
# Rapport du projet de calcul scientifique

Adam ROCHDI & Youssef ELAALOUCHE

# Sommaire

# 1 Introduction

La compression d'images joue un rôle essentiel dans le traitement d'image, permettant une gestion efficace des ressources en réduisant le volume de données requises pour le stockage et la transmission. Ce projet se concentre sur l'application de la décomposition en valeurs singulières (SVD) et explore diverses méthodes de décomposition matricielle pour la compression d'images. Les techniques étudiées incluent la SVD classique, la méthode des valeurs propres *eig*, et plusieurs approches de l'itération de sous-espace. L'objectif est de comparer leur efficacité en termes de fidélité de reconstruction et de performance computationnelle.

# 2 Partie 1: méthodes d'itération de sous-espace

## 2.1 Limitations de la méthode de la puissance

La méthode de la puissance permet de déterminer le vecteur propre lié à la valeur propre la plus grande (en valeur absolue).

### 2.1.1 Question 1:

Nous avons utilisé le fichier **'test_v11.m'** pour comparer les temps d'exécution des fonctions **'power_v11'** et de la fonction standard 'eig' de Matlab, en variant les tailles et les types de matrices. Les résultats de cette comparaison sont détaillés dans le tableau suivant.

| Taille | Type | eig | Puissance itérée |
|---|---|---|---|
| 10x10 | 1 | 0.00 | 3.125e-02 |
| | 2 | 0.00 | 6.250e-02 |
| | 3 | 0.00 | 1.562e-02 |
| | 4 | 0.00 | 1.562e-02 |
| 50x50 | 1 | 0.00 | 9.375e-02 |
| | 2 | 0.00 | 6.250e-02 |
| | 3 | 0.00 | 1.094e-01 |
| | 4 | 0.00 | 1.094e-01 |
| 200x200 | 1 | 6.250e-02 | 1.422 |
| | 2 | 1.562e-02 | 7.812e-02 |
| | 3 | 1.562e-02 | 2.344e-01 |
| | 4 | 0.00 | 1.76 |

Table 1: Comparaison des temps de calcul

En examinant les temps d'exécution pour les matrices de différentes tailles (10x10, 50x50 et 200x200), nous observons les tendances suivantes :

- La méthode `eig` est généralement plus rapide que la méthode de la puissance itérée pour les matrices de différentes tailles et types.
- Les temps de calcul pour la méthode `eig` sont relativement stables, indépendamment de la taille de la matrice. Par exemple, pour le type 1, les temps sont proches de 0 seconde pour les tailles 10x10, 50x50 et de environ 0.01 seconde pour la taille 200x200.
- Pour la méthode de la puissance itérée, les temps de calcul augmentent avec la taille de la matrice. Par exemple, pour le type 1, ils sont de 0.01 seconde pour la taille 10x10, de 0.06 seconde pour la taille 50x50, et de 1.44 secondes pour la taille 200x200.

#### 2.1.2 Question 2:

En étudiant en détail l'algorithme proposé, nous avons identifié deux produits de matrices par vecteur, $A \cdot \mathbf{v}$, à l'intérieur de la boucle. En restructurant les opérations, nous avons élaboré un algorithme qui n'intègre qu'un seul produit de matrice par vecteur dans la boucle, comme illustré dans la Figure 2. Cet algorithme, implémenté dans MATLAB dans le fichier `power_v12.m`, s'est révélé être deux fois plus rapide que l'algorithme initial.

```
while (~convg && k < m)

    k = k + 1;

    % méthode de la puissance itérée
    v = randn(n,1);
    z = A*v;
    beta = v'*z;

    % conv = || beta * v - A*v||/|beta| < eps
    % voir section 2.1.2 du sujet
    norme = norm(beta*v - z, 2)/norm(beta,2);
    nb_it = 1;

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

    end
```

Figure 1: Code optimisé

### 2.1.3 Question 3:

La méthode de la puissance itérée avec déflation est souvent moins performante car elle traite les valeurs propres et les vecteurs propres individuellement. À chaque itération, il faut répéter des opérations matricielles coûteuses sur une matrice actualisée de la forme $A - \beta \times (VV^T)$, où $\beta$ représente la valeur propre actuelle et $V$ le vecteur propre associé. Ce processus peut conduire à des temps de calcul accrus, particulièrement pour les grandes matrices.

```
>> test_v11v12

Matrice 200 x 200 - type 1

******* calcul avec la méthode de la puissance itérée ******

Temps puissance itérée = 1.734e+00
Nombre de valeurs propres pour attendre le pourcentage = 46

Matrice 200 x 200 - type 1

******* calcul avec la méthode de la puissance itérée améliorée ******

Temps puissance itérée = 9.844e-01
Nombre de valeurs propres pour attendre le pourcentage = 46
```

Figure 2: Résultat d'execution de test_v11v12

## 2.2 Extension de la méthode de la puissance pour calculer les vecteurs d'espace propre dominant

### 2.2.1 Question 4:

En appliquant la méthode de la puissance à un ensemble de $m$ vecteurs sans orthonormalisation, la matrice $V$ tend à converger uniquement vers le vecteur propre associé à la valeur propre la plus élevée de $A$, plutôt que vers $m$ vecteurs propres distincts. En l'absence d'orthonormalisation, les colonnes de $V$ sont progressivement dominées par le vecteur propre dominant. Ainsi, l'intégration d'une étape d'orthonormalisation dans les itérations de sous-espace est cruciale pour assurer une convergence appropriée vers des vecteurs propres distincts.

### 2.2.2 Question 5:

La raison pour laquelle il est acceptable de réaliser la décomposition spectrale complète de $H$ réside dans le fait que ses dimensions, $m \times m$, sont nettement inférieures à celles de $A$, qui sont de $n \times n$, où $n$ est typiquement beaucoup plus grand que $m$. De plus, l'objectif de l'Algorithme 2 et des autres variantes de la méthode de la puissance est de réduire la complexité des calculs. En se concentrant sur un sous-espace de dimension réduite, cette méthode permet d'obtenir rapidement et efficacement des approximations des valeurs propres dominantes et de leurs vecteurs propres, évitant ainsi les coûts élevés associés à la décomposition spectrale complète de $A$.

### 2.2.3 Question 6:

*Voir fichier matlab subspave_iter_v0.m*

### 2.2.4 Question 7:

Generate an initial set of m orthonormal vectors $V \in \mathbb{R}^{n \times m}$ : ***Line (48,49)***
$k = 0$ ***: (Line 38)***
PercentReached = 0 ***: (Line 40)***
**repeat**
k = k + 1 ***: (Line 54)***
Compute Y such that Y = AV ***: (Line 56)***

V = orthonormalisation of the columns of Y ***: (Line 58)***
Rayleigh-Ritz projection applied on matrix A and orthonormal vectors V ***: (Line 61)***
Convergence analysis step : save eigenpairs that have converged ***: (Line 92,94 )***
and update PercentReached ***: (Line 97)***
**until** $(PercentReached > PercentTraceornev = mork > MaxIter)$
***: (Line 52, 115)***

#### 2.2.5 Question 8:

Le calcul de $A^p$ implique de multiplier la matrice $A$ avec elle-même $p-1$ fois, ce qui a un coût de l'ordre de $O(2n^3)$ pour chaque multiplication. Ainsi, le coût total pour obtenir $A^p$ est de l'ordre de $O((p-1)2n^3)$. De plus, pour calculer $A^pV$, où $V$ est une matrice $n \times m$, il faut multiplier $A^p$ par $V$, ce qui coûte $O(2n^2m)$. Par conséquent, le coût total est de $O(2pn^3 + 2mn^2) \approx O(2pn^3)$.

Cependant, il est possible de réorganiser les calculs pour réduire ces coûts. Au lieu de calculer $A^p$ directement, nous pouvons effectuer $p$ multiplications successives de $A$ avec $V$, en stockant les résultats intermédiaires. Cette approche nécessite $p$ multiplications de matrices de dimensions $n \times m$, avec un coût total de $O(p(2n^2m))$.

En adoptant cette méthode, nous diminuons considérablement le nombre de flops nécessaire par rapport à la méthode de calcul de $A^p$ puis de $A^pV$, ce qui améliore l'efficacité de l'algorithme en restructurant les opérations.

#### 2.2.6 Question 9 :

Voir fichier ***matlab subspave_iter_v2.m***

#### 2.2.7 Question 10 :

D'abord, pour calculer les matrices de types 1 et 4, il faut généralement beaucoup plus d'itérations que pour les types 2 et 3. Cela s'explique par le fait que l'algorithme s'arrête dès qu'un certain pourcentage de la trace de la matrice est atteint. Pour les types 2 et 3, qui possèdent quelques valeurs propres dominantes très élevées et de nombreuses petites valeurs proches de zéro, atteindre rapidement ce pourcentage permet d'arrêter plus tôt les calculs. En revanche, pour

les types 1 et 4, les valeurs propres diminuent de manière linéaire et progressive, nécessitant l'obtention de bien plus de valeurs propres pour atteindre le pourcentage requis de la trace.

Ensuite, Augmenter la valeur de $\boldsymbol{p}$ réduit généralement le nombre d'itérations et le temps de calcul pour tous les types de matrices. Cependant, pour les types 2 et 3, le temps de traitement reste stable, même en augmentant $\boldsymbol{p}$ sur toute la plage des 25 valeurs. En revanche, pour les types 1 et 4, le temps de traitement continue de diminuer avec l'augmentation de $\boldsymbol{p}$. Cette observation souligne que l'efficacité de l'augmentation de $\boldsymbol{p}$ est fortement influencée par la distribution des valeurs propres de la matrice, indiquant que cette stratégie peut être particulièrement bénéfique pour certaines configurations de matrices mais pas pour d'autres

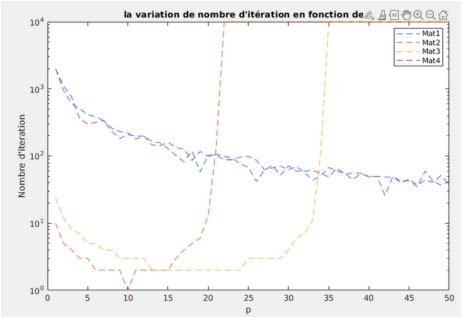


Figure 3: Nbre d'itérations en fct de p pour les 4 types de matrices (échl log))

#### 2.2.8 Question 11 :

Sur le tableau ci-dessous de *subspace_iter_v1*, nous constatons que la précision de la paire propre dominante est 106 fois meilleure que celle de la paire propre la moins dominante qui a été calculée. Cela est dû au fait que plus une paire propre est dominante, plus rapidement elle converge. Dans *subspace_iter_v1*, les paires propres qui ont déjà convergé (avec une précision suffisante selon nos critères) sont toujours traitées dans les boucles suivantes, en même temps que les paires propres qui n'ont pas encore convergé. Chaque boucle "supplémentaire" rend les paires propres déjà convergées encore plus précises, même si cela n'est pas nécessaire, tandis que, par exemple, la paire propre la moins dominante est moins précise car la boucle

s'arrête dès que la précision nécessaire (selon les critères donnés) est atteinte.

| qv pour les 11 vps |
|---|
| 5.95e-14 |
| 1.09e-13 |
| 2.00e-12 |
| 6.93e-12 |
| 1.48e-11 |
| 2.42e-10 |
| 1.46e-10 |
| 8.24e-10 |
| 1.51e-09 |
| 1.45e-08 |
| 8.53e-08 |

Table 2: Précision de chaque paire propre pour ***subspace_iter_v1***

#### 2.2.9 Question 12 :

Le nombre d'itérations nécessaires pour converger restera le même. Cependant, le nombre d'opérations sera grandement réduit puisque les vecteurs qui ont convergé ne seront plus traités. De plus, cela signifie que la précision sera beaucoup plus uniforme entre toutes les paires propres puisqu'il n'y aura plus de boucles *"bonus"* qui rendraient les paires propres les plus dominantes plus précises que les critères donnés.

#### 2.2.10 Question 13 :

Voir fichier ***matlab subspave_iter_v3.m***

#### 2.2.11 Question 14 :

Les matrices sont toutes pleines et symétriques, de même taille (n $\times$ n). La différence entre elles réside dans la répartition de leurs valeurs propres. La figure 8 montre que la courbe des valeurs propres des matrices de type 1 et 4 est linéaire, tandis que celles des types 2 et 3 ont une forme en 1/x . Il convient de noter que le type 1 possède des valeurs propres allant de 0 à 200, alors que pour les trois autres types, les valeurs propres restent entre 0 et 1.
Ces quatre types de matrices sont utilisés comme exemples spécifiques

de matrices qui pourraient être traitées par l'algorithme. Nous pouvons donc les tester pour déterminer quel algorithme, avec quels paramètres, est le mieux adapté à chaque type de matrice (comme nous l'avons vu pour la valeur de p dans la question 10).

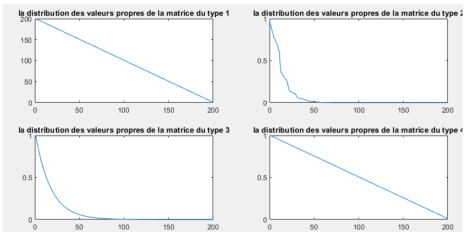


Figure 4: Répartition des valeurs propres pour chaque type de matrice

### 2.2.12 Question 15 :

Les figures montrent que, de manière générale, les grandes matrices et celles de type 1 et 4 demandent beaucoup plus de temps surtout en augmentant la taille de la matrice. En plus, Il est maintenant évident que la méthode de la puissance ne peut pas rivaliser avec la fonction 'eig' intégrée de MATLAB, qui a des temps de calcul très courts grâce aux plusieurs optimisations qu'elle a subit.

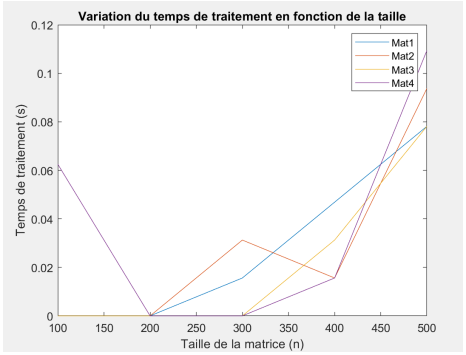


Figure 5: ***eig***

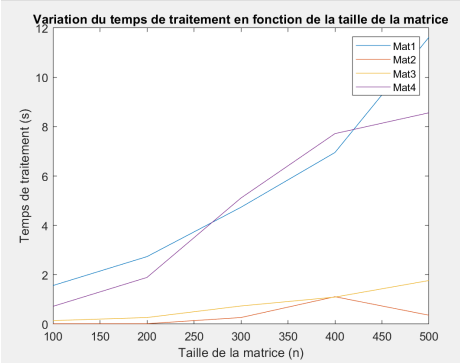


Figure 6: ***subspave_iter_v0***

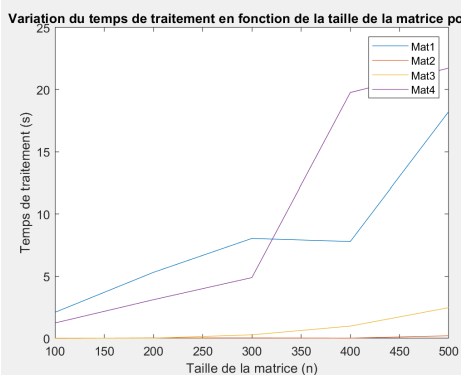


Figure 7: ***subspave_iter_v1***

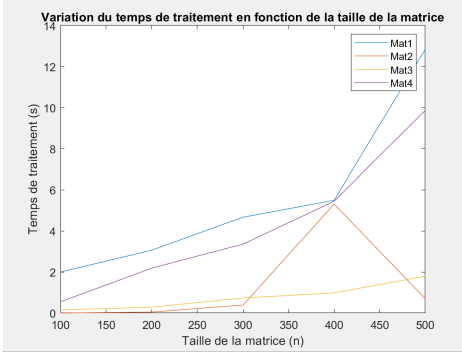


Figure 8: ***subspave_iter_v3***

# 3 Partie 2: Méthodes d'itération sur les sous-espaces Application à la compression d'images.

Dans cette section, nous examinons comment les méthodes d'itération sur les sous-espaces s'appliquent à la compression des images en niveaux de gris. Les images sont représentées par des matrices rectangulaires, et la compression est basée sur la décomposition en valeurs singulières (SVD) ainsi que sur le théorème de la meilleure approximation de rang réduit.

## 3.1 Question 1 :

Si on utilise une approximation de rang $k$ (où $k < p < q$) pour compresser une matrice d'image $I$ de taille $q \times p$, alors le triplet ($P_k$, $U_k$, $V_k$) aura les dimensions suivantes :

- $P_k$ : matrice diagonale de taille $k \times k$, contenant les $k$ premières valeurs singulières de la matrice d'image $I$. Elle aura $k$ éléments non nuls.
- $U_k$ : matrice de taille $q \times k$, contenant les $k$ premiers vecteurs propres associés aux $k$ vecteurs propres de $II^\top$. Elle aura $qk$ éléments.
- $V_k$ : matrice de taille $p \times k$, contenant les $k$ premiers vecteurs propres associés aux $k$ vecteurs propres de $I^\top I$. Elle aura $pk$ éléments.

En résumé, les dimensions des éléments du triplet ($P_k$, $U_k$, $V_k$) sont : $P_k$ : $k \times k$ ($k$ éléments non nuls), $U_k$ : $q \times k$ ($qk$ éléments), $V_k$ : $p \times k$ ($pk$ éléments).

## 3.2 Question 2 :

Le graphe ci-dessous montre comment l'erreur de reconstruction (RMSE) diminue avec l'augmentation du nombre de vecteurs singuliers utilisés, de 1 à 200. Cela indique une amélioration progressive de la qualité de l'image reconstruite lorsque plus de composantes principales sont incluses. En général, plus le nombre de vecteurs est élevé, meilleure est la précision de la reconstruction, mais les gains deviennent marginaux à des rangs plus élevés.

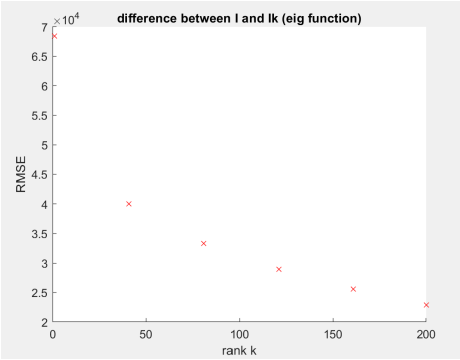


Figure 9: Graphe qui montre l'erreur en fonction de k

L'analyse des temps de reconstruction d'image révèle des différences significatives en termes de performance temporelle entre les méthodes utilisées:

- Les méthodes **SVD classique** et **avec eig** sont très rapides, avec des temps d'exécution autour de 0.65 secondes, profitant de l'optimisation des algorithmes intégrés dans MATLAB.
- Les **méthodes de l'itération de sous-espace (v0, v1, v3)** présentent des temps nettement plus élevés. La v0 et la v1 prennent environ 96 et 100 secondes respectivement, tandis que la v3 est plus rapide avec environ 25 secondes.

En conclusion, bien que les méthodes itératives puissent être théoriquement avantageuses pour certaines configurations, les méthodes directes (**SVD**, **eig**) demeurent préférables pour la compression d'images en termes de rapidité.

| Méthodes | Temps d'execution |
|---|---|
| SVD | 0.64 |
| eig | 0.63 |
| subspace_iter_v0 | 96.46 |
| subspace_iter_v1 | 99.69 |
| subspace_iter_v3(v2) | 24.77 |

# 4 Conclusion

Ce projet a exploré l'efficacité de différentes méthodes de décomposition matricielle pour la compression d'images. Les

résultats ont démontré que la méthode SVD classique et la décomposition via *eig* sont nettement plus rapides par rapport aux méthodes itératives de l'itération de sous-espace, spécialement pour les images de taille moyenne. Alors que la méthode SVD classique offre un équilibre optimal entre la vitesse de calcul et la qualité de reconstruction, les méthodes itératives, bien qu'intéressantes théoriquement, se révèlent moins pratiques pour des applications nécessitant une grande rapidité due à leur lourdeur computationnelle. Ces constatations suggèrent que la SVD reste préférable pour la compression d'images dans des contextes pratiques où la performance et la rapidité sont prioritaires.