Implementation of Jacobi and Gauss-Seidel Method

Jingmin Sun

March 14, 2019

Contents

1 Overview

In this project, I implement two algorithms (Jacobi and Gauss-Seidel) on simple and random linear system. In both algorithm, I set the stopping criteria to be the small tolerance of relative error, and the maximum number of iteration.

2 Algorithm

Algorithm (Jacobi Method):

Initialization: $A \succ 0$, diagonally dominant, $tol > 0$, b, x^0, $imax$, $error = \infty$ and x_r, such that $Ax_r = b$.

for *k=0,1,2,*$\cdots imax$ **do**

  if $error \leq tol$ **then**

    stop

  else

    $\delta = \sum_{j=1}^{n} A_{ij} \cdot x_j^k - A_{ii} \cdot x_i^k = A_{i,:} \cdot x^k - A_{ii} \cdot x_i^k$

    $x_i^{k+1} = \dfrac{b_i - \delta}{A_{ii}}$

  end

  $error = \dfrac{||x^{k+1} - x_r||}{||x_{k+1}||}$

end

Algorithm (Gauss-Seidel Method):

Initialization: $A \succ 0$, diagonally dominant, $tol > 0$, b, x^0, $imax$, $error = \infty$ and x_r, such that $Ax_r = b$.

for *k=0,1,2,*$\cdots imax$ **do**

  if $error \leq tol$ **then**

    stop

  else

    $\delta = \sum_{j=1}^{i-1} A_{ij} \cdot x_j^{k+1} + \sum_{j=i+1}^{n} A_{ij} \cdot x_j^k$

    $x_i^{k+1} = \dfrac{b_i - \delta}{A_{ii}}$

  end

  $error = \dfrac{||x^{k+1} - x_r||}{||x_{k+1}||}$

end

3 Implementation of Methods

3.1 Jacobi Method

Listing 1: Jacobi Method

```
%% Algorithm: Jacobi Method
%Input: A,b,x0,tol>0,imax>0
%where A is a matrix, b is a vector to solve *Ax=b*,
%x0 is the initial guess,
function [itr_vec, error1] = Jacobi(A,b,x,tol,imax)
fprintf ('\n Run Jacobi Method:\n');
itr=0;
error1 = [];itr_vec=[];
error=Inf;
n=size(x,1);
x1=A\b;
while error>tol %Stopping criterion: the relative error
    xold=x;

    for i=1:n
        delta = A(i,:) *xold - A(i,i)*xold(i);
        x(i)=(b(i)-delta)/A(i,i);
    end
    itr=itr+1;
    error= norm(x-x1)/norm(x);
    error1 = [error1;error];
    itr_vec = [itr_vec;itr];
     if itr>imax
       break
     end
end

 fprintf(' Total iterarion = %d \n Final answer=',itr);
    for i = 1:n
      fprintf('%15.14f ',x(i));
    end
    fprintf('error = %15.14f\n', error);
```

3.2 Gauss-Seidel Method

Listing 2: Gauss-Seidel method

```
%% Algorithm: Gauss Seidel Method
```

```
%%
function [itr_vec1, error2] = GaussSeidel(A,b,x,tol,imax)
fprintf ('\n Run Gauss-Seidel Method:\n');
error=Inf;
itr=0;itr_vec1=[];
error2 = [];
n=size(x,1);
x1=A\b;
while error>tol %Stopping criterion: the relative error
   for i=1:n
       delta = A(i,:) *x - A(i,i)*x(i);
       x(i)=(b(i)-delta)/A(i,i);
    end

    itr=itr+1;
    error= norm(x-x1)/norm(x);
    error2 = [error2;error];
    itr_vec1 = [itr_vec1;itr];

    if itr>imax
       break;
    end
end

 fprintf(' Total iterarion = %d \n Final answer=',itr);
    for i = 1:n
      fprintf('%15.14f ',x(i));
    end
    fprintf('error = %15.14f\n', error);
```

4 Test of the algorithm

4.1 Simple case

4.1.1 Code

Listing 3: Simple case

```
%% Solution of x in Ax=b using Jacobi Method and Gauss -Seidel Method with the Professor's example

```

```
clear all;clc;

% Initailize A b and intial guess x
A=[3 1 -1;2 4 1;-1 2 5];
b=[4 1 1]';
x=[0 0 0]';
tol=1e-15;
format long;

[itr_vec, error1] = Jacobi(A,b,x,tol,100);
[itr_vec1, error2] = GaussSeidel(A,b,x,tol,100);
semilogy(itr_vec, error1,'b-');
hold on
semilogy(itr_vec1, error2,'r-');
xlabel("# of iteration");
ylabel("Relative error");
legend("Jacobi", "Gauss-Seidel");
t0 = tic;
t1 = toc(t0);
fprintf('\n Total running time of my code is %15.14f\n', t1);
```

4.1.2 Result

The result is attached in the back of the report, and the graph is:

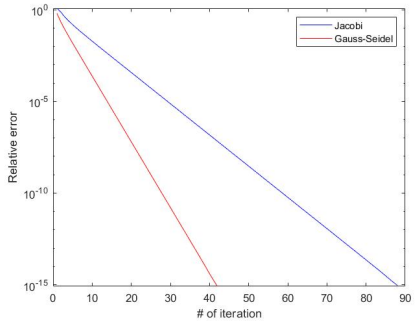


Figure 1: Simple case

4.1.3 Observation

We can easily see that the Jacobi method is much slower than the Gauss-Seidel method, and it is because that Gauss-Seidel uses the latest updated version of the x_j, so we do not waste our effort on working with

the old information.

4.2 Random case

4.2.1 The derivation of diagonally dominant matrix

The diagonal dominant matrix are generated simply by replacing the absolute value of diagonal entries by the sum of the absolute value of each entry in the row and a bit more.

4.2.2 Code

Listing 4: Random case

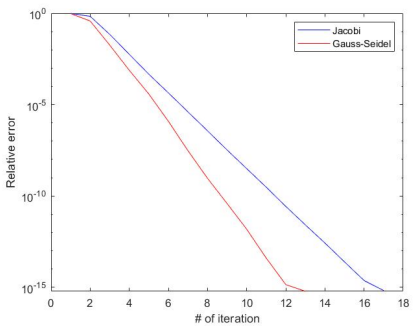
```
%% Solution of x in Ax=b using Jacobi Method and Gauss -Seidel Method
%%
clear all; clc;
tol=1e-15;
format long;


% Initailize A b and intial guess x
r = 100;
randn('seed',20190227);
A = randn(r);
b = randn(r,1);
x = randn(r,1);

% To make the matrix diagonally dominant, I simply made the absolute value
% of diagonal matrix to be the sum of all absolute value in the random
% matrix
for i =1:r
   for j = 1:r
       A(i,i) = abs(A(i,i))+abs(A(i,j));
   end
         q = randn();
       if q >=0
           q = 1;
       else
           q=-1;
       end
       A(i,i) = q * (A(i,i)+abs(randn()));
end
```

```
30
31 [itr_vec, error1] = Jacobi(A,b,x,tol,100);
32 [itr_vec1, error2] = GaussSeidel(A,b,x,tol,100);
33 semilogy(itr_vec, error1,'b-');
34 hold on
35 semilogy(itr_vec1, error2,'r-');
36 xlabel("# of iteration");
37 ylabel("Relative error");
38 legend("Jacobi", "Gauss-Seidel");
39 t0 = tic;
40 t1 = toc(t0);
41 fprintf('\n Total running time of my code is %15.14f\n', t1);
```

4.2.3 Result

The result is attached in the back of the report, and the graph is:

Figure 2: Random 100 x 100 case



4.2.4 Observation

We can get the similar result as before, but we can see that the difference between two method are relatively smaller than before, and the total number of iterations for both methods are small as well.

5 Development

As the claim I made before, I change the size of the matrix to different scale and gives me the following result:

And we can observe that as the size of the matrix increases, the number of iterations of both method are decreased, and the difference of two methods is decreasing as well.

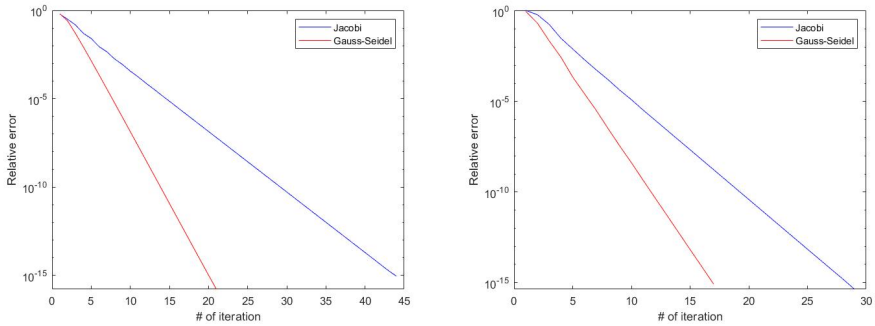


Figure 3: Random 4 x 4 on the left and 10 x 10 on the right

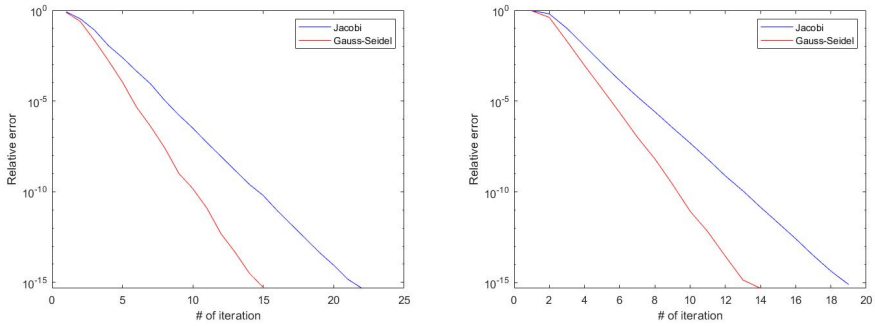


Figure 4: Random 20 x 20 on the left and 50 x 50 on the right

And this phenomenon can be explained: Since for the Jacobi method $B = D^{-1}(U + L)$, and for Gauss-Seidel method $B = (D+L)^{-1}U$, $\tilde{b} = D^{-1}b$, and the diagonal matrix is strictly dominant, with the knowledge of when r (size of the matrix) goes up, each element in a row plays a smaller and smaller role in the sum of that row, and the diagonal entries are completely dominant compared to other entries in matrix A, so that each entry in both B will be relative small, and x can be decrease sufficiently in each iteration, so that the number of iteration will be decrease. And at the same time, the difference between two Bs will decrease as well, and the difference between two methods will decrease.

We can check for that by checking the spectrum radius of B with the code

Listing 5: Spectral radius

```
%% Check spectrum radius
r=[4,10,20,50,100];
for ra = r
randn('seed',20190227);
A = randn(ra);
for i =1:ra
   for j = 1:ra
       A(i,i) = abs(A(i,i))+abs(A(i,j));
   end
```

```
10          q = randn();
11       if q >=0
12           q = 1;
13       else
14           q=-1;
15       end
16       A(i,i) = q * (A(i,i)+abs(randn()));
17  end
18
19  D = diag(diag(A));
20  U = triu(A,1);
21  L =  tril(A,-1);
22  B_Jacobi = inv(D)*(U+L);
23  B_GS = inv(D+L)*U;
24  fprintf(' r=%3i, The spectral radius of B for Jacobi is %5.4f, for GS is %5.4f\n',ra,
        max(abs(eig(B_Jacobi))), max(abs(eig(B_GS))));
25  end
```

and with the result that

Listing 6: Output

```
spec_rad_B
 r=  4, The spectral radius of B for Jacobi is 0.2744, for GS is 0.1211
 r= 10, The spectral radius of B for Jacobi is 0.2646, for GS is 0.0966
 r= 20, The spectral radius of B for Jacobi is 0.1763, for GS is 0.0690
 r= 50, The spectral radius of B for Jacobi is 0.1195, for GS is 0.0502
 r=100, The spectral radius of B for Jacobi is 0.0857, for GS is 0.0308
diary off
```

Since we know that the one with larger spectral radius in absolute value would have slower convergence rate, and that the convergence of x is closed related with the matrix B, the result above enhanced my statement above that GS method is always faster than Jacobi method, and with the increase of the size of the matrix, the algorithm converges faster.

6 Appendix

Listing 7: Simple output

```
Jaco_GS_test
```

```
 Run Jacobi Method:
 Total iterarion = 88
 Final answer=2.00000000000000 -1.00000000000000 1.00000000000000 error = 0.00000000000000

 Run Gauss-Seidel Method:
 Total iterarion = 42
 Final answer=2.00000000000000 -1.00000000000000 1.00000000000000 error = 0.00000000000000

 Total running time of my code is 0.00016326162164
diary off
```

Listing 8: Random output

```
Jaco_GS_test_rand

 Run Jacobi Method:
 Total iterarion = 17
 Final answer=-0.00953221293644 0.03102964159720 -0.02670288589201 0.01141034996126
     0.01146101895463 -0.02304369897164 -0.01299749688888 0.00719513030871 0.01428492000333
     -0.00676040847501 -0.01972940538038 -0.01051071864159 0.00129674178429 0.01355372973897
     0.00473909310659 -0.00396864740256 0.00086754286324 0.00782758969610 0.01506417751114
     0.01671716695877 0.00997399407190 -0.00569521204267 -0.00660199881864 0.01520867186502
     0.01557562818089 -0.00928948393313 0.01837281129422 -0.00113171590029 0.00679911912328
     -0.00039963100544 -0.00475246794837 -0.00526970647460 0.00410254708461 0.01919604186008
     0.00361697159943 0.00515891212947 -0.00050578680710 0.00871824863514 -0.00168223701078
     0.00883135454345 0.00206013977816 0.01274871251172 0.00440442739967 -0.00675708427118
     0.00693900359628 -0.00611678397905 0.01290455862183 -0.00615798112227 0.00730284481805
     0.00123347070814 0.00020307639780 -0.00517444873685 -0.00761110118353 0.00296607956376
     -0.00341359056213 -0.00294843977193 0.00312265119428 -0.01131796912347 0.00157298887629
     0.00621244928405 0.00253063715809 -0.00733349630990 -0.00639403741618 -0.00841807034993
     -0.00692266149524 -0.01444038598254 -0.00576231716805 0.00819879937534 0.00905700095553
     0.00540550114649 -0.00595049757610 -0.01189990467293 -0.00636973350297 -0.00238730613017
     0.00748623896884 -0.01511793390619 -0.01287261109632 0.00679700413495 -0.00894126685931
     -0.00242696894187 -0.00120227710553 0.00364602163489 0.00755764759453 -0.00157433487646
     -0.00955382747928 -0.00262015288562 -0.00518770865543 -0.00179552445352 0.01253654274029
     0.01046167958767 0.00528311646368 -0.00722559894358 0.00897931264537 -0.00411299351431
     0.00034393118885 -0.01763730911159 0.00073161555561 -0.00985187430089 -0.01211137368952
     -0.00197408135070 error = 0.00000000000000

 Run Gauss-Seidel Method:
```

```
 Total iterarion = 13
 Final answer=-0.00953221293644 0.03102964159720 -0.02670288589201 0.01141034996126
     0.01146101895463 -0.02304369897164 -0.01299749688888 0.00719513030871 0.01428492000333
     -0.00676040847501 -0.01972940538038 -0.01051071864159 0.00129674178429 0.01355372973897
     0.00473909310659 -0.00396864740256 0.00086754286324 0.00782758969610 0.01506417751114
     0.01671716695877 0.00997399407190 -0.00569521204267 -0.00660199881864 0.01520867186502
     0.01557562818089 -0.00928948393313 0.01837281129422 -0.00113171590029 0.00679911912328
     -0.00039963100544 -0.00475246794837 -0.00526970647460 0.00410254708461 0.01919604186008
     0.00361697159943 0.00515891212947 -0.00050578680710 0.00871824863514 -0.00168223701078
     0.00883135454345 0.00206013977816 0.01274871251172 0.00440442739967 -0.00675708427118
     0.00693900359628 -0.00611678397905 0.01290455862183 -0.00615798112227 0.00730284481805
     0.00123347070814 0.00020307639780 -0.00517444873685 -0.00761110118353 0.00296607956376
     -0.00341359056213 -0.00294843977193 0.00312265119428 -0.01131796912347 0.00157298887629
     0.00621244928405 0.00253063715809 -0.00733349630990 -0.00639403741618 -0.00841807034993
     -0.00692266149524 -0.01444038598254 -0.00576231716805 0.00819879937534 0.00905700095553
     0.00540550114649 -0.00595049757610 -0.01189990467293 -0.00636973350297 -0.00238730613017
     0.00748623896884 -0.01511793390619 -0.01287261109632 0.00679700413495 -0.00894126685931
     -0.00242696894187 -0.00120227710553 0.00364602163489 0.00755764759453 -0.00157433487646
     -0.00955382747928 -0.00262015288562 -0.00518770865543 -0.00179552445352 0.01253654274029
     0.01046167958767 0.00528311646368 -0.00722559894358 0.00897931264537 -0.00411299351431
     0.00034393118885 -0.01763730911159 0.00073161555561 -0.00985187430089 -0.01211137368952
     -0.00197408135070 error = 0.00000000000000

 Total running time of my code is 0.00015550404998
diary off
```