Master on Numerical Methods in Engineering

INTERNSHIP

Algorithms for computing eigenvalues in sparse matrices

Author : Seyed mohammadreza Attar Seyedi

External supervisors : Guillaume Houzeaux and Paula Cordoba

February 2018

Contents

1 Introduction

1.1 Motivation

Eigenvalues and eigenvectors play an important part in linear algebra applications of science and engineering. The type of matrices that we find in these fields, usually are large and sparse as they come from previously discretized PDE's with finite element, finite volume or finite difference techniques. This makes it difficult to find the eigenvalues of a matrix in the traditional way, this is solving the roots of its characteristic polynomial. In cases usually the eigenvalues of the different matrices are found using iterative methods. Depending on the type of application it is studied, the whole spectrum will be needed or just the biggest/smallest eigenvalue.

1.2 Objectives

The main objective is to find and develop a method to compute the eigenvalues in an efficient way. This has been done in the following way :

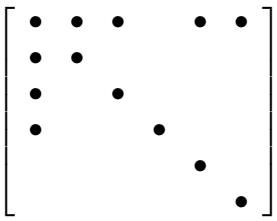
- Looking for bibliography and choosing an algorithm to compute the eigenvalues.
- Developing and writing the code for computing the eigenvalues for different sizes in sparse and dense matrices.
- Looking at CPU time and computing the condition number in each case.
- Finding the relationship between condition number and size of the matrix.

2 Pervious concepts needed

Before starting with my objectives, I needed to know well these concepts.

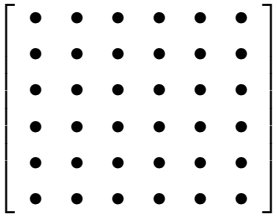
2.1 Sparse matrix

A sparse matrix is a matrix in which most of the elements are zero. Sparse matrices occur naturally in the solution of many real life problems. At more theoretically level sparse matrices arise in graph theory, linear programming, finite elements and the solution of ordinary and partial differential equations. Sparse matrices can solve linear system of equations or linear optimization problems that have thousands of variables, but whose coefficient are mostly zeros.



2.2 Dense matrix

A dense matrix is a matrix in which most of the elements are nonzero. Dense matrices store every entry in the matrix. They contain high percentage of occupied value in matrices.



2.3 Eigenvalues

The eigenvalue problem for a $(n \times n)$ matrix A relies on the determination of the nontrivial solutions u to :

$$Au = \lambda u$$

The eigenproblem is equivalent to find the roots of the characteristic equation :

$$det(A - \lambda I) = 0$$

There are several different techniques for solving the eigenproblem. The problem of computing the full spectrum is considered, in particular for the case of sparse matrices as they typically arise from Finite Element discretized

PDEs.

2.4 Condition number

The condition number plays an important role in numerical linear algebra. The condition number measures the sensitivity of the solution of a problem to perturbations in the data, it provides an approximate upper bound on the error in a computed solution. It depends on the norm used and can also be used to predict the convergence of iterative methods.
The convergence of an iterative process and the existence of linear system of equations depends on the form of the matrix A. The condition number is defined by :

$$k(A) = ||A||||A^{-1}||$$

To k make computation affordable is given in terms of singular value of the matrix.

$$k(A) = \frac{\sigma_{max}}{\sigma_{min}} = \frac{\sqrt{\lambda_{max}}}{\sqrt{\lambda_{min}}}$$

3 Methodology for computing eigenvalues

There are some iterative methods to find the eigenvalues like the power method, QR method, Jacobi method, Bisection method.
I will focus in the QR method and compare it with the power method, as they are appropriate ones for the type of matrices that are used mostly used in BSC, as most of the matrices are non-symmetric.

3.1 QR method

The QR method is a good method for computing eigenvalues in sparse and dense matrices. For calculating all eigenvalues we use QR method. In this method the cost per step in general is $o(n^3)$. Basically the QR method consists in using the QR decomposition to normalize and orthogonalize in every step.
First, the QR factorization has to be done, this is an expensive computation and it needs to be done in every time step, this is :

Every $n \times m$ matrix A has a matrix decomposition

$A = QR$

R is a $n \times m$ upper triangular matrix.

Q is a $n \times n$ orthogonal matrix.

The square matrix A can be decomposed as the product of an orthogonal matrix Q and an upper triangular matrix R.Now the order of the multiplication product of Q and R will be reversed product and R eliminated, $RQ = Q * AQ$. Since $Q * AQ$ is a similarity transformation of A, RQ has the same eigenvalues as A. More importantly, we will later see that by repeating this process. The matrix RQ will become closer and closer to upper triangular, such that the eigenvalues can be read off term the diagonal. That is QR method generate a sequence of the matrices A_k initialed with $A_0 = A$ and its given by :

$$A_k = R_k Q_k$$

Where Q_k and R_k represents a QR factorization of A_{k-1}

$$A_{k-1} = Q_k R_k$$

Algorithm 1 :
Basic QR algorithm is given by :

Input : A matrix $A \in C_{n\times n}$

Output: Matrices U and T such that $A = UTU^*$

Set $A_0 := A$ and $U_0 = I$ for k=1,...do

Compute QR factorization : $A_{k-1} := Q_k R_k$

Set $A_k := R_k Q_k$

Set $U_k := U_{k-1} Q_k$

end

Return $T = A^\infty$ and $U = U^\infty$

3.2 Power method

The power method is also known as single-vector iteration method as it is based on a simple recursion starting from an arbitrary nonzero vector. The power method computes the largest eigenvalue and it is corresponding eigenvector of symmetric and non-symmetric matrices.
In this method in every iteration we need to do one matrix $\times$ vector multiplications, which can be done in $o(n^2)$, where n is the length of the vector. Calculating the normalization constant is a summation of n elements ($o(n)$) and normalizing the vector with this constant is done by n multiplications, thus we find that in every iteration we need to do $o(n^2)$ operation.
The matrix is repeatedly applied to an arbitrary starting vector and renormalizes at each step.

Algorithm 2 :

The eigenvector corresponding λ_1 is the dominant vector

$$\lambda_1, \lambda_2, ..., \lambda_n$$

$$|\lambda_1| > |\lambda_i|, i = 1, 2, ..., n$$

$$x_1 = Ax_0$$

$$x_2 = Ax_1 = A(Ax_0) = A^2x_0$$

$$\lambda = \frac{Ax.x}{x.x}$$

x is an eigenvector of matrix A and $Ax = \lambda x$

$$\frac{Ax.x}{x.x} = \frac{\lambda x.x}{x.x} = \frac{\lambda(x.x)}{x.x} = \lambda$$

4 Numerical results

The numerical results will be focused in the QR method, which is the one that I have implemented in MATLAB, and it will be compared in the test cases with the power method. Basically the results are based on :

- Trying to calculate eigenvalues for random matrices in the QR method to see if it is fine.
- Comparing the largest eigenvalue in the test cases with the power method.
- Computing the eigenvalues for large matrices (sparse and dense) with the QR methods.

The results will be shown for :

- The number of iterations needed to compute the eigenvalues.
- Condition number.
- CPU time.

4.1 Random matrices

4.1.1 QR method

CPU time shows in Figure 1 and the number of iterations for random matrices shows in Figure 2.
In the QR method we can realize from figures that number of iterations and CPU time grow, when the size of the matrix start to increase. The eigenvalues and condition number grow with growth of the size of matrix like power method

Eigenvalues	10.837142	7.151864	4.88934	2.121630

Iterations	24
CPU time	0.14620
Condition number	5.1079

Table 1: $[4 \times 4]$ matrix

Eigenvalues	10.911626	7.483538	9.933118	0.399592	4.804090	3.468036

Iterations	51
CPU time	0.15625
Condition number	27.3069

Table 2: $[6 \times 6]$ matrix

Iterations	101
CPU time	0.21451
Condition number	53.759

Eigenvalues	16.578694	10.415816	4.672964	6.163960	-0.904298	-0.409561
	-1.667196	1.149621				

Table 3: $[8 \times 8]$ matrix

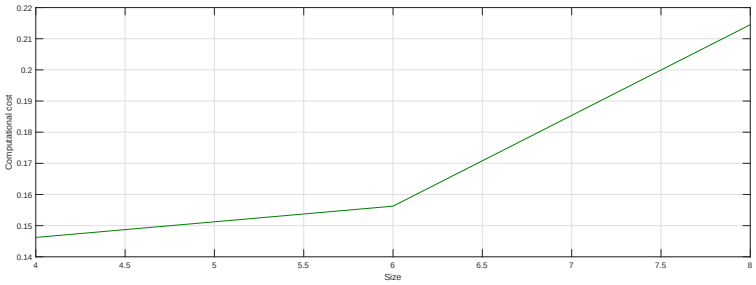


Figure 1: CPU time in the QR method for random matrices

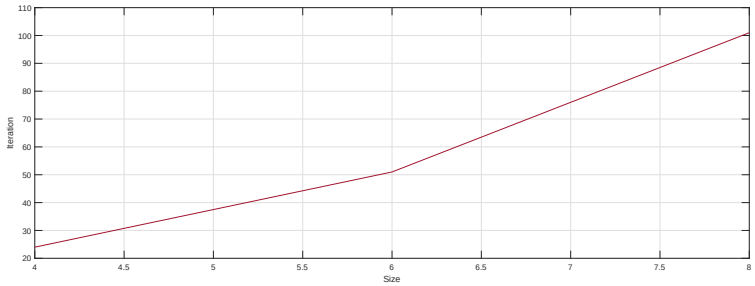


Figure 2: The number of iterations in the QR method for random matrices

4.1.2 Power method

The largest eigenvalue and corresponding eigenvector is shown for the power method. Also the condition number, the number of iterations and CPU time needed is shown.
Figure 3 shows CPU time and Figure 4 shows the number of iterations in power method for random test matrices.

Eigenvectors	0.18371	-0.663219	0.722126	0.070126

Largest eigenvalue	10.837142
Iterations	25
CPU time	0.14673
Condition number	5.1079

Table 4: $[4 \times 4]$ matrix

Eigenvectors	0.167742	-0.704622	0.642538	0.007577	0.242324	-0.061123

Largest eigenvalue	10.911634
Iterations	76
CPU time	0.15620
Condition number	27.3069

Table 5: $[6 \times 6]$ matrix

Eigenvectors	0.450650	0.299429	0.537054	0.414099	0.148789	0.130994
	0.152812	0.429770				

Largest eigenvalue	16.578695
Iterations	110
CPU time	0.25241
Condition number	53.759

Table 6: $[8 \times 8]$ matrix

In the power method, we can understand that the eigenvalues, the number of iterations, condition number and CPU time grow with the growth of the size of the matrix.

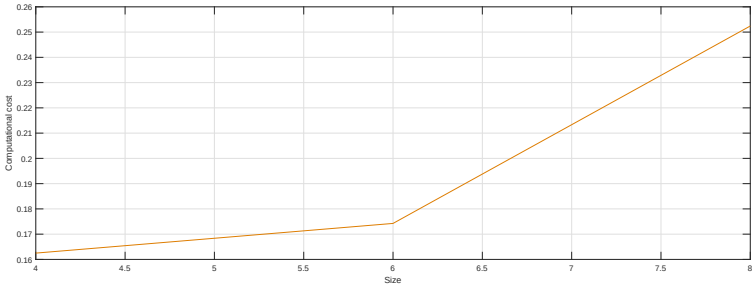


Figure 3: CPU time in the power method for random matrices

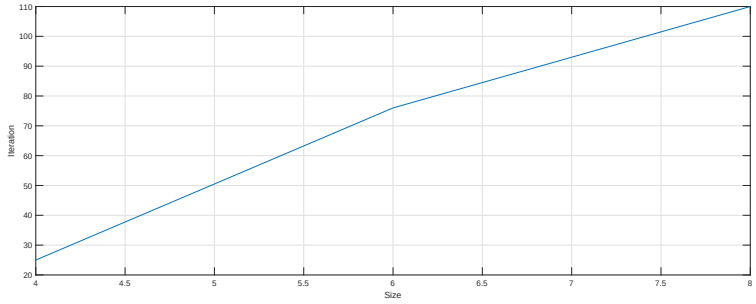


Figure 4: The number of iterations in the power method for random matrices

4.2 Comparison of sparse and dense matrices with QR method

4.2.1 Sparse matrices

The eigenvalues obtained by using the QR method for sparse matrix :

Sparse matrix [100 × 100]	Sparse matrix [1000 × 1000]
Eigenvalues	
$49.8046 + 0.000i$	4.9975+0.000i
$-2.7206 + 1.1224i$	-0.0793+0.0468i
$2.0850 + 1.8783i$	-0.0793-0.0468i
$2.0850 - 1.8783i$	-0.0650+0.0649i
$2.7484 + 0.000i$	0.0667+0.0631i
Condition number	
$5.4845e + 3$	$1.5612e + 5$

Table 7: Eigenvalues for [100 × 100] and [1000 × 1000] sparse matrix

The condition number for sparse matrices shows in Table 8.
The condition number in the sparse matrix is larger than the condition number in the dense matrix.

Condition number	
[100 × 100]	2.4213e+3
[200 × 200]	7.5129e+3
[700 × 700]	8.6751e+4
[1000 × 1000]	1.3444e+5

Table 8: Sparse matrices [A]

Figure 5 shows the condition number starts from [100×100] to [1000×1000] respect to the size of the matrix.
Figure 6(a) shows the condition number starts from [100×100] to [3000×3000] respect to the size of the matrix for sparse matrices.
Figure 6(b) shows the CPU time starts from [100 × 100] to [3000 × 3000] respect to the size of the matrix for sparse matrices.

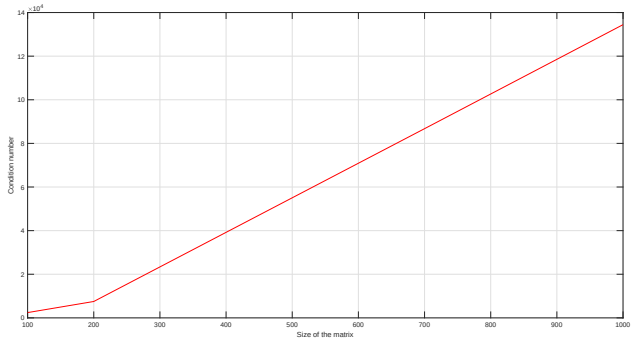
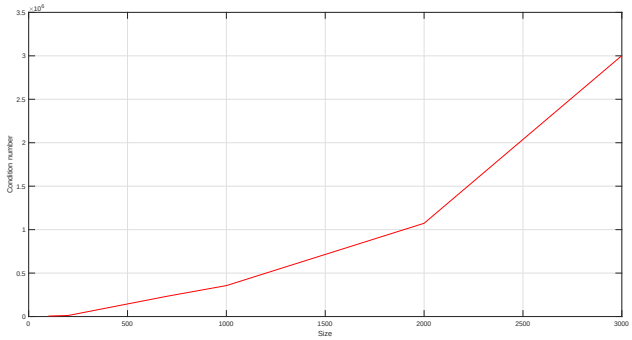


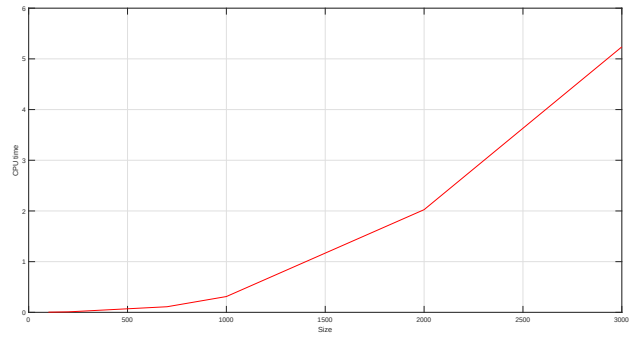
Figure 5: Sparse matrices [A] from [100 × 100] to [1000 × 1000] (Size - Condition number)

Size	Condition number	CPU time
[100 × 100]	5.4845e+3	0.003288
[200 × 200]	1.2258e+4	0.009270
[700 × 700]	2.3349e+5	0.110624
[1000 × 1000]	3.5612e+5	0.311733
[2000 × 2000]	1.0731e+6	2.025370
[3000 × 3000]	3.0024e+6	5.236781

Table 9: Sparse matrices [C]



(a) Sparse matrices [C] from $[100 \times 100]$ to $[3000 \times 3000]$ (Size-Condition number)



(b) Sparse matrices [C] from $[100 \times 100]$ to $[3000 \times 3000]$ (Size - CPU time)

Figure 6: Sparse matrices [C]

4.2.2 Dense matrices

The condition number shows in Table 10 for dense matrices.

Condition number	
$[100 \times 100]$	1.0274e+3
$[200 \times 200]$	3.5412e+3
$[700 \times 700]$	3.3736e+4
$[1000 \times 1000]$	4.6772e+4

Table 10: Dense matrices [B]

The CPU time in the dense matrix is bigger than sparse matrices in the QR method.
The condition number in dense matrices grows rapidly after $[1000 \times 1000]$.
Calculating the sparse and dense matrix bigger than $[7000 \times 7000]$ takes a long time.

Figure 7 shows the condition number starts from $[100 \times 100]$ to $[1000 \times 1000]$ respect to the size of the matrix for dense matrices.

Figure 8(a) shows the condition number starts from $[100 \times 100]$ to $[3000 \times 3000]$ respect to the size of the matrix for dense matrices.
Figure 8(b) shows the CPU time starts from $[100 \times 100]$ to $[3000 \times 3000]$ respect to the size of the matrix for dense matrices.

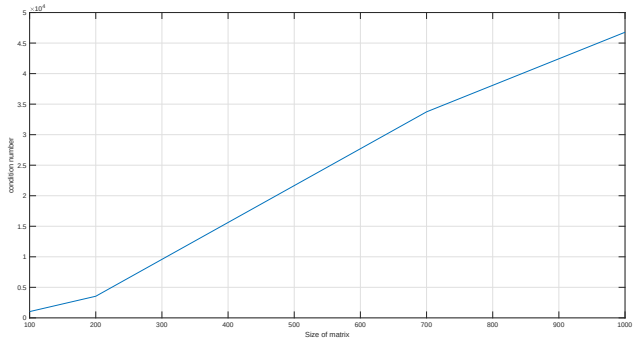
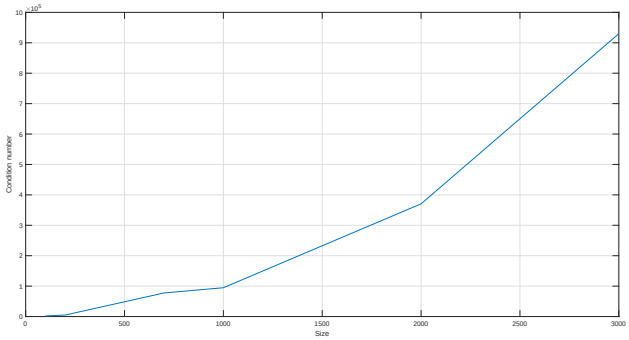


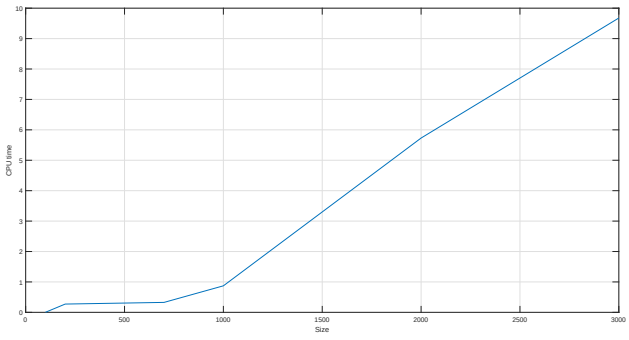
Figure 7: Dense matrices [B] from $[100 \times 100]$ to $[1000 \times 1000]$ (Size - Condition number)

Size	Condition number	CPU time
$[100 \times 100]$	1.8605e+3	0.001820
$[200 \times 200]$	4.9607e+3	0.272188
$[700 \times 700]$	7.7545e+4	0.327170
$[1000 \times 1000]$	9.4752e+4	0.871795
$[2000 \times 2000]$	3.7057e+5	5.730334
$[3000 \times 3000]$	9.3000e+5	9.676681

Table 11: Dense matrices [D]

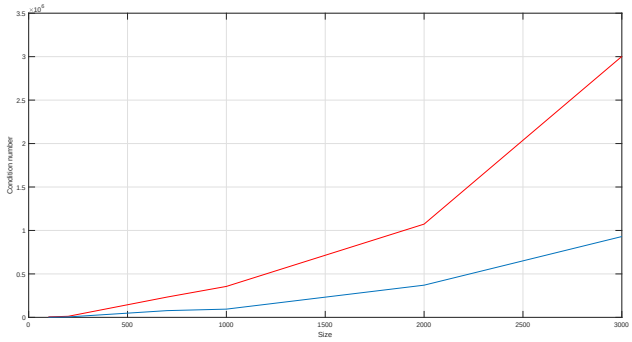


(a) Dense matrices [D] from $[100 \times 100]$ to $[3000 \times 3000]$ (Size - Condition number)

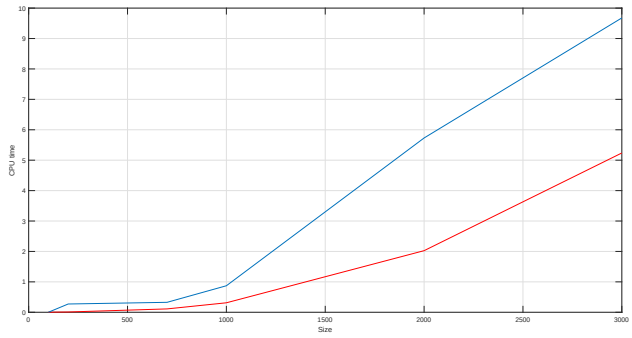


(b) Dense matrices [D] from $[100 \times 100]$ to $[3000 \times 3000]$ (Size - CPU time)

Figure 8: Dense matrices [D]



(a) Size - Condition number



(b) Size - CPU time

Figure 9: The red line corresponds to the dense matrices and the blue line to the sparse matrices

5 Conclusions

Iterative methods are used to compute eigenvalues and eigenvectors in large sparse and dense matrices. There are some iterative methods like the Power method, QR method, Jacobi algorithm and Bisection method.
The QR method can be used for all types of matrices and also it computes all the eigenvalues of the matrix, which it can be useful in some applications that the information of the whole spectrum is needed.

From what I have seen in the results, I can say that :

- The largest eigenvalues computed with the QR method for the initial test cases, matches with the one of the power method.

- The number of iterations, CPU time increase by growing of the size of the matrix, on the other hand the number of iterations and CPU time increase when we need to compute large sparse or dense matrices.

- There are strong relationships between the condition number and size of the matrix. The condition number grow with growth of the size of the matrix.

- The value of condition number in sparse matrices is bigger than dense matrices, but CPU time for dense matrices is larger than sparse matrices.