
Calcul Haut Performance et Simulation

# Rapport projet de Méthodes de Programmation Numérique Avancée

***Thème :***

---

## Décomposition en valeurs singulières SVD

---

***Préparé par :***

M$^{elle}$.Baya ABBACI

M$^{r}$.Hamza ELBARAGHI

M$^{r}$.Karim SMAIL

***Année 2020-2021***

# Table des matières

# Introduction

La décomposition en valeurs singulières du sigle SVD qui vient de "Singular Values Decomposition" pour les anglophones, permet la factorisation des matrices triangulaires, et peut être vu comme le procédé de diagonalisation pour les matrices carrées.

L'idée de la décomposition en valeurs singulières est similaire à la décomposition en valeurs propres, mais fonctionne pour n'importe quelle matrice A de taille m×n, c'est-à-dire que cette décomposition généralise la notion de valeurs propres aux matrices rectangulaires.

Le calcul de la décomposition en valeurs singulières est traité depuis le milieu des années 1960 par l'informaticien Gene Howard Golub [1] et le mathématicien William Morton Kahan, qui proposent en 1965 le premier algorithme pour ce calcul.

Cette méthode bien connue au 20 ème siècle, permet plus récemment de fournir des résultats en analyse de données avec par exemple la complétion de matrices aux données manquantes. Ce qui revient a résoudre des problèmes d'optimisation convexes.

Dans ce petit travail on va s'intéresser à cette décomposition en valeurs singulières, tout d'abord on va bien préciser le problème a résoudre, puis on expliquera la méthode utilisée pour le calcul, qui représente l'approche utilisée dans le code implémenté et amélioré en se servant des outils de parallélisation afin d'avoir les meilleurs performances possibles pour ce calcul.

# 1 Problématique

On sait que pour les matrices carrées diagonalisables, on a la possibilité de leurs appliquer une décomposition en valeurs propres [2], tel que :

$$A = XDX^{-1} \tag{1.1}$$

Avec :

D : une matrice diagonale de valeurs propres.
X : une matrice inversible de vecteurs propres.

Cette méthode ne peut pas être appliquée aux matrices rectangulaires et aux matrices non diagonalisables. Il fallait donc trouver une autre méthode applicable sur des matrices de n'importe quelle taille qu'elles soient diagonalisable ou pas, d'où l'apparition de la décomposition en valeurs singulières.

Son principe est très proche du principe de la décomposition en valeurs propres, sauf qu'elle s'applique sur n'importe quelle taille de matrice, y compris le cas ou le nombre de lignes m et beaucoup plus grand que le nombre de colonnes n.

Bien que la décomposition en valeurs singulière s'applique aussi bien aux matrices réelles que complexes, on contentera dans ce travail que des matrices à coefficients réels afin de faciliter un peu la tâche.

## 1.1 Le théorème de la SVD

Pour toute matrice A $\in \mathrm{R}^{m*n}$, il existe une matrice diagonale $U \in R^{m*m}$ et une matrice orthogonalee $V \in R^{rn*n}$ [3], telles que :

$$U^T AV = diag(\sigma_1, ..., \sigma_{min(m,n)}) = \Sigma \in R^{m*n} \tag{1.2}$$

Avec $\sigma_1 \geq \sigma_1 \geq ... \geq \sigma_{min(m,n)} \geq 0$.

En multipliant successivement l'équation (1.2) du théorème par $V^T$ puis par $U$, on obtient la forme de la décomposition en valeurs singulières la plus utilisée [4] :

$$A = U\Sigma V^T \tag{1.3}$$

Avec :

$U = [u_1, ..., u_m]$ : une matrice m×m unitaire.
$V = [v_1, ..., v_n]$ : une matrice n×n unitaire.
$\Sigma = diag[\sigma_1, ..., \sigma_r]$ : une matrice r×r diagonale avec coefficients réels et positifs, r=min(m,n) : représente le nombre de valeurs singulières, et les autres coefficients diagonaux sont nuls, on se retrouve avec les deux cas suivants :

- m ≥ n : Σ est de la forme : $(\Sigma_1, 0)$
- m ≤ n : Σ est de la forme : $\begin{pmatrix} \Sigma_1 \\ 0 \end{pmatrix}$

Le calcul de ces matrices peut être effectué à l'aide de la décomposition EVD des matrices carrées $A^T A$ et $AA^T$. En utilisant la décomposition (1.3) de A, on aura :

$$A^T A = V\Sigma U^T U\Sigma V^T = V\Sigma^2 V^T \tag{1.4}$$

$$AA^T = U\Sigma V^T V\Sigma U^T = U\Sigma^2 U^T \tag{1.5}$$

D'autre coté, $A^T A$ et $AA^T$ étant des matrices hermitiennes diagonalisables, elles se factorisent en produit de matrice diagonale $\Lambda$ et une matrice orthogonale des vecteurs propres X et Y, telles que :

$$A^T A = X\Lambda^2 X^T \tag{1.6}$$

$$AA^T = Y\Lambda^2 Y^T \tag{1.7}$$

Avec ces quatre dernières équations on obtient :

⋆ V=X et U=Y : les vecteurs propres de $A^T A$ et $AA^T$ respectivement.
⋆ $\Sigma^2 = \Lambda$ : les valeurs propres de $A^T A$ et $AA^T$ respectivement.

C'est de la qu'on déduit les valeurs singulières de A, qui sont les racines carrées des éléments de la diagonale de $\Lambda$ et les vecteurs singuliers à droite de A sont leur vecteurs propres correspondants.

## 1.2 Exemple de SVD

Voici un exemple explicite du calcul d'une décomposition en valeurs singulières de la matrice A suivante :

$$A = \begin{pmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{pmatrix}$$

On calcule $A^T A$ :

$$A^T.A = \begin{pmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{pmatrix} . \begin{pmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 11 & 1 \\ 1 & 11 \end{pmatrix}$$

Les valeurs propres correspondantes a cette matrice sont : $\sigma_1 = 12$ et $\sigma_2 = 10$.

Et leurs vecteurs propres sont : $v_1 = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$ $et$ $v_2 = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$ respectivement.

D'ou les deux matrices : $V = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$ $et$ $\Sigma = \begin{pmatrix} \sqrt{12} & 0 & 0 \\ 0 & \sqrt{10} & 0 \end{pmatrix}$

En effectuant le calcul $AA^T$, et suivre les mêmes étapes on obtient la matrice U suivant :

$$U = \begin{pmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{30}} \\ \frac{2}{\sqrt{2}} & \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{30}} \\ \frac{1}{\sqrt{6}} & 0 & \frac{-5}{\sqrt{30}} \end{pmatrix}$$

Le calcul des matrices U, V et $\Sigma$ intervenant dans la décomposition pour une telle matrice est vraiment simple, mais c'est très compliqué pour des matrices de très grandes taille, c'est pour cette raison la qu'il est intéressant de développer un algorithme qui fait le calcul, c'est ce qu'on va tenter de faire en se servant de l'approche de la bidiagonalisation des matrices qu'on va développer dans la section suivante.

# 2
# Approche utilisée

## 2.1 La bidiagonalisation d'une matrice

La bidiagonalisation est l'une des décompositions matricielles unitaires (orthogonales) telles que [5] :

$$B = U^T A V \tag{2.1}$$

B : une matrice bidiagonale supérieure.
A : une matrice qui peut être rectangulaire
U et V : des matrices unitaires.

La bidiagonalisation a une structure très similaire à la décomposition en valeurs singulières. Cependant, il est calculé au sein d'opérations finies, tandis que la décomposition en valeurs singulières nécessite des schémas itératifs pour trouver des valeurs singulières.

Pour les matrices denses, les matrices unitaires gauche et droite sont obtenues par une série de réflexions Householder appliquées alternativement à gauche et à droite. Ceci est connu sous le nom de bidiagonalisation Golub-Kahan. Pour les grandes matrices, elles sont calculées de manière itérative en utilisant la méthode de Lanczos , appelée méthode Golub-Kahan-Lanczos. [5]

Dans la section suivante on va proposer un algorithme de calcul de la décomposition en valeurs singulièstres qui se sert de cette approche de bidiagonalisation.

# 3
# Cas séquentiel

## 3.1 Description de l'algorithme proposé

Nous partons du principe que le calcul implicite du produit $A^TA$ pour évaluer les valeurs singulière et vecteurs singulières de matrice de départ $A$, celles-ci ont une relation directe avec la décomposition en valeurs propres de produit $A^TA$, mais malheureusement le calcul implicite de ce produit va multiplier le conditionnement de la patrice, et par conséquent on aura des résultats numériquement instable et le temps de calcul va se multiplier.

L'approche que nous allons suivre, comme indiqué précédemment, consiste à transformer d'abord la matrice de départ $A$ en une matrice bidiagonale $B$, et en suite travailler avec la matrice bidigonale obtenue en faisant appel à la décomposition en valeurs propres.

Nous proposons pour notre implémentation deux approches suivantes :

La première approche consiste à implémenter l'algorithme Golub-Kahan-Lanczos pour la bidiagonalisation d'une matrice quelconque, et par la suite en exploitant le fait que le produit de la transposée d'une matrice bidiagonal par elle même donne une matrice tridiagonal symétrique, il suffit donc de calculer la décomposition en "valeurs propres" de cette matrice pour qu'on puisse extraire les valeurs singulieres (le racine carré des valeurs propres de la matrice tridigonal), et de leurs vecteurs singuliers correspondantes (cela ce fait par accumulation des produits matricielles des matrices issues de la bidiagonalisation et de la décomposition en valeurs propres de la matrice tridiagonale, des deux cotés ). Dans cet approche la décomposition en valeurs propres de la matrice tridiagonale est implémenté en adoptant des routines de la librairie LAPACKE adaptés, à savoir : stevx..

Cet algorithme est basé sur une relation de récurrence double donnée par :
(voir la référence [6])

La deuxième approche, consiste à commencer du même algorithme de bidiagonalisation (Golub-Kahan-Lanczos), par contre la deuxième phase est mise en place par la méthode shifted-QR permettant d'accélérer la convergence de la méthode QR standard, et dans cette méthode phase on fait appel qu'à la routine LAPACKE de la factorisation QR.

## 3.2 Étude de performance théorique

L'algorithme de Golub-Kahan-Lanczos permet de bidiagonaliser la matrice d'entrée en un nombre fini d'itération, précisément la taille de la matrice est le nombre d'itérations nécessaire.

On revenant sur l'itération de Golub-Kahan-Lanczos on peut remarquer que la globalité de calcul se fait dans la partie multiplication matrice-vecteur, alors si on considère pas cette phase, le reste de l'algorithme peut se faire en $O(n)$ opérations arithmétique, et la partie multiplication matrice vecteur se fait en $O(d * n)$ avec $d$ le nombre moyenne des éléments non nuls dans une ligne de la matrice, ainsi la complexité totale peut être estimée à $O(d * n * m)$ ou $O(dn^2)$ si $n = m$, cet algorithme peut donc être le plus adapté pour les matrices creuses.

Concernant la partie transformation en matrice tridiagonale correspondante prend en générale $O(2n) = O(n)$, en remarquant que le produit $B^T B$, avec $B$ matrice bidiagonale, est une matrice tridiagonale symétrique. et donc nous allons stokcer que les deux parties : diagonale et sur-diagonale(= sous-diagonale) dans deux tableaux de 1 dimension chacun.

La partie décomposition en valeurs propres de la matrice tridiagonale, a été faite par deux approches, la première consistait à faire appel à la fonction LAPACKE_dstevx qui est une méthode généralisée pour les problèmes en valeurs propres de matrices tridiagonale symetriques. La deuxième approche consistait à implémenter la méthode QR en choisissant dans chaque itération le shift comme étant l'élément $T(n, n)$ de la matrice, dans chaque itération on fait appel à la factorisation QR en adoptant les routines LAPACK afin d'extraire les matrice Q et R, chaque itération a une complexité de $O(n^3)$, l'algorithme est très lent pour des matrices plus larges.

## 3.3 Étude de performance pratique

Afin d'évaluer les performances et comparer les deux approches, nous avons mesurer les performances dans un premier temps de l'algorithme Golub-Kahan-Lanczos et de tester sur différentes taille de la matrice, la FIGURE 3.1 présente les résultats obtenus.

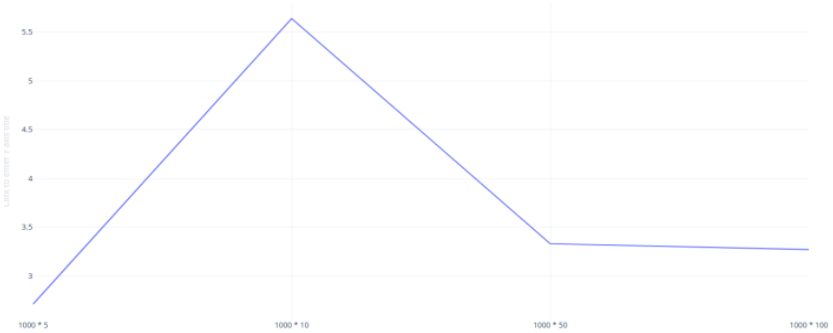


FIGURE 3.1 – Temps d'exécution (ms) en fonction de différentes tailles de la matrice pour l'algorithme de bidiagonalisation Golub-Kahan-Lanczos

Par la suite on a mesuré les performances de l'algorithme tout entier en utilisant les routines lapacke pour la deuxième phase, comme les montre la FIGURE 3.2 suivante :

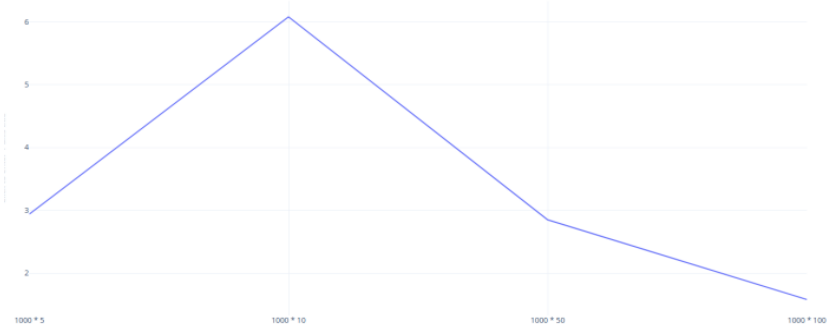


FIGURE 3.2 – Temps d'exécution (ms) en fonction des différentes tailles de la matrice pour l'algorithme svd avec les routines lapacke

Et puis en utilisant la méthode QR shifté, qu'on a représenté avec la FIGURE 3.3 la dessous :

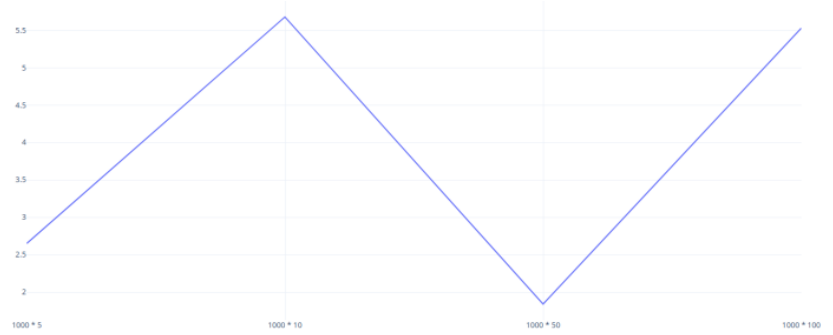


FIGURE 3.3 – Temps d'exécution (ms) en fonction des différentes tailles de la matrice pour l'algorithme svd avec la méthode QR shifté

## 3.4 Conclusion

On remarquant les courbes de performances, nous pouvons conclure que l'algorithme proposé, constitué par Golub-Kahan-Lanczos avec QR shifté présente des instabilité au niveau de résultats numériques et prend une bande passante plus large même pour les matrices modérées, ainsi que la partie QR prend la grand partie de l'algorithme et c'est cette partie qui génère l'instabilité numérique à cause des erreurs de calcul accumulés pendant les itérations on attendant la convergence, en plus on risque de perdre l'orthogonalité des vecteurs produites par la méthode QR ce qui conduira à une convergence lente même si on a choisie shifted-QR. Dans l'autre coté la bidiagonalisation ne pose pas de problème grave vu qu'il se fait en un nombre finit d'opération et l'algorithme de Golub-Kahan-Lanczos marche encore mieux avec les matrices larges et creuse, et se parallélise bien contre les autres méthodes comme l'utilisation des transformations de Householder.

# 4
# Cas parallèle

## 4.1 Description

La partie parallèle de notre implémentation consiste à introduire les directives OpenMP pour tirer profit des fonctionnalités de modèle de mémoire partagé.

L'idée principale est de paralléliser les boucles de multiplication matrice-matrice ou matrice-vecteur qui représente la majorité des parties de codes que se soit de coté bi-diagonalisation ou décomposition spectrale en shifted-QR.

## 4.2 Étude de performance théorique

L'algorithme de bidiagonalisation peut se parallèliser bien en openmp vu que les relations de récurrence –

## 4.3 Architectures parallèle visées

Dans notre implémentation parallèle, le modèle de programmation ciblé est la mémoire partagée avec Openmp comme indiqué précédemment, l'intérêt est de pouvoir diviser le travail sur les différents coeurs de calcul afin d'améliorer les performances de notre code.

## 4.4 Étude de performance pratique

Nous avons estimé les mesures de performances avec la commande time en linux afin d'avoir une idée sur le comportement des deux phases de l'algorithme (Bidiagonalisation et QR).

Et nous avons obtenu les résultats suivants pour les différents tailles de la matrice avec la variable OMP_NUM_THREAS=4 :

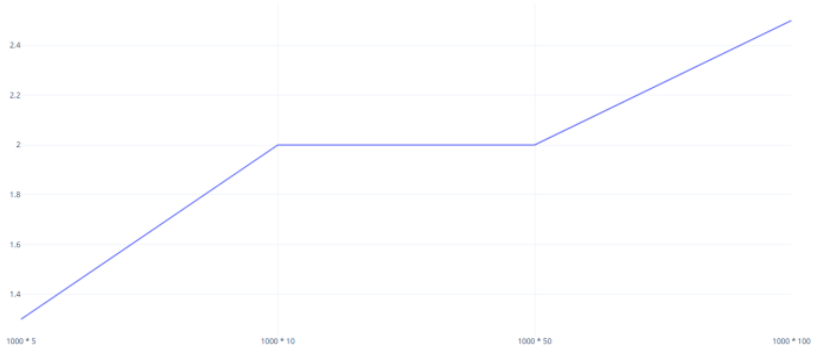


FIGURE 4.1 – Temps d'exécution dans le cas parallèle

## 4.5 Conclusion

Meme si on a utiliser le modèle openmp on voit pas beaucoup de différences au niveau du temps d'exécution, cela du au fait qu'il y a beaucoup de dépendances dans les boucles et qu'on utilise pas efficacement la mémoire cache, contrairement à l'implémentation Lapacke qui prend en considération tous ces mesures.

# Conclusion générale

Dans ce travail, nous avons implémenté premièrement une version séquentielle pour le calcul de la décomposition en valeurs et vecteurs singuliers d'une matrice réelle rectangulaire en utilisant la bidiagonalisation de Golub-Kahan-Lanczos dans un premier temps, et puis de faire appel à la décomposition spectrale de la matrice bidiagonale obtenue par deux méthodes : en évaluant la matrice tridiagonale symétrique associée à cette matrice bidigonale, nous avons utilisé les fonctions de la librairie lapacke adaptées aux problèmes symétrique de valeurs propres, et puis nous avons implémenté la méthode shifted-QR pour calculer les valeurs et vecteurs propres de la matrice tridiagonale et d'en extraire la décomposition en valeurs singuliers de la matrice de départ.

Les mesures de performances nous montrent qu'il faut utiliser les fonctions des librairies standard de calcul numérique qui sont déjà optimisées au niveau mathématique et algorithmique.

Les erreurs d'arrondi peuvent conduire à des fausses résultats et dans le cas par exemple de shifted-QR, à des pertes d'orthogonalité.

Il existe plusieurs algorithmes pour la bidiagonalisation et pour la SVD en générale, il faut en choisir la méthodes la plus adaptée pour chaque situation : pour les matrices larges et creuses, les matrices dont le nombre de lignes est très grandes par rapport aux nombre de colonnes ou l'inverse, etc.

# Bibliographie

[1] H.Gene Golub et F.Charles Van Loan, Matrix Computations, Third Edition, 1996.

[2] https ://fr.wikipedia.org/wiki/Dé9composition_en_valeurs_singulières.

[3] http ://josephsalmon.eu/enseignement/TELECOM/MDI720/SVD_fr.pdf.

[4] https ://fr.wikipedia.org/wiki/Décomposition_d'une_matrice_en_éléments_propres.

[5] https ://en.wikipedia.org/wiki/Bidiagonalization.

[6] http ://www.netlib.org/utk/people/JackDongarra/etemplates/node198.html.