Louis-Clément Olivier and Pierre Durollet

February 2024

Contents

1 Introduction

In this report we will compare different methods to compute the eigenpairs of a given matrix. first we will do a few tests using the Power Method as seen in the CTD of *Calcul Scientifique*. Then we will compare it to a more efficient method called the subspace iteration method. We will then review different improvements for this method to try and make it more efficient.

2 Low efficiency of the Power Method

n	*imat*	Temps eig (s)	Temps puissance (s)
200	1	0.01	1.380
200	2	0.01	0.01
200	3	0.03	0.05
200	4	0.001	1.2
300	1	0.03	4.8
300	2	0.02	0.03
300	3	0.01	0.15
300	4	0.02	6.57
500	1	0.04	28.51
500	2	0.03	0.1
500	3	0.04	0.61
500	4	0.07	28

Table 1: Time comparison between the power method and the eig function of matlab

Question 1 : As we can see in the table above, the power method struggles to get efficient results on matrices of type 1 and 4 whereas the eig function of matlab manages to stay under 0.1s even with 500x500 type 4 matrices which have low conditioning (which means slower convergence).

2.1 Improvement of the power method

Question 2 : We can improve the power method's algorithm by storing the value of $A \cdot v$ instead of computing it twice per loop. With this improvement we end up with this modified version of the algorithm:

Algorithm 1 Improved Power Method

Matrix $A \in \mathbb{R}^{n \times n}$
(λ_1, v_1) eigenpair associated to the largest (in module) eigenvalue.
$v \in \mathbb{R}^n$ given.
$\beta = v^T \cdot A \cdot v$
repeat
    $y = A \cdot v$
    $\beta_{old} = \beta$
    $\beta = v^T \cdot y$
    $y = y/||y||$
until $|\beta - \beta_{old}|/|\beta_{old}| < \epsilon$
$\lambda_1 = \beta, v_1 = v$

By testing with the same size and type of matrices we end up with the following results, as we can see they are about twice as fast with the improved method.

n	*imat*	Temps power_v1 (s)	Temps power_v2 (s)
200	1	1.380	0.7
200	2	0.01	0.02
200	3	0.05	0.06
200	4	1.2	0.8
300	1	4.8	2.59
300	2	0.04	0.03
300	3	0.15	0.07
300	4	6.57	2.96
500	1	28.51	11.4
500	2	0.1	0.04
500	3	0.61	0.38
500	4	28	11.6

Table 2: Time comparison between the regular power method and the improved

2.2 Main disadvantage of the power method

Question 3 : The deflated power method can be very slow to converge. Mostly because it is very dependant on the initial value of the starting vector v. and if multiple eigenvalues are close in magnitude, the convergence slows down even more. This method is also quite heavy in computation which doesn't help the efficiency.

3 Extending the Power Method to Compute Dominant Eigenspace Vectors

3.1 Subspace iter v0: a basic method to compute a dominant eigenspace

Question 4 : We cannot simply apply the power method to a matrix V made up of m vectors. If we did so the output would be a matrix where each row is v1, the largest eigenvector, as the power method is an algorithm that simply computes the largest eigenpair of a given matrix.

Question 5 : The reason why it is fine to compute the entire spectral decomposition of H while it is a problem to do the same with A is because we have :

$$H = V^T \cdot A \cdot V,\ V \in \mathbb{R}^{n \times m},\ A \in \mathbb{R}^{n \times n} \tag{1}$$

Therefore H is of size $m \times m$ which means we are only computing m eigenvectors instead of the n eigenvectors that we would have computed if we had tried the same on A.

4 Subspace_iter_v1 Improved Version Making use of Raleigh-Ritz's Projection

Algorithm 2 : Subspace iteration method v1 with Raleigh-Ritz projection

Input: Symmetric matrix $A \in \mathbb{R}^{n \times n}$, tolerance ε, $MaxIter$ (max nb of iterations) and PercentTrace the target percentage of the trace of A
Output: n_{ev} dominant eigenvectors V_{out} and the corresponding eigenvalues Λ_{out} .
Generate an initial set of m orthonormal vectors $V \in \mathbb{R}^{n \times n}$; $k = 0$;
PercentReached=0
repeat
    $k = k + 1$
    Compute Y such that $Y = A \cdot V$
    $V \leftarrow$ orthonormalisation of the columns of Y
    Rayleigh-Ritz projection applied on matrix A and orthonormal vectors V
    Convergence analysis step : save eigenpairs that have converged and update *PercentReached*
until (PercentReached > PercentTrace or $n_{ev} = m$ or $k > MaxIter$)

Question 7 : Here are the parts of the Matlab code that corresponds to each step :

step 3 :

```
Vr = randn(n, m);

Vr = mgs(Vr);
```

step 7 :

```
Y = A*Vr;
```

step 8 :

```
Vr = mgs(Y);
```

step 9 :

```
[Wr, Vr] = rayleigh_ritz_projection(A, Vr);
```

step 10 :

```
analyse_cvg_finie = 0;

        % nombre de vecteurs ayant convergé à cette itération
        nbc_k = 0;

        % nb_c est le dernier vecteur à avoir convergé à l'itération précédente
        i = nb_c + 1;

        while(~analyse_cvg_finie)
            % tous les vecteurs de notre sous-espace ont convergé
            % on a fini (sans avoir obtenu le pourcentage)
            if(i > m)

                analyse_cvg_finie = 1;

            else
                % est-ce que le vecteur i a convergé

                % calcul de la norme du résidu
                aux = A*Vr(:,i) - Wr(i)*Vr(:,i);
```

```
        res = sqrt(aux'*aux);

        if(res >= eps*normA)
            % le vecteur i n'a pas convergé,
            % on sait que les vecteurs suivants n'auront pas convergé non plus
            % => itération finie
            analyse_cvg_finie = 1;

        else
            % le_vecteur i a convergé
            % un de plus
            nbc_k = nbc_k + 1;
            % on le stocke ainsi que sa valeur propre
            W(i) = Wr(i);

            itv(i) = k;Extending the power method to compute dominant eigenspace vectors

            % on met à jour la somme des valeurs propres
            eigsum = eigsum + W(i);

            % si cette valeur propre permet d'atteindre le pourcentage
            % on a fini
            if(eigsum >= vtrace)

                analyse_cvg_finie = 1;

            else
                % on passe au vecteur suivant
                i = i + 1;

            end

        end

    end

end

% on met à jour le nombre de vecteurs ayant convergés
nb_c = nb_c + nbc_k;

% on a convergé dans l'un de ces deux cas
conv = (nb_c == m) | (eigsum >= vtrace);
```

5 Subspace iter v2 and Subspace iter v3: Towards an Efficient Solver

5.1 Block approach (subspace_iter_v2)

Question 8 : If we decide to compute A^p by simply computing each products one by one we get a total of $p-1$ times n^3 multiplication and $n^2(n-1)$ additions. therefore raising the total number of flops to compute A^p to $(p-1)\cdot(2n^3-n^2)$. finally to compute the product with the Matrix V which is a $n \times m$ Matrix we need $mn^2 + mn(n-1)$ flops. Therefore to compute $V \cdot A^p$ we need a total of $(p-1)\cdot(2n^3-n^2) + 2mn^2 - mn$ flops.

This is a quite expensive way of computing $V \cdot A^p$. We can greatly reduce the amount of flops by using the exponentiation, by squaring for example. But a simpler way of reorganizing the code to reduce the length of computing time is to simply compute A^p once just before the main loop and using that value for all iterations.

Question 10 : as p increases two things happen :

- The number of iterations decreases until a certain threshold of p is reached from which it starts increasing again.
- The quality of the eigenvector increases proportionally to p and is better than the v1 and v2 subspace iteration methods.

This is coherent, we are effectively doing p iterations of the program each time we do one iteration. this results in more precise eigenvectors in less iterations. but as p gets too large computing it at the start of the code becomes too long (reducing the effectiveness of the method again).

5.2 Deflation method (subspace iter v3)

Question 12 : The vectors with larger eigenvalues will be far more accurate. This is because the vectors with the largest eigenvalues converge faster. Therefore as the algorithm subspace_iter_v1 (and v2) is trying to reach the expected quality on the vectors with smaller eigenvalues it is still updating the vectors that are already considered "converged" and making them far more accurate than needed for the algorithm to finish.

Question 13 : With subspace_iter_v3 the the vectors of eigenvalues who are already considered "converged" are no longer updated. Therefore all the vectors will have the same quality as they are no longer unnecessarily updated. this will make the algorithm more efficient at the cost of a relative loss of accuracy.

6 Numerical Experiments

Question 14 : Each type of matrix is built around its eigenvalues, which are all first created in different ways :

- Type 1 :

$$\lambda_i = i \tag{2}$$

- Type 2 : these matrices have random eigenvalues of high conditioning.

$$\lambda_i = random(\frac{1}{cond}, 1),\ cond = 1e^{10} \tag{3}$$

- Type 3 : the matrixes of this type have an intermediate conditioning, with their logarithmic values evenly spread

$$\lambda_i = cond^{(\frac{i-1}{n-1})},\ cond = 1e^{5} \tag{4}$$

Their logarithmic spacing is :

$$\delta_i = \log(\lambda_{i+1}) - \log(\lambda_i) = \frac{-1}{n-1} \cdot \log(cond) \tag{5}$$

- Type 4 : the matrices of this type have a low conditioning

$$\lambda_i = 1 - (\frac{i-1}{n-1}) \cdot (1 - \frac{1}{cond}),\ cond = 1e^{2} \tag{6}$$

Their spacing is :

$$\delta_i = \lambda_{i+1} - \lambda_i = \frac{1}{n-1} \cdot (1 - \frac{1}{cond}). \tag{7}$$

In the next two figures we can see the distribution of the eigenvalues in each type of matrice, (200x200) sorted in descending order.

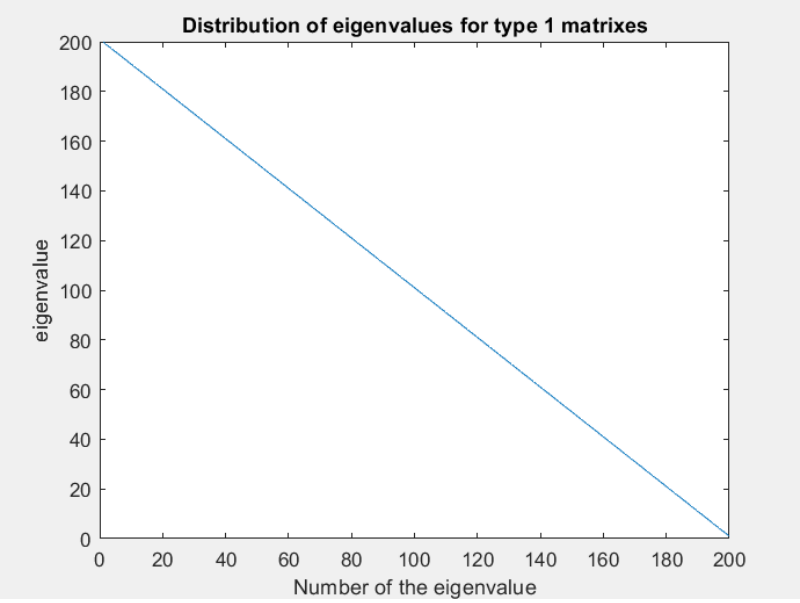


Figure 1: Eigenvalues of a type 1 matrix

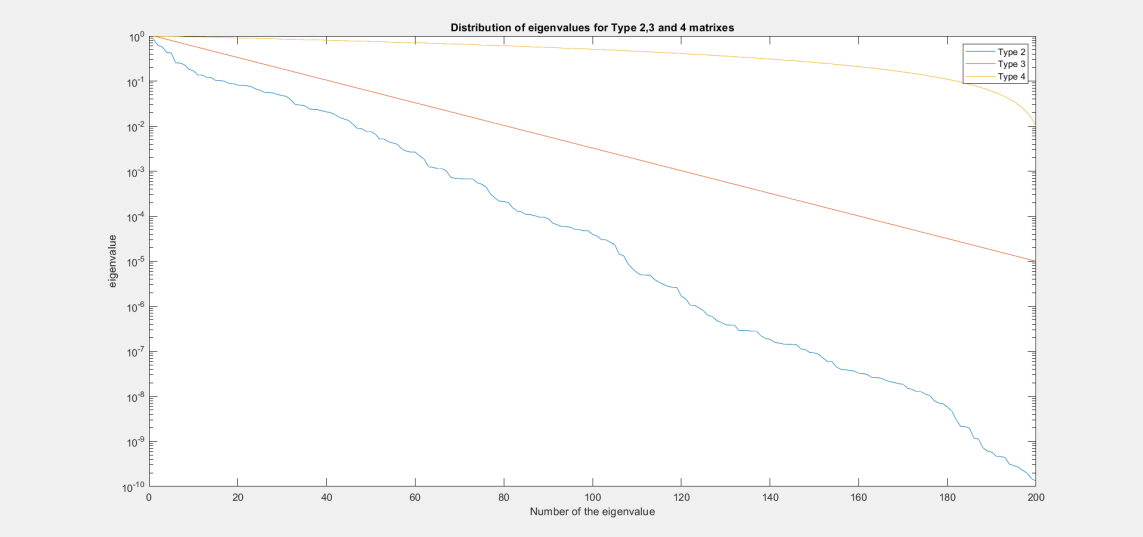


Figure 2: Eigenvalues of type 2,3 and 4 matrixcs on a log scale

The second graph clearly shows us how type 3 matrices have their logarithmic eigenvalues evenly spread.

7 Image Compression

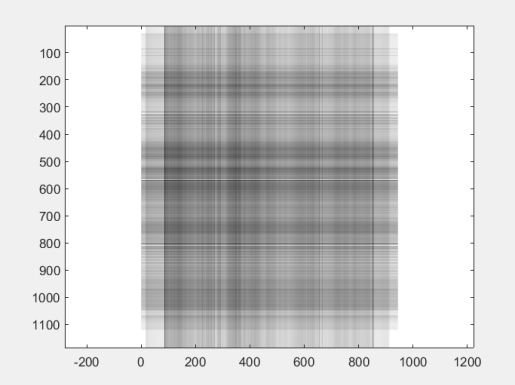
We have seen two algorithms with an increasing efficiency, the second one being based on the *Rayleigh quotient*. We will apply these algorithms to images, seen as rectangular matrices. This will rely on both the Singular Value Decomposition (SVD) and Best Low-Rank Approximation theorems.

An image can be represented as a matrix I of size $q \times p$. To compress said image, we are going to use its k-rank approximation, I_k

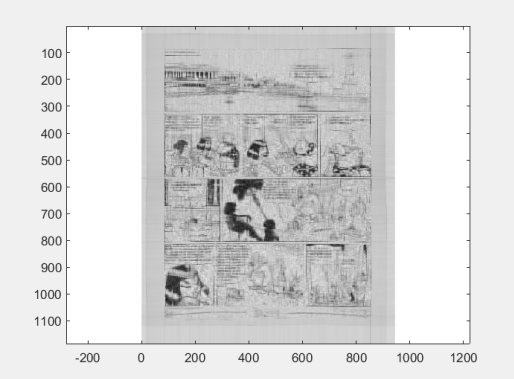
In order to generate the matrix Ik :

- We create a matrix $M = II^T$ (or $M = I^T I$ if $p < q$)
- We find the k eigenpairs of M
- We create the matrix Σ_k with the k eigenvalues
- We create the matrix U_k with the k eigenvectors (or V_k)
- We create the matrix V_k by using the relation between the vectors of U_k and those of V_k (or the matrix U_k with V_k)
- $I_k k = U_k \cdot \Sigma k \cdot V_k^T$

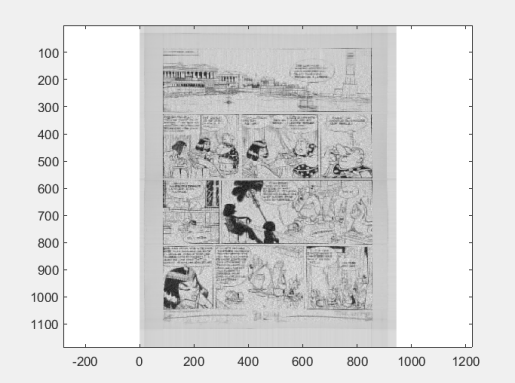
Question 1 : whether $p > q$ or $p < q$ we have
$U_k \in \mathbb{R}^{q \times k}$, $V_k \in \mathbb{R}^{p \times k}$ and $\Sigma_k \in \mathbb{R}^{k \times k}$



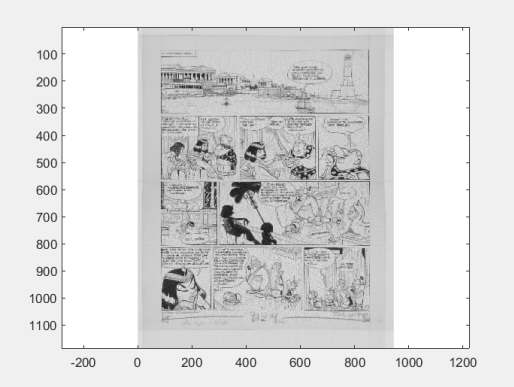
(a) k = 1



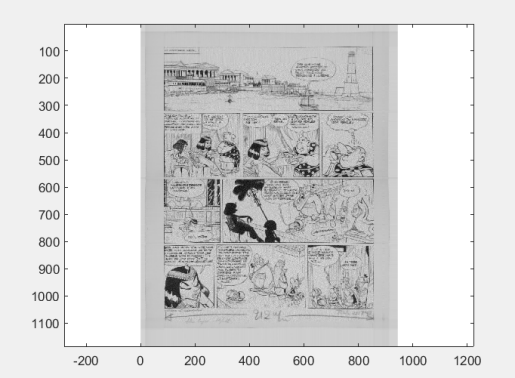
(b) k = 40



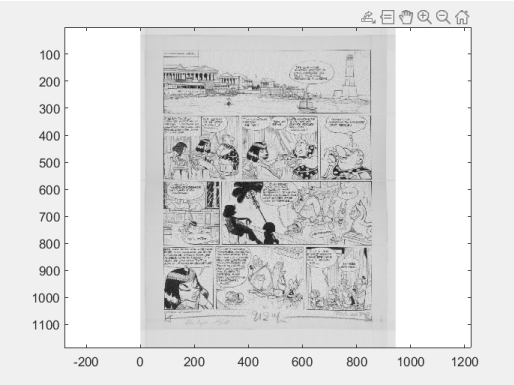
(c) k = 80



(d) k = 120



(e) k = 160



(f) k = 200

Figure 3: evolution of the reconstruction as the amount of eigenvectors increase

Question 2 : All methods of calculating the eigenvalues ends up yielding about the same RSME no matter what's the value of k. The efficiency increases does not lie in the quality of the resulting image (as all algorithm have the same quality criterias for stopping as each other). The improvement instead lies in computing time as clearly seen in the earlier parts of this report. As we can see in the images above at $k = 200$ the image is pretty much fully reconstructed while being quite compressed.