FRANCISCO CASTILLO APM 505 PROJECT

Contents

Initialization of the code

```
clear all    % Clear workspace
close all    % Close all figures
clc          % Clear command window
% Create four column vectors for storing the number of iterations for each method.
nGJ=zeros(5:1);
nGS=zeros(5:1);
nSOR=zeros(5:1);
normGJ=zeros(5:1);
normGS=zeros(5:1);
normSOR=zeros(5:1);
% Assign values of N and tol for each one of the four cases given in the assignment.
for i=1:5
```

```
    switch i
        case 1
            N=8;
            tol=1e-3;
        case 2
            N=16;
            tol=1e-4;
        case 3
            N=32;
            tol=1e-5;
        case 4
            N=64;
            tol=1e-6;
        case 5
            N=128;
            tol=1e-7;
    end
```

Obtain analytical solution

```
x=linspace(0,1,N+1);
[Y,X]=meshgrid(x,x);
U=exp(pi*X).*cos(pi*Y);
```

Run the different iteration methods

```
[uGJ,nGJ(i)]=GaussJacobi(N,tol);
[uGS,nGS(i)]=GaussSeidel(N,tol);
[uSOR,nSOR(i)]=SOR(N,tol);
```

Store the difference with the analytical solution

```
normGJ(i)=norm(uGJ-U);
normGS(i)=norm(uGS-U);
normSOR(i)=norm(uSOR-U);
```

```
end
```

Present the results in a table

```
T=table;
T.N = [8;16;32;64;128];
T.tol = ['1e-3';'1e-4';'1e-5';'1e-6';'1e-7'];
```

```
T.nGJ= nGJ';
T.nGS= nGS';
T.nSOR= nSOR';
T.NormGJ=normGJ';
T.NormGS=normGS';
T.NormSOR=normSOR'
```

```
T =

  5×8 table

    N     tol      nGJ      nGS     nSOR     NormGJ       NormGS       NormSOR
   ___    ____    _____    _____    ____    _________    _________    _________

     8    1e-3       43       47      19       0.14493      0.14887      0.14825
    16    1e-4      208      211      43      0.074341      0.07623     0.076049
    32    1e-5      972      941      99      0.037543     0.038294     0.038253
    64    1e-6     4430     4145     217      0.018868     0.019164     0.019155
   128    1e-7    19865    18103     515     0.0094655    0.0095833    0.0095812
```

Function for the Gauss-Jacobi iteration method

```
type GaussJacobi.m
```

```
function [u,n]=GaussJacobi(N,tol)
% This function uses the Gauss-Jacobi iteration method to come with the solution of the problem for given values of N and tol
h=1/N;          % Define the step
x=h*(0:N)';     % Create vector x
y=h*(0:N)';     % Create vector y
% Create and initialize the solution to zero since is the mean value of the boundary conditions
u=zeros(N+1,N+1);
% Boundary conditions
for j=1:N+1
    u(1,j)=cos(pi*y(j));
    u(end,j)=exp(pi)*cos(pi*y(j));
    u(j,1)=exp(pi*x(j));
    u(j,end)=-exp(pi*x(j));
end
% Create the random matrix uprev to enter the while loop
uprev=rand(N+1,N+1);
% Initialize the counter of iterations n
n=0;
% While loop to keep iterating until the tolerance is met
while norm(uprev-u)>tol
    % Increase the value of n
    n=n+1;
    % Save the previous solution
    uprev=u;
    for i=2:N
        for j=2:N
            % Update the value of the matrix u with the values of the previous iteration
            u(i,j)=(uprev(i+1,j)+uprev(i-1,j)+uprev(i,j+1)+uprev(i,j-1))/4;
        end
    end
end
end
```

Function for the Gauss-Seidel iteration method

```
type GaussSeidel.m
```

```
function [u,n]=GaussSeidel(N,tol)
% This function uses the Gauss-Seidel iteration method to come with the solution of the problem for given values of N and tol
h=1/N;          % Define the step
x=h*(0:N)';     % Create vector x
y=h*(0:N)';     % Create vector yuprev=rand(N+1,N+1);
% Create and initialize the solution to zero since is the mean value of the boundary conditions
u=zeros(N+1,N+1);
% Boundary conditions
for j=1:N+1
    u(1,j)=cos(pi*y(j));
    u(end,j)=exp(pi)*cos(pi*y(j));
    u(j,1)=exp(pi*x(j));
    u(j,end)=-exp(pi*x(j));
end
% Create the random matrix uprev to enter the while loop
uprev=rand(N+1,N+1);
```

```
% Initialize the counter of iterations n
n=0;
% While loop to keep iterating until the tolerance is met
while norm(uprev-u)>tol
    % Increase the value of n
    n=n+1;
    % Save the previous solution
    uprev=u;
    for i=2:N
        for j=2:N
            % Update the value of the matrix u with the values of the previous iteration and the updated values available
            u(i,j)=(u(i+1,j)+u(i-1,j)+u(i,j+1)+u(i,j-1))/4;
        end
    end
end
```

Function for the SOR iteration method

```
type SOR.m
```

```
function [u,n]=SOR(N,tol)
% This function uses the SOR iteration method to come with the solution of the problem for given values of N and tol
h=1/N;          % Define the step
x=h*(0:N)';     % Create vector x
y=h*(0:N)';     % Create vector yuprev=rand(N+1,N+1);
% Obtain the value of omega
xi=1-2*(sin(pi/(2*(N+1))))^2;
w=2/(1+sqrt(1-xi^2));
% Create and initialize the solution to zero since is the mean value of the boundary conditions
u=zeros(N+1,N+1);
% Boundary conditions
for j=1:N+1
    u(1,j)=cos(pi*y(j));
    u(end,j)=exp(pi)*cos(pi*y(j));
    u(j,1)=exp(pi*x(j));
    u(j,end)=-exp(pi*x(j));
end
% Create the random matrix uprev to enter the while loop
uprev=rand(N+1,N+1);
% Initialize the counter of iterations n
n=0;
% While loop to keep iterating until the tolerance is met
while norm(uprev-u)>tol
    % Increase the value of n
    n=n+1;
    % Save the previous solution
    uprev=u;
    for i=2:N
        for j=2:N
            % Update the value of the matrix u using omega and with the values of the previous iteration and the updated values
 available
            u(i,j)=w*(u(i+1,j)+u(i-1,j)+u(i,j+1)+u(i,j-1))/4+(1-w)*u(i,j);
        end
    end
end
```

Published with MATLAB® R2017a