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

And yields to a 4-point discretized solution

*“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.

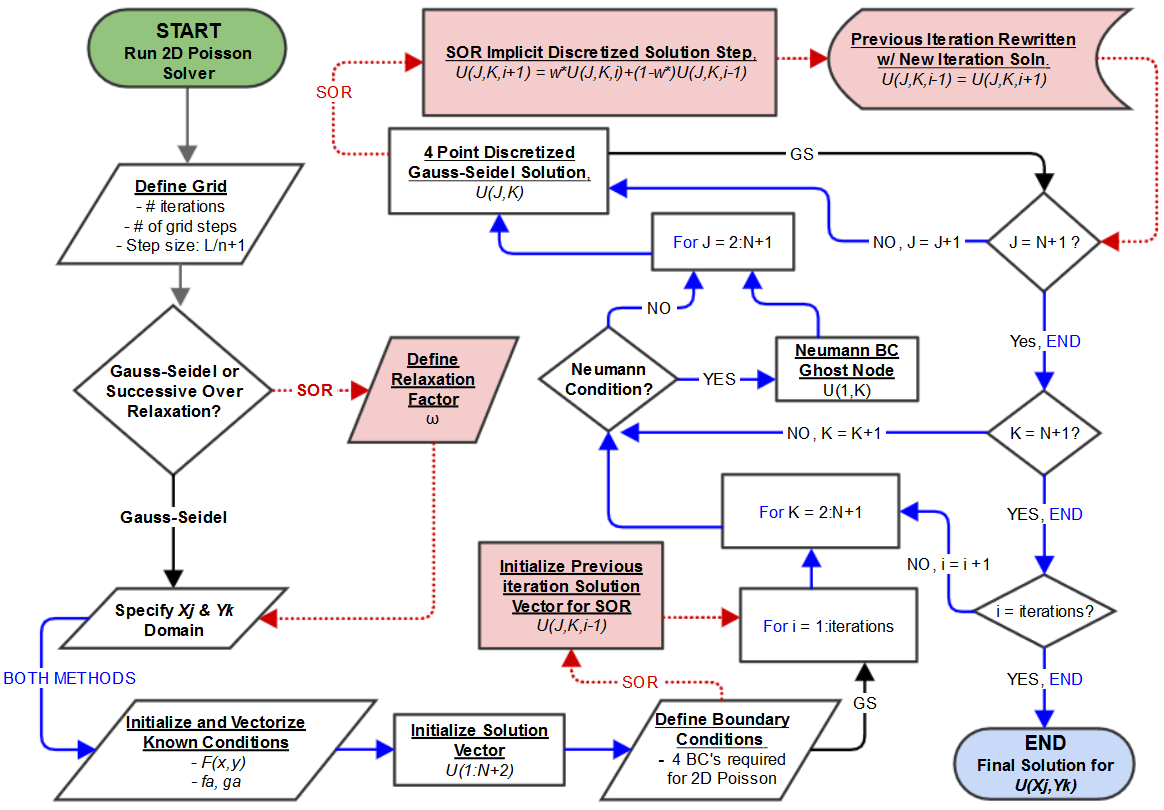


Figure 1: *2D Poisson Iterative Solver Algorithm for Gauss-Seidel & Successive Over-Relaxation*

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**