Projet Calcul Scientifique

Sakun Withanage Perera
Sidi Mohamed Raid GHALLABI

April 2025

Table des matières

1 Introduction

Dans ce projet, il s'agit d'implémenter et de tester différents algorithmes de calcul de couples propres de matrices, de tailles variées, présentant des valeurs propres plus ou moins uniformément réparties.

2 Limites de la méthode de la puissance itérée

2.1 Comparaison avec la fonction eig

Nous comparons ici la méthode de la puissance iterée avec la fonction eig de matlab.

Dimension	Type	Eig (en s)	Power Method (en s)
1000x1000	Type 1	1.8	non convergence
	Type 2	1.2	17
	Type 3	1.1	83
	Type 4	2	non convergence
200x200	Type 1	5×10^{-02}	8×10^{-01}
	Type 2	4×10^{-02}	1×10^{-01}
	Type 3	4×10^{-02}	6×10^{-02}
	Type 4	6×10^{-02}	8×10^{-01}
100x100	Type 1	3×10^{-02}	2×10^{-01}
	Type 2	2×10^{-02}	3×10^{-02}
	Type 3	1×10^{-02}	3×10^{-02}
	Type 4	2×10^{-02}	1×10^{-01}
50x50	Type 1	0	5×10^{-02}
	Type 2	0	3×10^{-02}
	Type 3	0	2×10^{-02}
	Type 4	0	6×10^{-02}
10x10	Type 1	0	1×10^{-02}
	Type 2	0	0
	Type 3	0	1×10^{-02}
	Type 4	0	1×10^{-02}

TABLE 1 – Comparaison entre les méthodes `Eig` et `power_v11`

Dans ce tableau nous voyons assez clairement que la fonction eig est plus rapide que l'algorithme de puissance iterée pour quasi tout taille et type de matrice, pour des matrice de petite taille la différence reste assez minime mais la différence s'accentue pour les matrice de grande taille. Pour certaine la convergence n'est même pas atteinte par exemple pour les matrice de type imat1 et imat4 de taille 1000×1000. La qualité des valeurs et vecteurs propres semble néanmoins similaires.

2.1.1 Modification : power v_12

Nous avons simplifié l'algorithme de la méthode de la puissance en le réorganisant pour qu'il n'effectue qu'une seule multiplication matrice–vecteur par itération comme suivant :

$1 : z = A.v$
$2 : \beta = v^T.z$
$3 :$ **repeat**
$4 :$ $\quad v = \frac{z}{\|z\|}$
$5 :$ $\quad z = A.v$
$6 :$ $\quad \beta_{old} = \beta$
$7 :$ $\quad \beta = v^T.z$
$8 :$ **until** $\frac{\|\beta - \beta_{old}\|}{\|\beta_{old}\|} < \epsilon$
$9 : \lambda_\mathbf{1} = \beta$ **and** $\mathbf{v_1} = \mathbf{v}$

Dimension	Type	Eig	Power v_11	Power v_12
1000x1000	Type 1	1.8	non convergence	non convergence
	Type 2	1.2	17	9
	Type 3	1.1	81	40
	Type 4	2	non convergence	non convergence
200x200	Type 1	5×10^{-02}	8×10^{-01}	4.3×10^{-01}
	Type 2	4×10^{-02}	1×10^{-01}	1×10^{-01}
	Type 3	4×10^{-02}	6×10^{-02}	3.1×10^{-01}
	Type 4	6×10^{-02}	8×10^{-01}	4.4×10^{-01}

TABLE 2 – Comparaison entre les méthodes `Eig` `power v_11` et `power v_12`

Ainsi dans la Table 2 nous voyons assez clairement que le temps d'exécution est environ divisé par deux en modifiant légèrement l'algorithme de power v_11. Cependant on voit clairement que ceci n'est pas suffisant pour atteindre les performances de la fonction eig de matlab (nottement pour les matrice de grande taille), dans le cas des matrices de type 1 et 4 de taille 1000×1000, on n'atteint même oas la convergence comme pour power v_11.

2.1.2 Les inconvénients de la méthode des puissances

Cette méthode présente plusieurs inconvénients. En effet, elle nécessite un grand nombre d'itérations pour converger, ce qui la rend relativement coûteuse en temps et en ressources de calcul. Par ailleurs, le choix aléatoire du vecteur initial peut ralentir considérablement l'algorithme si celui-ci est trop éloigné du vecteur des valeur propres recherchées.

2.2 Généralisation de la méthode de la puissance pour le calcul des vecteurs de l'espace propre dominant

2.2.1 subspace_iter_v0 : une méthode de base pour calculer un espace propre dominant

Question 4 (application de l'algoritme de puissance itérée à un ensemble de m vecteurs) :

Si l'on applique l'algorithme 1 (puissance) à un ensemble de m vecteurs au lieu d'un seul, il converge vers une matrice dont chaque colonne est un vecteur propre associé à la même valeur propre, et non vers m vecteurs propres associés à des valeurs propres différentes.

Question 5

Si on prend la variante de cette méthode incluant la matrice H, il n'y a pas de difficulté particulière à effectuer sa décomposition spectrale. En effet, H est une matrice de dimension $m \times m$, ce qui est plus petit que la matrice A, de dimension $n \times n$. Par conséquent, cette opération n'est pas un obstacle, car elle n'ajoute qu'un surcoût négligeable en temps de calcul par rapport aux autres étapes de l'algorithme.

Question 6

Voir le fichier matlab

2.2.2 subspace_iter_v1 : version améliorée utilisant la projection de Rayleigh-Ritz

Question 7 :

Algorithm 1 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 $V \in \mathbb{R}^{n \times m}$; - **ligne 47**
$k = 0$; - **ligne 38**
$PercentReached = 0$ - **ligne 45**
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 70**
until ($PercentReached > PercentTrace$ or $n_{ev} = m$ or $k > MaxIter$) - **ligne 52**

3 subspace_iter_v2 et subspace_iter_v3 : vers un solveur plus efficace

Ici, notre objectif est de construire un algorithme qui combine à la fois l'approche par blocs et la méthode de déflation, afin d'accélérer la convergence du solveur.

3.1 Approche par block : subspace_iter_v2

Question 8 : Coût en FLOPs du calcul de A^p puis $A^p \cdot V$

Soit $A \in \mathbb{R}^{n \times n}$ une matrice carrée, $V \in \mathbb{R}^{n \times m}$ une matrice de vecteurs colonnes, et $p \in \mathbb{N}^*$.

Calcul explicite de A^p puis $A^p \cdot V$:

— Calcul de A^p par $p-1$ multiplications successives de matrices $n \times n$:

Il faut effectuer pour calculer chaque coefficient de la matrice n multiplications et n-1 additions et il y a n^2 coefficients dans une matrice $n \times n$ ainsi il y a $n^2(2n-1) = 2n^3 - n^2$ opérations par multiplication de deux matrices et ainsi avec p-1 multiplications :

$$\text{Coût} = 2(p-1)n^3 - (p-1)n^2 \text{ FLOPs}$$

On peut arrondir ceci à $2(p-1)n^3$ terme dominant asymptotiquement (pour les grand n)

— Multiplication finale $A^p \cdot V$: (avec le même raisonnement et arrondi)

$$\text{Coût} = 2n^2 m \text{ FLOPs}$$

Total :

$$\boxed{2(p-1)n^3 + 2n^2 m \quad \text{FLOPs}}$$

Méthode optimisée :

Plutôt que de calculer A^p explicitement, on effectue p produits successifs de la forme :

$$A^p \cdot V = A \cdot (A \cdot (\dots (A \cdot V)))$$

Chaque produit $A \cdot V$ coûte $2n^2m$ FLOPs, et est répété p fois :

$$\boxed{2pn^2m \quad \text{FLOPs}}$$

Or $2pn^2m = 2n^2m + 2(p-1)n^2m$

et vu que $m < n$ alors $2(p-1)n^2m < 2(p-1)n^3$

donc $2pn^2m < 2(p-1)n^3 + 2n^2m$

Question 9

Voir le fichier matlab

Question 10

L'algorithme amélioré `subspace_iter_v2` s'avère plus rapide que `subspace_iter_v1`, mais introduit un nouveau paramètre crucial : p. On observe qu'une augmentation de p permet de réduire significativement le nombre d'itérations nécessaires à la convergence, rendant ainsi le calcul des vecteurs propres plus efficace. Toutefois, si p devient trop grand, `subspace_iter_v2` peut perdre en stabilité. En effet, lorsque $A^p \cdot V$ tend vers une matrice de rang 1 (d'après la section 2.1 de l'énoncé, l'action répétée de A amplifie uniquement la direction associée à la valeur propre dominante, rendant les colonnes de la matrice $A^p.v$) l'orthonormalisation devient impossible, compromettant ainsi la poursuite de l'algorithme.

3.2 Méthode de la déflation : subspace_iter_v3

Question 11 :

Un autre aspect important de cette adaptation concerne l'impact du critère d'arrêt sur la qualité des couples propres obtenus. Dans la version `subspace_iter_v1`, ce critère est évalué individuellement pour chaque couple propre, ce qui présente l'avantage d'être peu coûteux, mais peut entraîner une variabilité dans la précision des résultats. À l'inverse, la version `subspace_iter_v3` évalue le critère de convergence de manière collective sur un ensemble de couples propres, ce qui permet une évaluation plus cohérente et précise.

Question 12 :

Cependant, `subspace_iter_v3` interrompt l'itération sur un vecteur dès qu'il est considéré comme convergé, ce qui l'empêche de bénéficier d'itérations supplémentaires qui pourraient améliorer sa précision. On peut donc supposer que, dans certains cas, la qualité des résultats obtenus avec `subspace_iter_v3` sera légèrement inférieure à celle de `subspace_iter_v2`, qui continue à affiner tous les vecteurs en parallèle.

Question 13 :

Voir le fichier matlab

4 Applications numériques

Question 14 :

Voici des courbes représentant la distribution de valeurs propres

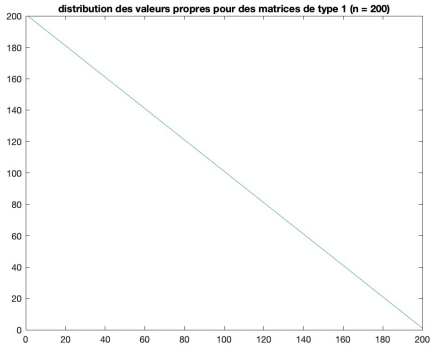


FIGURE 1 – Distribution des valeurs propres pour les matrices de type 1

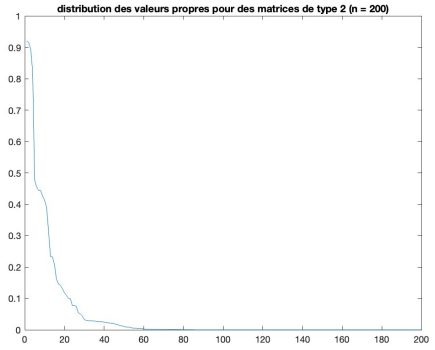


FIGURE 2 – Distribution des valeurs propres pour les matrices de type 2

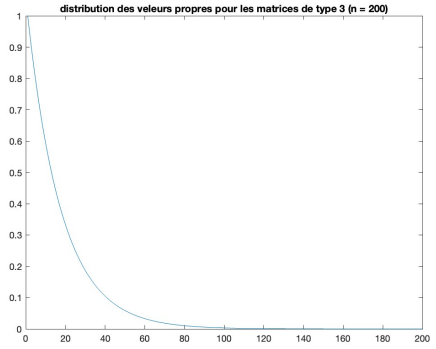


FIGURE 3 – Distribution des valeurs propres pour les matrices de type 3

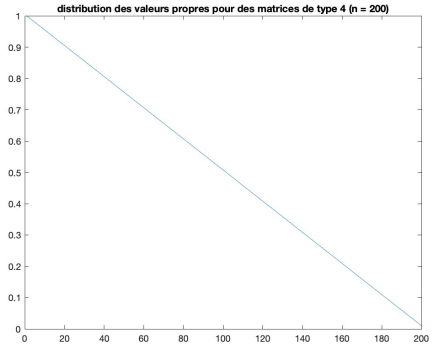


FIGURE 4 – Distribution des valeurs propres pour les matrices de type 4

Nous pouvons observer que la distribution des valeurs propres pour les matrices de type 1 et 4 est linéaire tandis que pour les matrices de type 2 et 3, on retrouve une distribution exponentielle. Ceci expliquerait le fait que les type 2 et 3 sont plus rapide à traiter comparés aux types 1 et 4. En effet, la norme des valeurs propres décroît très rapidement, ce qui permet d'atteindre la précision imposée par le quotient de Rayleigh avec un nombre réduit de valeurs propres. Par conséquent, les premières valeurs propres dominantes convergent plus vite, ce qui diminue le nombre total d'itérations nécessaires.

Question 15 :

Performances subspace_iter_v0 :

Type matrice	100x100	200x200
type 1	0.18s	0.64s
type 2	0.01s	0.16s
type 3	0.02s	0.07s
type 4	0.24s	0.7s

TABLE 3 – Temps de calcul pour différents types et tailles de matrices (subspace_iter_v0)

Performances subspace_iter_v1 :

Type matrice	100x100	200x200
type 1	0.04s	0.15s
type 2	0.01s	0.03s
type 3	0.01s	0.01s
type 4	0.06s	0.35s

TABLE 4 – Temps de calcul pour différents types et tailles de matrices (subspace_iter_v1)

Performances subspace_iter_v2 pour p = 10 :

Type matrice	100x100	200x200
type 1	0.01s	0.04s
type 2	convergence non atteinte	0.01s
type 3	0.00s	0.03s
type 4	0.02s	0.05s

TABLE 5 – Temps de calcul pour différents types et tailles de matrices (subspace_iter_v2)

Performances subspace_iter_v3 :

Type matrice	100x100	200x200
type 1	0.01s	0.65s
type 2	0.02s	0.02s
type 3	0.01s	0.01s
type 4	0.02s	0.74s

TABLE 6 – Temps de calcul pour différents types et tailles de matrices (subspace_iter_v3) pour p = 10

Les résultats précédents montrent que les algorithmes que nous avons développés deviennent progressivement plus rapides grâce aux optimisations successives, qui ont permis une amélioration significative des performances par rapport à la méthode de la puissance itérée vue en cours.

Par ailleurs, nos algorithmes convergent systématiquement, contrairement à la version naïve de la méthode de la puissance itérée.

On observe également que notre dernière version est plus rapide que la fonction `eig` de MATLAB pour les matrices de type 2 et 3, caractérisées par une décroissance rapide de leurs valeurs propres. Cette propriété permet d'éviter de calculer l'ensemble du spectre de A, ce qui accélère considérablement l'algorithme.

En revanche, pour les matrices de type 1 et 4, dont les valeurs propres décroissent plus lentement et de façon linéaire, notre implémentation reste moins performante que la fonction `eig` de MATLAB.

5 Application au compression d'image (Partie 2)

Dans cette deuxième partie du projet, nous étudions une application concrète : la reconstitution d'image a partir des algorithmes crées dans la partie 1.Ce travail repose sur l'utilisation de la *décomposition en valeurs singulières* (SVD) et du *théorème de la meilleure approximation de rang k*.

Objectif

L'objectif est de reconstruire une image à partir d'une approximation de rang k, ce qui permet de la compresser en ne conservant qu'une petite partie de l'information (les composantes principales), tout en maintenant une qualité visuelle acceptable.

Pour cela, nous appliquerons la SVD à l'image $I \in \mathbb{R}^{q\times p}$ pour obtenir les matrices U, Σ et V, puis reconstruire l'image à l'aide du triplet (U_k, Σ_k, V_k) formé des k premiers vecteurs et valeurs singulières.

- $\Sigma_k \in \mathbb{R}^{k\times k}$ est une matrice diagonale contenant les k plus grandes valeurs singulières de l'image. Ces valeurs sont les racines carrées des k plus grandes valeurs propres de I^TI (ou II^T).
- $U_k \in \mathbb{R}^{q\times k}$ est la matrice contenant les k premiers vecteurs propres de II^T. Ces vecteurs sont appelés vecteurs singuliers gauches.
- $V_k \in \mathbb{R}^{p\times k}$ est la matrice contenant les k premiers vecteurs propres de I^TI, appelés vecteurs singuliers droits.

Les dimensions des éléments du triplet (Σ_k, U_k, V_k) sont : $\Sigma_k : k \times k$ (avec k éléments non nuls), $U_k : q \times k$ (soit qk éléments), et $V_k : p \times k$ (soit pk éléments).

La reconstruction de l'image compressée I_k s'effectue alors par :

$$I_k = U_k\Sigma_kV_k^T$$

Ce triplet permet une compression efficace, car il nécessite de stocker seulement $k(p+q+1)$ valeurs au lieu des pq valeurs initiales de la matrice image complète, lorsque $k \ll \min(p, q)$.

La courbe ci-dessus met en évidence la baisse progressive de l'erreur quadratique moyenne (RMSE) lorsque le rang k de l'approximation augmente. Cela signifie qu'en conservant un plus grand nombre de composantes principales issues de la décomposition en valeurs singulières (SVD), la reconstruction de l'image gagne en fidélité. Toutefois, à partir d'un certain seuil, l'ajout de nouvelles composantes n'apporte qu'une amélioration marginale, les informations les plus significatives étant déjà capturées par les premières composantes.

L'évaluation des performances en temps de calcul met en avant les écarts notables entre les différentes approches utilisées pour la reconstruction d'image. Les méthodes directes, telles que la décomposition SVD classique et l'utilisation de la fonction `eig` de MATLAB, se distinguent par leur rapidité, avec des durées d'exécution avoisinant les 0.7 secondes grâce aux optimisations internes de MATLAB.

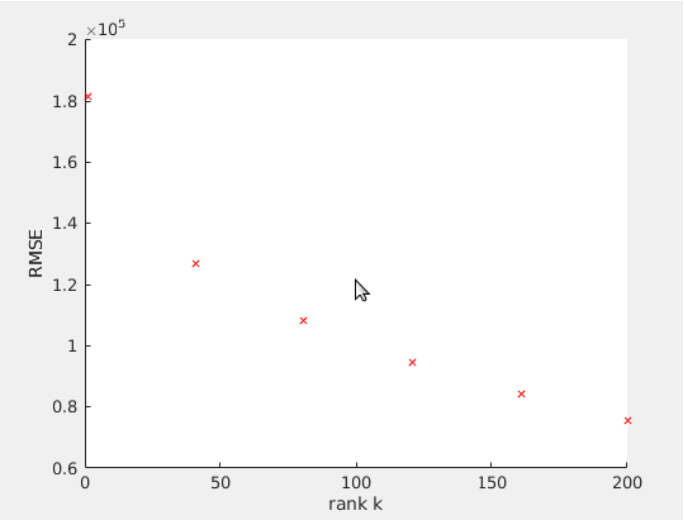
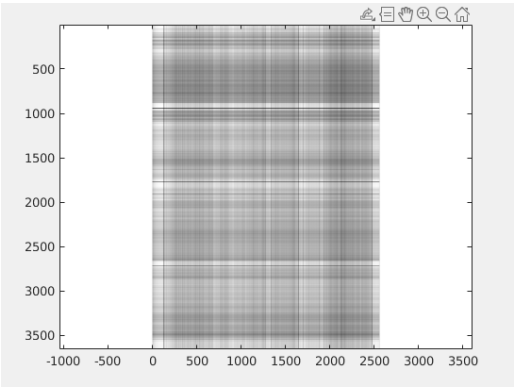
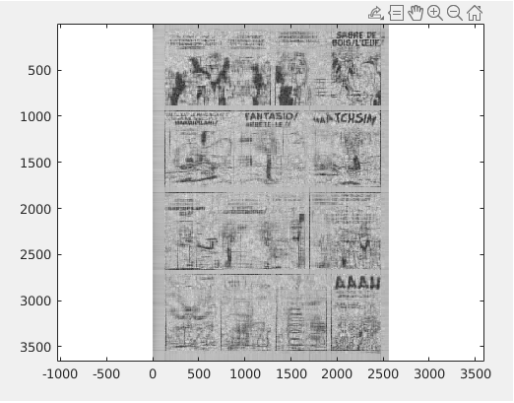
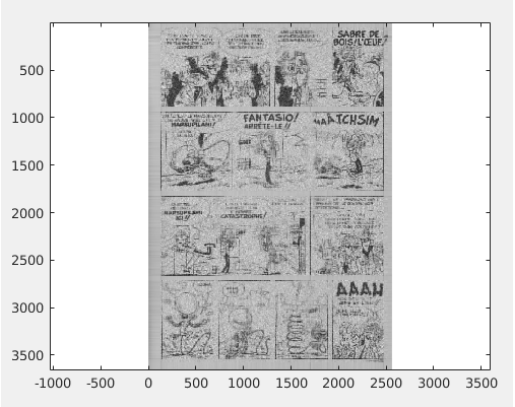
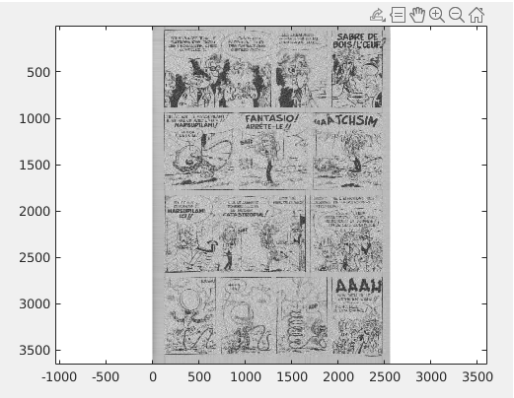


FIGURE 5 – Évolution de l'erreur RMSE en fonction du rang k de l'approximation.



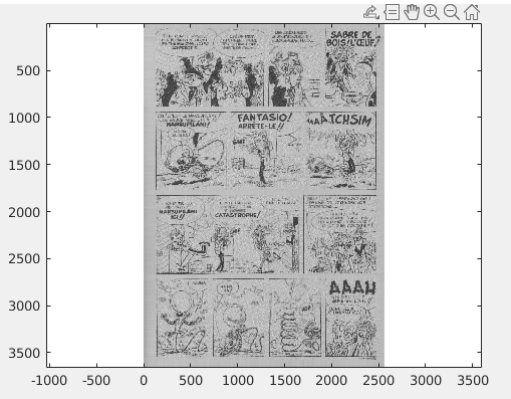


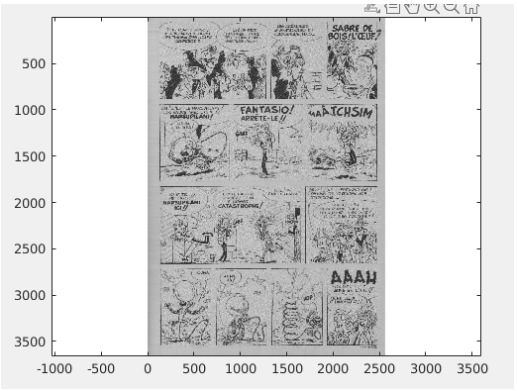




À l'inverse, les variantes de l'itération sur les sous-espaces présentent des temps de traitement beaucoup plus importants. Les versions v0 et v1 nécessitent respectivement environ 95 et 99 secondes, tandis que la version v2 et v3 se montre plus efficiente, avec un temps réduit à environ 22 secondes.

Ces résultats montrent que, malgré l'intérêt théorique des méthodes itératives dans certains contextes, les méthodes directes restent globalement préférables pour la compression d'images lorsque la rapidité d'exécution est un critère essentiel.

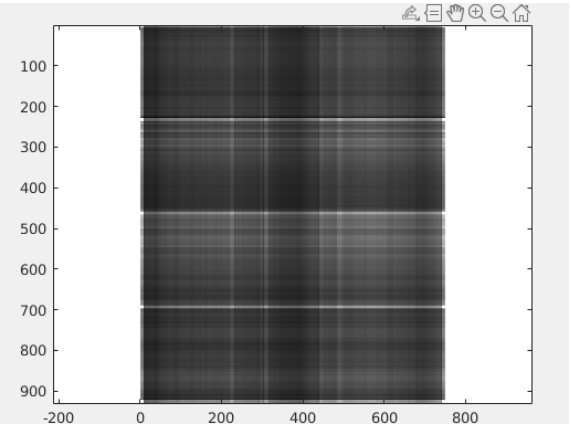




Méthode	Temps d'exécution
SVD	0.71
eig	0.98
subspace iter v0	95.20
subspace iter v1	99.66
subspace iter v2	22.67
subspace iter v3	22.67

Cas d'image avec couleur.

FIGURE 9 – Image BD avec couleur.



Conclusion

Ce projet nous a permis de mieux comprendre les enjeux liés au calcul de valeurs propres et à leur utilisation dans des applications concrètes comme la compression d'images. nous avons pu observer les différences importantes entre les méthodes directes (comme `eig` ou `svd`) et les méthodes itératives (subspace iteration), notamment en termes de temps de calcul et de qualité de reconstruction. En expérimentant différents paramètres, nous avons aussi vu l'impact réel qu'ils peuvent avoir sur les résultats finaux. Même si les méthodes itératives sont plus lentes, elles restent intéressantes pour de très grandes matrices. Ce travail illustré concrètement le lien entre les notions théoriques vues en cours et leur mise en œuvre concrète. Ce travail met en lumière l'importance d'un bon compromis entre performance, précision et complexité algorithmique selon le contexte d'utilisation.