16 octobre 2024

1ère année en SN

LAGRANGE – MORISSEAU
Calcul Scientifique – Rapport de Projet
ENSEEIHT

Élèves :
Angel LAGRANGE
Albin MORISSEAU

Enseignants
S. Zhang

Département de Science du Numérique

Table des matières

1 Introduction

Dans ce projet, nous comparons différentes méthodes pour obtenir la décomposition spectrale d'une matrice. Tout d'abord, nous mettrons en évidence les limites de la méthode de la puissance. Ensuite, nous introduirons une méthode plus efficace appelée méthode d'itération du sous-espace, qui est basée sur le concept mathématique du quotient de Rayleigh. Nous étudierons cette méthode sous quatre versions, avec comme objectif global une amélioration maximale de la complexité de calcul. Enfin, nous appliquerons ces algorithmes à un problème concret de compression d'images.

2 Méthode d'itération du sous-espace

2.1 Limites de la méthode de la puissance

Algorithm 1 Vector power method

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
$\beta = v^T \cdot A \cdot v$
repeat
  $y = A \cdot v$
  $v = y / \|y\|$
  $\beta_{old} = \beta$
  $\beta = v^T \cdot A \cdot v$
until $|\beta - \beta_{old}| / |\beta_{old}| < \varepsilon$
$\lambda_1 = \beta$ and $v_1 = v$

FIGURE 1 – Algorithme de la puissance iterée

Question 1

Comme le montrent les tableaux 1 et 2, la fonction Matlab `eig` est bien plus efficace en terme de temps de calcul et de précision comparée à la méthode de la puissance iterée. De plus elle calcule tous les couples porpres contrairement à l'autre qui n'atteint pas toujours la convergence.

TABLE 1 – Temps d'exécution en secondes pour la méthode de la puissance

Type/Taille de matrice	100	300	500	1000
Type 1	7×10^{-2}	4.78×10^{-2}	$2.731 \times 10^{+1}$	Non convergence
Type 2	1×10^{-2}	4×10^{-2}	1.2×10^{-1}	$2.91 \times 10^{+0}$
Type 3	7.8×10^{-2}	1.6×10^{-1}	6.1×10^{-1}	$1.52 \times 10^{+1}$
Type 4	9×10^{-2}	$4.78 \times 10^{+0}$	$2.816 \times 10^{+1}$	Non convergence

TABLE 2 – Temps d'exécution en secondes pour la fonction Matlab `eig`

Type/Taille de matrice	100	300	500	1000
Type 1	0×10^{0}	3×10^{-2}	5×10^{-2}	3.2×10^{-1}
Type 2	$0 \times 10^{+0}$	4×10^{-2}	4×10^{-2}	2.4×10^{-1}
Type 3	0×10^{0}	1×10^{-2}	4×10^{-2}	2.2×10^{-1}
Type 4	1×10^{-2}	3×10^{-2}	6×10^{-2}	2.5×10^{-1}

Plusieurs raisons expliquent ces différences de temps et de précision. Tout d'abord, comme son nom l'indique, la méthode de puissance mise en œuvre est considérée comme "basique". En revanche, la fonction `eig` exploite la structure et les propriétés inhérentes de la matrice pour améliorer à la fois l'efficacité et la précision. En outre, elle peut utiliser des techniques avancées telles que la factorisation QR.

Question 2

Un examen plus approfondi de l'algorithme fourni montre qu'il existe deux produits matrice × vecteur, désignés par $A.v$, dans la boucle. Cette redondance nuit à l'efficacité.
Nous présentons l'algorithme suivant, qui optimise la séquence d'opérations pour réduire le nombre de produits matrice × vecteur à un seul.

Algorithm 2 Vector power method modified

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

FIGURE 2 – Version améliorée de l'algorithme de la puissance

Il est clair que cette simple modification réduit presque de moitié le temps d'exécution.

TABLE 3 – Temps d'exécution en secondes de l'algorithme amélioré de la méthode de la puissance

Type/Type de matrice	100	300	500	1000
Type 1	7×10^{-2}	$2.56 \times 10^{+0}$	$1.45 \times 10^{+1}$	Non convergence
Type 2	0×10^{0}	2×10^{-2}	5×10^{-2}	$1.53 \times 10^{+0}$
Type 3	3.125×10^{-2}	1.8×10^{-1}	3.5×10^{-1}	$7.73 \times 10^{+0}$
Type 4	7×10^{-2}	$2.53 \times 10^{+0}$	$1.493 \times 10^{+1}$	Non convergence

Question 3

La principale limite de la méthode de la puissance déflatée réside dans l'obligation de calculer la décomposition spectrale complète de la matrice A, qui est une matrice carrée $nfoisn$. Un autre inconvénient, bien que moins important, est que dans les cas où une matrice possède plusieurs valeurs propres de magnitude similaire, la déflation peut s'avérer inefficace, nécessitant de multiples itérations pour converger vers les valeurs propres restantes.

2.2 Première version de la méthode d'itération du sous espace

Algorithm 3 Subspace iteration method, first version

Input : Matrix $A \in \mathbb{R}^{n \times n}$, number of required eigenpairs , tolerance ε and MaxIter
Output : m dominant eigenvectors V_{out} and the corresponding eigenvalues Λ_{out}
Generate a set of m orthonormal vectors $V \in \mathbb{R}^{n \times m}$; $k = 0$
repeat
  $k = k + 1$
  $Y = A \cdot V$
  $H = V^T \cdot A \cdot V$ or $H = V^T \cdot Y$
  Compute $acc = \|A \cdot V - V \cdot H\| / \|A\|$
  $V \longleftarrow$ orthonormalisation of the columns of Y
until $(k > MaxIter$ **or** $acc \leq \varepsilon)$
Compute the spectral decomposition $X \cdot \Lambda_{out} \cdot X^T = H$ where the eigenvalues of $H(diag(\Lambda_{out}))$ are arranged in descending order of magnitude.
Compute the corresponding eigenspace $V_{out} = V \cdot X$

FIGURE 3 – Première version de la méthode subspace iteration

Question 4

La méthode de la puissance itérée renvoie le vecteur propre associé à la plus grande valeur propre de la matrice. Sans déflation, cet algorithme renverra toujours la même paire propre dominante.

Ainsi, si nous appliquons l'algorithme de la puissance itérée aux m vecteurs colonnes de la matrice V, alors V convergera vers une matrice dont toutes les colonnes correspondent au vecteur propre de la plus grande valeur propre de A.

Question 5

Le calcul de la décomposition spectrale complète de H à l'aide de l'algorithme itératif du sous-espace ne pose pas de problème. En effet, H est de taille $m < n$. Le calcul de la décomposition spectrale de H est donc assez efficace et ne nécessite pas trop de mémoire ou de temps de calcul.

2.3 Seconde version de la méthode d'itération du sous-espace avec projection de Rayleigh

Algorithm 4 Rayleigh-Ritz projection

Input : Matrix $A \in \mathbb{R}^{n \times n}$ and an orthonormal set of vectors V
Output : The approximate eigenvectors V_{out} and the corresponding eigenvalues Λ_{out}
Compute the Rayleigh quotient $H = V^T \cdot A \cdot V$
Compute the spectral decomposition $X \cdot \Lambda_{out} \cdot X^T = H$ where the eigenvalues of $H(diag(\Lambda_{out}))$ are arranged in descending order of magnitude.
Compute $V_{out} = V \cdot X$

FIGURE 4 – Projection de Rayleigh

Algorithm 5 Subspace iteration method with Rayleigh-Ritz projection

Input : Symmetric matrix $A \in \mathbb{R}^{n \times n}$, tolerance ε and $MaxIter$ (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 $V \in \mathbb{R}^{n \times m}$; $k = 0$; $PercentReached = 0$
repeat
  $k = k + 1$
  Compute Y such that $Y = A \cdot V$
  $V \longleftarrow$ 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 ($PercentReached < PercentTrace$ $n_{ev} = m$ or $k > MaxIter$)

FIGURE 5 – Seconde version de la méthode d'itération du sous-espace avec projection de Rayleigh

Question 7

Voici les lignes du fichier `subspace_iter_v1.m` qui correspondent aux différentes étapes de l'algorithme précédent :

1. Générer un ensemble initial de m vecteurs orthonormés V ; (lignes 48 et 49)
2. $k = 0$; (ligne 38)
3. $PercentReached = 0$; (ligne 40)
4. $k = k + 1$; (ligne 54)
5. Calculer Y tel que $Y = A \cdot V$; (ligne 56)
6. $V \leftarrow$ orthonormalisation des colonnes de Y ; (ligne 58)
7. Projection de Rayleigh-Ritz appliquée à la matrice A et aux vecteurs orthonormés V ; (ligne 61)
8. - Étape de l'analyse de convergence : sauvegarde des paires propres qui ont convergé et mise à jour de $PercentReached$; (lignes 88 à 97)

2.4 Troisième version de l'itération du sous-espace avec la projection de Rayleigh et accélération

Question 8

Si nous calculons A^p en utilisant le produit matriciel classique, le coût en termes de flops serait $O(n^3p)$, car nous aurions besoin d'effectuer $p-1$ multiplications de matrices et le coût d'une multiplication de matrices n par n par une autre est $O(n^3)$.

Soit M une matrice $n \times n$ et V une matrice $n \times m$, si nous calculons $M \cdot V$ en utilisant le produit matriciel classique, le coût en termes de flops serait $O(n^2m)$.
Ensuite, si nous supposons que nous devons calculer A^p en utilisant le produit matriciel classique, le coût en termes de flops pour $A^p \cdot V$ serait $O(n^3p + n^2m)$, ce qui dans le cas où $m \ll n$ serait $O(n^3p)$.
Cependant, si nous multiplions d'abord la matrice $n \times n$ A avec la matrice $n \times m$ V puis nous multiplions $p-1$ fois par A du côté gauche, nous aurons $p-1$ produits de matrices $n \times n$ par $n \times m$, ce qui implique un coût moindre en termes de flops si $m \ll n$.En effet, le coût en termes de flops serait $O(pn^2m)$, ce qui est inférieur à $O(n^3p)$.
Question 10

Nous avons testé l'influence de l'augmentation du paramètre p pour des tailles et des types de matrices différentes. Voici ce que nous avons observé :

TABLE 4 – Temps d'exécution en secondes de la version 2 de la méthode du sous espace pour une matrice de taille 200x200 et de type 4

Valeurs de p	Temps de calcul
1	1.25×10^{-1}
10	4.68×10^{-2}
100	3.125×10^{-2}
300	6.25×10^{-2}

TABLE 5 – Temps d'exécution en secondes de la version 2 de la méthode du sous espace pour une matrice de taille 1000x1000 et de type 2

Valeurs de p	Temps de calcul
1	1.25×4^{-1}
2	4.68×2^{-1}
3	3.125×1.3^{-1}
4	6.25×1.6^{-1}

Lorsque nous augmentons p dans la méthode d'itération du sous-espace, la convergence tend à s'accélérer. Néanmoins, si p devient excessivement grand, la convergence échoue. En effet, l'augmentation de p augmente la charge de calcul de chaque itération ; le calcul de $A^p \cdot V$ nécessite p multiplications matricielles, ce qui peut être coûteux pour les matrices de taille importante. Par conséquent, p doit être judicieusement choisi pour équilibrer la vitesse de convergence et l'efficacité de calcul.

Question 11

Lorsque l'ensemble initial de vecteurs, sélectionné de manière aléatoire, capture effectivement une partie significative de l'espace propre, les paires propres qui en résultent tendent à être de qualité élevée. Inversement, si l'ensemble initial de vecteurs ne couvre pas une partie importante de l'espace propre, la convergence peut être lente, ce qui se traduit par un calcul moins précis des paires propres.

Question 12

Avec cette dernière méthode, la précision des couples propres diminue par rapport aux méthodes précédentes. Cela peut s'expliquer par le fait que l'étape de déflation peut entraîner la perte d'informations sur

les vecteurs propres calculés précédemment. Cette perte d'informations spectrales peut compromettre la précision des approximations ultérieures effectuées sur les couples propres, en particulier pour le couple final.

Question 14

Voici les 4 types de matrices différents avec lesquelles nous testons nos alogrithmes :
Type 1

Le premier type de matrice est une matrice à diagonale dominante, c'est à dire telle que la valeur absolue de chaque élément sur la diagonale principale est strictement supérieure à la somme des valeurs absolues des autres éléments de la même ligne (ou colonne).

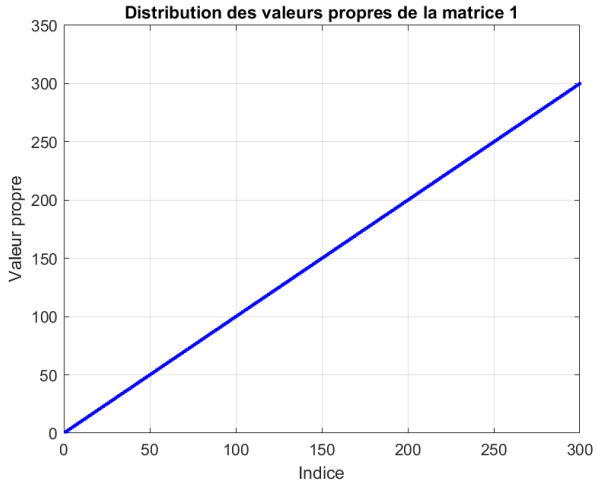


FIGURE 6 – Distribution valeurs propres de la matrice 1 de taille 300

Les valeurs propres suivent une distribution linéaire simple : D(i)=i Graphiquement, cela se traduit par une ligne droite lorsque on trace les valeurs propres en fonction de leur indice i. Chaque valeur propre est simplement égale à son indice.

Type 2

Le second type de matrice est aussi une matrice à diagonale dominante avec des valeurs absoluement inférieures à 1.

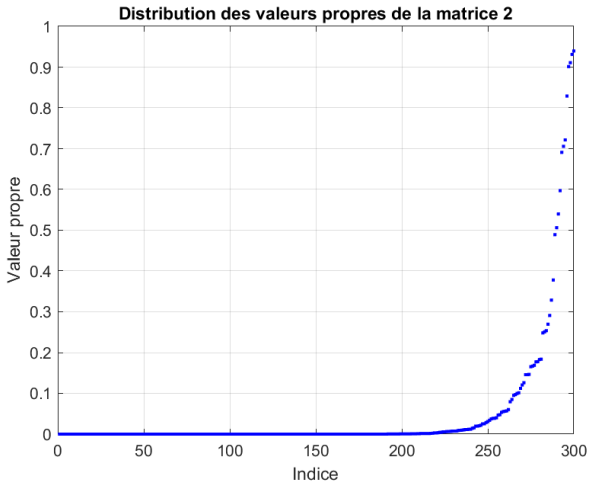


FIGURE 7 – Distribution valeurs propres de la matrice 2 de taille 300

Les valeurs propres sont générées de manière aléatoire, mais avec une densité de probabilité logarithmique uniforme. Les valeurs propres seront réparties de manière plus dense pour les valeurs plus petites et plus clairsemées pour les valeurs plus grandes.

Type 3

Le troisième type de matrice est constituée de valeur inférieure à 1.

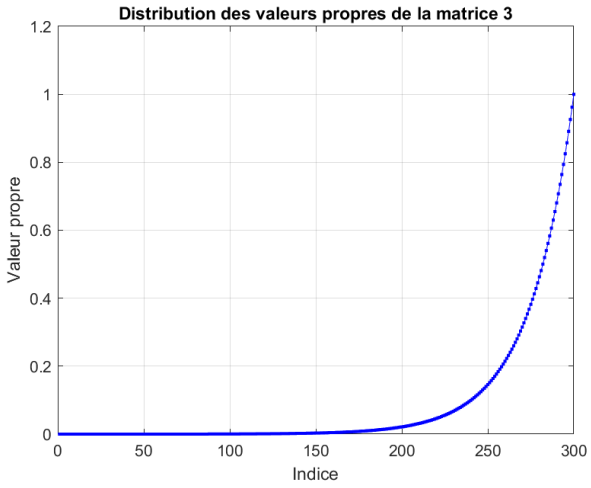


FIGURE 8 – Distribution valeurs propres de la matrice 3 de taille 300

Les valeurs propres décroissent de manière exponentielle en fonction de leur indice i, avec un facteur de décroissance contrôlé par le conditionement de la matrice. Graphiquement, cela se traduit par une décroissance exponentielle des valeurs propres lorsque on les trace en fonction de leur indice i. Les valeurs propres initiales seront relativement élevées et diminueront rapidement à mesure que i augmente.

Type 4

Le quatrième type de matrice est composée de valeurs infére ieures à 1 mais avec une plus importante présence de valeurs proches de 0.

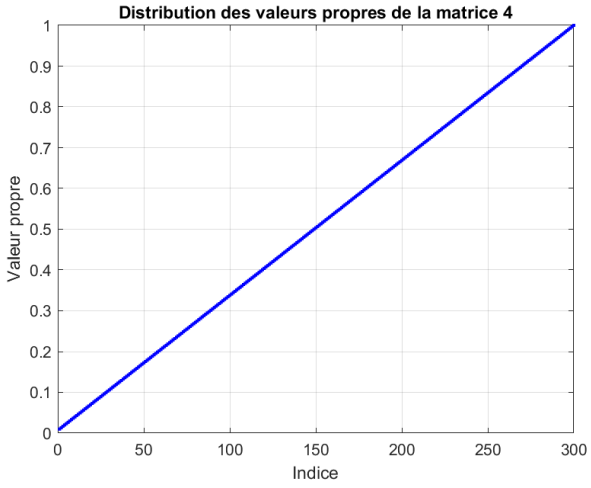


FIGURE 9 – Distribution valeurs propres de la matrice 4 de taille 300

Graphiquement, la courbe des valeurs propres en fonction de leur indice est une droite. Les valeurs propres

suivent une distribution uniforme.

Question 15

Nous avons comparé les perfomances en terme de temps de calcul des différents algorithmes pour différents types et différentes tailles de matrices significativement différentes. Nous avons choisi un pourcentage spectral à atteindre de 0.1 et un nombre de valeurs propres à calculer de $0.1 * n$ avec n la taille de notre matrice. De plus, pour les méthodes le nécessitant, nous avons choisi une puissance p égale à 3 suite aux observations de la question 10. Nous avons réuni les résultats dans le tableau suivant :

	eig	power method	v0	v1	v2	v3
Type 1 (*200* x *200*)	2.0e-02	9.9e-01	2.05	3.0e-01	1.2e-01	1.9e-01
Type 2 (*200* x *200*)	2.0e-02	3.0e-02	9.0e-02	8.0e-02	2.0e-02	4.0e-02
Type 3 (*200* x *200*)	2.0e-02	5.0e-02	2.3e-01	6.0e-02	9.0e-02	4.0e-02
Type 4 (*200* x *200*)	1e-02	1.05	1.56	3.5e-01	2.1e-01	2.0e-01
Type 1 (*1000* x *1000*)	3.2e-01	Non convergence	1.11e+02	9.3	3.85	3.92
Type 2 (*1000* x *1000*)	2.4e-01	1.5	1.28e+01	3.6e-01	1.5e-01	3.1e-01
Type 3 (*1000* x *1000*)	2.2e-01	7.73	1.45e+01	6.7e-01	2.8e-01	5.0e-01
Type 4 (*1000* x *1000*)	2.5e-01	Non convergence	1.13e+02	9.41	3.61	4.11

TABLE 6 – Tableau des temps d'éxécution en secondes des différents algorithmes.

Dans un premier temps, on observe que la fonction eig reste généralement la méthode la plus rapide. Cette constatation est cohérente car cette fonction a été spécialement conçue pour cette tâche et est celle implémentée dans Matlab donc celle jugée la plus efficace dans la majorité des cas.
On remarque aussi que la version 0 de $subspace_iter$, bien que parfois moins efficace en temps de calcul garanti au moins la convergence pour des matrices de plus grandes tailles.
L'implémentation de la version 2 de $subspace_iter$ semble efficace en temps de calcul et est parfois même plus efficace que la fonction Matlab *eig* bien que l'on reste dans les même ordres de grandeurs et que ces différences sont très peu significatives.
En revanche, on semble observer que la troisième et dernière version de $subspace_iter$ n'améliore pas le temps de calcul par rapport à la deuxième version et reste globablement dans les même ordres de grandeur. Or, avec l'implémentation de la déflation, on s'attendrait à ce que cette troisième version soit plus rapide. Etant donnée que nous trouvons quand même des ordres de grandeurs très similaires, nous supposons que la différence se fait principalement pour des matrices de très grande tailles.

3 Application à la compression d'images

Question 1

Voici les dimensions des différentes matrices du triplet :

$$U_k \in \mathbb{R}^{q \times k}$$
$$\Sigma_k \in \mathbb{R}^{k \times k}$$
$$V_k \in \mathbb{R}^{p \times q}$$

Question 2
Nous avons essayé de reconstruire différentes images avec l'approximation de rang faible en utilisant les différents algorithmes permettant de calculer les valeurs propres d'une matrice implentées précédement. Nous avons principalement fait varié les paramètres *puiss* et $search_space$ qui correspondent respectivement à notre p et notre m des parties précédentes.
Notre premier constat a été de réaliser que la méthode de la puissance ne nous permetait pas d'aboutir à l'obtention d'une image compressée. Nous avons interpreté cela par le fait que les images soient des matrices de très grandes tailles et que, comme nous l'avons vu précédemment, ces algorithmes ne convergeaient pas pour des matrices de taille importante.

Pour les autres méthodes, nous avons réuni les résultats des temps de calcul dans le tableau suivant :

	eig	SVD	v1	v2	v3
BD Asterix (*m=400* x *p=1*)	3.9e-01	1.02	3.431	3.327	3.41
BD Spirou (*m=400* x *p=1*)	1.26	15.26	19.49	17.97	16.7
BD Thorgal (*m=400* x *p=1*)	72.31	128.2	x	x	x
BD Spirou (*m=400* x *p=2*)	6.1	14.95	18.86	13.86	10.6
BD Spirou (*m=50* x *p=1*)	5.6	14.67	20.23	12.74	9.9

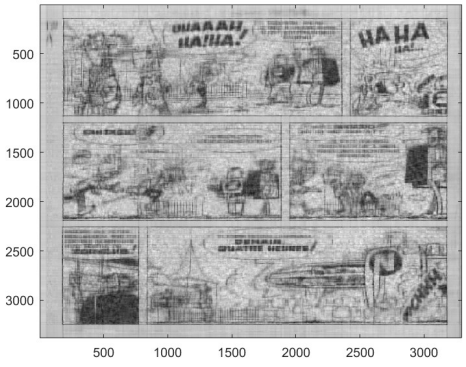
TABLE 7 – Tableau des temps d'éxécution en secondes des différents algorithmes de calcul de couple propre dans un cadre de reconstitution d'image.

Premièrement, notre hyopthèse émise en fin de partie précédente stipulant que la troisième version de $subspace_iter$ était plus efficace que la seconde pour de très grande matrices semble se vérifier. En effet, la version 2 est plus rapide pour une bande dessinée Asterix (1187x1920) mais la version 3 est bien plus rapide pour les bande dessinées Spirou(3385x3296) et Thorgal(7229x5634) qui sont de tailles bien plus importantes.

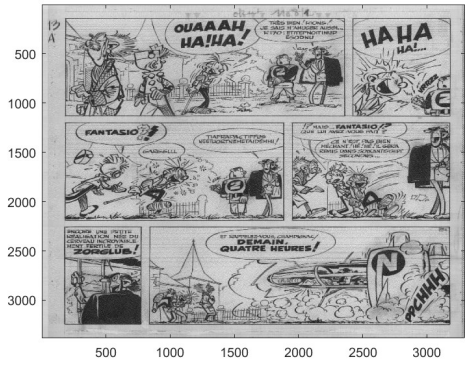
Nous avons remarqué que pour la Bande dessinée Thorgal, les méthodes du sous espaces iterérées, quelles que soient la version, ne convergent pas où sont trop coûteuses pour des valeurs de m trop élevées . Nous en déduisons qu'elles sont moins efficaces que la méthode SVD ou que la fonction eig de Matlab pour avoir une meilleure qualité. Elles trouvent leurs limites dans le cas de matrice de tailles très importantes.

INFLUENCE DU PARAMETRE *PUISS*
Comme le montre la compraison entre la ligne 2 et 4 du tableau précédent, on observe que si on augmente la valeur de p, le temps de calcul de l'image s'améliore mais on ne gagne pas spécialement en qualité.

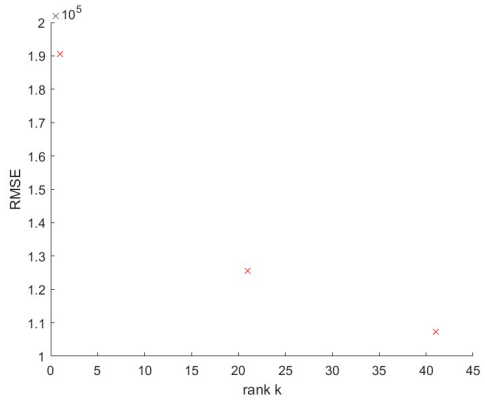
INFLUENCE DU PARAMETRE *SEARCHSPACE*
Au contraire, comme illustré dans les lignes 2 et 5 du tableau précédent, si on modifie le paramètre subspace correspondant à la valeur de m dans nos algorithmes implentés, le temps de calcul s'améliore. Cependant, comme le montre les figures ci-dessous, la qualité de l'image finale est diminuée ce qui est logique car on récupère moins d'information spectrale : on met moins de temps de calcul mais on augmente l'erreur de reconstruction car on perd en information.



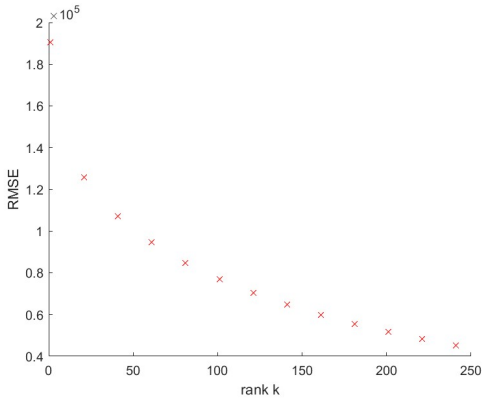
(a) Qualité de la BD Spirou pour search space = 50



(b) Qualité de la BD Spirou pour search space = 400



(a) Evolution du REM pour search space = 50



(b) Evolution du REM pour search space = 400

Ainsi, pour conclure, on constate que la fonction eig de matlab reste la plus performante. Nos algorithmes de recherche de valeurs propres tendent à s'améliorer et à se rapprocher des résultats de la fonction eig mais ils restent bien moins efficace en particulier lorsque la taille de la matrice devient trop importante.