

MCS 471 Project Three : Iterative Methods in Numerical Linear Algebra

The goal of this project is to investigate some iterative methods to solve linear systems and to compute eigenvalues. In practice, these methods are used on large structured matrices. We will use MATLAB in our numerical investigations.

1. The methods of Jacobi and Gauss-Seidel

To solve the linear system $A\mathbf{x} = \mathbf{b}$, we can view A as $A = M - N$. Then the original linear system $A\mathbf{x} = \mathbf{b}$ is equivalent to the fixed-point problem $M\mathbf{x} = N\mathbf{x} + \mathbf{b}$, giving rise to the iterative method

$$M\mathbf{x}^{(k+1)} = N\mathbf{x}^{(k)} + \mathbf{b}, \quad k = 0, 1, \dots$$

For M the diagonal of A and $N = M - A$, we obtain the method of Jacobi. The formula to update the solution vector \mathbf{x} componentwise at each step is

$$x_i^{(k+1)} = x_i^{(k)} + \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^n a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \dots, n.$$

One immediate improvement to the method of Jacobi is to use the j -th component of the new solution vector $\mathbf{x}^{(k+1)}$ in the update of the i -th component, for $j < i$:

$$x_i^{(k+1)} = x_i^{(k)} + \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i}^n a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \dots, n.$$

The formula above is used in the method of Gauss-Seidel, which corresponds to taking M as the lower triangular part and diagonal of A . Read also section 2.10 of the text book.

Assignments

- 1.1 Write an implementation of the method of Gauss-Seidel in MATLAB. You can clone off from the `jacobi.m` in Appendix 1. See Appendix 2 for the prototype of `gauss_seidel.m`.
- 1.2 Compare the convergence of the methods of Jacobi and Gauss-Seidel on ten random cases. As suggested in the example of `jacobi.m`, generate the data sets (A, \mathbf{x}) as follows:

```
>> Q = orth(rand(5)); D = diag(rand(1,5));
>> A = Q*D*Q'; z = rand(5,1); b = A*z;
>> x0 = z + rand(5,1)*1e-4;
>> [x,dx] = jacobi(A,b,x0,1e-8,50)
>> [x,dx] = gauss_seidel(A,b,x0,1e-8,50)
```

Make a table with three columns: number of test case, number of iterations needed by Jacobi, and number of iterations needed by Gauss-Seidel.

- 1.3 Find an example where the method of Jacobi diverges (the vector \mathbf{dx} should have norm $> 10^{10}$) and the method of Gauss-Seidel converges. Take sufficiently many iterations to obtain a large norm.

2. The Power Method

We say that λ is an eigenvalue of the matrix $A \in \mathbb{R}^{n \times n}$ if there exists a $\mathbf{v} \neq \mathbf{0}$ such that $A\mathbf{v} = \lambda\mathbf{v}$. In general, the n eigenvectors $\mathbf{v}^{(i)}$ (with corresponding eigenvalues λ_i) of A form a basis for \mathbb{R}^n . This means that we can write every vector $\mathbf{x} \in \mathbb{R}^n$ as a linear combination of the eigenvectors:

$$\mathbf{x} = \sum_{i=1}^n \alpha_i \mathbf{v}^{(i)}, \quad \alpha_i \in \mathbb{R}.$$

Observe what happens if we multiply A with \mathbf{x} :

$$A\mathbf{x} = \sum_{i=1}^n \alpha_i A\mathbf{v}^{(i)} = \sum_{i=1}^n \alpha_i \lambda_i \mathbf{v}^{(i)} = \lambda_1 \sum_{i=2}^n \alpha_i \frac{\lambda_i}{\lambda_1} \mathbf{v}^{(i)}.$$

If λ_1 has the largest modulus, then $\left| \frac{\lambda_i}{\lambda_1} \right| < 1$ and $A^k \mathbf{x}$ converges to a vector parallel to \mathbf{v}_1 , as $k \rightarrow \infty$. This observation forms the idea behind the power method, see pages 543-544 in the text book.

Assignments

- 2.1 Run the power method (see Appendix 3 for powers.m) on a random 5-by-5 matrix with a random start vector. Observe the plot of the logarithms of the errors. What is the order of convergence? Linear, or quadratic, or etc... Explain your answer.
- 2.2 Do the following experiment:

```
>> A = rand(5);
>> [v,d] = eig(A);
>> x0 = v(:,2);
>> [lambda,x,error] = powers(A,x0,eps,50)
```

In the experiment above, the start vector is one of the eigenvectors. What do you observe on the plot of the logarithms of the errors? Is there a difference with the case of a random start vector? In any case, explain.

3. The deadline is Friday 8 March, at 1PM

Bring your project solution to class. It should contain the following:

1. The listing of the MATLAB code to implement the method of Gauss-Seidel.
2. Numerical output, with numbers formatted in scientific format. Include print outs of the plots made with the power method.
3. Answers to the questions in the assignments. Please write complete grammatically correct sentences and avoid spelling mistakes.

In addition, you could also hand in the log files created via the **diary** command, eventually after some proper editing. Typing **diary** followed by the name of a file in a MATLAB session creates a new file with the given name which will contain everything you see on the screen during the session. While this file created by **diary** is by no means a substitute for the other items in the project solution, it could help you to gain some (partial) credit when some answers are (partially) wrong.

See <http://www.math.uic.edu/~jan/MCS471> for the hypertext version of this project. In particular, to avoid typing errors, you may want to download the .m files.

If you have questions, comments, or difficulties, feel free to come to my office for help.

Appendix 1: MATLAB implementation of Jacobi's method

Below is a very simple implementation of Jacobi's method, save it as jacobi.m.

```
function [x,dx] = jacobi(A,b,x,eps,N)
%
% The function jacobi applies Jacobi's method to solve A*x = b.
% On entry:
%   A      coefficient matrix of the linear system;
%   b      right-hand side vector of the linear system;
%   x      initial approximation;
%   eps    accuracy requirement: stop when norm(dx) < eps;
%   N      maximal number of steps allowed.
% On return:
%   x      approximation for the solution of A*x = b;
%   dx     vector last used to update x,
%          if success, then norm(dx) < eps.
%
% Example:
%   Q = orth(rand(5)); D = diag(rand(1,5));
%   A = Q*D*Q'; z = rand(5,1); b = A*z;
%   x = z + rand(5,1)*1e-4;
%   [x,dx] = jacobi(A,b,x,1e-8,50)
%
n = size(A,1);
fprintf('Running the method of Jacobi...\n');
for i = 1:N
    dx = b - A*x;
    for j = 1:n
        dx(j) = dx(j)/A(j,j);
        x(j) = x(j) + dx(j);
    end;
    fprintf(' norm(dx) = %.4e\n', norm(dx));
    if (norm(dx) < eps)
        fprintf('Succeeded in %d steps\n', i);
        return;
    end;
end;
fprintf('Failed to reached accuracy requirement in %d steps.\n', N);
```

Appendix 2: MATLAB prototype of the method of Gauss-Seidel

For the implementation of the method of Gauss-Seidel, you should use the following prototype (save as gauss_seidel.m):

```
function [x,dx] = gauss_seidel(A,b,x,eps,N)
%
% The function gauss_seidel applies the method of Gauss-Seidel
% to solve  $A*x = b$ .
% On entry:
%   A      coefficient matrix of the linear system;
%   b      right-hand side vector of the linear system;
%   x      initial approximation;
%   eps    accuracy requirement, stop when  $\text{norm}(dx) < \text{eps}$ ;
%   N      maximal number of steps allowed.
% On return:
%   x      approximation for the solution of  $A*x = b$ ;
%   dx     vector last used to update x,
%          if success, then  $\text{norm}(dx) < \text{eps}$ .
%
% Example:
%   Q = orth(rand(5)); D = diag(rand(1,5));
%   A = Q*D*Q'; z = rand(5,1); b = A*z;
%   x = z + rand(5,1)*1e-4;
%   [x,dx] = gauss_seidel(A,b,x,1e-8,50)
%
```

Appendix 3: MATLAB implementation of the power method

Save the following as powers.m:

```
function [lambda,x,err] = powers(A,x,eps,N)
%
% Applies the power method to find the dominant eigenvalue lambda
% and corresponding eigenvector x of the matrix A.
% On entry:
%   A      a matrix;
%   x      initial approximation for the eigenvector;
%   eps    accuracy requirement: stops when norm(x-A*x) < eps;
%   N      maximal number of iterations allowed.
% On return:
%   lambda modulus of dominant eigenvalue of A;
%   x      eigenvector corresponding with lambda;
%   err    estimate for the errors;
%          the logarithms of the errors are plotted.
%
% Example:
%   A = rand(5); x = rand(5,1);
%   [lambda,x,error] = powers(A,x,1.0e-13,50)
%
x0 = x/norm(x);
fprintf('Running the power method...\n');
for i = 1:N
    x = A*x0;
    x = x/norm(x);
    err(i) = norm(x-x0);
    fprintf(' error : %.4e\n', err(i));
    if err(i) < eps
        fprintf('Succeeded in %d steps.\n', i);
        lambda = norm(A*x);
        plot(1:i,log10(err),'*');
        return;
    end;
    x0 = x;
end;
fprintf('Failed to reach accuracy requirement in %d steps.\n', N);
lambda = norm(A*x);
plot(1:N,log10(err),'*');
```