



Rapport de Calcul Scientifique

Projet

Sommaire

1	Introduction	2
1.1	Contexte	2
1.2	Rappels et introduction	2
1.3	Objectifs	2
2	Méthode d’itération des sous-espaces	3
2.1	Limites de la méthode de la puissance itérée	3
2.2	Extension de la méthode de la puissance itérée	6
2.3	<code>subspace_iter_v2</code> et <code>subspace_iter_v3</code> : vers un solveur efficace	8
2.4	Expérimentations numériques	10
3	Application à la compression d’image	12
3.1	Rappels et Introduction	12
3.2	Compression d’images	13
4	Conclusion	22

Réalisé par :

Ralph Khairallah
Thomas Canisares–Morère

Date : May 7, 2025

1 Introduction

1.1 Contexte

1.2 Rappels et introduction

Dans plusieurs problèmes de data analysis, on s'intéresse seulement à quelque couples propres dominant d'une matrice sans faire toute la décomposition spectrale. Par exemple la méthode ACP qu'on a vu en analyse de donne, et dans ce projet on verra la compression d'image. La power method de base calcule un couple propre dominant mais prend du temps pour atteindre la convergence car la méthode repart de 0 pour calculer les valeurs propres restantes (non dominantes).

Dans ce projet on étudiera plusieurs version de méthodes pour traiter ce problème , on verra 2 méthodes de power method v11 et v12 et 4 variante de la subspace iteration method v0 v1 v2 v3 et les comparer tous ensemble avec la méthode classique de Matlab `eig`.

Dans la première Partie, nous implémentons tout ces algorithmes et nous allons comparer leur performances sur plusieurs matrices , de taille et de types différentes afin de tirer des conclusion sur les algorithmes les plus performants et dans quel cas.

Dans la Partie 2, nous allons utiliser ses algorithmes implantés dans la Partie 1 dans un pipeline de compression d'images et pareil nous allons essayer de les comparer sur plusieurs images.

1.3 Objectifs

1. Algorithmes

- Implémenter `power_v12` (un seul produit Matrice-Vecteur par itération) qui est une méthode amélioré de la `power_v11` et 4 méthodes de `subspace_iter_v0`, `v1`, `v2` et `v3`.
- Completer les fichiers des algorithmes ainsi que les fichiers de tests pour comparer les algorithmes eux.

2. Étude numérique

- Comparer les algorithmes entre eux et les présenter dans des tables.
- Mesurer le temps d'exécution de chaque algorithme pour différent types de matrices et de tailles différentes, calculer le nombre d'itération avant d'atteindre a convergence.

3. Analyse comparative

- Identifier les types de matrices qui accélèrent ou ralentissent la convergence.
- Identifier les algorithmes qui présente le meilleur rapport en temps d'exécution et nombre d'itération ainsi que précision, comme par exemple l'impact du paramètre p , pour atteindre la convergence.

2 Méthode d’itération des sous-espaces

2.1 Limites de la méthode de la puissance itérée

Question 1 : Comparaison des temps d’exécution entre `eig` de MATLAB (décomposition spectrale complète) et `power_v11` (méthode de la puissance itérée avec déflation).

Type 1 — Valeurs propres $D(i) = i$

Taille	Méthode	Temps (s)	# EigPairs	Eigpair Quality	Eigval Quality
100	<code>eig</code>	0.190	-	[5.94e-16, 3.85e-14]	[0, 7.36e-15]
	<code>power_v11</code>	0.160	6	[9.94e-09, 1.42e-08]	[0, 9.95e-15]
500	<code>eig</code>	0.280	-	[6.18e-16, 2.25e-13]	[0, 1.50e-13]
	<code>power_v11</code>	146.5	26	[1.14e-16, 1.64e-08]	[-, 5.03e-14]
1000	<code>eig</code>	1.150	-	[5.74e-16, 5.54e-13]	[0, 1.33e-13]
	<code>power_v11</code>	fail	—	[0, 0]	[0, 0]

Table 1: Résultats pour les matrices de type 1

Commentaires : Dans ce cas on voit bien que `power_v11` converge rapidement pour les matrices de petites tailles ($n = 100$). Mais pour $n = 500$, la méthode devient beaucoup plus lente, et échoue totalement pour $n = 1000$. Alors que `eig` reste très rapide et précis.

Type 2 — Valeurs propres $D(i) = \text{random}(1/\text{cond}, 1)$

Taille	Méthode	Temps (s)	# EigPairs	Eigpair Quality	Eigval Quality
100	<code>eig</code>	0.030	-	[2.63e-16, 5.04e-07]	[0, 1.70e-07]
	<code>power_v11</code>	0.010	1	[7.14e-09, 7.14e-09]	[4.38e-16]
500	<code>eig</code>	0.230	-	[5.12e-16, 3.45e-06]	[0, 2.70e-07]
	<code>power_v11</code>	0.610	2	[9.96e-09, 1.53e-08]	[6.18e-16, 1.25e-15]
1000	<code>eig</code>	1.030	-	[5.34e-16, 1.31e-06]	[0, 2.12e-07]
	<code>power_v11</code>	5.89	4	[9.98e-09, 1.49e-08]	[1.24e-15, 6.97e-15]

Table 2: Résultats pour les matrices de type 2

Commentaires : Ici, la méthode `power_v11` arrive à extraire les valeurs propres dominantes avec précision. Quand on augmente la taille de la matrice, le nombre d’itérations croît avec, ce qui augmente le temps. Encore ici `eig` est plus précis.

Type 3 — Valeurs propres $D(i) = \text{cond}^{(-(i-1)/(n-1))}$

Taille	Méthode	Temps (s)	# EigPairs	Eigpair Quality	Eigval Quality
100	eig	0.040	-	[4.00e-16, 1.94e-11]	[0, 5.76e-12]
	power_v11	0.010	1	[9.38e-09, 9.38e-09]	[8.88e-16]
500	eig	0.240	-	[5.89e-16, 3.15e-11]	[0, 1.06e-11]
	power_v11	2.46	5	[9.83e-09, 1.44e-08]	[0, 4.33e-15]
1000	eig	1.020	-	[5.40e-16, 5.14e-11]	[0, 1.14e-11]
	power_v11	25.99	10	[9.94e-09, 1.43e-08]	[0, 8.33e-15]

Table 3: Résultats pour les matrices de type 3

Commentaires : Ici, on voit que plus les valeurs propres sont proches, plus il faut d’itérations. **power_v11** est plus rapide pour les matrice de petite taille mais en général **eig** reste plus rapide stable.

Type 4 — Valeur propres $D(i) = 1 - ((i-1)/(n-1)) * (1 - 1/\text{cond})$

Taille	Méthode	Temps (s)	# EigPairs	Eigpair Quality	Eigval Quality
100	eig	0.040	-	[6.49e-16, 5.24e-14]	[0, 1.27e-14]
	power_v11	0.090	6	[9.94e-09, 1.42e-08]	[1.16e-16, 9.77e-15]
500	eig	0.250	-	[5.52e-16, 4.52e-14]	[0, 1.87e-14]
	power_v11	173.1	26	[9.99e-09, 1.64e-08]	[0, 5.04e-14]
1000	eig	1.100	-	[4.95e-16, 6.49e-14]	[0, 1.51e-14]
	power_v11	fail	—	[0, 0]	[0, 0]

Table 4: Résultats pour les matrices de type 4

Commentaires : La méthode **power_v11** devient instable pour $n = 1000$ et très lente dès $n = 500$. **eig** reste rapide et stable pour n’importe quelle matrice .

Conclusion générale

On voit donc bien que la **Méthode eig** est en général rapide stable et précise pour tout type et taille de matrices.

Ceci n’est pas le cas pour la **Méthode power_v11** qui est acceptable pour des petites matrices avec un spectre espacé. Elle est parfois très lente ou n’arrive même pas à la convergence ce qui rend la **Méthode eig** plus avantageuse

Finalement le type de matrice utilisé est très important et doit être pris en considération car il influence sur les performances des méthodes , par exemple pour les spectres mal conditionnés (type 2, 3, 4), la **Méthode power_v11** ne performe pas bien.

Question 2 : Comparaison des temps d'exécution entre la méthode de la puissance itérée `power_v11` et de la puissance itérée amélioré `power_v12`

Dans cette partie nous cherchons à améliorer l'algorithme 1, pour ne pas calculer le produit Av deux fois dans la boucle. En effet ceci augmente considérablement le temps d'exécution quand la taille de la matrice augmente et nous obtenons l'algorithme suivant:

Algorithm 1 Méthode puissance itérée amélioré

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

```

 $z \leftarrow A \cdot v$ 
 $\beta \leftarrow v^T \cdot z$ 
 $v \leftarrow z / \|z\|$ 
 $\beta_{\text{old}} \leftarrow \beta$ 
 $z \leftarrow A \cdot v$ 
 $\beta \leftarrow v^T \cdot z$ 
 $|\beta - \beta_{\text{old}}| / |\beta_{\text{old}}| < \varepsilon$ 
 $\lambda_1 \leftarrow \beta$  and  $v_1 \leftarrow v$ 
```

Ainsi voici le changement qu'on effectuera dans le code Matlab :

Listing 1 Méthode puissance itérée amélioré (`power_v12.m`)

```

1 while (~convg  $\&\&$  k < m)
2     k = k + 1;
3
4     % méthode de la puissance itérée améliorée
5     v = randn(n,1);
6     z = A*v;
7     beta = v'*z;
8
9     % conv = || beta * v - A*v || / |beta| < eps
10    % voir section 2.1.2 du sujet
11    norme = norm(beta*v - z, 2)/norm(beta,2);
12    nb_it = 1;
13
14    while (norme > eps  $\&\&$  nb_it < maxit)
15        y = z;
16        v = z / norm(z,2);
17        beta = v'*z;
18        norme = norm(beta*v - z, 2)/norm(beta,2);
19        nb_it = nb_it + 1;
20    end
```

Ainsi avec cette algorithme voici les résultats que nous obtenons pour différentes types de matrices de tailles différentes:

Taille	Type	Temps power_v11 (s)	Temps power_v12 (s)	Accélération
100	1	5.763e-02	9.850e-03	5.85x
500	1	1.116e+01	2.856e-02	390.66x
1000	1	1.604e+00	5.972e-02	26.86x
100	2	1.184e-02	6.782e-03	1.75x
500	2	6.200e-02	2.146e-02	2.89x
1000	2	4.817e-01	6.080e-02	7.92x
100	3	1.174e-02	8.169e-03	1.44x
500	3	1.603e-01	2.174e-02	7.38x
1000	3	2.169e+00	6.294e-02	34.47x
100	4	4.959e-02	1.003e+00	4.94x
500	4	1.200e+01	2.744e-02	437.37x
1000	4	1.646e+00	6.431e-02	25.60x

Table 5: Comparaison des temps d'exécution entre `power_v11` et `power_v12` selon la taille et le type de matrice

Commentaire : La méthode de la puissance itérée amélioré accelere la convergence en evitant de calculer le nombre de multiplications dans chaque iteration. Ceci est interessant surtout quand on augmente la taille de la matrice et comme on peut le voir `power_v12` est parfois $1.4\times$ jusqu'à $400\times$ plus rapide que `power_v11` dans certains cas.

Question 3 : Principal inconvenient de la méthode de la puissance itérée avec déflation en terme de temps de calcul.

Le principal inconvenient de la méthode de la puissance avec déflation (implémentée dans `power_v11`) est qu'elle relance une boucle complète de puissance itérée pour chaque couple propre, alors que les valeurs propres suivantes sont généralement plus difficiles à isoler en raison de leur proximité spectrale. Par conséquent :

- La méthode passe mal à l'échelle lorsqu'on cherche un grand nombre de valeurs propres.
- Elle devient très lente pour les grandes matrices, chaque itération nécessitant un produit matrice-vecteur ce qui rend la convergence non atteinte dans certains cas.
- On a perte de précision progressivement à cause des erreurs numériques accumulées et de la perte d'orthogonalité.

2.2 Extension de la méthode de la puissance itérée

Question 4 : Matrice de convergence V de la méthode de la puissance itérée si on l'applique à plusieurs vecteurs m .

Si on applique la méthode de la puissance itérée simultanément à une matrice V de m vecteurs sans forcer l'orthonormalisation entre elles, les colonnes de V ne convergent pas vers les m vecteurs propres de A . Ils chercheront à s'aligner au vecteur propre associé à la plus grande valeur propre en module.

Dans ce cas nous aurons une matrice V quasiment de rang 1 et nous ne pourront plus extraire les différents couples propres.

Ainsi la méthode de subspace iteration fait en sorte d'orthonormaliser à chaque itération ce qui permet de garder les colonnes de V indépendantes et donc une convergence à l'espace propre dominant de dimension m .

Question 5 :

Dans l'algorithme fournit du subspace iteration on calcule la décomposition spectrale complète de $H = V^T A V$. Ceci ne pose aucun problème car :

- H est de dimension $m \times m$, avec m le nombre de vecteurs dans le sous-espace et $m \ll n$.
- Diagonaliser H qui est de complexité ($\mathcal{O}(m^3)$) est négligeable par rapport à $A \cdot V$ qui est de ($\mathcal{O}(n^2 m)$);

Calculer donc la décomposition spectrale complète de H n'est pas problématique car H est une matrice de taille réduite par rapport à A . Cela permet de réaliser des calculs efficaces et rapides, tout en obtenant les informations nécessaires sur les valeurs propres et vecteurs propres dominants de A .

Question 6 : Comparaison des temps d'executions et nombre d'itérations `subspace_iter_v0` et `subspace_iter_v1`

Taille	Type	Méthode	Temps (s)	# Itérations
100	1	subspace_iter_v0	3.680	1629
100		subspace_iter_v1	1.140	87
300	1	subspace_iter_v0	37.47	4869
300		subspace_iter_v1	8.77	678
100	2	subspace_iter_v0	0.150	39
100		subspace_iter_v1	0.060	4
500	2	subspace_iter_v0	2.840	219
500		subspace_iter_v1	1.330	17

Table 6: Comparaison `subspace_iter_v0` et `subspace_iter_v1` pour les matrices de type 1 et 2

Commentaire: On voit clairement que la version de `subspace_iter_v1` est plus efficace que la première avec beaucoup moins de nombre d'itérations et des temps d'exécutions plus rapides.

Question 7 : Identification des étapes de l'algorithme 4

Voici l'algorithme Subspace iteration method v1 with Raleigh-Ritz projection annoté avec en bleu les lignes correspondantes à chaque étape du `subspace_iter_v1`.

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 % lignes 21–22
 Output: n_{ev} dominant eigenvectors V_{out} and the corresponding eigenvalues Λ_{out} . % lignes 120–125

Generate an initial set of m orthonormal vectors $V \in R^{n \times m}$; $k = 0$; $PercentReached = 0$ % lignes 37–48

repeat

- $k = k + 1$ % 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 $PercentReached$ % lignes 63–115

until ($PercentReached > PercentTrace$ or $n_{ev} = m$ or $k > MaxIter$) % ligne 52

2.3 subspace_iter_v2 et subspace_iter_v3 : vers un solveur efficace

Question 8 : Cout de calcul de A^p suivi de $A^p \cdot V$.

Le calcul de A^p suivi de $A^p \cdot V$ est coûteux en terme de flops. En effet si A est une matrice $n \times n$ et V est $n \times m$, alors le calcul de A^p nécessite $(p - 1)$ produit matrice-matrice avec chacune une complexité de $\mathcal{O}(n^3)$ flops.

Ensuite le produit de A^p par V demande encore $\mathcal{O}(n^2m)$ flops. Le coût total sera donc dominé par $\mathcal{O}(pn^2m)$ et ceci pour des valeurs où $m \ll n$.

En effet une autre approche serait d'éviter de calculer A^p explicitement. Le calcul $A^p \cdot V$ pourrait être fait itérativement en effectuant successivement p produits matricielles, de la manière suivante :

$$V \leftarrow A \cdot (A \cdot (\cdots (A \cdot V) \cdots)).$$

Dans ce cas, on effectue p produit matricielles avec un coût chacun de $\mathcal{O}(n^2m)$, avec un total donc de $\mathcal{O}(pn^2m)$ flops ce qui nous évite de calculer et stocker A^p ce qui pourrait être pénible.

Question 10 : Impact du paramètre p .

Méthode	Taille Matrice	p (pour v2)	Temps (s)	Itérations
v0	100	-	2.46	1629
v1	100	-	0.58	87
v2	100	5	0.64	18
v2	100	10	0.63	9
v2	100	20	0.60	5
v2	100	40	0.70	3
v0	200	-	14.04	3192
v1	200	-	2.24	263
v2	200	5	2.65	53
v2	200	10	1.86	27
v2	200	20	1.80	14
v2	200	40	2.99	7

Table 7: Comparaison des 3 méthodes subspace iteration pour $n = 100$ et $n = 200$, matrice de type 1

Commentaire: On voit bien d'après la table que `subspace_iter_v2` effectue beaucoup moins d'itération que `subspace_iter_v0` et `subspace_iter_v1`, ce qui est dû à la boucle `for` introduite pour $(A^p \cdot V)$.

Aussi pour $n = 100$ et $n = 200$, à chaque fois qu'on augmente p , on voit que le nombre d'itérations diminue et atteint plus rapidement la convergence.

Par contre si on augmente p beaucoup, le calcul devient pénible et coûteux pour chaque itération ce qui augmente le temps total ou même parfois n'atteint pas la convergence.

Il faut donc éviter de choisir plus que 40 ou 50 pour p et rester $p = 5-50$ où le temps d'exécution et la convergence attendue est optimale.

Question 11 : Précision `subspace_iter_v1` et `subspace_iter_v2`.

Dans le cas des deux méthodes `subspace_iter_v1` et `subspace_iter_v2` on remarque que la qualité de précision des couples propres mesurée par $\|Av - \beta v\|/|\beta|$, change d'un vecteur propre à l'autre.

En effet dans la `subspace_iter_v1` la convergence n'est pas atteinte de la même manière (rapidité) pour tous les vecteurs propres. Dans le cas de la subspace iteration, celle-ci capte d'abord les vecteurs propres qui sont associés aux plus grandes valeurs propres en module, ce qui fait que ces quelques vecteurs propres dominent la convergence rapidement.

Ainsi, les vecteurs propres restant dans V (ceux associés aux plus petites valeurs propres), vont converger plus lentement avec moins de précision.

Pareil pour `subspace_iter_v2`, ici l'accélération qu'on a ajoutée par rapport à A^p accélère la convergence en général mais n'arrive toujours pas à identifier les vecteur propre dominant qui ont déjà été calculés, des vecteurs propre restant qui pourrait rencontrer des difficultés lors de la convergence, ce qui peut causer une divergence dans certains cas.

Question 12 : Anticipation couples propres subspace_iter_v3.

Dans le cas de la `subspace_iter_v3`, l'idée serait de isoler les couples propre déjà traité et qui ont déjà convergé, afin de ne plus les recalculer dans les prochaines itération ce qui évite d'orthonormaliser et de faire la projection de Rayleigh-Ritz sur des vecteurs déjà traités.

Ceci devrait donc augmenter la précision et stabilité , car seulement les vecteurs propres qui non pas encore convergé seront modifier dans les prochaines itération, cela diminuera donc le coût de calcul, surtout quand on a un grand nombre de couples propres.

2.4 Expérimentations numériques

Question 14 : Comparaison 4 types de matrices.

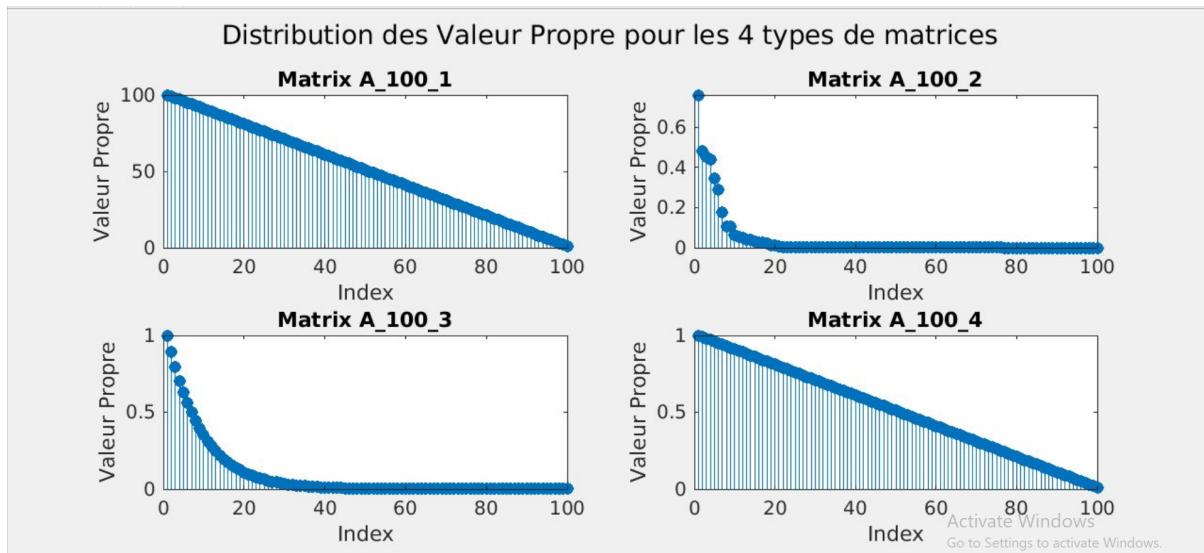


Figure 1: Distributions des valeur propres pour les 4 types de matrice.

Dans cette figure on voit bien les distributions des valeurs propres pour les 4 types de matrices de taille 100×100 .

- **imat = 1:** Les valeurs propres décroît linéairement de 100 à 1. Convergence plus lente que les autres.
- **imat = 2:** Ici on voit d'abord les grandes valeurs propre qui décroît fortement, ce qui montre que les couples propres dominants donc isolé rapidement. Convergence rapide des vecteurs propres dominants.
- **imat = 3:** Proche du cas où $\text{imat} = 2$ avec une décroissance plus légère.
- **imat = 4:** Pareil que pour $\text{imat} = 1$ avec des valeurs répartis de 0 à 1.

On voit bien comment la distribution des valeurs propres affectent la convergence, qui est plus rapide quand les cas 2 et 3.

Question 15 : Comparaison performances algorithmes.

Dans le tableau ci dessous on peut voir les résultat trouvées pour les différente méthodes, pour différent types de matrices de différentes tailles.

Matrice Type	Taille	Méthode	Temps (s)	Itérations	Couple Qualité	Valeur Qualité
1	100	v0	3.02	1629	1.22e-10	5.68e-16
	100	v1	0.54	87	4.22e-09	1.95e-15
	100	v2	0.52	18	2.26e-09	2.39e-15
	100	v3	0.47	16	1.00e-02	1.56e-14
	100	eig	0.0231	—	—	—
1	300	v0	34.18	4869	4.38e-11	3.79e-16
	300	v1	5.82	678	7.98e-14	8.00e-14
	300	v2	6.35	136	7.91e-14	7.93e-14
	300	v3	7.19	147	3.33e-03	1.27e-13
	300	eig	0.0742	—	—	—
2	100	v0	0.11	39	6.34e-10	4.38e-16
	100	v1	0.07	4	5.29e-09	4.38e-16
	100	v2	0.06	1	5.18e-08	2.92e-16
	100	v3	0.07	1	3.68e-01	2.92e-16
	100	eig	0.0428	—	—	—
2	300	v0	0.78	104	3.79e-10	3.71e-16
	300	v1	0.21	11	2.48e-09	3.71e-16
	300	v2	0.29	3	3.11e-12	1.24e-16
	300	v3	0.36	3	1.20e-09	1.33e-15
	300	eig	0.8539	—	—	—
3	100	v0	0.42	150	4.58e-10	4.44e-16
	100	v1	0.11	9	4.20e-09	2.22e-16
	100	v2	0.14	2	4.68e-10	0.00e+00
	100	v3	0.12	2	1.94e-10	0.00e+00
	100	eig	0.0299	—	—	—
3	300	v0	3.81	449	1.77e-10	3.33e-16
	300	v1	0.41	22	1.16e-08	4.44e-16
	300	v2	0.48	5	1.02e-09	1.33e-15
	300	v3	0.51	5	7.18e-09	1.33e-15
	300	eig	0.0773	—	—	—
4	100	v0	3.29	1629	1.22e-10	1.11e-16
	100	v1	0.96	87	4.22e-09	2.22e-15
	100	v2	1.02	18	2.26e-09	2.92e-15
	100	v3	0.91	16	1.00e-02	1.48e-14
	100	eig	0.03497	—	—	—
4	300	v0	37.51	4901	4.36e-11	0.00e+00
	300	v1	11.49	682	7.55e-14	7.54e-14
	300	v2	12.04	137	7.61e-14	7.62e-14
	300	v3	12.80	148	3.31e-03	1.29e-13
	300	eig	0.7377	—	—	—

Table 8: Comparaison des performances des méthodes pour différents types et tailles de matrices

On voit bien d'après la table que en terme de :

Temps d'exécution:

- Pour les 2 tailles de matrices ($n = 100$) et ($n = 300$), `eig` est beaucoup plus plus rapide que les autres méthodes.
- Parmis les méthodes de subspace , `v1` and `v2` sont en général les plus rapides et consistent.

Convergence:

- `v0` est la plus lente pour atteindre la convergence et effectue un très grand nombre d'itération par rapport aux autres ce qui devient très pénible plus on a de grande matrices.

- v1, atteint la convergence plus vite que v0 avec beaucoup moins d’itérations. v2 et v3 diminuent encore plus leur nombre d’itération par rapport aux autres , avec v2 plus rapide en général. On voit aussi que v3 a plus du mal sur les précision probablement du a l’étape de déflation qu’on a ajouter (comme on peut le voir dans la colonne Couple Qualité).

Impact du type de matrice utilisé:

- Pour les matrice de type `imat = 2` and `imat = 3` les méthode de subspace 1 et 2 sont efficaces en terme de temps et de précision, peut être a cause de leur forme qui sépare bien les valeur propre dominantes.
- Pour les matrices de types `imat = 1` and `imat = 4` cela est plus compliqué , on voit que le temps d’execution est plus lent et moin de precision.
- `eig` est rapide et efficace pour les 4 types de matrice.

Précision:

- v0, v1, and v2 sont en general precis , avec des valeur proche du epsilon, mais v3 n’est pas aussi precis que les autres avec des qualité pour les couples et valeur propres proche de 10^{-2} ou pire.

En conclusion on peut dire que :

- `eig` reste le meilleur surtout en terme de convergence.
- v1 and v2 sont les meilleur parmi les subspace en terme de convergence et précision.

3 Application à la compression d’image

3.1 Rappels et Introduction

Comme nous l’avons vu avant , les images sous formats numérique sont représentés par un ensemble de grande matrices et chaque valeur représente la couleur ou l’intensité de gris d’un pixel. Dans certains cas cette représentation peut être simple pour travailler sur les images mais aussi pénible et nécessite des traitements supplémentaires. Quand on compresse les images, on cherche a garder dans les matrices les éléments essentiels pour visualiser l’image mais on essaye de diminuer la taille pour stocker l’image plus efficacement.

On trouve plusieurs méthodes intéressantes de nos jours pour faire tout cela et notamment les méthodes *linéaires et algébriques*. En effet, la majorité des images peuvent être décrites et représentées par un petit nombre de mode dominants (ce qui complète ce qu’on avait vu pour les méthodes d’itération avec les couples propre dominants). Donc pour calculer ces couples il suffit d’extraire des valeurs singulières de la matrice de l’image en question et les couples propres dominants.

Dans cette partie du rapport nous verrons donc:

1. La notion d'approximation de faible rang avec décomposition en valeurs singulières et théorème de la meilleure approximation.
2. Comment la compression est mis en œuvre sur différentes images de BD en gris avec l'extraction des couples propres dominant grâce aux algorithmes développés dans la Partie 1, et la reconstruction des images compressé.

3.2 Compression d'images

Question 1 : Dimensions de (Σ_k, U_k, V_k) .

Les dimensions de chaque élément du triplet sont:

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

Dans le cas où $q < p$:

Dans ce cas, la SVD a lieu à travers $M = I^T I \in R^{p \times p}$, ce qui donne :

- $V_k \in R^{p \times k}$
 - $U_k \in R^{q \times k}$ est calculé avec :
- $$U_k = \frac{1}{\sigma_i} I V_k$$
- $\Sigma_k \in R^{k \times k}$

Donc les dimensions restent inchangées: $U_k \in R^{q \times k}$, $\Sigma_k \in R^{k \times k}$, $V_k \in R^{p \times k}$.

Question 2 : Reconstruction d'images

Dans cette section, nous cherchons à reconstruire une image compressée à l'aide de différentes méthodes de calcul de valeurs propres, notamment `eig`, les méthodes de la puissance itérée (`power_v11`, `power_v12`) et les quatre variantes de la méthode d'itération de sous-espace (`v0` à `v3`).

L'image utilisée est illustrée ci-dessous :



Les paramètres utilisés pour toutes les méthodes sont les suivants :

- Tolérance : `eps = 1e-8`
- Nombre maximal d’itérations : `maxit = 10000`
- Taille de l’espace de recherche : `search_space = 400`
- Pourcentage de trace à atteindre : `percentage = 0.99`
- Nombre de redémarrages pour les méthodes v2 et v3 : `puiss = 1`

Résultats visuels obtenus avec `power_v11`

Nous présentons ci-dessous les images reconstruites à l’aide de la méthode `power_v11` pour différentes valeurs du rang k .

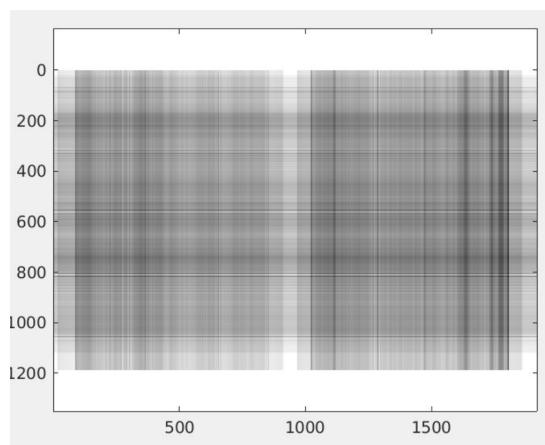


Figure 2: Image reconstruite avec $k = 1$

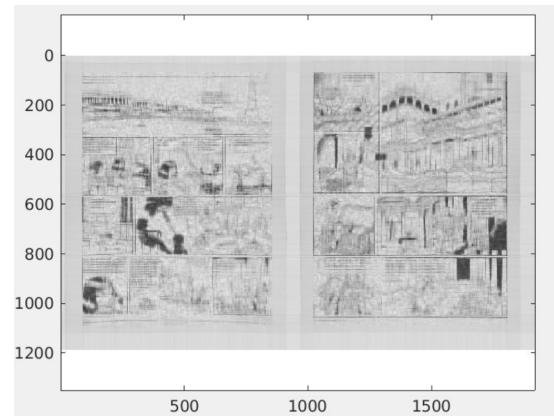


Figure 3: Image reconstruite avec $k = 41$

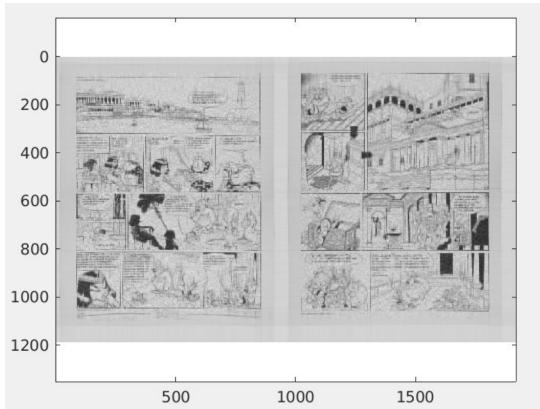


Figure 4: Image reconstruite avec $k = 81$

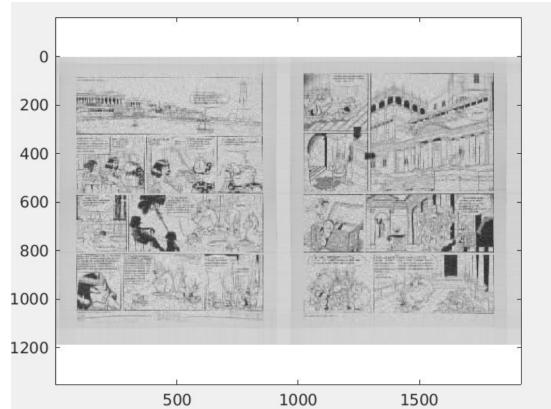


Figure 5: Image reconstruite avec $k = 121$

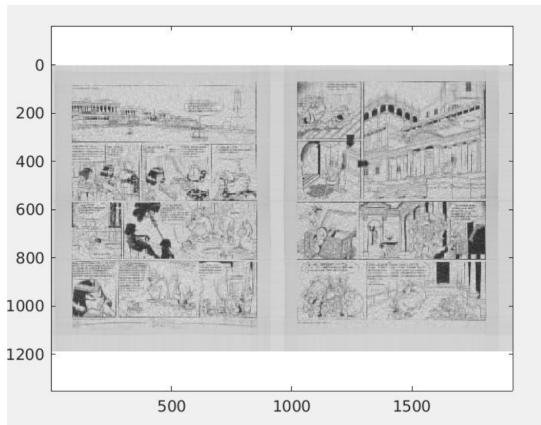


Figure 6: Image reconstruite avec $k = 201$

Comparaison de la reconstruction de l'image entre les différentes méthodes

Dans cette section, nous comparons les résultats de la reconstruction d'image en utilisant différentes méthodes de calcul des valeurs propres. Les images reconstruites sont présentées pour chaque méthode avec des valeurs spécifiques de k .

Méthode `eig`

La méthode `eig` est une approche classique pour le calcul des valeurs propres et vecteurs propres d'une matrice. Elle fournit une décomposition spectrale complète, ce qui permet d'obtenir une reconstruction d'image de très haute qualité et parvient à capturer presque tous les détails de l'image originale. Cependant, cette précision a un coût : la méthode nécessite une valeur très élevée de k , ce qui la rend computationnellement intensive. Cela signifie que, bien qu'elle soit idéale pour des reconstructions de haute qualité, elle peut ne pas être pratique pour des applications nécessitant des calculs rapides ou des ressources limitées.



Figure 7: Reconstruction avec `eig`

Méthode `power_v11`



Figure 8: Reconstruction avec `power_v11`,

Méthode `power_v12`

La méthode `power_v12` est la version améliorée de `power_v11`, optimisée pour réduire le nombre de multiplications matricielles par itération.



Figure 9: Reconstruction avec power_v12

Méthodes subspace_iter_v0 à subspace_iter_v3

Les méthodes d’itération de sous-espace, allant de `subspace_iter_v0` à `subspace_iter_v3`, représentent une approche plus sophistiquée pour le calcul des valeurs propres. Ces méthodes fonctionnent en itérant sur un sous-espace de vecteurs et en utilisant des techniques comme la projection de Rayleigh-Ritz pour extraire les valeurs propres dominantes. Les figures ci-dessous montrent les résultats de reconstruction pour chaque version de la méthode. On observe une amélioration progressive de la qualité de reconstruction à mesure que les versions deviennent plus avancées. Ces méthodes sont particulièrement efficaces pour les grandes matrices et offrent un bon compromis entre coût computationnel et qualité de reconstruction.



Figure 10: Reconstruction avec subspace_iter_v0



Figure 11: Reconstruction avec subspace_iter_v1



Figure 12: Reconstruction avec `subspace_iter_v2`



Figure 13: Reconstruction avec `subspace_iter_v3`

Analyse comparative

L’analyse comparative des différentes méthodes révèle que la méthode `eig` offre la meilleure qualité de reconstruction, mais à un coût élevé en termes de k . Les méthodes de puissance (`power_v11` et `power_v12`) sont limitées par la réduction du pourcentage cible, ce qui affecte la qualité de reconstruction mais arrive quand même à reconstruire les images. En revanche, les méthodes d’itération de sous-espace (`subspace_iter_v0` à `subspace_iter_v3`) montrent une amélioration progressive de la qualité de reconstruction, avec des résultats particulièrement satisfaisants pour les versions les plus avancées. Ces méthodes sont plus efficaces en termes de coût computationnel tout en offrant une bonne qualité de reconstruction, ce qui les rend adaptées pour des applications pratiques nécessitant un équilibre entre performance et ressources.

Visualisation RMSE

On a tracer les différentes **RMSE** pour chaque méthode avec les différentes valeur de k , voici nos résultats :

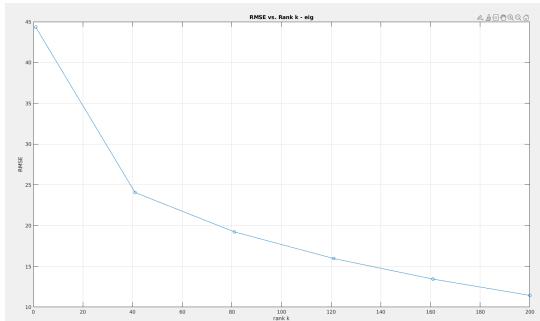


Figure 14: Variation de la différence I et I_k pour la méthode `eig`

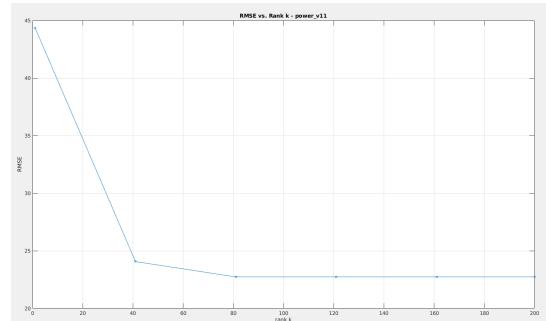


Figure 15: Variation de la différence I et I_k pour la méthode `power_v11`

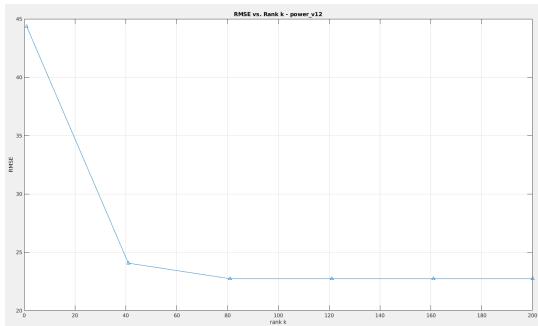


Figure 16: Variation de la différence I et I_k pour la méthode `power_v12`

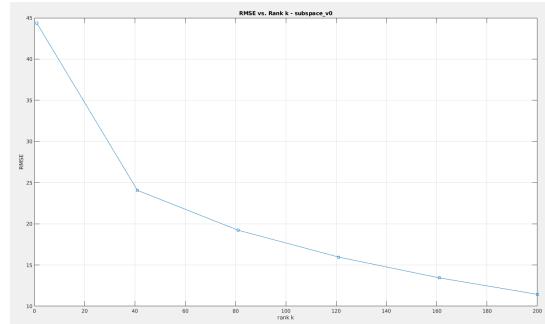


Figure 17: Variation de la différence I et I_k pour la méthode `subspace_iter_v0`

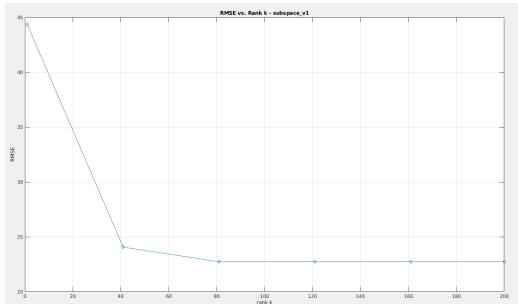


Figure 18: Variation de la différence I et I_k pour la méthode `subspace_iter_v1`

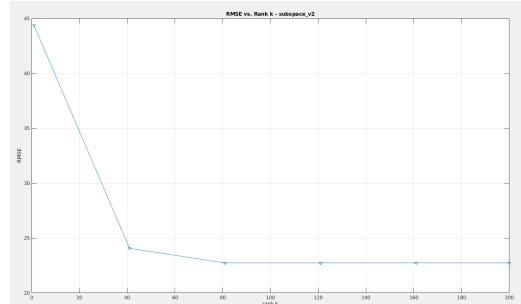


Figure 19: Variation de la différence I et I_k pour la méthode `subspace_iter_v2`

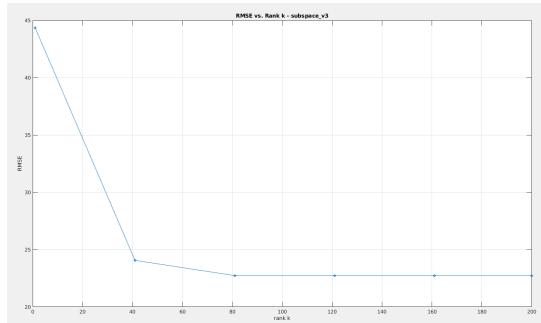


Figure 20: Variation de la différence I et I_k pour la méthode `subspace_iter_v3`

Comparaison des performances (RMSE). Les courbes RMSE obtenues pour les différentes méthodes montrent une tendance globale similaire : une diminution rapide de l'erreur aux faibles rangs suivie d'une stagnation. La méthode `eig` reste la plus performante en termes de qualité de reconstruction, avec une décroissance régulière du RMSE

même pour des valeurs élevées de k . Les méthodes `power_v11` et `power_v12` convergent rapidement vers un plateau, mais n'atteignent pas les mêmes niveaux d'approximation que `eig`. Les versions `subspace_v1`, `v2` et `v3` offrent des performances proches, avec une convergence vers une erreur minimale dès $k = 80$, ce qui témoigne de leur efficacité pour extraire les modes dominants. En revanche, `subspace_v0` se distingue par une convergence plus lente, similaire à celle de `eig`, mais sans en atteindre la précision finale.

Question Bonus: Reconstruction d'images en couleur

Nous avons essayé de construire les images en noir et blanc dans l'étude d'avant, nous allons essayer de le faire maintenant mais avec l'image colorié ci-dessous :



Figure 21: Image colorié avant la compression

Pour cela nous utilisons le même algorithme que pour la reconstruction d'image en noir et blanc mais cette fois pour les 3 canaux RGB et nous superposons les 3 images reconstruites pour avoir la version finale.

Bien que les méthodes que nous avons mises en œuvre soient capables de reconstruire une image couleur, leur temps d'exécution élevé peut poser un obstacle à leur utilisation dans des applications pratiques qui exigent une reconstruction rapide et précise des images couleur.

Cependant, nous avons remarqué que la reconstruction d'une image couleur prend beaucoup plus de temps que celle d'une image en niveaux de gris, en raison de leur volume plus important et du traitement de trois matrices au lieu d'une seule.

Voici par exemple la reconstruction de nos images pour `subspace_iter_v2`

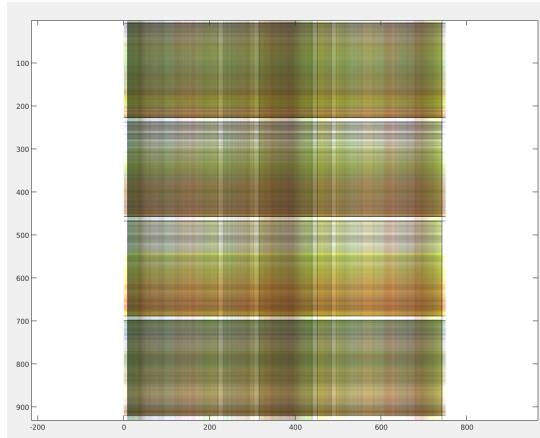


Figure 22: Image reconstruite avec $k = 1$



Figure 23: Image reconstruite avec $k = 41$



Figure 24: Image reconstruite avec $k = 81$



Figure 25: Image reconstruite avec $k = 121$



Figure 26: Image reconstruite avec $k = 161$



Figure 27: Image reconstruite avec $k = 201$

4 Conclusion

Dans ce rapport, nous avons exploré diverses méthodes pour le calcul des valeurs propres et leur application à la compression d'images. Nous avons commencé par examiner les limites de la méthode de la puissance itérée et avons introduit des améliorations avec la méthode de la puissance itérée améliorée (`power_v12`). Ces améliorations ont montré des gains significatifs en termes de temps d'exécution, particulièrement pour les grandes matrices.

Nous avons ensuite étudié les méthodes d'itération de sous-espace, en commençant par les versions de base (`subspace_iter_v0`) et en progressant vers des versions plus avancées (`subspace_iter_v3`). Ces méthodes ont démontré une efficacité croissante, tant en termes de nombre d'itérations nécessaires que de précision des valeurs propres calculées. Les expérimentations numériques ont révélé que les méthodes d'itération de sous-espace, en particulier les versions plus avancées, offrent un bon compromis entre coût computationnel et qualité des résultats.

Dans la partie sur l'application à la compression d'image, nous avons appliqué ces méthodes pour reconstruire des images à partir de leur représentation compressée. Les résultats ont montré que les méthodes d'itération de sous-espace permettent une reconstruction de qualité satisfaisante avec un coût computationnel raisonnable. La méthode `eig`, bien qu'offrant la meilleure qualité de reconstruction, est beaucoup plus coûteuse en termes de valeurs de k nécessaires.

Pour la reconstruction d'images en couleur, Nous arrivons bien à reconstruire l'image pour la plupart des méthodes , avec un peu de mal pour les power v11 et v12 qui sont des méthodes permettent de calculer rapidement les premières valeur singulière dominante mais ne garantissent pas une approximation optimale de l'image compressé contrairement a la méthode SVD ou aux 3 méthodes de subspace qui font bien le job.

En conclusion, ce rapport a mis en lumière l'importance de choisir la méthode appropriée en fonction des contraintes computationnelles et des exigences de qualité. Les

méthodes d’itération de sous-espace, en particulier les versions les plus avancées, se sont révélées être des outils efficaces pour la compression d’images, offrant un bon équilibre entre performance et coût.