Rapport Maths des Données : Probabilistic algorithms for constructing approximate matrix decompositions

Jérémy Dourville

Septembre 2025

Abstract

Les décompositions matricielles jouent un rôle central en mathématiques appliquées, en informatique théorique mais encore en traitement des données. Elles permettent d'analyser la structure interne des matrices, de réduire la complexité des calculs et d'optimiser le stockage de l'information.

Dans ce rapport, nous présentons l'approximation de décomposition, en particulier la diagonalisation. Nous mettons en évidence les avantages liés à la représentation compressée des matrices de rang faible, qui permet des calculs rapides.

1 Introduction

Les décompositions matricielles sont des outils fondamentaux en mathématiques appliquées et en traitement des données. Elles permettent de simplifier les calculs, d'analyser les structures internes des matrices et de stocker efficacement des informations.

Dans ce rapport, nous nous intéressons particulièrement à l'approximation de la décomposition QR, SVD ainsi que la diagonalisation approchée en limitant le rang de l'approximation. Les idées seront principalement tiré de [Tro11]

2 Quelques notations et définitions:

- La distribution normale, notée $\mathcal{N}(0,1)$, est la distribution de probabilité définie par la densité : $x \mapsto \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$
- Soit $M \in \mathcal{M}_{n,k}(\mathbf{C})$, on définit $M^* := \overline{M}^T \in \mathcal{M}_{k,n}(\mathbf{C})$.
- Une matrice carré M est dite hermitienne si et seulement si $M^* = M$. Dans ce cas, elle est diagonalisable en base orthonormée.
- Une matrice $Q \in \mathcal{M}_{n,l}(\mathbf{C})$ est dite orthonormée si et seulement si ses colonnes sont orthogonales et de norme 1, c'est-à-dire $Q^*Q = I_l$.

Dans le cas d'une matrice carré, on dit que Q est unitaire.

- Une matrice $R \in \mathcal{M}_{l,k}(\mathbf{C})$ est dite triangulaire supérieure si et seulement si tous les coefficients sous la diagonale sont nuls, i.e $R_{ij} = 0 \quad \text{pour } i > j$.

On se place dans l'espace $\mathcal{M}_{n,k}(\mathbf{C})$ munis

- du produit scalaire usuel : $< A|B >= \sum_{i,j} \overline{a_{ij}} b_{i,j}$.
- de la norme suivante : $||A||^2 = \sum_{i,j} |a_{ij}|^2$

Ainsi $A \approx BC$ signifie $||A - BC|| \leq \epsilon$ pour un ϵ fixé.

Pour $X, Y \in \mathbb{R}^n$ on définit $d(X,Y) := \min_{\sigma \in \mathcal{S}_n} \sum_{i=1}^{n} |x_i - y_{\sigma(i)}|$. d est une distance entre les multi-ensembles (ensemble avec multiplicité des éléments) et non sur les vecteurs.
Preuve que d est une distance :

- d est à valeur positives
- $d(X,Y) = 0 \Leftrightarrow \exists \sigma \in \mathcal{S}_n, X = \sigma(Y) \Leftrightarrow X = Y$ (car on étudie les ensembles, pas les éléments de R^n).
- Pour l'inégalité triangulaire, il suffit de composer chacune des permutations pour laquelle on obtient le minimum.

3 Décomposition exacte de matrice

3.1 Exemple sur les matrices de rang 1

3.1.1 Décomposition d'une matrice de rang 1

Une matrice $M \in \mathcal{M}_{n,k}(\mathbf{C})$ de rang 1 peut toujours s'écrire comme le produit d'une colonne $C \in \mathcal{M}_{n,1}(\mathbf{C})$ et d'une ligne $L \in \mathcal{M}_{1,k}(\mathbf{C})$:$M = CL$.
Cette décomposition permet de stocker efficacement une matrice de rang 1 comme le couple (C, L), donc en espace $O(n+k)$ et non en $O(nk)$.

Voici un algorithme simple pour calculer C et L à partir de M en temps $O(nk)$:

Algorithm 1: Décomposition d'une matrice de rang 1

Input: Une matrice $M \in \mathcal{M}_{n,k}(\mathbf{C})$ de rang 1
Output: Vecteur colonne $C \in \mathcal{M}_{n,1}(\mathbf{C})$ et vecteur ligne $L \in \mathcal{M}_{1,k}(\mathbf{C})$ tels que $M = CL$

Choisir un indice i tel que $M_{i,.} \neq 0$; `// une ligne non nulle`
$L \leftarrow M_{i,:}$; `// la ligne choisie`
Choisir un indice j tel que $L_j \neq 0$;
$C \leftarrow \frac{M_{.,j}}{L_j}$; `// le vecteur colonne`
return C, L

On remarque par ailleurs que quitte à renormaliser C, on a une décomposition QR de M (cf. section 3.2).

3.1.2 Puissance d'une matrice de rang 1

Pour une matrice carrée $M = CL$ de rang 1, on peut calculer facilement sa puissance M^p :

$$M^p = (CL)^p = C(LC)^{p-1}L = (LC)^{p-1}M.$$

- $LC \in \mathcal{M}_{1,1}(\mathbf{C})$ est un scalaire complexe.
- Le calcul de LC nécessite $O(n)$ opérations.
- Le calcul de $(LC)^{p-1}$ à partir de LC nécessite seulement $O(\log p)$ multiplications grâce à l'exponentiation rapide.

Ainsi, M^p peut être calculée en $O(n + \log p)$ opérations.

Cette méthode illustre l'avantage de stocker une matrice de rang 1 sous-forme de couple de vecteurs : les calculs sont beaucoup plus rapides et économes en mémoire.

3.2 Décomposition d'une matrice de rang r

On peut faire une décomposition analogue à celle aux matrices de rang de 1 pour un rang quelconques.
Une matrice $A = (A_1|\dots|A_k) \in \mathcal{M}_{n,k}(\mathbf{C})$ est de rang si et seulement s'il existe $C_1 \dots C_r$ des colonnes indépendantes tels que
$\forall i, \exists \lambda_{1,i} \dots \lambda_{r,i}, A_i = \sum_{j=1^r} \lambda_{j,i} C_j$.

Cela se reformule de la manière suivante : $A = (C_1|\dots|C_r) * \begin{pmatrix} \lambda_{1,1} & \dots & \lambda_{n,1} \\ \vdots & & \vdots \\ \lambda_{1,r} & \dots & \lambda_{n,r} \end{pmatrix}$.

Cette décomposition sera utilisée par la suite pour générer une matrice de bas rang. En effet, si l'on tire une matrice de $\mathcal{M}_{n,r}(\mathbf{C})$ et une matrice de $\mathcal{M}_{r,k}(\mathbf{C})$ alors on obtient une matrice de $\mathcal{M}_{n,k}(\mathbf{C})$ de rang au plus r.(cf section 4.1.3)

3.3 Décomposition QR

3.3.1 Présentation de la décomposition QR

La décomposition QR est une factorisation fondamentale en algèbre linéaire. Elle permet de représenter une matrice $A \in \mathcal{M}_{n,k}(\mathbf{C})$ comme le produit d'une matrice orthonormée Q et d'une matrice triangulaire supérieure R : $A = QR$

3.3.2 Utilisation de la décomposition QR

La factorisation QR est très utilisée en calcul scientifique, et par exemple dans la résolution de systèmes.
Si $A = QR$ le système $Ax = b$ se réécrit $QRx = b$. C'est à dire $Rx = Q^*b$.
Or R étant triangulaire supérieure, cela permet de résoudre efficacement ce système (en $O(n^2)$ opérations contre $O(n^3)$ pour un algorithme de pivot sans hypothèse).

Par ailleurs, la décomposition QR permet de faire d'autres décompositions telles que la décomposition SVD.

3.3.3 Calculabilité de la décomposition QR

Il existe plusieurs méthodes afin d'obtenir la décomposition QR, parmi elles ont peu citer la méthode de Householder, la méthode de Givens et la méthode de Gram-Schmidt.
Ces algorithmes lorsque $M \in \mathcal{M}_{n,l}(\mathbf{C})$ sont en complexité $O(nl^2)$.

3.4 Décomposition SVD

3.4.1 Présentation de la décomposition SVD

La décomposition SVD est une autre décomposition classique, elle est une généralisation de la diagonalisation.
Ici, on représente une matrice $A \in \mathcal{M}_{m,n}(\mathbf{C})$ à partir de deux matrices orthonormées $U \in \mathcal{M}_{m,m}(\mathbf{C}), V \in \mathcal{M}_{n,n}(\mathbf{C})$ et d'une matrice diagonale a coefficient positif $\Sigma = Diag(\sigma_1 \dots \sigma_k) \in \mathcal{M}_{m,n}(\mathbf{C})$ tels que : $A = U\Sigma V^*$.

Par exemple pour une matrice symétrique réel, la diagonalisation en base orthonormée fournie par le théorème spectrale correspond à la décomposition SVD.

Ici aussi, la décomposition n'est pas unique (si (U, V, Σ) est une décomposition alors $(-U, -V, \Sigma)$ en est aussi une).

3.4.2 Complexité de la décomposition SVD

Dans [Hig19] les auteurs annonces l'état de l'art sur la décomposition SVD pour les matrices carrées. La complexité est de $O(n^3)$.

4 Décompositions approchées d'une matrice

Le but est de reprendre l'idée de décomposition vue sur le cas des matrices de rang 1, mais cette fois, on va demander une approximation de notre matrice initiale.
L'idée correspond donc pour une matrice $A \in \mathcal{M}_{m,n}(\mathbf{C})$ et un entier k fixé, de trouver deux matrices $B \in \mathcal{M}_{m,k}(\mathbf{C}), C \in \mathcal{M}_{k,n}(\mathbf{C})$ tels que $A \approx BC$.
Cela peut être bénéfique dans de multiples domaines, que ce soit pour le stockage de données, mais aussi bien pour gagner en temps de calcul.
Pour cela, 2 étapes vont être nécessaires afin de pouvoir obtenir cette décomposition.

4.1 Approximation de base orthonormale

4.1.1 Calcul d'une approximation de base orthonormale

Ici, on va chercher à calculer une matrice Q orthogonales telle que $A \approx QQ^*A$

Algorithm 2: Approximation d'une base orthonormale une projection aléatoire

Input: $A \in \mathcal{M}_{n,k}(\mathbf{C}),\ l \in \mathbb{N}$
Output: $Q \in \mathcal{M}_{n,l}(\mathbf{C})$ telle que $A \approx QQ^*A$
$\Omega \in \mathcal{M}_{k,l}(\mathbf{C})$; // $\forall i,j,\ \Omega_{i,j} \sim \mathcal{N}(0,1)$
$Y \leftarrow A\Omega$
Calculer Q, R une décomposition QR de Y
return Q

Correction de l'algorithme

Par définition de la décomposition QR, on a $Q^*Q = I_l$.
En particulier dans le cas où $l = n$ on a $QQ^* = I_n$ et ainsi l'approximation faite est exacte.
Sinon $QQ^* \approx I_n$

Évolution de l'approximation selon l

Cet algorithme à une complexité de : $O(kl + nkl + nl^2) = O(nl(k + l))$ en supposant la génération d'une valeur suivant une loi normale en temps constant, et un algorithme naïf de multiplication matricielle.

4.1.2 Estimation de l'approximation de base orthonormale

On voudrait tester si l'approximation est réussie, c'est-à-dire $||A - QQ^*A|| \leq \epsilon$
Or, on peut évaluer facilement évaluer $||(A - QQ^*A)\omega||$ pour ω un vecteur.
Et on a le lemme suivant :

Soit $\{\omega^i\}_{1 \leq i \leq r}$ un ensemble de vecteur gaussien indépendant, $M \in \mathcal{M}_{n,k}(\mathbf{C})$, on a:

$$\forall \alpha, \mathbb{P}(||B|| \leq \alpha\sqrt{\frac{2}{\pi}} max_i ||B\omega^i||) \geq 1 - \alpha^r$$

Des grandeurs typiques d'erreurs en informatiques sont de 10^{-10} et donc en évaluant sur 10 valeurs, on peut être assez serein d'avoir une majoration de $||B||$ ([Wik25]).

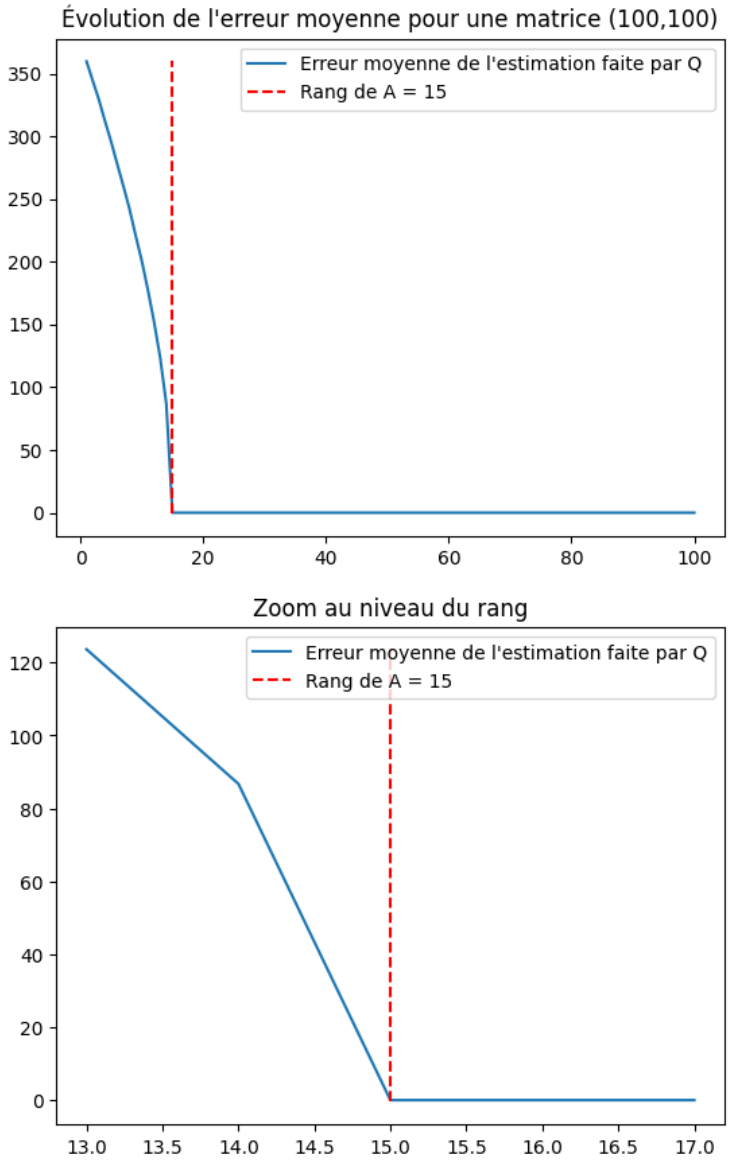
On peut donc appliquer cela pour tester si $||A - QQ^*A|| \leq \epsilon$ avec probabilité 10^{-10} en temps $O(n(k + l))$. Par ailleurs, cette estimation peut être effectuée en même temps que le calcul de Q, ce qui permet d'amortir le coût de calcul. Et tant que la condition $||A - QQ^*A|| \leq \epsilon$ n'est pas respecté, on peut relancer l'algorithme afin d'obtenir une matrice qui satisfasse ce critère.

4.1.3 Analyse expérimentale de l'approximation en base orthonormée

On remarque dans l'algorithme 2 que Q est de rang au plus l. Et donc on peut s'attendre lorsque $l < rg(A)$ à avoir une mauvaise approximation. Tandis qu'à l'opposé pour $l \geq rg(A)$ on peut espérer avoir une décomposition exacte.

Algorithm 3: Évolution de l'erreur selon le rang maximale autorisée

Input: $A \in \mathcal{M}_{n,n}(\mathbf{C})$ une matrice générée aléatoirement,
nb_rep : le nombre de répétition
Pour l=1 à n:
  Pour j=1 à nb_rep :
    Calculer une matrice $Q_{j,l}$ selon l'algorithme 2 avec la taille l .
    Calculer l'écart $||A - Q_{j,l}Q_{j,l}^{*}A||$.
  Faire la moyenne des écarts sur j notée m_l.
Tracer la courbe $m_l = f(l)$



La manière de générer A que j'ai choisie pour expérimenter cela est de générer une matrice

de rang limité vu à la section 3.2 .
On constate que la matrice Q approxime aussi précisément que l'on veut lorsque la taille de Q est au moins le rang.

4.2 Construction d'une approximation

4.2.1 Construction d'une approximation de décomposition SVD

Ici, on suppose avoir une matrice Q tels que $A \approx QQ^*A$.

Algorithm 4: Approximation d'une décomposition SVD

Input: $A \in \mathcal{M}_{n,k}(\mathbf{C}), Q \in \mathcal{M}_{n,l}(\mathbf{C})$
Output: U, V, Σ une décomposition SVD
$B = Q^*A \in \mathcal{M}_{l,k}(\mathbf{C})$
Calculer une décomposition SVD $B = \tilde{U}\Sigma V^*$
Calculer $U = Q\tilde{U}$
return *Décomposition SVD* (U, Σ, V)

4.2.2 Étude de l'approximation de décomposition SVD

Correction de l'algorithme :
On remarque tout d'abord que $U\Sigma V^* = Q\tilde{U}\Sigma V^* = QB = QQ^*A \approx A$.
V est bien orthonormée, et Σ est une diagonale positive par définition de la décomposition SVD.
De plus $U^*U = \tilde{U}^*Q^*Q\tilde{U} = \tilde{U}^*\tilde{U} = I$ donc U est bien orthogonale.
On a donc bien une décomposition SVD.

Précision de l'algorithme :
On a $||A - U\Sigma V^*|| = ||A - Q\tilde{U}\Sigma V^*|| = ||A - QB|| = ||A - QQ^*A||$.
Autrement dit, excepté les erreurs de calcul naturellement faites par un ordinateur, notre décomposition SVD et aussi précise que l'approximation en base orthonormales.

Complexité de l'algorithme :
La complexité de l'algorithme est $O(lkn + SVD(l, k))$.
En pratique $l \leq k$ et donc on a une complexité de $O(lkn + k^3)$.
Le point de comparaison est la SVD exacte ou la complexité est $O(max(k, n)^3)$.
En pratique l est le rang d'approximation et si $l << n$ et $k << n$ alors on observe un grand gain dans le temps de calculs ($O(lkn + k^3)$ contre $O(n^3)$).

4.2.3 Analyse expérimentale de l'approximation de la décomposition SVD

On obtient des courbes similaires à la sections 4.2.3 du fait que :
$||A - U\Sigma V^*|| = ||A - QQ^*A||$.

5 Utilisation au cas de la diagonalisation

La diagonalisation est l'une des décompositions les plus élémentaires avec de multiples applications. C'est pour cela que j'ai souhaité tester sur cette décomposition l'efficacité de cette approche.

5.1 Complexité de la diagonalisation exacte

D'après [Sri23], la complexité de la diagonalisation est en $O(T_{MN}(n)log^2(\frac{\delta}{n}))$ avec TM la complexité de la fonction de multiplication matricielle qui est numériquement stable, et δ une constante de précision.
On peut donc en prenant une bonne multiplication avoir une diagonalisation en $O(n^3)$ (On peut prendre la multiplication de Strassen).
J'ai quand même souhaiter estimer la complexité de la fonction numpy associer. Pour cela je fais le protocole suivant :

Algorithm 5: Estimation de la complexité de la diagonalisation de numpy

Input: n_max : la plus grande valeur de la dimension des matrices auquel ont effectue une diagonalisation
nb_rep : le nombre de répétition
Pour i=1 à n_max:
  Pour r=1 à nb_rep :
    Générer une matrice aléatoirement de taille (n, n) .
    Mesurer le temps de calcul de la diagonalisation.
  Faire la moyenne des différentes répétition := t_i.
Tracer la courbe $t_i = f(i)$

5.2 Algorithme d'approximation de la diagonalisation

Algorithm 6: Diagonalisation approchée

Input: $A \in \mathcal{M}_{n,n}(\mathbf{C})$ une matrice hermitienne, $Q \in \mathcal{M}_{n,l}(\mathbf{C})$ tels que $A \approx QQ^*A$
Output: $U, \Lambda \in \mathcal{M}_{n,n}(\mathbf{C})$ tels que $A \approx U\Lambda U^*$,Λ une matrice qui est diagonale réelle et U orthonormale
$B = Q^*AQ$
Calculer U, Λ tels que $B = V\Lambda V^*$
return $U = QV, \Lambda$

Correction de l'algorithme :
On remarque tout d'abord que $B^* = Q^*A^*Q = Q^*AQ = B$ car A est hermitienne. Ainsi B est diagonalisable d'après le théorème spectral en bas orthonormée. Autrement dit V est orthonormale.
Donc $U^*U = V^*Q^*QV = I$ car Q et V sont orthonormales.

Précision de l'algorithme :
On a $U\Lambda U^* = QV\Lambda V^*Q^* = QBQ^* = QQ^*AQQ^*$.
Par hypothèse sur Q on a donc :
$||AQ^*Q - U\Lambda U^*|| = ||A - QQ^*AQQ^*|| \leq ||A - QQ^*A||.||Q^*Q|| \leq \epsilon.1 = \epsilon$ car la norme est sous-multiplicative.
De plus $||A - AQQ^*|| = ||A^* - QQ^*A^*|| \leq ||A - QQ^*A|| \leq \epsilon$ car A est hermitienne.

Donc $||A - U\Lambda U^*|| \leq ||A - AQQ^*|| + ||AQ^*Q - U\Lambda U^*|| \leq 2\epsilon$.

Autrement dit, excepté les erreurs de calcul naturellement faites par un ordinateur, notre diagonalisation reste du même ordre de précision que l'approximation en base orthonormales.

Complexité de l'algorithme :
La complexité de l'algorithme est $O(n^2l + n^2l + DIAG(l)) = O(n^2l + l^3)$. Le terme $n^2l + n^2l$ vient du calcul de B.
On peut englober le coût de calcul de la matrice Q qui est $O(n^2l)$.
Donc on a un algorithme de diagonalisation approché fournissant A en $O(n^2l + l^3)$ contre $O(n^3)$ pour une diagonalisation exacte.
En prenant $l << n$ on obtient donc un gain sur le calcul de la diagonalisation.

5.3 Analyse expérimentale de l'approximation de la diagonalisation

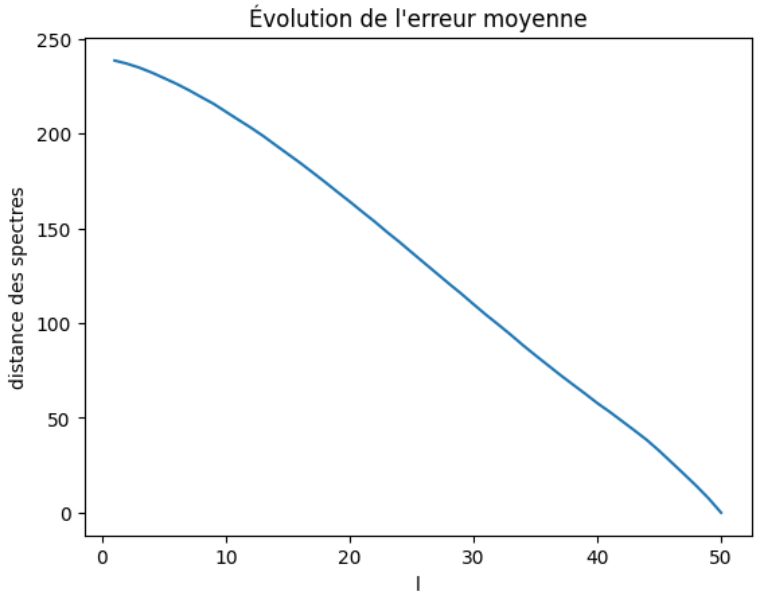
5.3.1 Analyse pour une matrice quelconque

Ici, les matrices sont générées selon la propriété vue en section 3.2 afin de fixer un bas rang. Mais de plus, chacune des deux matrices sont des matrices de loi normale (fonction np.random.randn(n, r) de numpy).

La notion que je souhaite comparer ici est l'estimation du spectre fait par cette méthode. On va donc comparer le spectre de A au spectre de Λ. Le problème principal est que Λ est un ensemble de taille l donc on le complète par autant de zéros que nécessaire.
Le choix d'ajouter des zéros et non des autres nombres est logiques, car Q est choisis pour approcher le comportement de A. C'est-à-dire que Q va approcher précisément les "grandes" valeurs propres.
Pour mesurer cette estimation, on va utiliser la distance vue en section 2.

Algorithm 7: Évolution de l'erreur sur le spectre selon le paramètre l

Input: $A \in \mathcal{M}_{n,n}(\mathbf{C})$ une matrice générée aléatoirement,
Output: Graphique de la précision du spectre au fur et a mesure du rang d'approximation
nb_rep : le nombre de répétition
Calculer S le spectre de A Pour l=1 à n:
  Pour j=1 à *nb_rep* :
    Calculer une matrice $Q_{j,l}$ selon l'algorithme 2 avec la taille l .
    Calculer le spectre $S_{j,l}$ de la matrice $B_{j,l}$ dans l'algorithme de diagonalisation.
    Compléter $S_{j,l}$ par autant de 0 que nécessaire pour avoir un ensemble à n éléments.
    Calculer $d(S, S_{j,l})$.
  Faire la moyenne des écarts sur j notée m_l.
Tracer la courbe $m_l = f(l)$



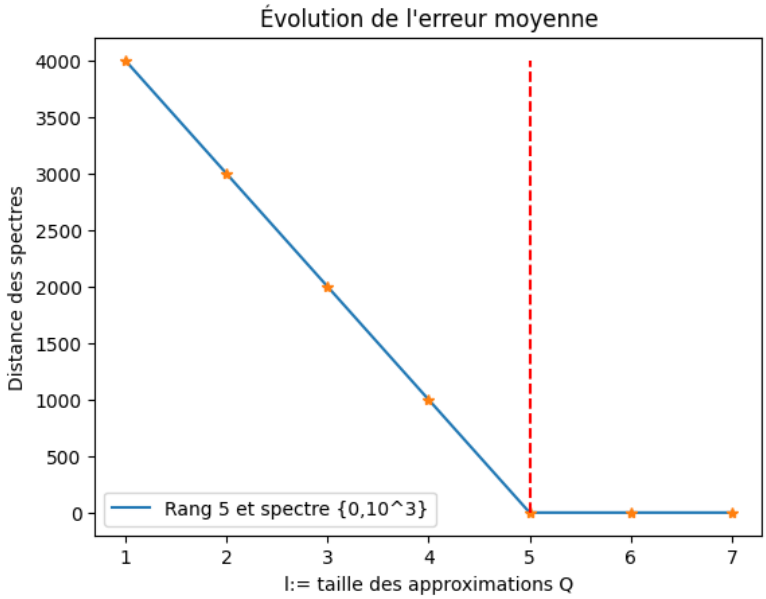
Le fait que l'erreur soit à tendance décroissante est assez naturel; on permet à notre estimation d'avoir de plus en plus de précision.

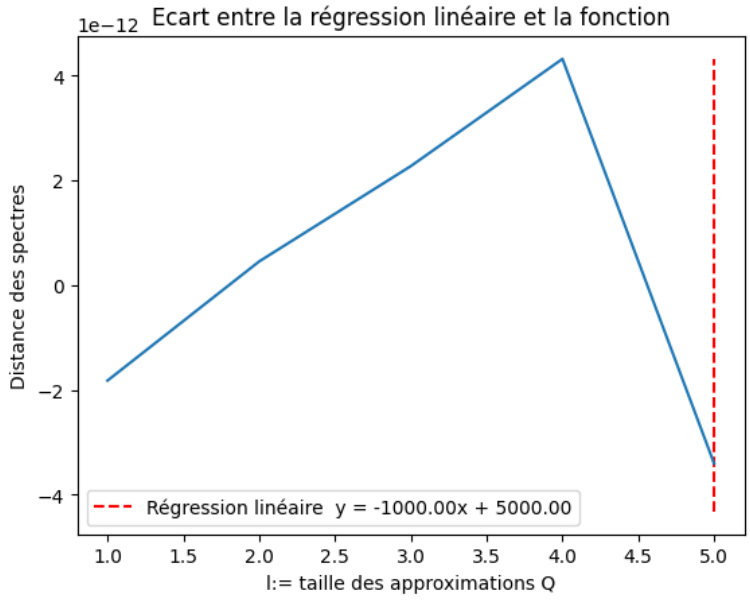
5.3.2 Analyse pour une matrice avec 1 valeur propre fixée

Je génère une matrice de rang r et de valeur propre v de la manière suivante :

Algorithm 9: Génération d'une matrice a une valeur propre

Input: n, r, v
Output: $A \in \mathcal{M}_{n,n}(\mathbf{C})$ tels que $rg(A) = r$ et $sp(A) = \{0, v\}$
$D = Diag(vID_r, 0_{n-r})$ avec r fois le v.
Tant que True:
  Générer $P \in \mathcal{M}_{n,n}(\mathbf{C})$ par des lois normales
  Si P est inversible :
    Renvoyer PDP^{-1}

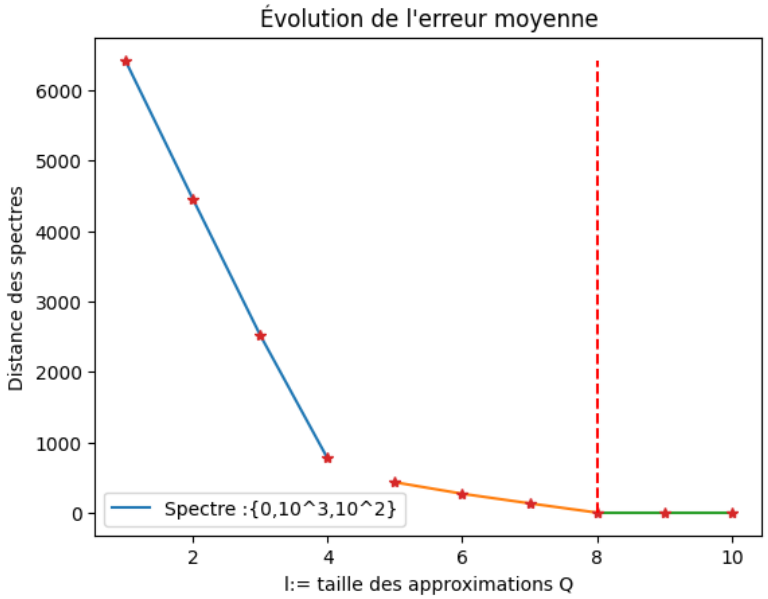




On analyse expérimentalement qu'on approxime 1 axe propre par degré de rang que l'on s'autorise lors de l'approximation.Ce qui entraîne l'évolution du spectre approximé.

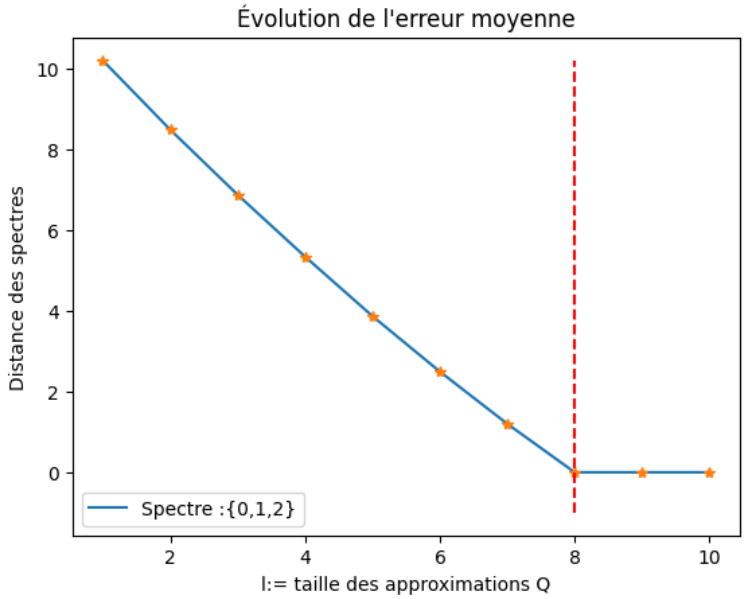
5.3.3 Analyse pour une matrice avec 2 valeurs propre fixée

On fait la même chose que précédemment en partant de :
$D = Diag(v_1 ID_{r_1}, v_2 ID_{r_2}, 0_{n-r})$.



On constate que l'approximation du spectre a 3 comportements linéaires disjoints :

- On approxime la grande valeur propre, la pente de la droite vaut $\approx -2.10^3$
- On approxime la petite valeur propre, la pente de la droite vaut $\approx -10^2$
- On a finis d'approximer



On constate un problème si les deux valeurs propres sont proches (1 et 2 ici). L'approximation n'est pas faite sur un axe propre puis un autre. Sur la partie associer à la "grande valeur propre" (partie bleu de la courbe précédemment) on a une pente ≈ -1.5 ce qui est la moyenne

des 2 valeurs propres.
Ainsi, on comprend que si on veut approximer les "grandes valeurs propres", le fait que les valeurs propres sont distantes favorise une stabilité numérique.

6 Conclusions

Ce rapport a montré que lorsqu'une matrice est de bas rang, il est possible de l'approximer efficacement dans une base orthonormée, représentée par une matrice Q. Cette approximation constitue une étape essentielle dans la construction d'autres décompositions fondamentales, telles que la décomposition en valeurs singulières (SVD) ou la diagonalisation approchée.
Nos expériences ont mis en évidence que, pour des matrices de rang faible par rapport à n, on peut obtenir un gain asymptotique notable sur le temps de calcul des approximations. Cette observation souligne l'intérêt des méthodes de projection aléatoire et de réduction de la dimension de calcul.
Plusieurs prolongements de ce travail sont envisageables. D'une part, il serait intéressant d'étudier les gains en espace mémoire que peuvent offrir ces décompositions. D'autre part, une piste toute aussi intéréssante consisterait à évaluer l'applicabilité de ces approches dans le cadre du machine learning, où les data set sont massifs, et pourrait potentiellement être modélisés par des matrices de bas rang.
Enfin, un axe de recherche plus théorique serait d'adapter ces techniques au cadre des algorithmes de streaming, dans lequel les données arrivent progressivement et ne peuvent être stockées intégralement en mémoire. Généraliser les décompositions approchées dans ce contexte permettrait de traiter le cas des matrices de tailles immenses mais de "petit" rangs.

References

[Hig19] Pierre Blanchard; Mawussi Zounon; Jack Dongarra; Nick Higham. *Novel SVD Algorithms*. 2019. URL: https://www.nlafet.eu/wp-content/uploads/2019/04/D2-9-SVDalgorithms-final.pdf?utm_source=chatgpt.com.

[Sri23] Nikhil Srivastava. *The Complexity of Diagonalization*. 2023. URL: https://arxiv.org/pdf/2305.10575.

[Tro11] N.Halko; P.G.Martinsson; J.A. Tropp. *Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions*. 2011. URL: http://users.cms.caltech.edu/~jtropp/papers/HMT11-Finding-Structure-SIREV.pdf.

[Wik25] Wikipedia. *Mémoire à code correcteur d'erreurs*. 2025. URL: https://fr.wikipedia.org/wiki/M%C3%A9moire_%C3%A0_code_correcteur_d%27erreurs.

Annexes

A Calcul de la distance entre ensembles:

On souhaite pouvoir calculer $\min_{\sigma \in \mathcal{S}_n} \sum_{i=1}^{n} |x_i - y_{\sigma(i)}|$ avec les $\forall i, x_i, y_i \in \mathbf{C}$.

On va définir la matrice de coût $C[i,j] = |x_i - y_j|$.
Et on cherche a minimiser la quantité $\min_{\sigma \in \mathcal{S}_n} \sum_i C[i, \sigma(j)]$.
Cela correspond à l'algorithme hongrois est nous permet d'avoir un algorithme en $O(n^3)$ [1].

Algorithm 10: Idée de la méthode hongroises

Input: Une matrice $M \in \mathcal{M}_{n,k}(\mathbf{C})$
Output: $\min_{\sigma \in \mathcal{S}_n} \sum_i C[i, \sigma(j)]$

(0): Pour chaque ligne, soustraire dans chaque case l'élément minimal de la ligne.
(0'): Pour chaque colonne, soustraire dans chaque case l'élément minimal de la colonne.
(1): Sélectionner le nombre maximal de zéros indépendants (différentes lignes et différentes colonnes).
(2): Si n cases non pas été sélectionnées:
(2a): Couvrir les colonnes ou il y a un 0 séléctionnés.
(2b): Couvrir les lignes ou il y a un 0 recouverts qui n'est pas sélectionnés, et découvrir sa colonne.
(2c) Appliquer la méthode hongroises sur la sous-matrice nons couvertes.

[1] L'algorithme présenté ici sera en $O(n^4)$.