MECE 5397: Scientific Computing for Engineers

Final Project Report

Poisson Equation in 2D

By: Nha Nguyen

Due date: 5/5/2017

Professors: Andrea Prosperretti, Amit Amritkar

**Proposed Problem**

I started with the 2D Poisson Equation and I am supposed to use Matlab to create an approximate solution using the discretized methods. The form of the 2D Poisson Equation looks like:

Where u is the 2D grid-like matrix solution over a 2D x and y domain that I’m trying to obtain. F is the right-hand side equation:

The domain in question is from and

And there are one Neumann boundary condition for one side and the other three sides are bounded by Dirichlet conditions. The Neumann boundary condition is:

The other three Dirichlet boundary conditions were defined as follow:

The equation on the right hand side of the Poisson PDE, the F is given as:

**The Discretized Method**

The method to approximate the 2D matrix solution U is to use a discretize method. The equation is as below:

and assuming == that are generated from and to reordering the equation to write in Matlab, we have:

and the Matlab code for solving for the U matrix is:

clc

clear

% Putting down constants

n=1000;

ax=0; bx=2\*pi; ay=0; by=2\*pi;

% Generating x and y values

x=linspace(ax,bx,n);

ix=1:n;

y=linspace(ay,by,n);

iy=1:n;

% The boundary conditions

fa=(y-ay).^2.\*cos(y); ga= y.\*(y-ay).^2;

ubx=ga;

uax= fa;

u=zeros(n);

uby = (by-ay).^2.\*cos(by) + (x-ax)/(bx-ax)\*(by\*(by-ay)^2-(by-ay)^2\*cos(by));

hx= 2\*pi/(n-1); hy=hx; h=hy;

u(:,1)=uax; u(:,n)=ubx';

u(n,:)=uby;

%F = 0;

% Loop time

for j=2:n-1

for i=2:n-1

f(i,j) = sin(pi.\*(x(i)-ax)./(bx-ax)).\*cos(pi/2\*(y(j)-ay)./(by-ay)+1);

u(i,j)=1/4\*(u(i-1,j)+u(i,j-1)+u(i+1,j)+u(i,j+1))+h^2\*f(i,j);

end

% Ghost node Neumann Conditions du/dy(y=ay)= 0

u(:,1)=u(:,3);

end

F= zeros(n);

F1 = sin(pi.\*(x(1)-ax)./(bx-ax)).\*cos(pi/2\*(y(1)-ay)./(by-ay)+1);

F(1,1)=F1; F(1,2:end)=0; F(2:end,1)=0;

F(2:end,2:end)= f;

U=u

surf(x,y,U)

The Neumann boundary condition will have a ghost node, so the discretize version of it is:

and was applied in the code at the boundary y = ay inside the j loop but outside the i loop, Where , and using i of all values from 1 to n, and j=1.

The code lists the U matrix at the end and the 3D surface plot of U in the x,y plain at the end.

**The Gauss Seidel Method**

The next method to use to obtain an error value of the discretize method above is the Gauss Seidel method. It is unique that it attempts to solve for new values of U by using the old values of U obtained from the discretize method from above.

An example of how the Gauss Seidel is to use an example of a matrix multiplied set Ax=b, where A is a 2D square matrix, x is a column vector that will be the new value of U that we are trying to obtain of size, nx1; b is a column vector of size nx1 as well and in the example it will be of some random values, but for the Poisson Equation it stands for F. Assume A is size 3x3, x is 3x1, and b is also 3x1 we have:

Where the iterations are also in the order of subscripts of x. In the equation, the and are unknowns and are assumed to be 0s there. In the second iteration with equation the value is known from first iteration and will be reused here, while is unknown and assumed to be 0. This pattern repeats for all iterations to the last one. To apply the Gauss Seidel method to the Poisson Equation solved for above, the U matrix obtained from the discretized method will be set to equal to A, and the b column vector is F, where the F will be of limited values of just the diagonal of matrix F from the discretized method. The code of the Gauss Seidel Method is shown below:

function[X\_solution,iteration\_table,Error\_val]=Gauss\_Seidel(U,Errorinput,F)

% This function aims to solve any 2D PDE equation in the Gausse Seidel

% method. I will use my Poisson Equation for my project as my input values

% of U and F. This function is unique that it solves for each value in the

% form of matrix multiplication style Ax=b, where A=U inut matrix, b=F of

% input column vector, and x is each iteration of individual element U.

A=U;

b=diag(F);

n=length(b);

% Initial values of X assume set-up to be 0

X=zeros(n,1);

Errorvalue=1;

iteration=0;

while (Errorvalue)>Errorinput

iteration=iteration+1; %making as many iterations until it fals within the error given

Z=X;

for i=1:n

j=1:n;

j(i)=[]; % This empties out the part where there are no solutions and coefficients are not needed.

Xtemp=X; % Here Xtemp takes the same values as X.

Xtemp(i)=[];% This reduce the Xtemp vector and eliminates spots with no answers.

X(i)=(b(i)-sum(A(i,j)\*Xtemp))/(A(i,i)); % The Gausse Seidel equation.

end

Xsol(:,iteration)=X;

% The error for the first unknown is usually larger than the rest of

% the other errors so it will be used as the error value.

Errorvalue=abs((X(2,1)-Z(2,1))/(X(2,1)));

end

iteration\_table=[1:iteration;Xsol]';

X\_solution=X;

Error\_val=Errorvalue;

The value for the error equation is also included in this function file. The error equation is:

For the Gauss Seidel method the second term iteration gives the largest error so that point was chosen.

**The SOR Method**

An added technique to the Gauss Seidel Method is Successive Overrelaxation or the SOR technique. The Gauss Seidel is carried out like normal but with an added equation:

Where is a value between 0 and 2. If =1, then the result stays the same, but if is between 0 and 1, the result is the weighted average between the present and previous iterations. For between 1 and 2 the ewuation is considered overrelaxation because it pushes the answer to the correct answer.

The code is a modified Gausse Seidel code with the equation above put in:

function[SolutionSOR,iteration\_table,Error\_val]=Gauss\_SeidelwSOR(U,Errorinput,F,lambda)

% This function aims to solve any 2D PDE equation in the Gausse Seidel

% method. I will use my Poisson Equation for my project as my input values

% of U and F. This function is unique that it solves for each value in the

% form of matrix multiplication style Ax=b, where A=U inut matrix, b=F of

% input column vector, and x is each iteration of individual element U.

A=U;

b=diag(F);

n=length(b);

% Initial values of X assume set-up to be 0

X=zeros(n,1);

Errorvalue=1;

iteration=0;

while (Errorvalue)>Errorinput

iteration=iteration+1; %making as many iterations until it fals within the error given

Z=X;

for i=1:n

j=1:n;

j(i)=[]; % This empties out the part where there are no solutions and coefficients are not needed.

Xtemp=X; % Here Xtemp takes the same values as X.

Xtemp(i)=[];% This reduce the Xtemp vector and eliminates spots with no answers.

X(i)=(b(i)-sum(A(i,j)\*Xtemp))/(A(i,i)); % The Gausse Seidel equation.

old=X(i);

x(i)=lambda\*sum+ (1-lambda)\*old;

end

Xsol(:,iteration)=X;

xsol(:,iteration)=x;

% The error for the first unknown is usually larger than the rest of

% the other errors so it will be used as the error value.

Errorvalue=abs((X(2,1)-Z(2,1))/(X(2,1)));

end

iteration\_table=[1:iteration;xsol]';

SolutionSOR=x;

Error\_val=Errorvalue;