Chapter 7: Iterative Solvers

Joseph Sepich

Contents

1 Problem 1 Comparing Various Methods **1**

1.1 a. Tridiagonal Gaussian elimination 1

1.2 b. Jacobi's Method 2

1.3 c. Gauss-Seidel Method 3

1.4 d. SOR Method 3

2 Problem 2 SOR in Matlab **4**

3 Problem 3 Jacobi Iterations in Matlab **6**

4 Problem 4 **8**

4.1 a. Perform one step of Jacobi iteration with starting vector = [1,1,1]. 8

4.2 b. Perform one step of Gauss-Seidel iteration with startin vector = [1,1,1]. 9

4.3 c. Do either method converge on this system? Why? 9

1 Problem 1 Comparing Various Methods

Solve the following system using the four methods.

$$\begin{pmatrix} 4 & 3 & 0 \\ 3 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 24 \\ 30 \\ -24 \end{pmatrix}$$

1.1 a. Tridiagonal Gaussian elimination

Step 1: $a_1 = a_1/4$

$$\begin{pmatrix} 1 & 0.75 & 0 \\ 3 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 6 \\ 30 \\ -24 \end{pmatrix}$$

Step 2: $a_2 = a_2 - 3a_1$

$$\begin{pmatrix} 1 & 0.75 & 0 \\ 0 & 1.75 & -1 \\ 0 & -1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 6 \\ 12 \\ -24 \end{pmatrix}$$

Step 3: $a_2 = a_2/1.75$

$$\begin{pmatrix} 1 & 0.75 & 0 \\ 0 & 1 & -1.75 \\ 0 & -1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 6 \\ 6.85714 \\ -24 \end{pmatrix}$$

Step 4: $a_1 = a_1 - 0.75a_2$

$$\begin{pmatrix} 1 & 0 & 0.42857 \\ 0 & 1 & -1.75 \\ 0 & -1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0.85714 \\ 6.85714 \\ -24 \end{pmatrix}$$

Step 5: $a_3 = a_3 + a_2$

$$\begin{pmatrix} 1 & 0 & 0.42857 \\ 0 & 1 & -1.75 \\ 0 & 0 & 3.4285714 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0.85714 \\ 6.85714 \\ -17.14285714 \end{pmatrix}$$

Step 6: $a_3 = a_3/3.4285714$

$$\begin{pmatrix} 1 & 0 & 0.42857 \\ 0 & 1 & -1.75 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0.85714 \\ 6.85714 \\ -5 \end{pmatrix}$$

Step 7: $a_1 = a_1 - 0.42857a_3$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -1.75 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 6.85714 \\ -5 \end{pmatrix}$$

Step 8: $a_2 = a_2 + a_3/1.75$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \\ -5 \end{pmatrix}$$

1.2 b. Jacobi's Method

Here we iterate on the vector x, setting it equal each time to the solution of our guess and it's equation. Our iteration scheme is:

$$x_1^{k+1} = \frac{1}{4}(24 - 3x_2^k)$$

$$x_2^{k+1} = \frac{1}{4}(30 - 3x_1^k + x_3^k)$$

$$x_3^{k+1} = \frac{1}{4}(-24 + x_2^k)$$

Let's start with the vector $x^0 = [6, 7.5, -6]'$, which is the best choice starting with a diagonally dominant matrix.

Iteration 1:

$$x^1 = [0.375, 1.5, -4.125]$$
$$x^2 = [4.875, 6.1875, -5.625]$$

After 2 iterations we are not very accurate at all.

1.3 c. Gauss-Seidel Method

Here we iterate on the vector x, doing the same as Jacobi, but using our new x values when we get them. Our iteration scheme is:

$$x_1^{k+1} = \frac{1}{4}(24 - 3x_2^k)$$
$$x_2^{k+1} = \frac{1}{4}(30 - 3x_1^{k+1} + x_3^k)$$
$$x_3^{k+1} = \frac{1}{4}(-24 + x_2^{k+1})$$

Let's start with the vector $x^0 = [6, 7.5, -6]'$, which is the best choice starting with a diagonally dominant matrix.

$$x^1 = [0.375, 5.71875, -4.57031]$$
$$x^2 = [1.71094, 5.07422, -4.73145]$$

Here the x$_3$ value is already getting quite close to it's actual value.

1.4 d. SOR Method

Here we iterate just like the previous Gauss-Seidel method, but give an accelerant to the new value of x. Our iteration scheme is (using w = 1.25):

$$x_1^{k+1} = (1-w)x_1^k + w\frac{1}{4}(24 - 3x_2^k) = (-0.25)x_1^k + 1.25\frac{1}{4}(24 - 3x_2^k)$$
$$x_2^{k+1} = (1-w)x_2^k + w\frac{1}{4}(30 - 3x_1^{k+1} + x_3^k) = (-0.25)x_2^k + 1.25\frac{1}{4}(30 - 3x_1^{k+1} + x_3^k)$$
$$x_3^{k+1} = (1-w)x_3^k + w\frac{1}{4}(-24 + x_2^{k+1}) = (-0.25)x_3^k + 1.25\frac{1}{4}(-24 + x_2^{k+1})$$

Let's start with the vector $x^0 = [6, 7.5, -6]'$, which is the best choice starting with a diagonally dominant matrix.

$$x^1 = [-1.0313, 6.5918, -3.9401]$$
$$x^2 = [1.578, 5.0164, -4.9474]$$

The value of x$_3$ is even closer than the previous two. You can see such a noticable increase in the accuracy of the last variable compared to the first two in this iteration and the Gauss-Seidel over the Jacobi method, because the third variable is able to use the new value obtained from the iteration already whereas x$_1$ uses no new values and x$_2$ still uses x$_3$'s old value when you calculate them. You could also clearly see the affect of having the over relaxtion technique.

2 Problem 2 SOR in Matlab

My Function:

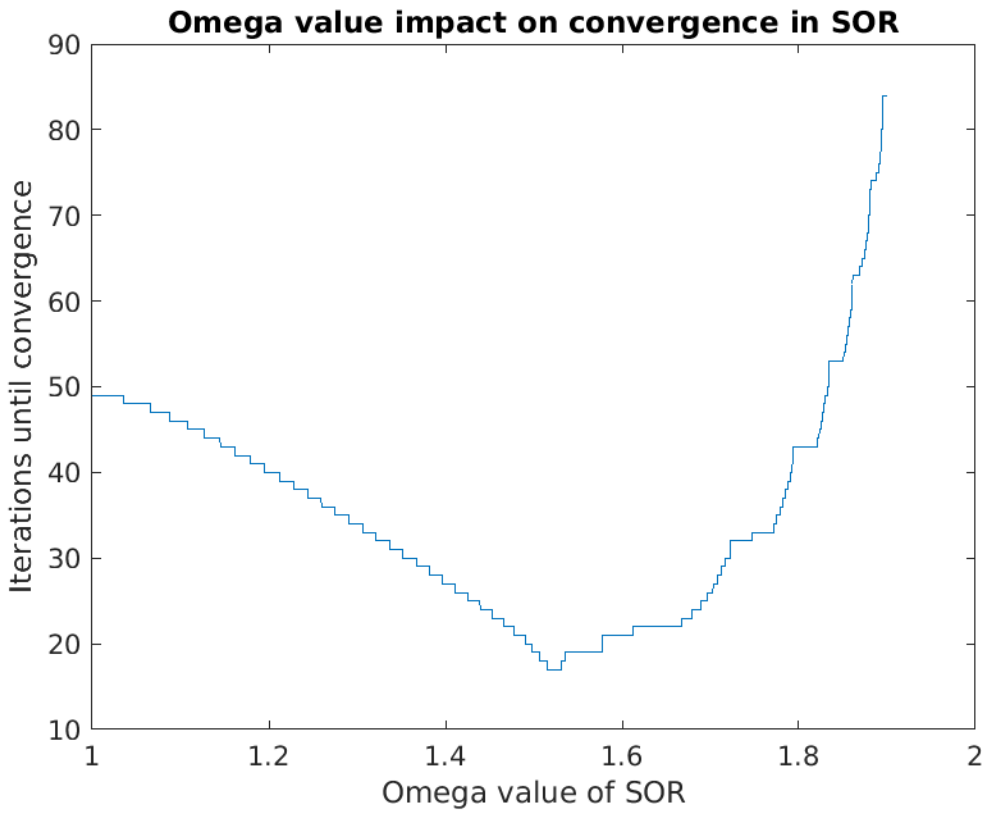
```
function [x,nit]=sor(A,b,x0,w,d,tol,nmax)
% SOR : solve linear system with SOR iteration
% Usage: [x,nit]=sor(A,b,x0,omega,d,tol,nmax)
% Inputs:
%   A : an n x n matrix
%   b : the rhs vector, with length n
%   x0 : the start vector for the iteration% tol: error tolerance
%   w: relaxation parameter, (1 < w < 2),
%   d : band width of A.
% Outputs::
%   x : the solution vector
%   nit: number of iterations

D = diag(diag(A)); % first diag gets diag of A, second turns into diagonal matrix
L = tril(A)- D; % lower triangle, but remove the main diagonal
U = triu(A)- D; % upper triangle, but remove the main diagonal

% check for convergence of system with matrix norm of M
M = inv(D + w*L) * ( (1-w)*D - w*U);
y = inv(D + w*L)*b;
e= max(eig(M)); % L2 norm
if abs(e) >= 1
    disp ('Cannot converge with given parameters')
end

% set initial guess
x = zeros(length(b),nmax);
x(:,1) = x0;

nit = 1;
err1= norm(x(:,1)-b); %set intial error ||xk - xk-1|| <= tol, Axk - b <= tol
err2= norm(A*x(:,1)-b);
while err1 > tol & err2 > tol & nit <= nmax% go until nmax, within error tol for residual or xDiff norm
    x( : ,nit + 1 ) = y + M*x(:,nit);
    err1 = norm(x(:,nit+1)-x(:,nit));
    err2 = norm(A*x(:,nit+1)-b);
    nit = nit + 1;
end

fprintf ('The final ans obtaibed after %d iterations is  \n', nit)
x = x(:,nit-1);
disp(x);

end
```

My Script

```
omega = 1.0:0.0001:1.9;
numIterations = zeros(length(omega),1);
```

```
% given inputs
A = ...
  [-2.011 1 0 0 0 0 0 0 0;
   1 -2.012 1 0 0 0 0 0 0;
   0 1 -2.013 1 0 0 0 0 0;
   0 0 1 -2.014 1 0 0 0 0;
   0 0 0 1 -2.015 1 0 0 0;
   0 0 0 0 1 -2.016 1 0 0;
   0 0 0 0 0 1 -2.017 1 0;
   0 0 0 0 0 0 1 -2.018 1;
   0 0 0 0 0 0 0 1 -2.019];

x = [0.95; 0.9; 0.85; 0.8; 0.75; 0.7; 0.65; 0.6; 0.55];
xResult = zeros(length(x),length(omega));


b = [-0.994974; 0.0015740700000000001; -0.00089667700000000013; -0.0027113700000000003; ...
     -0.00407407; -0.00511719; -0.00592917; -0.00657065; -0.507084];


for i = 1:length(omega)
    [xResult(:,i), numIterations(i)] = sor(A,b,x,omega(i),3,10^-4,100);
end

disp(numIterations)

figure
plot(omega, numIterations)
xlabel("Omega value of SOR")
ylabel("Iterations until convergence")
title("Omega value impact on convergence in SOR")

% After reviewing the plot 1.52 looks to be about the fastest value

sor(A,b,x,1.52,3,10^-4,100);
```

My Figure



My output from fastest convergence:

sor(A,b,x,1.52,3,10^-4,100);

The final ans obtaibed after 17 iterations is

- 0.598282174892330
- 0.548504459835072
- 0.506345876671595
- 0.470228835561903
- 0.438847388887656
- 0.411390422599940
- 0.387154972681848
- 0.365605562729382
- 0.346316779821866

It appears that having a value close to 1.5 is optimal, but it quickly get's harder/less quick to converge after 1.5, versus a much less steep slope in the decrease before the steep increase that we see.

3 Problem 3 Jacobi Iterations in Matlab

My Function

```
function [x,nit] = jacobi(A,b,x0,tol,nmax)
%JACOBI Summary of this function goes here
%   Detailed explanation goes here
```

```
D = diag(diag(A)); % first diag gets diag of A, second turns into diagonal matrix
L = tril(A)- D; % lower triangle, but remove the main diagonal
U = triu(A)- D; % upper triangle, but remove the main diagonal

% check for convergence of system with matrix norm of M
M = -1 * inv(D)*(L + U);
y = inv(D)*b;
e= max(abs(eig(M))); % L2 norm
if abs(e) >= 1
    disp ('Cannot converge with given parameters')
end

% set initial guess
x = zeros(length(b),nmax);
x(:,1) = x0;

nit = 1;
err1= norm(x(:,1)-b); %set intial error ||xk - xk-1|| <= tol, Axk - b <= tol
err2= norm(A*x(:,1)-b);
while err1 > tol & err2 > tol & nit <= nmax% go until nmax, within error tol for residual or xDiff norm
    x( : ,nit + 1 ) = y + M*x(:,nit);
    err1 = norm(x(:,nit+1)-x(:,nit));
    err2 = norm(A*x(:,nit+1)-b);
    nit = nit + 1;
end

fprintf ('The final ans obtaibed after %d iterations is  \n', nit)
x = x(:,nit-1);
disp(x);
end
```

My Script

```
% Set parameters
omega = 1.0:0.0001:1.9;
numIterations = zeros(length(omega),1);
tol = 10^-4;
nmax = 100;


% given inputs
A = ...
  [-2.011 1 0 0 0 0 0 0 0;
   1 -2.012 1 0 0 0 0 0 0;
   0 1 -2.013 1 0 0 0 0 0;
   0 0 1 -2.014 1 0 0 0 0;
   0 0 0 1 -2.015 1 0 0 0;
   0 0 0 0 1 -2.016 1 0 0;
   0 0 0 0 0 1 -2.017 1 0;
   0 0 0 0 0 0 1 -2.018 1;
   0 0 0 0 0 0 0 1 -2.019];

x = [0.95; 0.9; 0.85; 0.8; 0.75; 0.7; 0.65; 0.6; 0.55];
xResult = zeros(length(x),length(omega));
```

```
b = [-0.994974; 0.0015740700000000001; -0.00089667700000000013; -0.0027113700000000003; ...
     -0.00407407; -0.00511719; -0.00592917; -0.00657065; -0.507084];

jacobi(A,b,x,tol,nmax);
```

Output

The final ans obtaibed after 84 iterations is

- 0.909577773388009
- 0.834158809943258
- 0.770274149932905
- 0.715433109122108
- 0.667816804205437
- 0.626053655120232
- 0.589110712523574
- 0.556181082386154
- 0.526643205159853

This method took about as long as the longest convergence of the SOR method. This method is definitely slower than SOR and I cannot see anyway that Jacobi would be better, since SOR builds off of it.

4 Problem 4

Consider the following system.

$$\begin{pmatrix} 5 & 4 & -2 \\ -2 & 8 & -3 \\ 1 & 1 & -7 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 \\ 6 \\ 5 \end{pmatrix}$$

4.1 a. Perform one step of Jacobi iteration with starting vector = [1,1,1].

$$x^{k+1} = \frac{1}{5}(2 - 4y^k + 2z^k)$$
$$y^{k+1} = \frac{1}{8}(6 + 2x^k + 3z)$$
$$z^{k+1} = -\frac{1}{7}(5 - x^k - y^k)$$

Performing one step:

$$vector = [0, 0.875, -0.42857]$$

4.2 b. Perform one step of Gauss-Seidel iteration with startin vector = [1,1,1].

$$x^{k+1} = \frac{1}{5}(2 - 4y^k + 2z^k)$$
$$y^{k+1} = \frac{1}{8}(6 + 2x^{k+1} + 3z)$$
$$z^{k+1} = -\frac{1}{7}(5 - x^{k+1} - y^{k+1})$$

Performing one step:

$$vector = [0, 1.125, -0.55357]$$

4.3 c. Do either method converge on this system? Why?

Well we can take a look at both the actual evaluation of the sytem and the norm of M in the matrix vector form. If we plug the equation into a calculator we can find the actual value with Gaussian elimination to be [0.186969, 0.569405, -0.606232]. You would expect the Gauss-Seidel first iteration to be a better guess than the first Jacobi iteration, but it is further off, so let's look at the value of the matrix norms.

Jacobi we have $M = -D^{-1}(L + U)$ and G-S we have $-(D + L)^{-1}U$. Using a calculator we can see these norms are 0.50411 for the Jacobi method and 0.20701 for the Gauss-Seidel method. That means that both methods must converge.