Rapport de projet Calcul Scientifique

Subspace Iteration Methods Application to Image Compression

Myriam ROBBANA & Ikram BERROUG

Groupe EF-04

Avril 2024

Contents

Partie I

1 Limitations of the power method

La "basic power method" est rappelée dans l'Algorithme 1. Elle est utilisée pour déterminer le vecteur propre associé à la plus grande (en module) valeur propre.

Algorithm 1 Vector power method

Input: Matrix $A \in R^{n \times n}$
Output: (λ_1, v_1) eigenpair associated to the largest (in module) eigenvalue.
$v \in R^n$ 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 :

Type de la matrice	n (taille de la matric)	eig method	power method v11
imat 1	50	0.05	0.297
	200	0.05	1.75
	1000	0.31	convergence non atteinte
imat 2	50	0	0.01
	200	0.01	0.02
	1000	0.02	3.75
imat 3	50	0	0
	200	0	0.08
	1000	0.28	21.14
imat 4	50	0	0.01
	200	0.03	1.57
	1000	0.27	convergence non atteinte

Table 1: Comparaison des temps de calcul entre eig et powerv11

On remarque qu'avec eig de Matlab, le temps augmente certes au fur et à mesure que l'on augmente la taille de la matrice, mais cette variation est très minime. Cependant, le temps augmente de manière beaucoup plus importante avec power$_v$11, $lorsqu'onaugmentelatailledelamatrice$.

Question 2 : Voici la portion de code du fichier powerv12 permettant de réorganiser les opérations pour n'avoir qu'un seul produit matrice × vecteur dans la boucle

```
31      ......
32      % somme des valeurs propres
33      eig_sum = 0.0;
34
35      % indicateur de la convergence (pourcentage atteint)
36      convg = 0;
37
38      % numéro du couple propre courant
39      k = 0;
40      % méthode de la puissance itérée
41      v = randn(n,1);
42      z = A*v;
43      beta = v'*z;
44
45      while (~convg && k < m)
46          k = k + 1;
47
48          % conv = || beta * v - A*v||/|beta| < eps
49          % voir section 2.1.2 du sujet
50          norme = norm(beta*v - z, 2)/norm(beta,2);
51          nb_it = 1;
52
53          while(norme > eps && nb_it < maxit)
54            v = z / norm(z,2);
55            z = A*v;
56            beta = v'*z;
57            norme = norm(beta*v - z, 2)/norm(beta,2);
58            nb_it = nb_it + 1;
59          end
60      ......
```

Le tableau ci-dessous réunit les résultats des tests pour évaluer powerv11 et powerv12

Type de la matrice	n (taille de la matrice)	power method v11	power method v12
imat 1	50	0.297	0.01
	200	1.75	0.15
	1000	convergence non atteinte	convergence non atteinte
imat 2	50	0.01	0.01
	200	0.02	0.02
	1000	3.75	0.99
imat 3	50	0	0.01
	200	0.08	0.02
	1000	21.14	1.11
imat 4	50	0.01	0.01
	200	1.57	0.12
	1000	convergence non atteinte	convergence non atteinte

Table 2: Comparaison des temps de calcul entre powerv11 et powerV12

On remarque grace à ce tableau, que l'algorithme de la puissance itéré est plus efficace pour powerv12 que pour powerv11. En effet, on voit bien que les temps obtenus pour les différentes tailles de matrices et les differents types de matrices, sont inférieurs dans powerv12 que dans powerv11. De plus, on remarque comme précedemment, que plus la taille de la matrice est grande et plus le temps

de calcul des vecteurs/valeurs propres augmente. D'ailleurs, on remarque même que pour des tailles trop importantes de matrice (1000), la convergence n'est pas atteinte à temps.

Question 3 : Afin d'évaluer l'influence de la déflation dans l'algorithme de la puissance itéré, nous avons réaliser deux jeux de tests, le premier sans la déflation (En commentant la ligne : $A = A - \beta \cdot (v \cdot v')$), et le second avec la déflation (en décommentant la ligne précedente). On obtient donc le tableau suivant pour imat=1 :

	Sans déflation	Avec déflation
Temps puissance itéré	1.142e+01	1.737e+01
Nombre de valeurs propres	26	26
Qualité des couples propres	[9.980e-09 , 1.000e-08]	[9.996e-09 , 1.644e-08]
Qualité des valeurs propres	[5.025e-14 , 5.263e-02]	[0.000e+00 , 5.025e-14]

Table 3: Comparaison des résultats avec et sans déflation

Les temps de calculs plus ou moins long dans les méthodes de puissances itératives proviennent du fait que l'on réalise de nombreuses opérations matricielles, ce qui est assez couteux en temps mais également en mémoire !

Cependant, après avoir comparé l'algorithme **avec** et **sans** la déflation, on remarque que celle-ci n'influe pas réellement sur la précision des vecteurs propres, mais plutôt sur celle de valeurs propres !

Ceci est du au fait qu'avec la méthode de déflation, le calcul d'une valeur propre nécessite à chaque fois l'utilisation des valeurs propres précédentes qui comportent déjà une erreur de calculs. Ainsi en modifiant A à tour de boucle, on diminue les erreurs au fur et à mesure ce qui implique une meilleure qualité.

En revanche, le temps observé **avec** l'algorithme de déflation est plus grand que celui **sans** déflation car il y a une étape supplémentaire qui fait la différence : $A = A - \beta \cdot (v \cdot v')$;

2 Extending the power method to compute dominant eigenspace vectors

2.1 subspace iter v0: a basic method to compute a dominant eigenspace

2.1.1 Orthonormalisation

Question 4 : La différence entre la power method et la subspace iteration méthod v0 est la matrice A prise en entrée. En effet, dans subspace iteration method v0, on suppose que la matrice $A \in R^{n \times n}$ est symétrique à coefficients réels. Ainsi, d'après le théorème spectrale, A est diagonalisable et donc on peut orthonormaliser les vecteurs propres. Cependant, ce n'est pas le cas dans la power method car on ne suppose pas que la matrice A est symétrique ce qui implique qu'on n'aura pas forcément une base.

2.1.2 Stopping criterion

2.1.3 Rayleigh quotient

Question 5 : La matrice H est définie par :

$$H = V^T \cdot A \cdot V$$

Puisque $\dim(V) = n \times m$ et $\dim(A) = n \times n$, la dimension de H est $\dim(H) = m \times m$. Ainsi, il est plus avantageux de calculer la décomposition spectrale de H plutôt que celle de A, étant donné que m est considérablement plus petit que n. Cela se traduit par moins de calculs nécessaires.

Question 6 : Le code suivant correspond à l'implémentation de l'Algorithme 2 dans le fichier subspace-iter-v0.m :

```
function [ V, D, it, flag ] = subspace_iter_v0( A, m, eps, maxit )

    % calcul de la norme de A (pour le critère de convergence)
    normA = norm(A, 'fro');

    n = size(A,1);

    % indicateur de la convergence
    conv = 0;
    % numéro de l'itération courante
    k = 0;

    % on génère un ensemble initial de m vecteurs orthogonaux

    V = randn(n, m);
    V = mgs(V);
    % rappel : conv = invariance du sous-espace V : ||AV - VH||/||A|| <= eps
    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' * Y;

        % vérification de la convergence
        acc = norm(A*V - V*H)/norm(A);
        conv = acc <= eps;

        % orthonormalisation
        V = mgs(Y);

    end

    % décomposition spectrale de H, le quotient de Rayleigh
    [X,LambdaOut] = eig(H);

    % on range les valeurs propres dans l'ordre décroissant
    [W, idx] = sort(diag(LambdaOut), 'descend');

    % on permute les vecteurs propres en conséquence
    X_sorted = X(:, idx);

    % les m vecteurs propres dominants de A sont calculés à partir de ceux de H
    V = V * X_sorted;
    D = diag(W);

    it = k;

    if (conv)
      flag = 0;
    else
      flag = -3;
    end
```

```
57
58 end
```

Pour les résultats du test-v0v1, consultez la question 13 où tous les résultats des tests sont regroupés.

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

2.2.1 First improvements

2.2.2 Convergence analysis step

Question 7 : Pour chaque étape de l'algorithme ci-dessous, les lignes de code correspondantes dans le fichier subspace-iter-v1.m sont affichées à droite de manière **bleu gras**.

Algorithm 2 Subspace iteration method v1 with Raleigh-Ritz projection

Input: Symmetric matrix $A \in R^{n \times n}$, 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 $V \in R^{n \times m}$ **(lignes 48 et 49)**; $k = 0$ **(ligne 38)**; $PercentReached = 0$ **(ligne 40)**
repeat
  $k = k + 1$ **(ligne 54)**
  Compute Y such that $Y = A \cdot V$ **(ligne 56)**
  $V \longleftarrow$ 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 64 à 115)**
until ($PercentReached > PercentTrace$ or $n_{ev} = m$ or $k > MaxIter$) **(lignes 52 et 115)**

Remarque : L'amélioration de la subspace iter v0 à la subspace iter v1, qui repose sur l'utilisation de la Raleigh-Ritz projection, semble être significative en termes de performances. Pour une matrice A de taille 200×200, la méthode v1 nécessite un temps de calcul considérablement réduit par rapport à la méthode v0, tout en effectuant nettement moins d'itérations (10 fois moins d'itérations). Cette différence illustre l'efficacité de la méthode v1.
Exemple de comparaison des performances :

- subspace iter v0 : Temps de calcul = 26.90, Nombre d'itérations = 2341
- subspace iter v1 : Temps de calcul = 0.28, Nombre d'itérations = 263

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

3.1 Block approach (subspace iter v2)

Question 8 : Lorsqu'on multiplie une matrice M de taille $n_1 \times n_2$ par une matrice L de taille $n_2 \times n_3$, chaque élément de la matrice résultante C de taille $n_1 \times n_3$ est calculé comme suit :

$$c_{ij} = \sum_{k=1}^{k=n_2} m_{ik} \times n_{kj}$$

Ainsi, pour chaque élément de C, il faut effectuer n_2 multiplications et $n_2 - 1$ additions. Sachant que C est composé de $n_1 \times n_3$ éléments, le coût du calcul de C est exactement $n_1 \times n_3 \times (2 \cdot n_2 - 1)$.

Donc le coût du calcul de A^p est le coût de $(p-1)$ multiplications de matrices de taille $n \times n$:

$$A^p \sim \sum_{i=1}^{p-1} (2n-1)(n^2) \sim n^3 \cdot p$$

Comme $dim(V) = n \times m$, le coût de calcul de A^p suivi de sa multiplication par V est :

$$n^3 \cdot p + n^2 \cdot m$$

Il est donc plus pratique d'organiser les calculs différemment pour réduire le coût de calcul.
Puisque $m \ll n$, il serait judicieux de faire une boucle pour calculer $A^p \cdot V$:

```
%% Y <- Ap*V
Y = A * Vr;
for i = 1:(p-1)
    Y = A*Y;
end
```

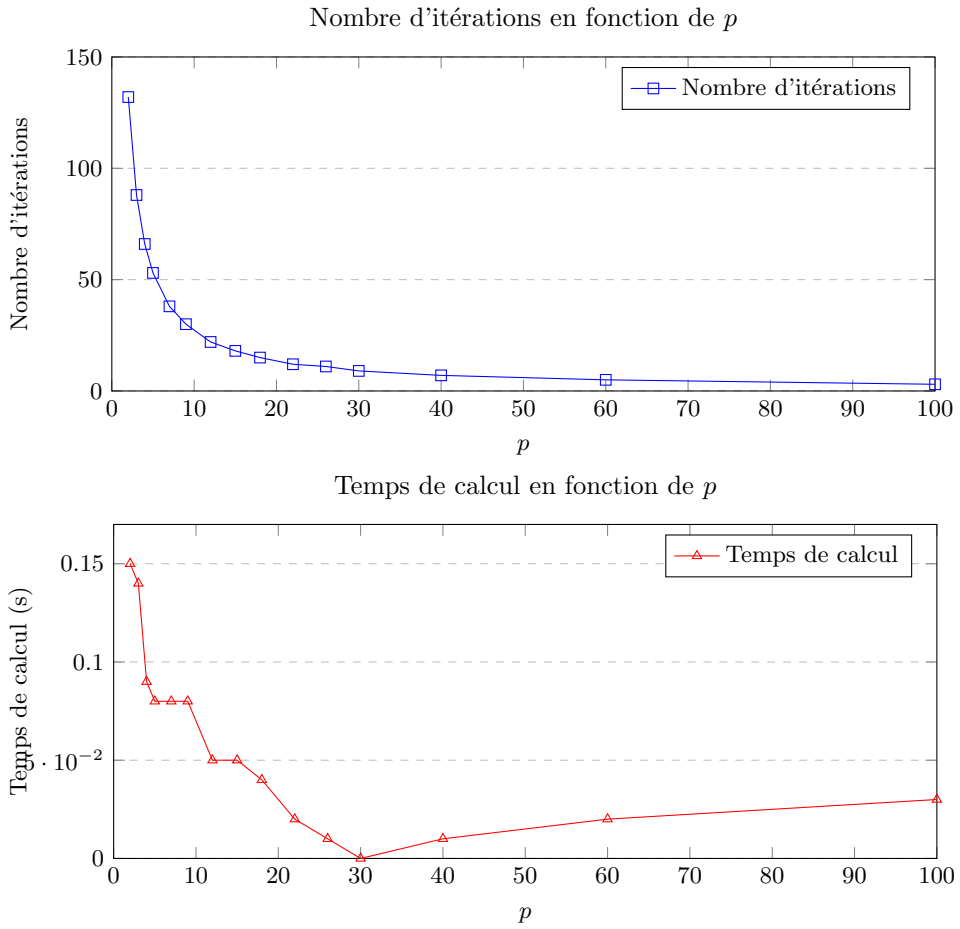
Le coût de cette méthode sera donc :

$$\sum_{i=1}^{p-1} n \times m \times (2n-1) \sim p \cdot n^2 \cdot m$$

Ce qui est mieux, car il n'y a pas de facteur cubique de n, qui est supposé être plus grand que p et m. En utilisant cette méthode, les calculs seront réduits et le temps d'exécution aussi.

Question 9 : cf fichier subspace-iter-v2.m

Question 10 : Pour une matrice A de taille 200×200 et de type $imat = 1$, on teste la fonction subspace-iter-v2 pour différentes valeurs de p, les résulats correspondants sont représentés dans la figure suivante :



Observations :

Les courbes représentant le temps de calcul et le nombre d'itérations sont étroitement liées, comme prévu. Initialement, avec un p plus petit, le nombre d'itérations nécessaires pour atteindre la convergence est plus élevé, ce qui entraîne un temps de calcul plus long. Ainsi, lorsque p augmente, le nombre d'itérations diminue, ce qui se traduit par une réduction du temps de calcul.

Cependant, malgré cette tendance à la baisse du temps de calcul avec p, pour certaines valeurs de p, telles que $p = 150$ et $p = 200$, nous constatons que le temps d'exécution ne diminue pas significativement par rapport à des valeurs de p plus faibles, au contraire le pourcentage pour atteindre les valeurs propres n'est pas atteint, en d'autres termes, on n'atteint pas le résultat souhaité de l'algorithme. Cela suggère qu'au-delà d'un certain point, l'augmentation de p ne se traduit plus par une amélioration significative des performances, suggérant ainsi l'existence d'une valeur optimale de p à déterminer.

Explications :

Dans la version v2 de l'algorithme, au lieu de multiplier la matrice A à chaque itération de la boucle principale, nous accélérons le processus en effectuant une multiplication par A^p . Cette approche équivaut à itérer la boucle de la version v1 p fois sans effectuer les nombreuses opérations coûteuses qui se produisent à l'intérieur de la boucle. En multipliant par A^p , nous évitons ainsi de répéter ces opérations à chaque itération, ce qui conduit à une réduction significative du temps de calcul.

De plus, le calcul de A^p est effectué en dehors de la boucle principale et le résultat est enregistré. En faisant cela, nous évitons de recalculer A^p à chaque itération de la boucle, ce qui réduit encore le temps de calcul. Ce stockage externe de A^p permet une utilisation efficace de la puissance de calcul et des ressources système, car cette opération coûteuse n'est effectuée qu'une seule fois, indépendamment

du nombre d'itérations de la boucle. Ainsi, en combinant l'accélération bloc avec l'enregistrement de A^p , la version v2 de l'algorithme parvient à réduire considérablement le temps de calcul par rapport à la version v1, surtout pour des valeurs élevées de p.

3.2 Deflation method (subspace iter v3)

Question 11 :

Voici le vecteur qv obtenu représentant la qualité de chacun des couples :

```
qv =

   1.0e-07 *

   0.000000470044229
   0.000000857945483
   0.000016549168609
   0.000057918882055
   0.000125449771256
   0.002065881434669
   0.001261870568878
   0.007170895545475
   0.013294953252659
   0.129221537046485
   0.766979101683572
```

On remarque que l'erreur de calcul du vecteur propre de la valeur propre la plus grande est beaucoup plus petite que la denière valeur propre. En effet, la première composante du vecteur qv vaut 4.6e-7, alors que la dernière valeur vaut 7.7e-1.

Explication de cette différence : Le vecteur propre associer à la première valeur propre de qv sera le premier à converger du fait qu'il soit associer à la plus grande valeur propre. Cependant, on observe que lorsque les autres vecteurs continuent à converger, lui aussi continue ce qui explique l'erreur de précision qui devient de plus en plus petite (d'où la valeur très petite de la première composante de qv). Les autres vecteurs obéissent à ce même principe également, et le dernier vecteur lui s'arrête directement dès sa première convergence car vu qu'il s'agit du dernier à considérer, le programme s'arrête.

Finalement, ceci explique les précisions des composantes de qv, qui vont du plus précis (tout en haut) au moins précis.

Question 12 :
Avec la méthode space iter v3 on s'assure de la convergence de chaque vecteur individuellement. Ainsi, dès qu'un vecteur a convergé, celui-ci est mis de côté et stoqué dans Vc et n'est plus jamais modifié. Il s'agit donc à ce moment là d'essayer de faire converger les autres vecteurs de Vnc.

Voici le vecteur qv obtenu avec l'algorithme space iter v3 :

```
qv =

   1.0e-07 *

   0.692488435933760
   0.657821077158874
   0.583432188653107
   0.787780516638381
   0.563450863772427
   0.783117447817166
   0.333119918360105
   0.685939436526111
   0.665589423887919
   0.748193817866724
   0.766979695991013
```

On voit bien que cette fois-ci, les équarts de précision des différentes valeurs propres reste du même ordre de grandeur et n'est pas grand. Les vecteurs qui ont convergé ne continuent pas à converger !

Question 13 : cf fichier subspace-iter-v3.m

4 Numerical experiments

Question 14 : Les quatre types de matrices sont :

- **Matrice de Type 1 :** matrice symétrique dont les valeurs propres décroissent. Les valeurs propres commencent à n et décroissent jusqu'à 1, ainsi $k(A) = n$.
- **Matrice de Type 2 :** matrice symétrique avec des valeurs propres distribuées exponentiellement. Les valeurs propres sont générées de manière aléatoire à l'aide d'une distribution exponentielle, avec un conditionnement : $k(A) = 10^{10}$.
- **Matrice de Type 3 :** matrice symétrique avec des valeurs propres distribuées géométriquement. Les valeurs propres décroissent géométriquement de 1 jusqu'à une petite valeur, déterminée selon un conditionnement : $k(A) = 10^5$.
- **Matrice de Type 4 :** matrice symétrique avec des valeurs propres distribuées linéairement entre une petite valeur déterminée et 1 selon un conditionnement : $k(A) = 10^{10}$.

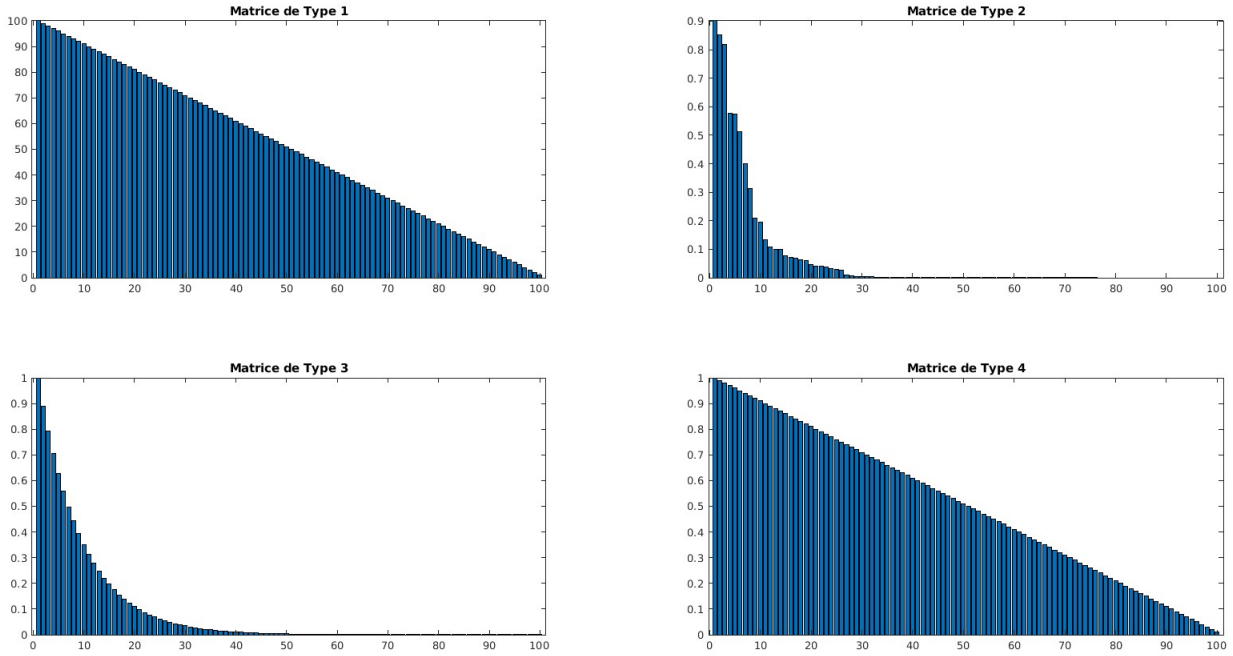


Figure 1: distributions de valeurs propres pour chaque type de matrice (n = 100)

Afin de voir quelle représentation matricielle est la meilleure, il faut réaliser le rapport des valeurs propres consécutives. Ainsi, on peut remarquer que, dans la matrice 1 et 4, les valeurs propres consécutives sont très proches, donc leur rapport est très proche de 1. Or pour trouver la meilleure représentation matricielle, il faut minimiser la valeur de ce rapport, et donc essayer d'approcher au maximum la valeur 0. Ceci s'observe plus pour le type de matrice 2 et 3 qui assurent donc une meilleure convergence !

Question 15 : imat = 1: (Pour les algorithmes subspace-iter-v2 et subspace-iter-v3, on a pris $p = 20$).

Taille de la matrice	Algorithme	Temps de calcul	Nombre d'itérations
50	eig	0.01	-
	power-v11	0.02	-
	power-v12	0.02	-
	subspace-iter-v0	0.72	481
	subspace-iter-v1	0.17	37
	subspace-iter-v2	0.08	2
	subspace-iter-v3	0.05	2
200	eig	0.01	-
	power-v11	1.73	-
	power-v12	0.14	-
	subspace-iter-v0	2.87	2341
	subspace-iter-v1	0.32	263
	subspace-iter-v2	0.05	14
	subspace-iter-v3	0.05	14
400	eig	0.08	-
	power-v11	16.56	-
	power-v12	0.51	-
	subspace-iter-v0	247.2	5348
	subspace-iter-v1	pourcentage non atteint	pourcentage non atteint
	subspace-iter-v2	pourcentage non atteint	pourcentage non atteint
	subspace-iter-v3	convergence non atteinte	convergence non atteinte

Table 4: Comparaison des performances entre les algorihtmes implémentés et ceux fourni pour des matrices de type 1 de différentes tailles

imat = 2:
(pour un grand p, la convergence n'est pas atteinte, on a pris p = 4)

Taille de la matrice	Algorithme	Temps de calcul	Nombre d'itérations
50	eig	0.03	-
	power-v11	0.02	-
	power-v12	0.01	-
	subspace-iter-v0	0.11	48
	subspace-iter-v1	0.01	2
	subspace-iter-v2	0.02	4
	subspace-iter-v3	0.01	4
200	eig	0.01	-
	power-v11	0.02	-
	power-v12	0.03	-
	subspace-iter-v0	0.82	76
	subspace-iter-v1	0.04	8
	subspace-iter-v2	0.01	2
	subspace-iter-v3	0.01	2
400	eig	0.07	-
	power-v11	0.06	-
	power-v12	0.03	-
	subspace-iter-v0	137	3299
	subspace-iter-v1	0.005	16
	subspace-iter-v2	0.02	4
	subspace-iter-v3	0.02	4

Table 5: Comparaison des performances entre les algorihtmes implémentés et ceux fourni pour des matrices de type 2 de différentes tailles

imat = 3:

Taille de la matrice	Algorithme	Temps de calcul	Nombre d'itérations
50	eig	0.01	-
	power-v11	0.01	-
	power-v12	0.01	-
	subspace-iter-v0	0.16	59
	subspace-iter-v1	0.02	4
	subspace-iter-v2	0.01	2
	subspace-iter-v3	0.05	2
200	eig	0.01	-
	power-v11	0.04	-
	power-v12	0.02	-
	subspace-iter-v0	2.65	248
	subspace-iter-v1	0.04	15
	subspace-iter-v2	0.03	4
	subspace-iter-v3	0.03	4
400	eig	0.06	-
	power-v11	0.44	-
	power-v12	0.09	-
	subspace-iter-v0	24.34	554
	subspace-iter-v1	0.1	35
	subspace-iter-v2	0.06	9
	subspace-iter-v3	0.08	9

Table 6: Comparaison des performances entre les algorihtmes implémentés et ceux fourni pour des matrices de type 3 de différentes tailles

imat = 4:

Taille de la matrice	Algorithme	Temps de calcul	Nombre d'itérations
50	eig	0.02	-
	power-v11	0.01	-
	power-v12	0.01	-
	subspace-iter-v0	0.77	473
	subspace-iter-v1	0.05	37
	subspace-iter-v2	0.02	10
	subspace-iter-v3	0.04	10
200	eig	0.04	-
	power-v11	1.71	-
	power-v12	0.16	-
	subspace-iter-v0	29.73	2353
	subspace-iter-v1	0.3	265
	subspace-iter-v2	0.12	67
	subspace-iter-v3	0.11	67
400	eig	0.05	-
	power-v11	15.87	-
	power-v12	0.54	-
	subspace-iter-v0	248.3	5387
	subspace-iter-v1	pourcentage non atteint	pourcentage non atteint
	subspace-iter-v2	pourcentage non atteint	pourcentage non atteint
	subspace-iter-v3	convergence non atteinte	convergence non atteinte

Table 7: Comparaison des performances entre les algorihtmes implémentés et ceux fourni pour des matrices de type 4 de différentes tailles

Partie II

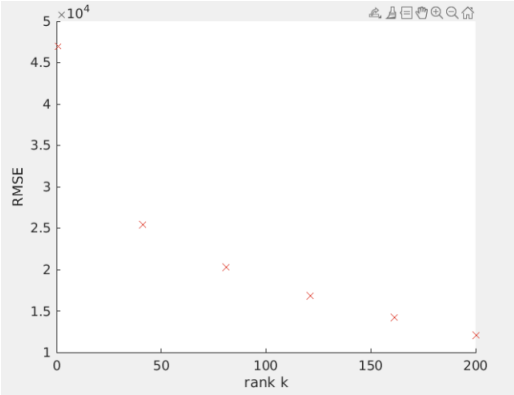
5 Application to Image Compression

Question 1 : Les dimensions des éléments du triplet (Σ_k, U_k, V_k) sont :

- $dim(\Sigma_k) = k \times k$
- $dim(U_k) = q \times k$
- $dim(V_k) = p \times k$

Question 2 :

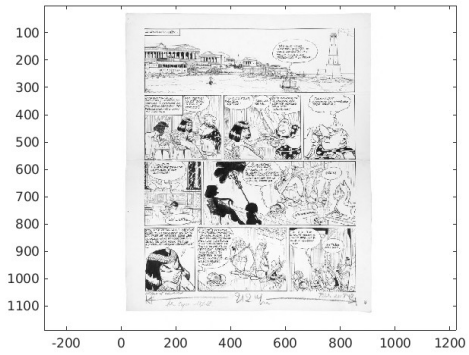
Voici la courbe obtenue représentant le RMSE (root mean square error) en fonction du nombre de valeurs propres.



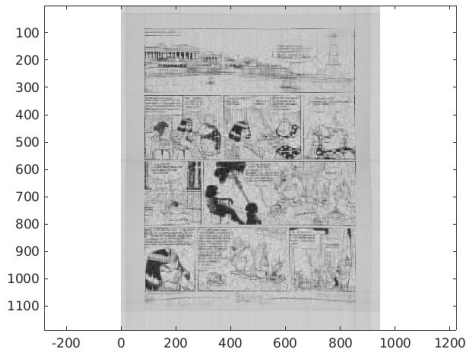
On remarque dans cette courbe que plus le nombre de valeurs propres augmente et plus l'erreur diminue !

Figure 2: image originale sans compression

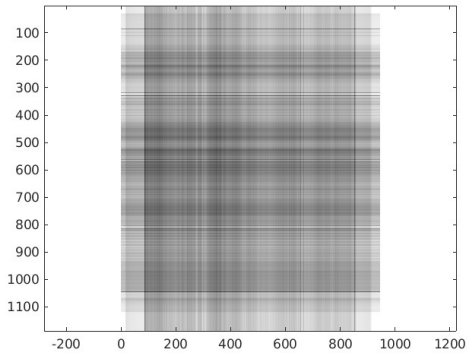
En comparant les différentes images reconstruites de la BD obtenues, on se rend compte que l'algorithme le plus efficace est celui de eig qui donne la meilleure reconstruction. Cependant, au contraire, l'algorithme power v12 pein à reconstruire la BD qui devient méconaissable.



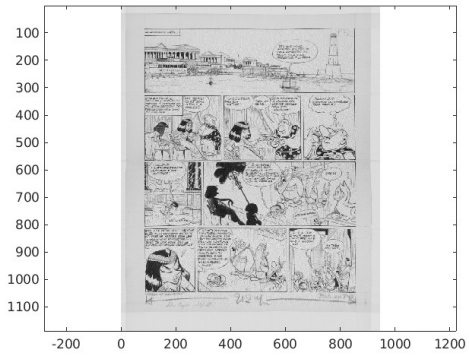
(a) image compressée en utilisant eig



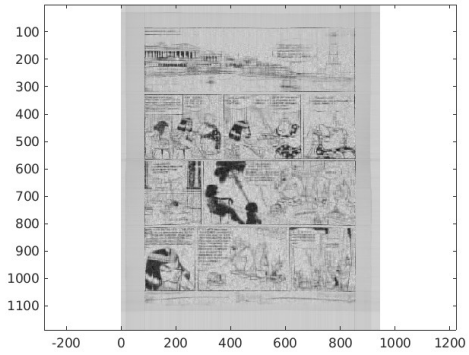
(b) image compressée en utilisant power v11



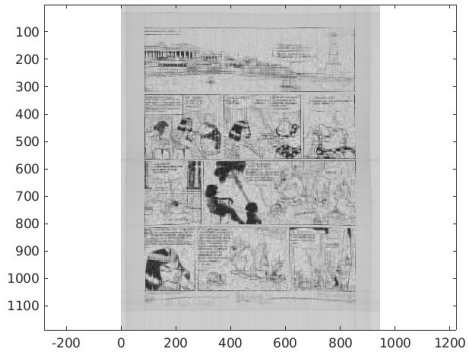
(c) image compressée en utilisant power v12



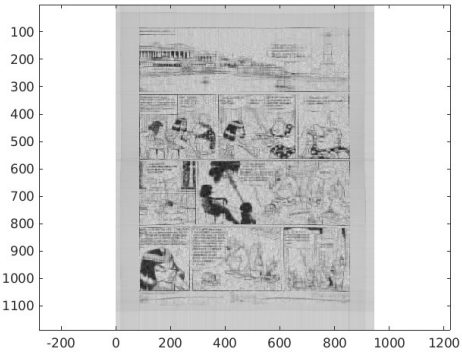
(d) image compressée en utilisant subspace-iter-v0



(e) image compressée en utilisant subspace-iter-v1



(f) image compressée en utilisant subspace-iter-v2



(g) image compressée en utilisant subspace-iter-v3