracked numerically by tracking sample solution sets as the number of iterations increased. Table 1 is a side by side comparison of solution vector samples for I = 2k, 16k, 28k, 32k, 34k.

Table 1: Solution Vector Samples at Increasing Iterations I



As the iterations increase, the expected trend of an iterative solution method like Gauss-Seidel becomes apparent. The solutions are gradually decreasing to the point of convergence to the smallest set of values when I = 32k. When the iterations increase to I = 34k, the values begin increasing again which may be an indicator of rounding error from truncation.

For the *Gauss Seidel* grid convergence study, solution vector sets were sampled from the yk axis spatial “midpoint” at different grid sizes. Table 2 below shows the sample vector sets at y axis solution “midpoint” at different spatial grid resolutions.

Table 2: Grid Sensitivity Vector Samples at Different Grid Sizes N



Initially, grid independence is tested by halving the number of spatial steps and comparing the solution results while the number of iterations is held constant at I = 32k. Halving the grid fineness from N = 600 to N = 300 is quite an extreme change to the solution resolution and with a slow iterative method like *Gauss Seidel*, did not yield very good results. As different levels of increasing grid fineness are evaluated, it appears the solution is grid independent for only small changes in grid resolution. By decreasing the steps from N = 600 to N = 598, the discretization maintains reasonable accuracy. Beyond a small decrease in step, truncation error will begin to grow.

The *Gauss-Seidel* code verification scheme utilized the method of manufactured solutions, which states if an equation of the form:

Where the exact solution is so,

Which means if,

Therefore, if *u(x,y)* is solved using *Gauss Seidel* Method correctly,

and

The verification code yielded favorable results with an average absolute error of

and a maximum absolute error of .

*Successive Over Relaxation (SOR):*

The Successive Over Relaxation method for solving the 2D Poisson equation is a method easily adapted to Gauss Seidel discretization with significant improvements in speed and accuracy. The parameters used for defining the grid were:

I = # of iterations, N = # of spatial intervals, Δ = Step Size, ω = Relaxation Factor

3D contour grids where used to visualize how different parameters affect the numerical solution. Figure 4 below shows how SOR with a proper relaxation factor can yield a numerical solution equal to or better than the best Gauss Seidel solution attempt (I = 32k) with only 6000 iterations.

I = 6000 , ω = 1.71 N = 600 , Δ = .0105 , Processing Time = 60 s

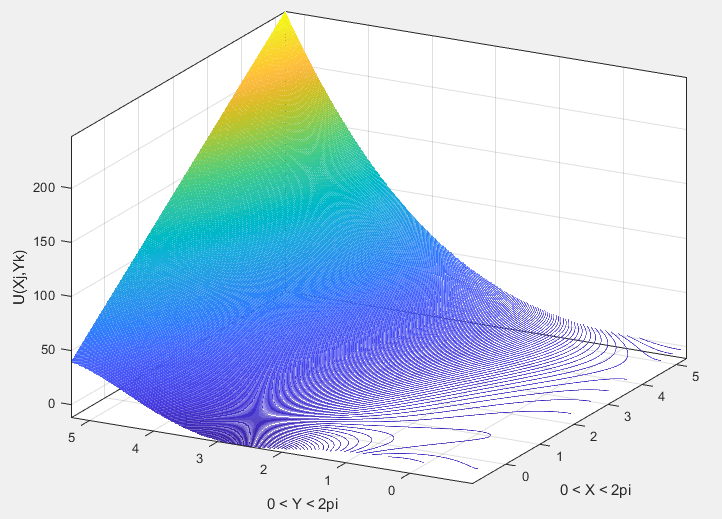


Figure 4: *SOR Numerical Solution Plot I = 6k and* ω = *1.71*

Table 3 shows how the 6k iteration SOR method solution is more accurate than the best Gauss Seidel attempt (I=32k) in only 60 seconds.

Table 3: SOR vs. Gauss Seidel



With this first attempt at an optimized SOR solution, it became apparent that more precise solutions may exist if the number of iterations and relaxation factor were further refined. Interestingly, if the number of iterations is *decreased* and the relaxation factor is *increased* accordingly, the final SOR solution increases in precision and runs faster. Table 4 compares the results of several sample solution vectors with decreasing iterations, and respective relaxation factor. It is important to note the spatial discretization is held constant at N = 600, and all of the contour grids for the solutions in Table 4 look virtually the same, so they are not shown.

Table 4: *SOR Solution Performance for Varying I and ω*



There does come a point where the relaxation reaches its upper limit of ω = 2, as well as using too few iterations. Figure 5 is the result when ω = 2 and the number of iterations is too small to “relax”. The upper x and y bounds get “chopped” off by the enforced boundary conditions.

I = 200, ω = 2, N = 600 , Δ = .0105 , Processing Time = 3 s

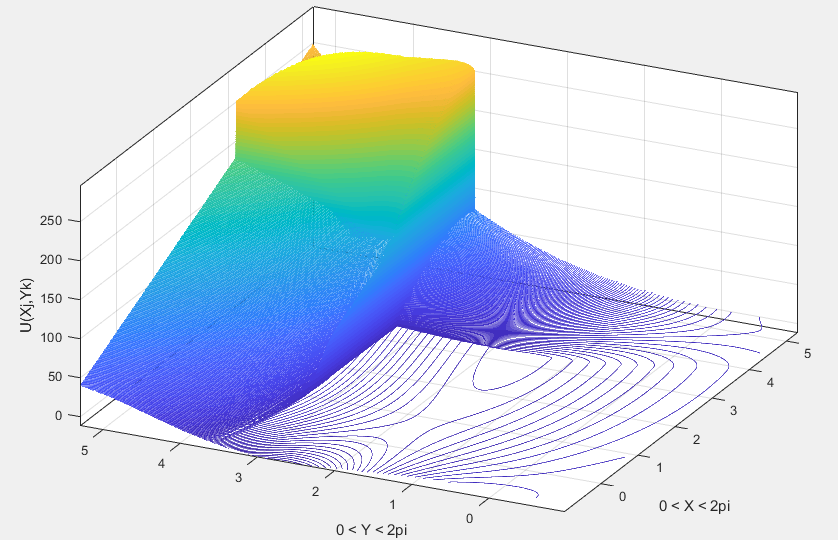
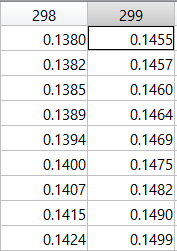
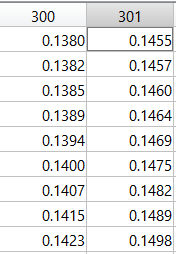


Figure 5: *How Not To Do SOR w/ I = 200 and ω = 2*

The grid convergence study revealed the SOR method was very sensitive to grid changes. Simply halving the grid steps will wildly throw off results because of the method’s reliance on a very specific w relaxation factor. Despite this, grid independence appears to be maintained only when the number of discretized steps is decreased slightly, and the relaxation factor is adjusted accordingly, as seen in the samples below in Table 6. SOR appears to be very grid sensitive, at least, if the relaxation factor is not adjusted appropriately. The SOR method used in the code is grid independent for only tiny changes in spatial discretization.

Table 5: *Example of Small Grid Changes for I = 6k*



SOR code verification was done with the same Method of Manufactured Solution scheme as the Gauss Seidel code. The verification code yielded identical results as the Gauss Seidel solution from before. An SOR solution with parameters:

I = 4000 , ω = 1.81, N = 600 , Δ = .0105 , Processing Time = 38 s

Yielded an average absolute error of and a maximum absolute error of . The code was then further investigated with several more parameter scenarios:

I = 3000 , ω = 1.85, N = 600 , Δ = .0105 , Processing Time = 30 s

I = 2000 , ω = 1.8899, N = 600 , Δ = .0105 , Processing Time = 19.37 s

Each scenario consistently yielded the same error of:

and