



Projet Calcul Scientifique

Auteur :

AZDAD Bilal
MISSOURI Taha

Département Sciences du Numérique - Première année
2022-2023

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 3 |
| 2 | Limitations of the Power Methods | 3 |
| 3 | Extending the Power Method to Compute Dominant Eigenspace Vectors | 4 |
| 3.1 | subspace_iter v0: a basic method to compute a dominant eigenspace | 4 |
| 3.1.1 | Orthonormalisation | 4 |
| 3.1.2 | Rayleigh quotient | 4 |
| 3.2 | subspace iter v1: improved version making use of Raleigh-Ritz projection | 5 |
| 3.2.1 | Convergence analysis step | 5 |
| 4 | subspace iter v2 and subspace iter v3: toward an efficient solver | 6 |
| 4.1 | Block approach (subspace iter v2) | 6 |
| 4.2 | Deflation method (subspace iter v3) | 8 |
| 5 | Numerical experiments | 8 |
| 6 | Image Compression | 12 |

1 Introduction

In this project, we will see that this specific algorithm is not efficient in terms of performance. Then we will present a more efficient method called subspace iteration method, based on an object called Rayleigh quotient. We will study four variants of this method.

2 Limitations of the Power Methods

Algorithm 1 méthode puissance itérée basique

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

Question 1 : Using the file `test_v11.m`, compare the running time of the function `power_v11` to compute a few eigenpairs with the running time of the function `eig` of Matlab that compute all the eigenpairs for different sizes and types of matrices.

la fonction `eig` est visiblement meilleur a calculé toutes le valeurs propres 300 fois plus rapidement que la fonction `power_v11`, qui plus est ne calcul que les premiers couples propres dans l'ordre décroissant des valeurs propres, voici quelques résultats :

| Type matrice | 1 | 2 | 3 | 4 |
|--------------------------|-----------|-----------|-----------|-----------|
| <code>eig</code> | 9.000e-02 | 7.000e-02 | 4.000e-02 | 6.000e-02 |
| Puissance itérée basique | 3.727e+01 | 4.400e-01 | 9.700e-01 | 3.550e+01 |

Table 1: Comparaison fonction `eig`/power method taille (matrice 200*200)

Question 2 : Looking closely at the proposed algorithm, two matrix \times vector products, $A.v$, are identified inside the loop. By rearranging the operations, propose an algorithm that will have only one matrix \times vector product in the loop

En réarangant la boucle comme suit

```

while(norme > eps && nb_it < maxit)
    v = z / norm(z,2);
    z = A*v;
    beta = v'*z;
    norme = norm(beta*v - z, 2)/norm(beta,2);
    nb_it = nb_it + 1;
end

```

Figure 1: réduction de la complexité de calcul

Il ne reste qu'un produit matrice*vecteur ce qui réduit la complexité de calcul et donc réduit le temps de résolution de power_v12 par rapport à power_v11

Après l'execution du test de comparaison il s'avère que power_v12 est deux fois plus rapide que la fonction pwer_v11, on peut l'observer sur le tableau suivant.

| Type matrice | 1 | 2 | 3 | 4 |
|----------------------------|-----------|-----------|-----------|-----------|
| Puissance itérée améliorée | 1.843e+01 | 3.200e-01 | 5.400e-01 | 1.835e+01 |
| Puissance itérée basique | 3.515e+01 | 4.300e-01 | 8.700e-01 | 3.542e+01 |

Table 2: Comparaison power method basique /power method taille améliorée (matrice 200*200)

Question 3 : What do you think to be the main drawback of the deflated power method in terms of computing time ?

Le principal inconvénient de la méthode de puissance défléchie est le fait qu'elle calcule toute la décomposition spectrale de la matrice A, qui est une matrice carrée $n \times n$; un processus qui s'avère exceptionnellement chronophage.

3 Extending the Power Method to Compute Dominant Eigenspace Vectors

3.1 subspace_iter v0: a basic method to compute a dominant eigenspace

3.1.1 Orthonormalisation

Question 4 : Towards which matrix does V converge the power method algorithm if we apply it to a set of m vectors ?

L'algorithme de la puissance iterée basique s'applique sur un vecteur quelconque de R^n , on obtient en résultat un vecteur associé à la valeur propre la plus grande en valeur absolue, changer ce vecteur v par une matrice V de R^{n*m} , va simplement résulter en l'application de la power méthode pour chaque colonne de la matrice V indépendamment ce qui va entraîner une matrice V dont les colonnes vont converger vers un vecteur propre (pas forcément le même, dépend de la colonne initiale de V) associé à la valeur propre la plus grande en valeur absolue.

3.1.2 Rayleigh quotient

Question 5 :We are looking at variants of the power method in order to avoid computing the whole spectral decomposition of the matrix A. But in Algorithm 2, a computation of the whole spectral decomposition of the matrix H is performed. Explain why it is not a problem to compute the whole spectral decomposition of H by investigating its dimensions ?

La matrice et de dimensions $m \times m$ et comme le nombre des valeurs propres qu'on a besoin pour atteindre le pourcentage est m , calculé la décomposition spectrale de H n'est pas un problème si on considère le calcul de la décomposition spectrale de A qui est de taille $n > m$

Question 6 : In the file subspace_iter_v0.m, fill in the function to get Algorithm 2

```
while (~conv && k < maxit)
    k = k + 1;
    % calcul de Y = A.V
    Y=A*V;
    % calcul de H, le quotient de Rayleigh H = V^T.A.V
    H = V'*A*V;
    % vérification de la convergence
    acc=norm(Y-V*H,'fro')/normA;
    conv=(acc<=eps);
    % orthonormalisation
    V = mgs(Y);
```

Figure 2: implatation sans la projection de Rayliegh

```
while (~conv && k < maxit)
    k = k+1;
    %% Y <- A*V
    Y = A*Vr;
    %% orthogonalisation
    Vr = mgs(Y);
    %% Projection de Rayleigh-Ritz
    |[Wr, Vr] = rayleigh_ritz_projection(A, Vr);
```

Figure 3: implatation avec la projection de Rayliegh

3.2 subspace iter v1: improved version making use of Raleigh-Ritz projection

3.2.1 Convergence analysis step

Question 7: In the file subspace_iter_v1.m, identify all the steps of Algorithm 4

Algorithm 4 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 $V \in \mathbb{R}^{n \times m}$; $k = 0$; $PercentReached = 0$ [Ligne 47-49](#)

repeat [Ligne 52](#)

$k = k + 1$ [Ligne 54](#)

 Compute Y such that $Y = A \cdot V$ [Ligne 56](#)

$V \leftarrow$ orthonormalisation of the columns of Y [Ligne 58](#)

Rayleigh-Ritz projection applied on matrix A and orthonormal vectors V [Ligne 61](#)

Convergence analysis step: save eigenpairs that have converged and update $PercentReached$ [Ligne 70-115](#)

until ($PercentReached > PercentTrace$ or $n_{ev} = m$ or $k > MaxIter$) [Ligne 52-73](#)

Figure 4: Identification algorithme 4/ implantation matlab

4 subspace iter v2 and subspace iter v3: toward an efficient solver

4.1 Block approach (subspace iter v2)

Question 8: what is the cost in term of flops of the computation of Ap , then $Ap \cdot V$? How organize differently this computation to reduce this cost?

le cout du calcul de Ap est $p * n^3$ soit $O(p * n^3)$ et le cout pour calculer $Ap * v$ est $p * n^3 + 2 * n^2 * m$ soit aussi $O(p * n^3)$

Pour réduire le coût de ce calcul, Nous pouvons calculer Ap en élévant A au carré et à ses puissances successivement, ce qui peut être réalisé en utilisant $O(\log(p))$ multiplications de matrices. Plus précisément, nous pouvons calculer $A^{(2^k)}$ pour $k = 0, 1, \dots, \log_2(p)$ pour les k qui vérifie le k -ième bit de p en binaire est égale à 1 , puis multiplier les matrices résultantes pour obtenir Ap . Cela réduit le coût de calcul de Ap à $O(n^3 \times \log(\log(p)))$, ce qui est bien inférieur au coût de calcul de Ap directement.

Une fois que nous avons calculé Ap , nous pouvons alors le multiplier par V en utilisant la multiplication matrice-vecteur, qui a un coût de $O(n^2)$. Par conséquent, le coût total de calcul de $Ap \times V$ en utilisant l'exponentiation rapide est $O(n^3 \times \log(\log(p)) + n^2)$.

Question 9: Modify the file `subspace_iter_v2.m` to implement this acceleration (note that the initial code of this subprogram is the v1 version of the method, the only difference is the input p).

```
% Calcule A^p·V en utilisant "the method of repeated squaring"

Ap = A; %

% Calcule A^(2^k) pour j=0,1,...,log2(p)
for j = 0:log2(p)
    if bitget(p, j+1) % vérifie si j-eme bit de p est 1
        Ap = Ap * A^(2^j);
    end
end
```

Figure 5: Identification algorithme 4/ implantation matlab

Question 10: Observe the behaviour of this approach when increasing p.

l'élévation d'une matrice A à une puissance élevée p aide à mettre en évidence la valeur propre dominante et le vecteur propre correspondant de A . La valeur propre dominante a la plus grande magnitude et le vecteur propre correspondant a le plus d'influence sur le produit résultant $A^p \times v$ lorsque p devient grand.

```
***** calcul avec subspace iteration v2 *****

Temps subspace iteration v2 = 1.080e+00
Nombre d'itérations : 58
Nombre de valeurs propres pour attendre le pourcentage = 46
Qualité des couples propres (par rapport au critère d'arrêt) = [2.112e-15 , 9.579e-08]
Qualité des valeurs propres (par rapport au spectre de la matrice) = [8.745e-16 , 3.21
Pour p = 7
Matrice 200 x 200 - type 1

***** calcul avec subspace iteration v2 *****

Temps subspace iteration v2 = 8.000e-01
Nombre d'itérations : 51
Nombre de valeurs propres pour attendre le pourcentage = 46
Qualité des couples propres (par rapport au critère d'arrêt) = [9.621e-16 , 9.050e-08]
Qualité des valeurs propres (par rapport au spectre de la matrice) = [3.056e-16 , 2.92
Pour p = 8
Matrice 200 x 200 - type 1

***** calcul avec subspace iteration v2 *****

Temps subspace iteration v2 = 6.300e-01
Nombre d'itérations : 45
Nombre de valeurs propres pour attendre le pourcentage = 46
Qualité des couples propres (par rapport au critère d'arrêt) = [1.630e-15 , 9.855e-08]
Qualité des valeurs propres (par rapport au spectre de la matrice) = [4.685e-16 , 3.46
Pour p = 9
Matrice 200 x 200 - type 1
```

Figure 6: réduction de la complexité de calcul

À mesure que p augmente, les contributions des plus petites valeurs propres et vecteurs propres diminuent rapidement, ce qui signifie que la méthode de l'itération de la puissance peut converger plus rapidement et avec plus de précision vers la paire propre dominante. Cela est dû au fait que la contribution de la paire propre dominante à $A^p \times v$ domine sur les autres paires propres, rendant l'algorithme plus rapide à converger vers cette paire propre lorsque p augmente. Cependant, l'élévation de A à une puissance élevée peut également entraîner une instabilité numérique, en

particulier si A est mal conditionnée, il faut donc prendre des précautions lors de la sélection de la valeur de p.

4.2 Deflation method (subspace iter v3)

Question 11: One result of the functions that compute the eigenpairs is the quality of each eigenpairs (qv) that is used as a convergence criteria, $\|Av - \beta v\| / |\beta|$, Explain why, with subspace iter v1 method, this accuracy differs for some of the vectors.

Comme la méthode subspace iter_v1_method affine le sous-espace de manière itérative, la précision des paires propres calculées peut varier pour différents vecteurs dans le sous-espace. Certains vecteurs peuvent converger rapidement vers des paires propres de haute qualité, tandis que d'autres peuvent converger plus lentement ou vers des paires propres de qualité inférieure. Cela est dû au fait que la projection de A sur le sous-espace peut affecter différents vecteurs de manière différente, en fonction de leur relation avec l'espace propre approximé. Par conséquent, le comportement de convergence et la précision des paires propres calculées peuvent varier en fonction du choix du sous-espace initial et de la méthode de projection utilisée.

Question 12: Try to anticipate what will occur, in term of accuracy of the eigenpairs, with the sub- space iter v3 method

La méthode du subspace_iter_v3 tire parti du fait que les colonnes de V convergent dans l'ordre, nous pouvons donc figer les colonnes convergées de V pour économiser des coûts de calcul. Plus précisément, nous figeons les premières nbc colonnes de V qui ont convergé et ne mettons à jour que les colonnes restantes de V. Cela permet de réaliser des économies importantes dans les étapes de produit matrice-vecteur, d'orthonormalisation et de projection Rayleigh-Ritz. Par conséquent, nous pouvons anticiper que la méthode de l'itération de sous-espace v3 convergera plus rapidement et avec plus de précision que la méthode de l'itération de sous-espace v1, car elle utilise une approche plus efficace pour calculer les paires propres. De plus, puisque nous ne mettons à jour que les colonnes de V qui n'ont pas convergé dans la méthode de l'itération de sous-espace v3, nous pouvons nous attendre à ce que la précision des paires propres calculées soit plus élevée pour ces vecteurs par rapport à la méthode de l'itération de sous-espace v1.

Copy the file subspace_iter_v2.m into a file subspace_iter_v3.m to implement this defla- tion.

5 Numerical experiments

Question 14: What are the differences between the 4 types of matrices ? Create some figures that show the eigenvalue distribution of these different types (the matrix and its spectrum are store in a file when you create the matrix).

les différences sont dans les titres des figures

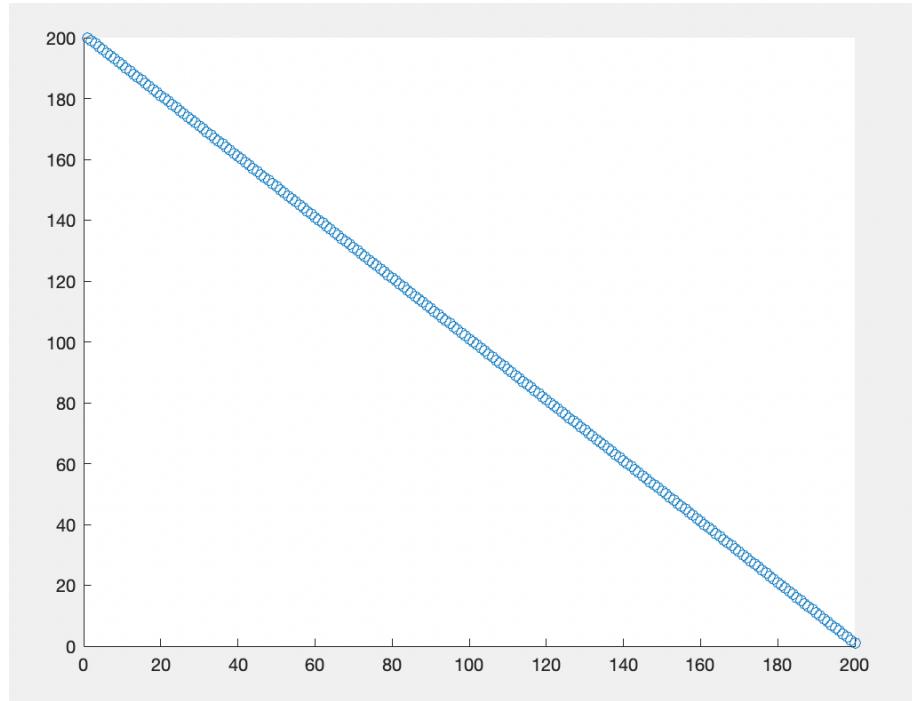


Figure 7: A matrice avec des valeurs proportionnelles à 1, 2, ..., n.

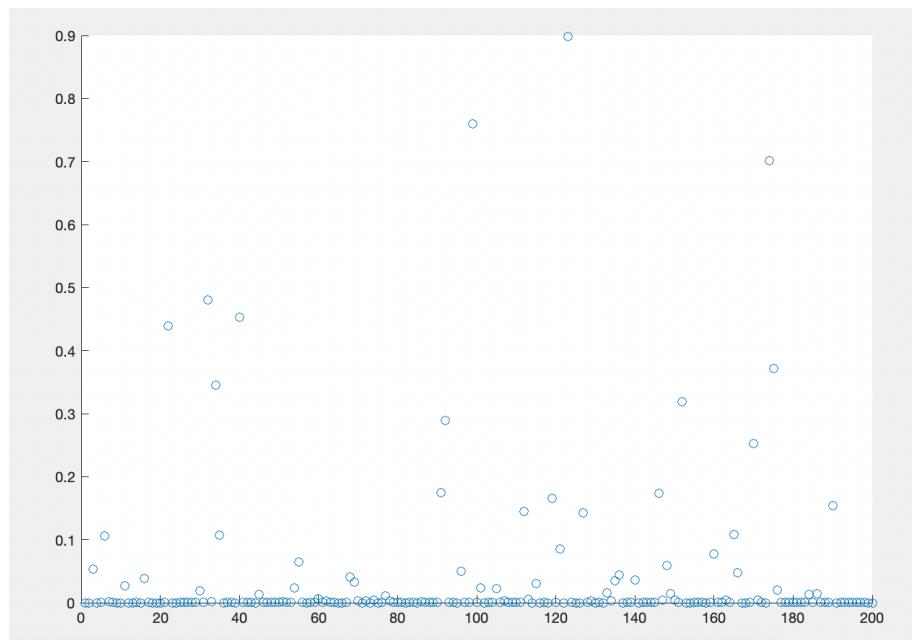


Figure 8: A une matrice avec des valeurs propres décroissantes de manière exponentielle

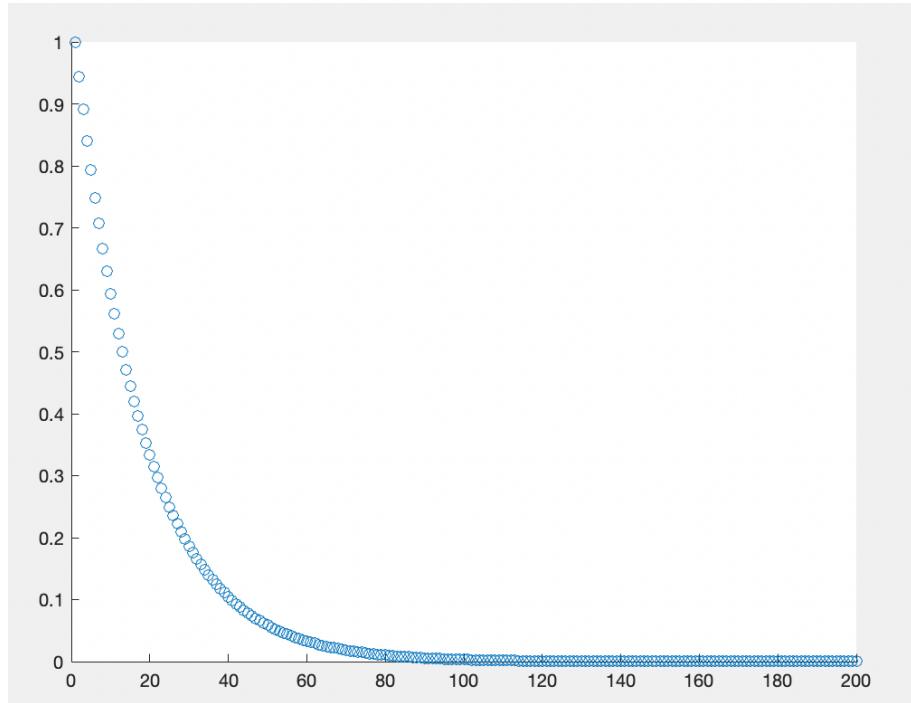


Figure 9: A une matrice avec des valeurs propres décroissantes de manière géométrique

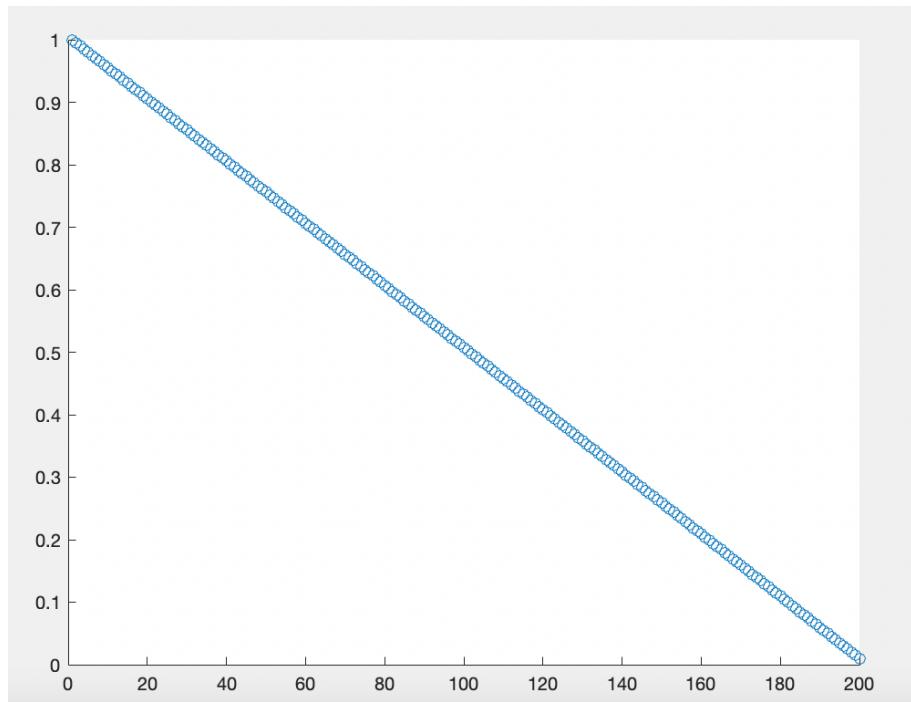


Figure 10: A Une matrice avec des valeurs propres qui sont une combinaison linéaire d'une constante et d'une fonction linéaire

Question 15: Compare the performances of the algorithms you have implemented and those provided (including eig) for different types and sizes of matrices.

| Type matrice | 1 | 2 | 3 | 4 |
|--------------|-----------|-----------|-----------|-----------|
| v0 | 1.209e+01 | 5.074e+01 | 1.570e+00 | 1.237e+01 |
| v1 | 4.970e+00 | 1.400e-01 | 2.500e-01 | 5.170e+00 |
| v2 | 7.300e-01 | 1.600e-01 | 1.500e-01 | 8.300e-01 |
| v3 | 5.600e-01 | 1.900e-01 | 1.900e-01 | 5.600e-01 |

Table 3: Comparaison méthodes en temps(matrice 200*200)

| Type matrice | 1 | 2 | 3 | 4 |
|--------------|-------------------------|-------------------------|-------------------------|-------------------------|
| v0 | [0.00e+00 , 1.00e+00] | [1.23e-16 , 1.00e+00] | [6.66e-16 , 1.00e+00] | [6.66e-16 , 1.00e+00] |
| v1 | [1.04e-15 , 3.83e-13] | [4.38e-16 , 1.50e-15] | [0.00e+00 , 4.11e-15] | [1.67e-15 , 3.97e-13] |
| v2 | [3.05e-16 , 2.92e-13] | [2.47e-16 , 2.69e-15] | [2.22e-16 , 5.48e-15] | [6.37e-16 , 3.34e-13] |
| v3 | [1.633e-16 , 2.870e-13] | [2.470e-16 , 2.691e-15] | [2.220e-16 , 5.486e-15] | [3.561e-16 , 3.324e-13] |

Table 4: Comparaison méthodes en précision (matrice 200*200)

| Type matrice | 1 | 2 | 3 | 4 |
|--------------|-----------|-----------|-----------|-----------|
| v0 | 5.000e-02 | 9.000e-02 | 1.600e-01 | 5.800e-01 |
| v1 | 6.000e-02 | 6.000e-02 | 7.000e-02 | 1.600e-01 |
| v2 | 1.100e-01 | 1.500e-01 | 1.500e-01 | 9.000e-02 |
| v3 | 1.300e-01 | 1.300e-01 | 1.700e-01 | 1.100e-01 |

Table 5: Comparaison méthodes en temps (matrice 50*50)

| Type matrice | 1 | 2 | 3 | 4 |
|--------------|-----------------------|-----------------------|-----------------------|-----------------------|
| v0 | [1.56e-15 , 1.00e+00] | [2.31e-16 , 1.00e+00] | [1.44e-15 , 1.00e+00] | [4.44e-16 , 1.00e+00] |
| v1 | [0.00e+00 , 4.08e-15] | [5.77e-16 , 1.71e-15] | [0.00e+00 , 3.02e-15] | [0.00e+00 , 2.59e-15] |
| v2 | [6.76e-16 , 2.91e-15] | [0.00e+00 , 2.45e-16] | [0.00e+00 , 1.59e-15] | [1.13e-16 , 2.53e-15] |
| v3 | [6.76e-16 , 2.91e-15] | [0.00e+00 , 2.45e-16] | [0.00e+00 , 1.59e-15] | [1.13e-16 , 2.53e-15] |

Table 6: Comparaison méthodes en précision (matrice 50*50)

Pour les matrices de type 1, la méthode subspace_iter_v2 offre la meilleure combinaison de temps et de précision, bien que la version v3 soit légèrement plus rapide. La version v2 est plus précise pour les derniers couples propres. Pour les matrices de type 2, la méthode subspace_iter_v3 est la meilleure en termes de temps et de précision, la version v1 étant trop longue et la version v2 manquant de précision. Pour les matrices de type 3, la méthode subspace_iter_v1 est la meilleure en termes de temps et de précision, la version v2 étant trop longue. Pour les matrices de type 4, la méthode subspace_iter_v2 est la meilleure en termes de temps et de précision pour les matrices de grande taille, mais la méthode subspace_iter_v1 est préférable pour les matrices de petite taille car elle est plus rapide. La version v3 n'est pas assez précise pour le temps qu'elle prend et ne doit donc pas être utilisée.

6 Image Compression

Question 1: what are the size of the elements the triplet

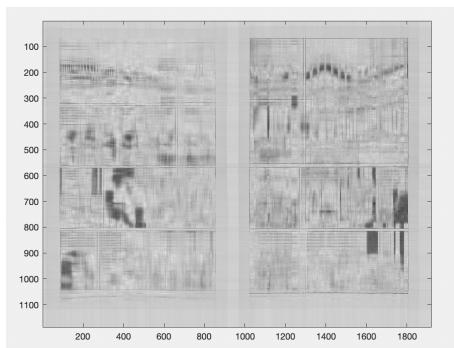
$$(\Sigma_k, U_k, V_k) ?$$

- Le triplet (Σ_k, U_k, V_k) est composé de Σ_k , une matrice diagonale de taille $k \times k$, U_k , une matrice de taille $q \times k$ contenant les vecteurs singuliers gauches correspondants de I , et V_k , une matrice de taille $p \times k$ contenant les vecteurs singuliers droits correspondants de I . Ainsi, le nombre total d'éléments dans le triplet est $k(2q+p)$, ce qui est inférieur à $q \times p$, le nombre total d'éléments dans l'image originale I , lorsque k est beaucoup plus petit que q et p . C'est le principe de la compression.

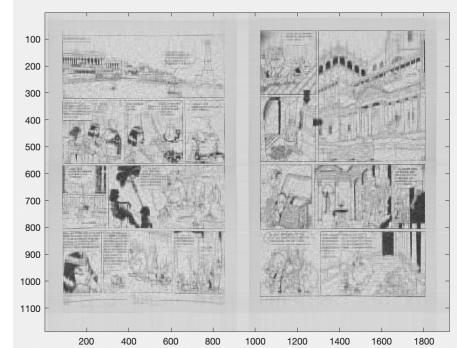
Question 2: .

eps=10⁻⁸ ; maxiter=100000 ;
searchspace=400 ; pourcentage=0.995

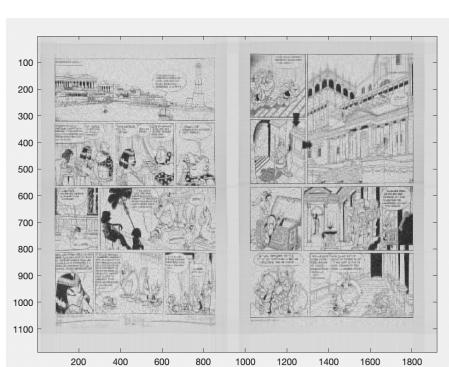
utilisation de la fonction SVD



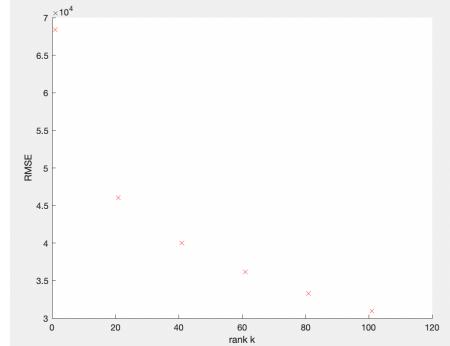
(a) itération 2 de iter



(b) itération 4



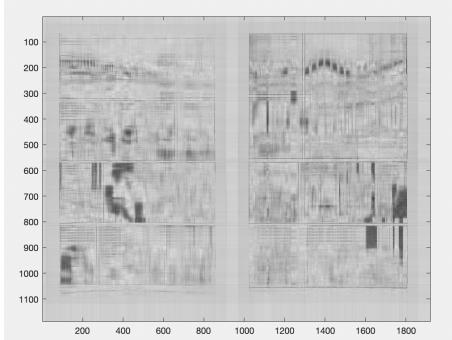
(a) itération 6



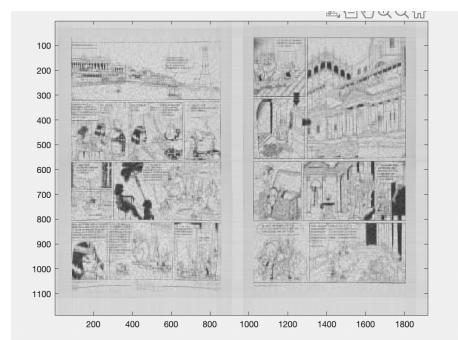
(b) Différence entre l'image réelle et l'image reconstruite

Figure 12: Utilisation de la fonction SVD

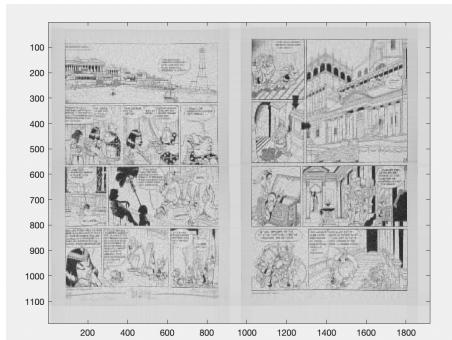
utilisation de la fonction eig



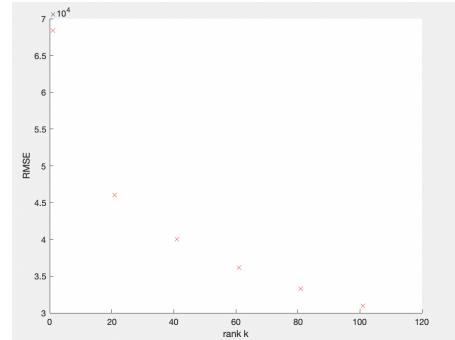
(a) itération 2



(b) itération 4



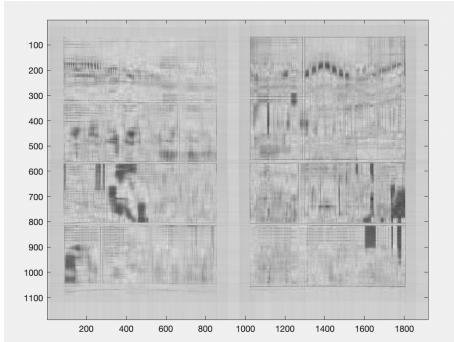
(a) itération 6



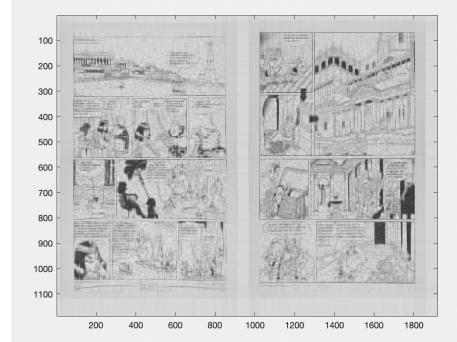
(b) Différence entre l'image réelle et l'image reconstruite

Figure 14: Utilisation de la fonction eig

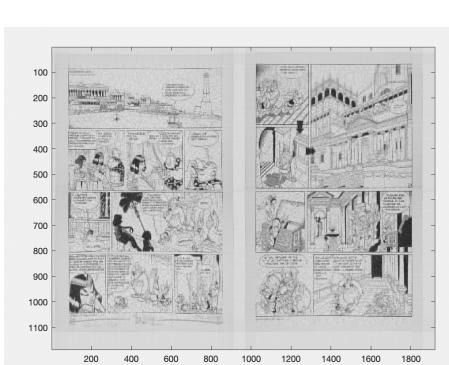
utilisation de la fonction power_methodV12



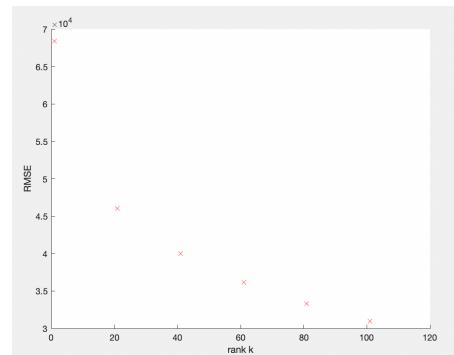
(a) itération 2



(b) itération 4



(a) itération 6



(b) Différence entre l'image réelle et l'image reconstruite

Figure 16: Utilisation de la fonction SVD

utilisation de la fonction subspace_iteration_V1

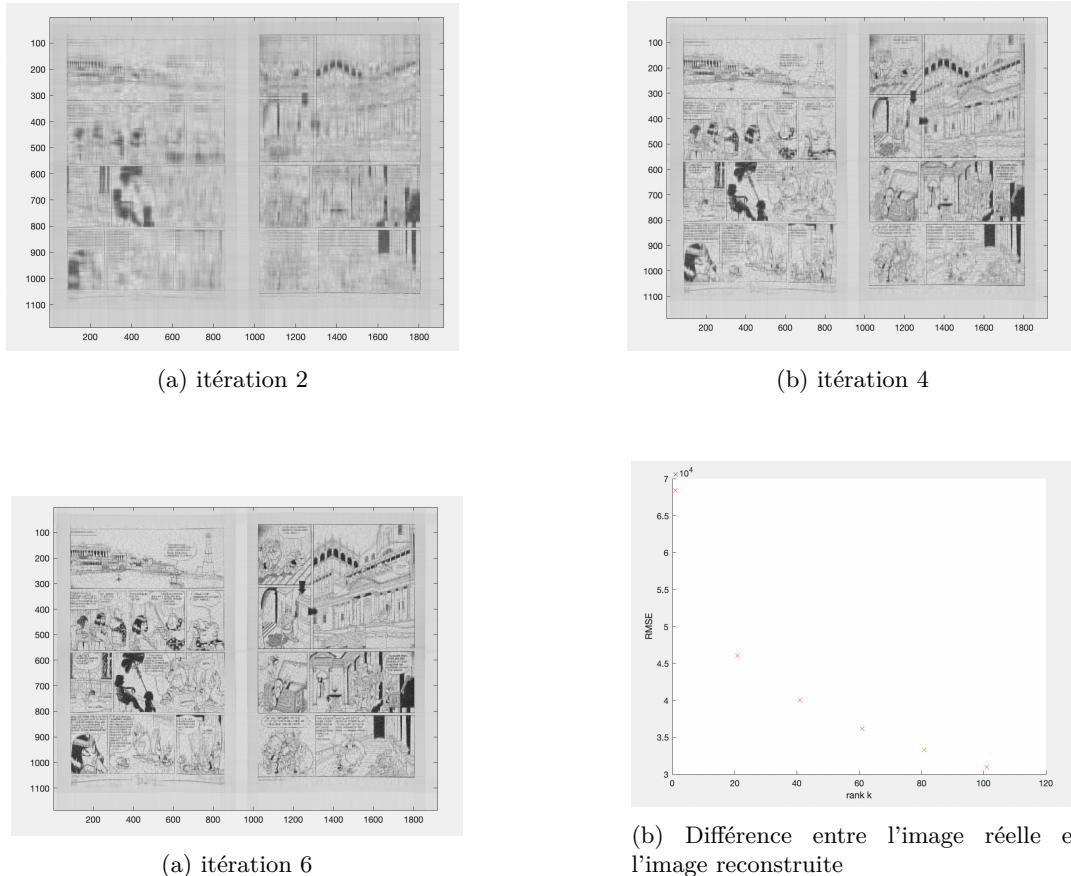
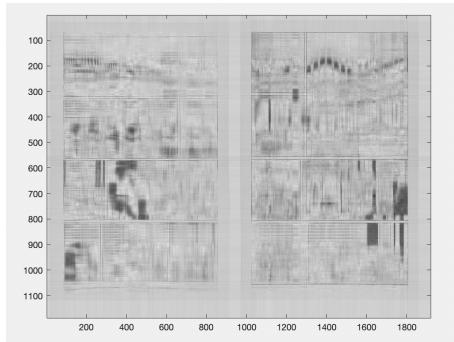
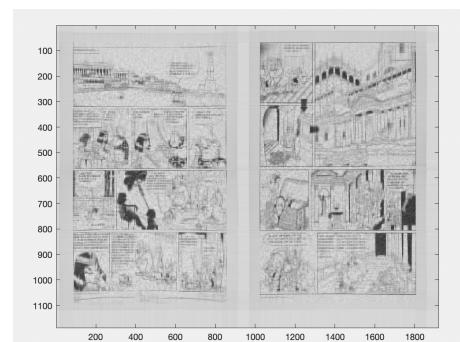


Figure 18: Utilisation de la fonction subspace_iter_v1

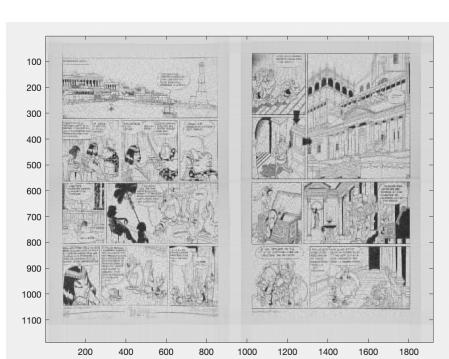
utilisation de la fonction subspace_iteration_V2



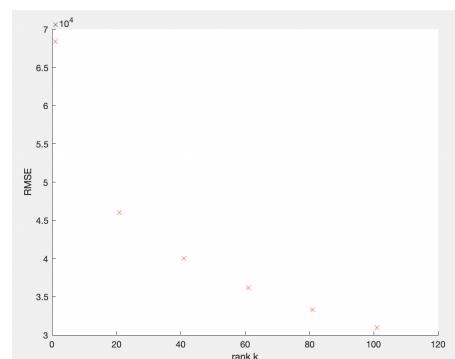
(a) itération 2



(b) itération 4



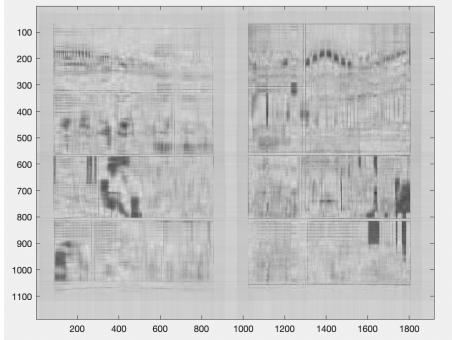
(a) itération 6



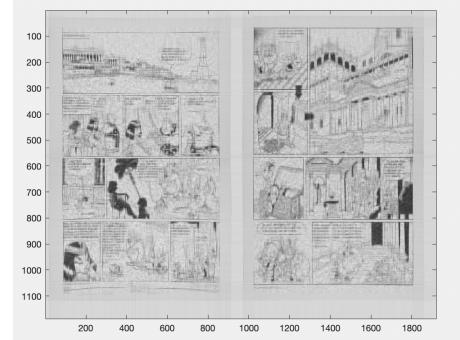
(b) Différence entre l'image réelle et l'image reconstruite

Figure 20: Utilisation de la fonction subspace_iter_v3

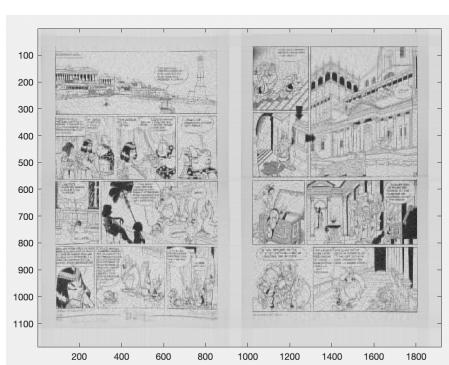
utilisation de la fonction subspace_iteration_V3



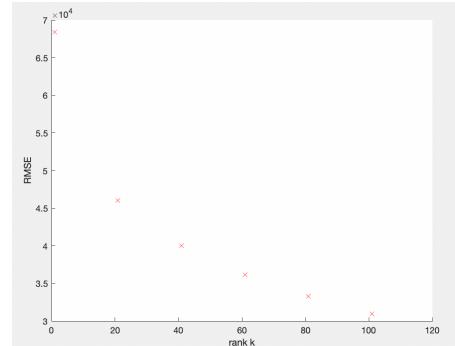
(a) itération 2



(b) itération 4



(a) itération 6



(b) Différence entre l'image réelle et l'image reconstruite

Figure 22: Utilisation de la fonction subspace_iter_3

La principale différence dans notre exemple ici est le temps d'exécution, cela est dû au fait que les paramètres utilisés n'étaient pas assez sélectifs. La méthode SVD a été la plus rapide, suivie de la méthode eig, puis les méthodes subspace_iter_v3 et subspace_iter_v2 sont à comparable en rapidité, suivies de subspace_iter_v1. Enfin, la méthode power_method_v12 a pris une quantité beaucoup de temps pour s'exécuter.