

PROJET ANALYSE NUMÉRIQUE

ENCADRÉ PAR MME ZIDANI HASNAA

Application de la décomposition en valeurs singulières (SVD) pour la compression d'image

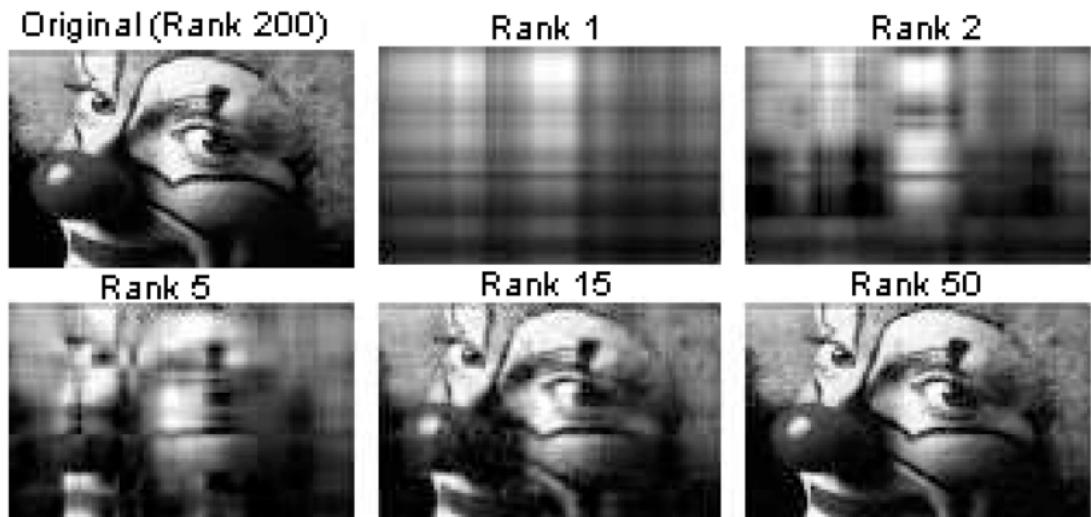


Table des matières

1	Introduction	3
2	Analyse théorique de la décomposition en valeurs singulières	4
2.1	Définitions et principes importants	4
2.2	Interprétation des valeurs propres et valeurs singulières	7
2.3	Principe général de la méthode SVD	7
2.4	Approximation de rang faible de matrices : le Théorème d'Eckart-Young	11
3	Implémentation numérique en Python	13
3.1	Explication des fonctions utilisées	13
3.2	Tests des fonctions sur des exemples basiques	15
3.3	Analyse des résultats sur divers exemples	16
3.4	Comparaison des fichiers avant et après compression	16
3.4.1	Image 1 - Lena512	17
3.4.2	Image 2 - Mandrill	18
3.4.3	Image 3 - Tigre	19
3.5	Performances de l'algorithme de compression	20
4	Conclusion	21
5	Sources	22

1 Introduction

La décomposition en valeurs singulières (SVD) constitue une généralisation de la notion de valeurs propres, spécifiquement adaptée aux matrices rectangulaires. Elle se présente comme un outil essentiel de factorisation pour de telles matrices, et propose ainsi une idée similaire à la diagonalisation des matrices carrées. Tout au long de ce projet, nous utiliserons couramment l'acronyme SVD pour faire référence à cette décomposition (Singular Values Decomposition pour les anglophones).

L'idée de réduire les matrices rectangulaires par le biais de la SVD est devenue une pratique classique depuis les années 1960, initiée par les travaux de l'informaticien Gene Howard Golub, et du mathématicien William Morton Kahan, qui ont proposé le premier algorithme de calcul de la SVD en 1965.

Cinq grands mathématiciens ont contribué au développement de cette méthode. Eugenio Beltrami (1835-1899), Camille Jordan (1838-1921), James Joseph Sylvester (1814-1897), Erhard Schmidt (1876-1959) et Hermann Weyl (1885-1955). Ces derniers ont été responsables de l'établissement de l'existence de la décomposition en valeurs singulières et du développement de sa théorie. Beltrami, Jordan et Sylvester sont arrivés à la décomposition à travers ce que nous appellerions maintenant l'algèbre linéaire ; Schmidt et Weyl l'ont approchée à partir d'équations intégrales.

Les premières démonstrations de la décomposition en valeurs singulières pour les matrices carrées réelles remontent à la fin du XIXe siècle lorsque Beltrami tout d'abord, puis Jordan ensuite, ont étudié les formes bilinéaires. Ils cherchaient à comprendre dans quelles conditions deux formes bilinéaires pouvaient être transformées l'une en l'autre par des opérations qui préservent les propriétés orthogonales. Alors que le travail de Beltrami a essentiellement porté sur les décompositions des matrices AA^T et A^TA , Jordan, quant à lui, a appliqué un raisonnement par récurrence à partir de la plus grande valeur singulière de A . En examinant les propriétés des valeurs singulières, Jordan a contribué à établir des méthodes plus générales pour la décomposition en valeurs singulières, complétant ainsi la perspective de Beltrami.

James Joseph Sylvester s'intéressa également à la décomposition en valeurs singulières en 1889. Ses travaux ont mis l'accent sur les formes quadratiques et les transformations associées aux matrices. En explorant la manière dont on peut diviser une matrice en valeurs singulières, Sylvester a examiné comment cela peut être utilisé dans le contexte des formes quadratiques.

Erhard Schmidt, quant à lui, a également joué un rôle important dans le développement de la théorie de la décomposition en valeurs singulières au début du XXe siècle. Ses contributions ont été marquées par une approche basée sur les équations intégrales. Schmidt a établi des liens entre la décomposition en valeurs singulières et les opérateurs linéaires définis par des équations intégrales, offrant ainsi une perspective différente et enrichissante sur cette méthode.

Hermann Weyl a contribué à la décomposition en valeurs singulières en apportant des nouvelles perspectives issues de l'analyse fonctionnelle et des opérateurs linéaires. De nos jours, une inégalité porte son nom mettant en corrélation les valeurs propres et les valeurs singulières.

La SVD sera explorée dans ce projet pour la compression d'images. L'objectif principal est d'implémenter une compression d'image en utilisant la SVD. Dans un premier temps nous définirons les notions mathématiques nécessaires à la compréhension de la SVD avant de détailler de manière théorique cette décomposition. Ensuite nous implémenterons la SVD en Python en illustrant sur divers exemples. Enfin, nous discuterons des résultats obtenus .

2 Analyse théorique de la décomposition en valeurs singulières

2.1 Définitions et principes importants

1. Valeurs propres

Considérons A une matrice réelle de taille $n \times n$.

Définition 2.1. Un scalaire λ est appelé valeur propre de A s'il existe un vecteur colonne non nul v tel que :

$$Av = \lambda v$$

Le vecteur v vérifiant cette relation est appelé vecteur propre correspondant à la valeur propre λ .

Définition 2.2. On appelle polynôme caractéristique associé à A et on note P_A le polynôme défini par :

$$P_A(\lambda) = \det(A - \lambda I_n)$$

Ce polynôme est de degré n et admet n racines complexes possiblement multiples.

Définition 2.3. On appelle les racines du polynôme caractéristique les valeurs propres de la matrice A . La multiplicité de la racine est appelée multiplicité algébrique.

Exemple :

Considérons A la matrice carrée

$$A = \begin{bmatrix} 2 & 1 \\ 4 & 3 \end{bmatrix}.$$

Le polynôme caractéristique P_A est donné par :

$$P_A(\lambda) = \det(A - \lambda I_2),$$

où I_2 est la matrice identité d'ordre 2.

Calculons le polynôme caractéristique :

$$P_A(\lambda) = \det \left(\begin{bmatrix} 2 - \lambda & 1 \\ 4 & 3 - \lambda \end{bmatrix} \right).$$

Calculons et simplifions cette expression :

$$P_A(\lambda) = (2 - \lambda)(3 - \lambda) - (1 \cdot 4) = \lambda^2 - 5\lambda + 5.$$

Maintenant, résolvons $P_A(\lambda) = 0$ pour obtenir les valeurs propres :

$$\lambda^2 - 5\lambda + 5 = 0.$$

Les solutions de cette équation quadratique sont les valeurs propres de A .

Appliquons la formule pour trouver les solutions d'un polynôme du second degré :

$$\lambda = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Dans notre cas, $a = 1$, $b = -5$, et $c = 5$. En substituant ces valeurs, nous obtenons les solutions de l'équation quadratique.

$$\begin{aligned}\lambda &= \frac{5 + \sqrt{(-5)^2 - 4(1)(5)}}{2 \cdot 1} \\ \lambda &= \frac{5 + \sqrt{25 - 20}}{2} \\ \lambda &= \frac{5 + \sqrt{5}}{2}\end{aligned}$$

et

$$\lambda = \frac{5 - \sqrt{5}}{2}.$$

Ainsi, les valeurs propres de la matrice A sont $\frac{5+\sqrt{5}}{2}$ et $\frac{5-\sqrt{5}}{2}$.

2. Valeurs singulières

Définition 2.4. Les valeurs singulières d'une matrice A réelle sont les racines carrées des valeurs propres de AA^T .

Exemple :

Considérons la matrice A suivante :

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

1. Calcul de $A^T A$

Pour obtenir $A^T A$, multiplions la transposée de A par A :

$$A^T A = \begin{bmatrix} 5 & 2 \\ 2 & 2 \end{bmatrix}$$

2. Calcul des valeurs propres $A^T A$

Pour $A^T A$, résolvons le polynôme caractéristique $\det(A^T A - \lambda I) = 0$:

$$\det \left(\begin{bmatrix} 5 - \lambda & 2 \\ 2 & 2 - \lambda \end{bmatrix} \right) = 0$$

Cela conduit à deux valeurs propres $\lambda_1 = 6$ et $\lambda_2 = 1$.

3. Calcul des valeurs singulières (σ_i) :

Les valeurs singulières sont les racines carrées des valeurs propres : $\sigma_1 = \sqrt{6}$, $\sigma_2 = \sqrt{1}$.

3. Définitions, Propriétés et Théorèmes utiles pour les démonstrations

Définition 2.5. Deux matrices carrées A et B sont dites semblables s'il existe une matrice inversible P telle que $A = P^{-1}BP$.

Définition 2.6. Une matrice U est dite unitaire si elle vérifie $UU^t = U^tU = I_d$

Proposition 2.1. Pour une matrice A ($m \times n$) quelconque,

- a) $A^T A$ ($n \times n$) est symétrique,
- b) Si λ est une valeur propre de $A^T A$ ou de AA^T , alors $\lambda \geq 0$.
- c) Deux matrices semblables ont les mêmes valeurs propres

Démonstration. Par les propriétés de la transposée,

$$(A^T A)^T = A^T (A^T)^T = A^T A,$$

□

Démonstration. Soit une valeur propre λ de $A^T A$ et un vecteur propre $\mathbf{v} \neq \mathbf{0}$, on peut écrire

$$\lambda \|\mathbf{v}\|^2 = \lambda(\mathbf{v} \cdot \mathbf{v}) = \mathbf{v} \cdot (\lambda \mathbf{v}) = \mathbf{v} \cdot (A^T A \mathbf{v}) = (\mathbf{A} \mathbf{v}) \cdot (\mathbf{A} \mathbf{v}) = \|\mathbf{A} \mathbf{v}\|^2 > 0.$$

Puisque $\|\mathbf{v}\| > 0$, on en déduit que $\lambda > 0$.

□

Démonstration. Soit A et B deux matrices semblables, c'est-à-dire qu'il existe une matrice inversible P telle que $B = P^{-1}AP$.

Soit λ une valeur propre de A et \mathbf{v} un vecteur propre correspondant, c'est-à-dire $A\mathbf{v} = \lambda\mathbf{v}$. Nous voulons montrer que λ est aussi une valeur propre de B avec le même vecteur propre \mathbf{v} .

Considérons $B\mathbf{v}$:

$$B\mathbf{v} = P^{-1}AP\mathbf{v}$$

Maintenant, nous savons que $A\mathbf{v} = \lambda\mathbf{v}$, nous pouvons substituer cela :

$$B\mathbf{v} = P^{-1}A(\lambda\mathbf{v})$$

En utilisant la propriété associée à l'inversibilité de P , nous pouvons réarranger cela :

$$B\mathbf{v} = \lambda P^{-1}\mathbf{v}$$

En multipliant des deux côtés par P , nous obtenons :

$$BP\mathbf{v} = \lambda\mathbf{v}$$

Cela montre que λ est une valeur propre de B avec le même vecteur propre \mathbf{v} .

Ainsi, si A et B sont semblables, elles partagent les mêmes valeurs propres. \square

Définition 2.7. Pour toute matrice $A = (a_{ij}) \in \mathbb{R}^{n \times m}$, on définit sa norme de Frobenius par :

$$\|A\|_F = \sqrt{\sum_{i=1}^n \sum_{j=1}^m a_{ij}^2} = \sqrt{\text{tr}(A^T A)},$$

où la trace d'une matrice est égale à la somme de ses éléments diagonaux.

Proposition 2.2. Soit A une matrice $m \times n$ et Q une matrice orthogonale $n \times n$, alors la norme de Frobenius est invariante par multiplication par une matrice orthogonale :

$$\|QA\|_F = \|A\|_F$$

Démonstration. Soit A une matrice, et Q une matrice orthogonale. Nous voulons montrer que $\|QA\|_F = \|A\|_F$. Par définition de la norme de Frobenius d'une matrice A , nous avons

$$\|A\|_F = \sqrt{\sum_{i,j} |a_{ij}|^2}$$

Maintenant, considérons $\|QA\|_F$ où Q est une matrice orthogonale c'est à dire $Q^T Q = Id$:

$$\begin{aligned} \|QA\|_F &= \sqrt{\text{tr}((QA)^T QA)} \\ &= \sqrt{\text{tr}(A^T Q^T QA)} \\ &= \sqrt{\text{tr}(A^T A)} \\ &= \|A\|_F \end{aligned}$$

Finalement, ceci est équivalent à la norme de Frobenius de A , donc $\|QA\|_F = \|A\|_F$. Cela démontre que la multiplication par une matrice orthogonale laisse la norme de Frobenius invariante. \square

Théorème 2.1. *Théorème spectral : Soit A une matrice symétrique réelle ($n \times n$). Alors, il existe une matrice orthogonale Q et une matrice diagonale D telles que*

$$A = QDQ^T,$$

où Q^T est la transposée de Q et D a les valeurs propres de A sur sa diagonale.

Proposition 2.3. *Soit $A \in \mathbb{M}_{n,p}(\mathbb{R})$, $P \in \mathbb{M}_p(\mathbb{R})$ inversible et $Q \in \mathbb{M}_n(\mathbb{R})$ inversible. Alors :*

- (a) $\text{rg}(AP) = \text{rg}(A)$
- (b) $\text{rg}(QA) = \text{rg}(A)$

4. Pseudo inverse

Définition 2.8. *Si A est une matrice de dimensions $m \times n$, la pseudo-inverse A^+ est définie par la relation :*

$$A^+ = (A^T A)^{-1} A^T$$

ou

$$A^+ = A^T (AA^T)^{-1}$$

Cette définition est basée sur la décomposition en valeurs singulières (SVD) de la matrice A . La pseudo-inverse généralise le concept d'inverse matriciel pour les matrices qui ne sont pas carrées ou de rang incomplet.

2.2 Interprétation des valeurs propres et valeurs singulières

Les valeurs propres et les valeurs singulières sont des outils mathématiques qui servent à simplifier et à comprendre des données dans différents domaines.

Par exemple, dans le cadre de la matière "Analyse de données" de ce second semestre, l'importance des valeurs propres se manifeste à travers des méthodes telles que l'Analyse en Composante Principale (ACP). Cette méthode consiste à résumer l'information qui est contenue dans une large base de données en un certain nombre de variables synthétiques appelées composantes principales. Les données vont être vues comme une matrice dans laquelle, on cherchera à calculer les valeurs propres. Ces valeurs propres vont nous permettre de déterminer le nombre de composantes principales optimal pour notre analyse en composantes principales. Cette méthode est d'ailleurs utilisée dans divers domaines, notamment en finance, par exemple pour l'analyse d'un portefeuille pour un client. Dans cet exemple, les valeurs propres peuvent être utilisées pour déterminer les principaux facteurs qui contribuent à la variabilité des rendements d'un portefeuille. D'autre part, les valeurs singulières sont importantes pour la décomposition en valeurs singulières (que nous détaillerons un peu plus loin dans ce projet). La SVD peut être appliquée à la matrice des rendements des actifs pour extraire des informations sur les composantes les plus significatives et pour construire de meilleurs portefeuilles.

Dans le cadre de ce projet, nous verrons que la décomposition en valeurs singulières (SVD) est utilisée pour compresser des images tout en conservant l'essentiel de l'information visuelle. Les valeurs singulières représentent l'importance des différents composants de l'image. Quant aux valeurs propres, elles sont utilisées pour représenter les caractéristiques principales de l'image.

2.3 Principe général de la méthode SVD

La diagonalisation d'une matrice est souvent très utile pour calculer des puissances de cette matrice ou pour avoir une idée de son comportement. Malheureusement, toutes les matrices ne sont pas diagonalisables, et ce procédé ne peut pas s'appliquer aux matrices qui sont rectangulaires. La décomposition en valeurs singulières est un procédé qui, dans certains cas, peut remplacer la diagonalisation.

Théorème 2.2. *Soit A une matrice $n \times m$ à coefficients complexes. Alors A peut s'écrire $M = U\Sigma V$ où*

- U est une matrice $n \times n$ orthogonale.
- V est une matrice $m \times m$ orthogonale.
- Σ est une matrice diagonale d'ordre n dont les coefficients sur la diagonale sont les valeurs singulières de M .

Cette décomposition s'appelle la décomposition en valeurs singulières de M .

Démonstration. Soit $A \in \mathcal{M}_{n,m}(\mathbb{R})$, montrons que $A = U\Sigma V^T$ avec $U \in \mathcal{M}_{n,n}(\mathbb{R})$ unitaire, $\Sigma \in \mathcal{M}_{n,m}(\mathbb{R})$ diagonale rectangulaire, et $V \in \mathcal{M}_{m,m}(\mathbb{R})$ unitaire.

On sait que $A^T A$ est symétrique positive, donc nous avons les deux propriétés suivantes :

- Elle admet des valeurs propres positives (car $A^T A$ est positive)
- D'après le Théorème spectral, il existe une base orthonormée $(V_i)_{i \in \llbracket 1, m \rrbracket}$ de vecteurs propres tels que $(AA^T = V\mathcal{D}_\lambda V^T)$, où V est une matrice unitaire (orthogonale) associée à $(V_i)_{i \in \llbracket 1, m \rrbracket}$, de dimension (m, m) , et \mathcal{D}_λ est la matrice diagonale de dimension (m, m) constituée des valeurs propres $(\sigma_i)_{i \in \llbracket 1, m \rrbracket}$ de $A^T A$.

$$\mathcal{D}_\lambda = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_m \end{bmatrix}$$

$\sqrt{(\lambda_i)_{i \in \llbracket 1, m \rrbracket}}$ sont appelés les valeurs singulières de A appellons les $(\sigma_i)_{i \in \llbracket 1, m \rrbracket}$.

Posons D tel que $\exists r \in \llbracket 1, \min(n, m) \rrbracket$ où $ir \Leftrightarrow \lambda_i = 0$

$$D == \begin{bmatrix} \sigma_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_r & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{bmatrix}$$

Construisons la matrice U de dimension (n, n) , nous allons réécrire U sous forme de matrice bloc de la manière suivante $U = [U^R \ U^N]$ où U^R est de dimension (n, r) et U^N est de dimension $(n, n - r)$. L'objectif de cette factorisation est de séparer les cas où nous avons des valeurs propres non nulles de celles où nous avons des valeurs propres nulles. Définissons U^R :

$$U^R = AV^R \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\sigma_r} \end{bmatrix}$$

Ici V^R représente les r premiers vecteurs de V . Les dimensions des matrices A , V^R et de l'autre matrice étant de dimension (n, m) , (m, r) et (r, r) le produit matriciel est bien défini. Par conséquent, la matrice U^R est de dimension (n, r) , le résultat recherché. Afin de correspondre à la définition de U , vérifions que les vecteurs de U^R constituent une famille orthonormée.

$$\begin{aligned} (U^R)^T U^R &= \left(AV^R \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\sigma_r} \end{bmatrix} \right)^T AV^R \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\sigma_r} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\sigma_r} \end{bmatrix} (V^R)^T A^T A V^R \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\sigma_r} \end{bmatrix} \end{aligned}$$

Démontrons que

$$(V^R)^T A^T A V^R = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_r \end{bmatrix}$$

Posons

$$D_\lambda^R = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_r \end{bmatrix}$$

On a

$$D_\lambda = \begin{bmatrix} D_\lambda^R & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_r & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{bmatrix}$$

On sait que $A^T A = V D_\lambda V^T \implies V^T A^T A V = D_\lambda$

Or

$$\begin{aligned} V^T A^T A V &= ([V^R \ V^M])^T A^T A \begin{bmatrix} V^R \\ V^M \end{bmatrix} \\ &= \begin{bmatrix} (V^R)^T \\ (V^M)^T \end{bmatrix} A^T A [V^R \ V^M] \\ &= \begin{bmatrix} (V^R)^T A^T A V^R & (V^R)^T A^T A V^M \\ (V^M)^T A^T A V^R & (V^M)^T A^T A V^M \end{bmatrix} \end{aligned}$$

avec V^M les $m - r$ derniers vecteurs propres

Par identification, on obtient $D_\lambda^R = (V^R)^T A^T A V^R$

$$\text{Donc } (U^R)^T U^R = \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\sigma_n} \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{bmatrix} \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\sigma_n} \end{bmatrix} = Id_r$$

On déduit le résultat de la dernière égalité car on effectue le produit de matrices diagonales donc commutatives entre elles, de plus on rappelle que $\sqrt{(\lambda_i)} = \sigma_i$. Par conséquent les vecteurs de U^R forment une base orthonormée. Pour finir complétons U en définissant à U^N , comme la famille de vecteur orthonormées entre-eux et à ceux de U^R , son existence nous est donnés par application du théorème de la base incomplète et du procédé de Gram-Schmidt, ainsi U est unitaire.

$$\Sigma = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad \Sigma = \begin{bmatrix} D^R & 0 \\ 0 & 0 \end{bmatrix} \quad D = \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_r \end{bmatrix}$$

$$\begin{aligned}
U \sum V^T &= [U^R \ U^N] \begin{bmatrix} D^R & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} (V^R)^T \\ (V^M)^T \end{bmatrix} \\
&= [U^R \ U^N] \begin{bmatrix} (D^R V^R)^T \\ 0 \end{bmatrix} \\
&= U^R D^R (V^R)^T \\
&= A V^R \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\sigma_r} \end{bmatrix} D^R (V^R)^T \\
&= A V^R \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\sigma_r} \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_r \end{bmatrix} (V^R)^T \\
&= A V^R I d(V^R)^T \\
&= A V^R (V^R)^T \\
&= A I d \\
&= A
\end{aligned}$$

Démonstration. Vérifions maintenant que les valeurs propres de A sont bel et bien les racines carrées des valeurs propres de AA^T .

Soit A une matrice $m \times n$ admettant une décomposition en valeurs singulières :

$$A = U \Sigma V$$

Où :

- U est une matrice unitaire de taille $m \times m$ ($UU^t = I_d$).
- Σ est une matrice diagonale $m \times n$ avec les valeurs singulières de A , $\sigma_1, \dots, \sigma_r$, sur la diagonale (r est le rang de A).
- V est une matrice unitaire de taille $n \times n$ ($VV^t = I_d$).

Nous obtenons ainsi :

$$AA^t = (U \Sigma V)(U \Sigma V)^t = (U \Sigma V)(V^t \Sigma^t U^t) = U \Sigma \Sigma^t U^t$$

On remarque ici que AA^t et $\Sigma \Sigma^t$ sont semblables car il existe U inversible ($UU^t = I_d$ d'où U admet comme inverse U^t) tel que $AA^t = U \Sigma \Sigma^t U^{-1}$.

$\Sigma \Sigma^t$ est une matrice diagonale admettant comme valeur propre $\sigma_1^2, \dots, \sigma_r^2$. Or comme AA^t et $\Sigma \Sigma^t$ sont semblables, les valeurs propres de AA^t sont aussi $\sigma_1^2, \dots, \sigma_r^2$.
($\sigma_1, \dots, \sigma_r$ sont les valeurs singulières de A)

On a donc montré que les valeurs singulières de A sont les racines carrées des valeurs propres de AA^t . □

Exemple de la SVD d'une matrice A quelconque : On reprend la matrice de l'exemple précédent

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

1. Calcul de $A^T A$ et AA^T

Pour obtenir $A^T A$, multiplions la transposée de A par A :

$$A^T A = \begin{bmatrix} 5 & 2 \\ 2 & 2 \end{bmatrix}$$

$$AA^T = \begin{bmatrix} 5 & 2 & 1 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

2. Calcul des valeurs propres et vecteurs propres de $A^T A$

Pour $A^T A$, résolvons le problème caractéristique $\det(A^T A - \lambda I) = 0$:

$$\det \left(\begin{bmatrix} 5 - \lambda & 2 \\ 2 & 2 - \lambda \end{bmatrix} \right) = 0$$

Cela conduit à deux valeurs propres $\lambda_1 = 6$ et $\lambda_2 = 1$. Les vecteurs propres associés sont obtenus en résolvant $(A^T A - \lambda I)v = 0$. Pour $\lambda_1 = 6$, $v_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$. Pour $\lambda_2 = 1$, $v_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$.

3. Calcul des valeurs propres et vecteurs propres de AA^T

Pour AA^T , résolvons le problème caractéristique $\det(AA^T - \lambda I) = 0$:

$$\det \left(\begin{bmatrix} 5 - \lambda & 2 & 1 \\ 2 & 1 - \lambda & 0 \\ 1 & 0 & 1 - \lambda \end{bmatrix} \right) = 0$$

Cela conduit à trois valeurs propres $\lambda_1 = 0$, $\lambda_2 = 1$, et $\lambda_3 = 6$. Les vecteurs propres associés sont obtenus en résolvant $(AA^T - \lambda I)u = 0$. Pour $\lambda_1 = 0$, $u_1 = \begin{bmatrix} -1 \\ 2 \\ 1 \end{bmatrix}$. Pour $\lambda_2 = 1$, $u_2 = \begin{bmatrix} 0 \\ -1 \\ 2 \end{bmatrix}$. Pour $\lambda_3 = 6$, $u_3 = \begin{bmatrix} 5 \\ 2 \\ 1 \end{bmatrix}$.

2. Construction des matrices U , Σ , et V

- U est formée par les vecteurs propres de AA^T normalisés :

$$U = \begin{bmatrix} \frac{\sqrt{5}}{\sqrt{6}} & 0 & -\frac{1}{\sqrt{6}} \\ \frac{\sqrt{2}}{\sqrt{15}} & -\frac{1}{\sqrt{5}} & \frac{\sqrt{2}}{\sqrt{3}} \\ \frac{1}{\sqrt{30}} & \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{6}} \end{bmatrix}$$

- Σ est une matrice diagonale avec les valeurs singulières :

$$\Sigma = \begin{bmatrix} \sqrt{6} & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

- V est formée par les vecteurs propres de $A^T A$ normalisés :

$$V = \begin{bmatrix} \frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{bmatrix}$$

Nous obtenons ainsi la décomposition en valeur singulière de notre matrice. A peut être écrite comme

$$A = U\Sigma V^T.$$

2.4 Approximation de rang faible de matrices : le Théorème d'Eckart-Young

Dans le domaine du traitement d'images par exemple, il peut être intéressant de travailler sur une matrice 'proche' de la matrice image (rappelons qu'une image de taille $m \times n$ est une matrice de $m \times n$ pixels, où chaque pixel admet une valeur comprise entre 0 et 255).

C'est en ce sens qu'intervient l'approximation de rang faible : si M est la matrice image sur laquelle on travaille, on recherche une matrice \tilde{M} de rang strictement inférieur au rang de M qui rend minimum une distance entre M et \tilde{M} . Cette matrice existe et est liée à la SVD que nous avons énoncée précédemment. Le théorème d'Eckart-Young énonce ces résultats.

Théorème 2.3. Théorème d'Eckart-Young : Soit M une matrice de dimensions $m \times n$ avec une décomposition en valeurs singulières (SVD) $M = U\Sigma V^T$, où U est une matrice orthogonale $m \times m$, Σ est une matrice diagonale $m \times n$ avec les valeurs singulières de M sur la diagonale, et V est une matrice orthogonale $n \times n$. Soit r le rang de M .

Alors, pour toute approximation de rang k ($1 \leq k \leq r$), la meilleure approximation de rang k selon la norme de Frobenius est donnée par :

$$\min_{\text{rang}(\tilde{M})=k} \|M - \tilde{M}\|_F = \sqrt{\sum_{i=k+1}^r \sigma_i^2}$$

où σ_i sont les valeurs singulières de M , et $\text{rang}(\tilde{M})$ est le rang de la matrice \tilde{M} . La matrice optimale \tilde{M} est obtenue en conservant les k plus grandes valeurs singulières de M et en les utilisant pour former la matrice Σ_k dans la décomposition $M_k = U\Sigma_k V^T$, où M_k est l'approximation de rang k de M .

Démonstration. Pour établir ce résultat, nous partons de la définition de la norme de Frobenius.

$$\|M\|_F^2 = \text{Tr}(M^T M).$$

Nous savons que la multiplication par une matrice orthogonale laisse la norme $\|\cdot\|_F$ invariante, nous obtenons ainsi la propriété suivante3.

$$\|QM\|_F = \|M\|_F (**)$$

$$\begin{aligned} \|M - \tilde{M}\|_F^2 &= \|U\Sigma V^t - \tilde{M}\|_F^2 \\ &= \|U^T(U\Sigma V^t - \tilde{M})V\|_F^2 && (***) \\ &= \|\Sigma - U^T \tilde{M}V\|_F^2 \\ &= \sum_{i=1}^m \sum_{j=1}^n (\sigma_{ii} - c_{ij})^2 && \text{par définition de la norme de Frobenius} \\ &= \sum_{i=1}^r (\sigma_{ii} - c_{ii})^2 + \sum_{i>r} (\sigma_{ii})^2 + \sum_{i \neq j} c_{ij}^2 \end{aligned}$$

où $C = U^T \tilde{M}V$ et les $c_{ij} = (U^T \tilde{M}V)_{ij}$

La dernière égalité est déduite par le fait que M est une matrice de rang r donc Σ est aussi une matrice de rang r . En effet puisque :

$$M = U\Sigma V^T \Leftrightarrow \Sigma = U^T M V$$

et que les matrices U^T et V sont orthogonales donc inversibles (par définition) par la propriété 2.3 Σ a bien le même rang que M .

Le but étant de minimiser $\|M - \tilde{M}\|_F^2$, nous allons minimiser chaque somme. On pose :

$$\begin{aligned} -c_{ii} &= 0 \text{ si } i > r \text{ et} \\ -c_{ij} &= 0 \text{ si } i \neq j. \end{aligned}$$

On aurait tendance à poser également $\sigma_{ii} - c_{ii} = 0$ mais cela impliquerait que $\sigma_{ii} = c_{ii}$ donc que le rang de la matrice C serait égal à celui de la matrice Σ à savoir r . Or étant donné que nous cherchons à minimiser la matrice M de rang r par une matrice de rang inférieur, nous allons trouver un autre moyen de minimiser ce dernier terme.

Ainsi, pour minimiser $\sum_{i=1}^r |\sigma_{ii} - c_{ii}|^2$ en ayant k valeurs propres non nuls parmi les c_{ii} , il est logique d'annuler les k plus grandes valeurs singulières donc les k premières car elles sont dans l'ordre décroissant. Nous avons ainsi

$$c_{ij} = \begin{cases} 0 & \text{si } i \neq j \\ 0 & \text{si } i = j \text{ et } i > k \\ \sigma_{ii} & \text{si } i = j \text{ et } i \leq k \end{cases}$$

Finalement, nous obtenons :

$$\begin{aligned}\|M - \tilde{M}\|_F^2 &= \sum_{i=1}^r (\sigma_{ii} - c_{ii})^2 + \sum_{i>r} (c_{ii})^2 + \sum_{i \neq j} c_{ij}^2 \\ &= \sum_{i=k+1}^r (\sigma_{ii})^2\end{aligned}$$

D'où :

$$\|M - \tilde{M}\|_F = \sqrt{\sum_{i=k+1}^r (\sigma_{ii})^2}$$

Nous venons ainsi de montrer que la meilleure approximation de rang faible d'une matrice au sens de la norme de Frobenius est une SVD tronquée de cette matrice. \square

3 Implémentation numérique en Python

Afin de pouvoir implémenter la décomposition en valeur singulière, nous devons construire les matrices U, V, Σ . Pour cela, nous allons procéder en plusieurs étapes et utiliser divers fonctions.

3.1 Explication des fonctions utilisées

La puissance itérée

La puissance itérée est une méthode qui permet de déterminer la plus grande valeur propre ainsi que le vecteur propre associé d'une matrice.

```
#determine la plus grande valeur propre ainsi que son vecteur propre associe
def puissancei(A,tol):
    n = A.shape[0]
    x0 = np.arange(n)#initialise x0 en tant que vecteur aleatoire de taille n
    x0 = x0/np.sqrt(np.dot(x0,x0)) #normalisation du vecteur initial
    y = np.dot(A,x0)
    x1 = y/np.sqrt(np.dot(y,y))

    while (abs(abs(np.dot(x0,x1))-1) > tol):
        x0 = x1
        y = np.dot(A,x0)
        x1 = y/np.sqrt(np.dot(y,y))

    Lam = np.sqrt(np.dot(y,y))
    return Lam,x1
```

Que fait la fonction ?

Cette fonction commence par initialiser un vecteur aléatoire, puis multiplie répétitivement ce vecteur par la matrice, le normalisant à chaque étape. La boucle while permet d'itérer le processus jusqu'à obtenir un degré de précision satisfaisant (tol) sur le vecteur propre. Une fois que la précision est atteinte, la fonction renvoie la valeur propre et le vecteur propre correspondant.

La méthode de déflation

La méthode de déflation est une technique permettant, en connaissant la valeur propre de plus grand module d'une matrice et un vecteur propre associé, de trouver la seconde valeur propre dont le module est le plus grand.

```
def deflation(A, p, tol) :
    #extrait le nombre de lignes de la matrice A :
    n = A.shape[0]

    #vecteur des valeurs propres :
    lam = np.zeros(p)

    #matrice des vecteurs propres :
    vect = np.zeros((n,p))

    #Calcul de la 1ere valeur propre en appelant la fonction puissance :
    lam[0], vect[:,0] = puissancei(A,tol)
    for k in range(1,p) :
        #calcul de la matrice deflattee A :
        A = A - (lam[k-1]) * np.outer(vect[:,k-1],vect[:,k-1])

        #calcul valeur et vecteur propre associe a la matrice deflatte A :
        val, vec = puissancei(A,tol)
        lam[k], vect[:,k] = val, vec

    #retourne les valeurs et vecteurs propres calcules :
    return lam, vect
```

Que fait la fonction ?

Le principe de notre fonction est de calculer de manière successive (grâce à une boucle for) les valeurs/vecteurs propres de la matrice A en utilisant la fonction puissance itérée. Pour cela, dans un premier temps nous calculons la première valeur propre ainsi que le premier vecteur propre associé de notre matrice A . Puis nous retranchons à A le produit du dernier vecteur propre avec sa transposée (nous obtenons une matrice), cela a pour effet de rendre ce vecteur orthogonal à la nouvelle matrice. Cette opération est effectuée grâce à la ligne de code $A = A - (\lambda_{k-1}) * \text{np.outer}(\text{vect}[:,k-1],\text{vect}[:,k-1])$ où np.outer est appelé le produit externe . Ainsi sa valeur propre associée est 0. Maintenant la deuxième valeur propre est la plus grande , nous pouvons donc appliquer la puissance itérée pour trouver celle-ci. Nous répétons ce processus au moyen d'une boucle for jusqu'à trouver toutes les valeurs et vecteurs propres.

La fonction SVD

Que fait la fonction ?

La méthode SVD se fait en plusieurs étapes, le but étant de calculer successivement V, S et U . Pour trouver V et S , nous devons juste appliquer la méthode de déflation (3) sur $A^t A$ (cela nous permet d'obtenir les vecteurs propres qui constituent V et les valeurs propres qui lorsque l'on applique la racine carrée nous donne la diagonale de S).

Pour calculer U , nous devons revenir à la partie théorie. Nous avons écrit U sous forme de matrice bloc $U = [U^R \ U^H]$ avec r tel que $i > r \iff \lambda_i = 0$. Afin de faciliter le calcul de la matrice U , nous avons considérer qu'il n'existe pas de i tel que $\lambda_i = 0$. Nous nous sommes permis cela car en informatique le 0 au sens de réel (float) est inexistant à cause des imprécisions de la machine ainsi $r = \min(n, m)$ où n est le nombre de lignes et m le nombre de colonnes. De plus pour une question de cohérence au niveau des dimensions de la matrice U , nous devons considérer deux cas le cas où $n \leq m$ et le cas où $n \geq m$. Dans le cas où $n \leq m$, nous avons $r = \min(n, m) = n$, par conséquent U^H est de dimension $(n, n-r)=(n,0)$ c'est à dire qu'elle n'existe pas. Nous obtenons ainsi :

$$U = U^R = AV^N \begin{bmatrix} \frac{1}{\lambda_1} & 0 & \dots & 0 \\ 0 & \frac{1}{\lambda_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\lambda_r} \end{bmatrix} = AV^N \begin{bmatrix} \frac{1}{\lambda_1} & 0 & \dots & 0 \\ 0 & \frac{1}{\lambda_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\lambda_n} \end{bmatrix}$$

avec V^N les n premiers vecteurs de V . Dans le cas contraire ($n \geq m$), nous transposons notre matrice et nous appliquons la SVD sur la matrice transposée , puis nous retransposons la matrice à la fin afin d'obtenir le résultat souhaité. Cela nous a amené au code suivant :

```

#Algo de SVD :
def svd(A,tol):
#Etape 1
    t = False
    n = A.shape[0] #nombre de lignes de A
    m = A.shape[1] #nombre de colonnes de A
    if n > m:
        t = True
        A = A.T
        n = A.shape[0]
        m = A.shape[1]

#Etape 2
    p = min(n,m) #nombre max de valeur propre pour une matrice n*m
    S = np.zeros((n,m))
    S_inv = np.zeros((m,m))
#Etape 3
    #construction de la matrice V :
    s, V = deflation(A.T @ A, m, tol)
#Etape 4
    #construction de la matrice S et S_inv:
    for i in range(p):
        S[i,i] = np.sqrt(s[i])
        S_inv[i,i] = 1/np.sqrt(s[i])
#Etape 5
    #construction de la matrice U :
    U = (A @ V @ S_inv)[:, :n]

#Etape 6
    if t==True:
        return V, S.T, U.T
    return U, S, V.T

```

1. Nous vérifions si $n \leq m$ ou le cas contraire (dans ce cas on transpose notre matrice et on déclare une variable booléenne t qu'on initialise à TRUE pour ne pas oublier de transposer notre résultat).
2. Nous initialisons deux matrices S et S_{inv} en tant que matrices nulles qui nous seront utiles pour la suite du code.
3. Nous appliquons déflation pour trouver la matrice V et la matrice diagonale S
4. Nous remplissons la matrice S_{inv} (utile pour calculer U) et S avec les valeurs singulières correspondant à la racine carrée des valeurs propres de $A^T A$
5. Nous calculons U
6. La fonction renvoie ensuite les matrices U, S et V^T , qui représentent la décomposition en valeurs singulières de la matrice A en fonction de la valeur de notre variable booléenne t .

3.2 Tests des fonctions sur des exemples basiques

Avant de tester notre fonction svd sur des images, nous l'avons testée sur une matrice tridiagonale de taille (n,n) :

$$A = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & -1 & 2 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix}$$

Tout d'abord, nous avons pris un n petit ($n=3,4,..$) puis au fur et à mesure nous avons augmenté la valeur de ce n .

On a aussi calculer la norme de la différence entre la matrice de départ et le produit USV^t

Par exemple, pour $n=3$, nous obtenons 2 matrices très semblables, nous le voyons avec l'erreur qui est de l'ordre 10^{-15} :

```

Reconstitution USV^T:
[[ 2.00000000e+00 -1.00000000e+00 -3.88578059e-16]
 [-1.00000000e+00 2.00000000e+00 -1.00000000e+00]
 [-1.66533454e-16 -1.00000000e+00 2.00000000e+00]]
Matrice A de départ:
[[ 2. -1. 0.]
 [-1. 2. -1.]
 [ 0. -1. 2.]]
erreur: 1.2749133095875076e-15

```

FIGURE 1 – Affichage des résultats pour n=3

De même pour n = 30, l'erreur est de l'ordre de 10^{-5} ce qui nous rassure sur l'efficacité de ce programme.

```
erreur: 2.4335699084413005e-05
```

FIGURE 2 – Erreur pour n=30

3.3 Analyse des résultats sur divers exemples

Dans cette partie, nous allons appliquer notre algorithme de SVD pour compresser des images. En effet, les images sont considérées comme des matrices avec une taille assez conséquente qui prend de la place au niveau de la mémoire mais aussi du temps par rapport au temps de calcul. C'est pour cela que nous allons appliquer un résultat issue du théorème d'Eckart-Young qui énonce qu'une matrice peut être approximée par une matrice de rang inférieur sans trop perdre d'informations en gardant ses valeurs propres dominantes. Dans le cas des matrices représentant une image, cela signifie que nous pouvons extraire les parties les plus importantes de l'image, comme les contours, les textures, ou les zones de forte variation de couleurs, en conservant uniquement les premiers valeurs/vecteurs propres significatifs. Ainsi, même avec une version simplifiée de l'image, nous pouvons toujours reconnaître les éléments essentiels de celle-ci. Ce procédé s'inscrit plus largement dans le domaine de la compression d'image.

Une image compressée est une version plus petite de l'image d'origine. Elle est créée en utilisant des techniques spéciales pour réduire la quantité de données nécessaires pour la stocker ou la transmettre. Il existe deux types principaux de compression d'image : la compression avec perte et la compression sans perte. Dans la compression avec perte, des données sont supprimées de manière intelligente de l'image pour réduire sa taille. Cela peut entraîner une perte de qualité visuelle, mais la compression est souvent plus efficace. C'est le cas des fichiers JPEG ou MPEG. En revanche, la compression sans perte réduit la taille de l'image sans perdre aucune information. Cela est réalisé en utilisant des algorithmes qui exploitent les données de l'image. Bien que la compression sans perte conserve la qualité de l'image, elle est généralement moins efficace que la compression avec perte en termes de réduction de taille de fichier. C'est le cas des fichiers de type PNG. Les images compressées sont utilisées partout, des photos sur votre téléphone aux images sur le web. Dans notre cas, nous utiliserons la compression avec perte d'information.

3.4 Comparaison des fichiers avant et après compression

En considérant une matrice A de taille $n \times m$, d'après le théorème d'Eckart-Young, nous pouvons trouver un rang k inférieur au rang de A tel que la matrice A_k approche au mieux A . Ainsi en appliquant la SVD à A et en gardant les k premières valeurs singulières, on obtiendra $A_k = U_k S_k V_k^T$ où U_k représente les k premières colonnes de U , S_k les k premières valeurs singulières de S et V_k^T les k premières lignes de V^T . On aura ainsi $k \times m + k \times 1 + k \times n$ éléments conservés. D'où la matrice compressé A_k de "taille" $k(n + m + 1)$.

Ceci nous conduit à introduire la notion de facteur de compression. Il détermine à quel point l'image est réduite . Le facteur de compression est donné par la formule suivante :

$$F_{\text{compress}} = \frac{\text{Taille originale}}{\text{Taille compress}} = \frac{n \times m}{k(n+m+1)}$$

Ce dernier représente la réduction de la taille des images après compression par rapport à celle d'origine et permet de comparer l'image compressée à l'image originale (plus ce facteur est grand plus l'image est compressée.).

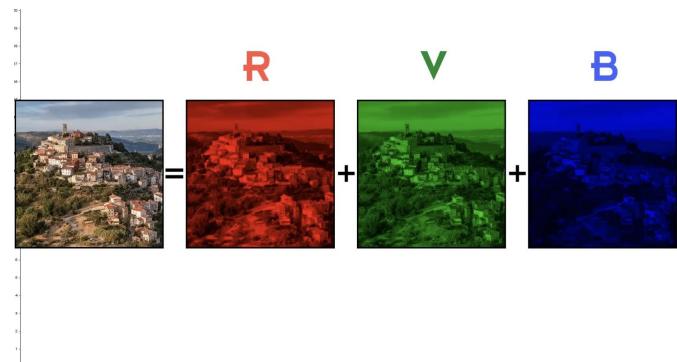
On remarquera que le facteur de compression est supérieur à 1. En effet la taille de l'image originale est supérieure à la taille de l'image compressée ce qui nous donne une condition sur la valeur de k :

$$\frac{n \times m}{k(n+m+1)} > 1 \Leftrightarrow k < \frac{n \times m}{n+m+1}$$

k est donc le plus petit entier vérifiant cette condition. Ainsi cette majoration nous permet de dire que si nous dépassons ce seuil, l'image compressée se rapprochera moins de l'image originale. Cependant, nous verrons par la suite que, pour certain type d'image, même en dépassant ce seuil, l'image compressée sera proche de l'originale.

Afin d'extraire des matrices U,S,V les matrices U_k, S_k, V_k , nous avons implémenté un algorithme très simple (que nous avons appelé compressé) qui prend en paramètre ces 3 matrices ainsi que l'entier k et renvoie les 3 matrices "compressées".

Nous allons maintenant comparer les résultats obtenus avec des images en couleur et des images en niveaux de gris. Il est important de noter que les images en couleur posent un défi supplémentaire en raison de leur complexité, avec plusieurs canaux de couleur à considérer. En effet, la couleur est définie par trois composantes qui sont le niveau de rouge, le niveau de vert et le niveau de bleu. Ainsi, nous aurons à considérer trois matrices pour effectuer l'algorithme de SVD afin d'aboutir à la compression d'image.



Nous avons donc choisi 3 images (2 de tailles carré et 1 rectangulaire) :



FIGURE 3 – Lena512

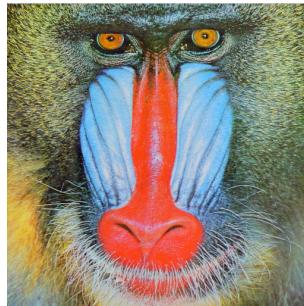


FIGURE 4 – Mandrill



FIGURE 5 – Tigre

3.4.1 Image 1 - Lena512

Commençons par comparer l'image "grise" pour différente valeur de k. Pour appliquer la SVD à ces images, nous avons utilisé la commande "mpimg.imread()" de la bibliothèque matplotlib.image qui permet de mettre une image sous forme de matrice avec comme coefficient la valeur de chaque pixel.

Enfin, nous utilisons la commande "np.clip(image, 0, 255)" qui nous permet de limiter les valeurs à l'intérieur du tableau image. Ainsi les valeurs plus petites que 0 seront égales à 0 et celles plus grandes que 255 seront égales à 255. Cela nous est particulièrement utile puisque chaque pixel correspond à une valeur entre 0 et 255.

Dans la suite nous avons testé l'algorithme de la SVD pour plusieurs valeurs de k. Cette image est de taille 512×512 , ainsi par la condition énoncé ci-dessus, on doit avoir $k < \frac{512 \times 512}{512+512+1} = 255,75$.



(a) $k = 10$

(b) $k = 100$

(c) $k = 255$

On remarque ici que si nous gardons 10 valeurs singulières (image (a)), l'image compressée ressortira avec des perturbations. Cela est due au fait que le k choisi est trop petit par rapport à la taille de la matrice. Cependant à partir d'un $k = 100$ on voit que l'image se rapproche beaucoup de l'originale, c'est-à-dire que prendre environ $\frac{1}{5}$ des données de l'image originale est suffisant pour approcher celle-ci. En prenant $k = 255$ l'image est aussi bien que l'originale.

Nous avons voulu tester en prenant un k supérieur à la condition ($k < 255,75$). En prenant $k=500$ par exemple :



FIGURE 7 – $k = 500$

Ici, l'image obtenue possède des perturbations. En effet notre algorithme de la SVD se base sur la recherche de valeurs propres et vecteurs propres ce qui implique des calculs de flottants. Or ce type de calculs se fait toujours avec une certaine précision et nous n'obtenons jamais les valeurs exactes.

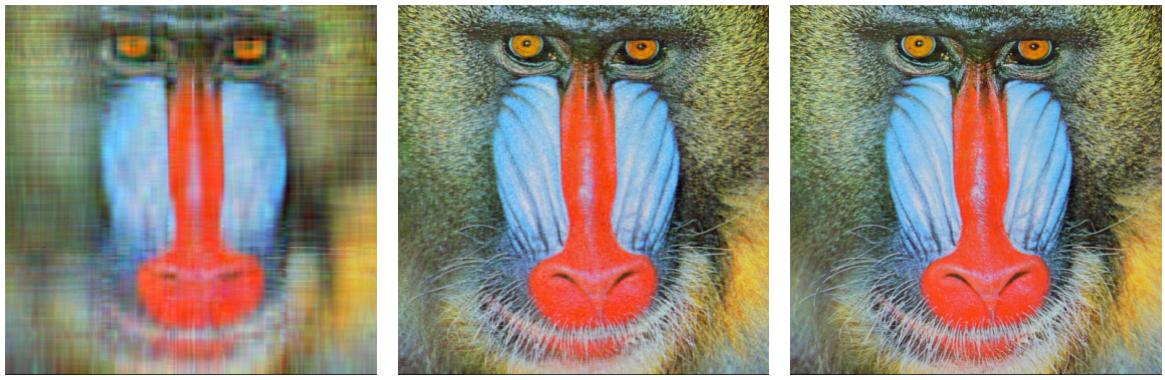
De plus l'algorithme de Déflation recherche les valeurs propres dans l'ordre décroissant donc les dernières valeurs propres sont souvent très proche de 0 ce qui explique ce manque de précision lorsque l'on rapproche notre k de la taille de l'image originale.

3.4.2 Image 2 - Mandrill

Ensuite, nous avons décidé d'appliquer la SVD sur une image colorée. On rappelle qu'une image colorée est composée de ce qu'on appelle 3 canaux, Rouge, Bleu et vert ainsi chaque composante de couleur peut être traitée séparément en les représentant par une matrice d'entiers compris entre 0 et 255 (et on appliquera la SVD à chacune de ces matrices). Nous ne pouvons donc pas appliquer directement la SVD à cette matrice. C'est pour cela qu'il a fallu extraire de l'image les 3 matrices correspondant aux 3 canaux.

Si on note "img" la matrice image alors la matrice correspondant au canal i est $\text{img}[:, :, i-1]$ pour $i = 1, 2, 3$. Après avoir obtenu les matrices U , S et V et fait le produit matricielle USV pour chaque canal, nous avons utiliser la commande "np.stack()" de Python qui nous permet d'empiler les 3 matrices obtenues en 1 seul matrice. Le paramètre "axis=-1" précise dans quel ordre nous les empilons, ici chaque matrice sera empiler à la fin de la "grande" matrice.

Testons maintenant l'algorithme de SVD pour différentes valeurs de k . Ici la taille de l'image est $(512, 512, 3)$ donc la condition pour le k : $k < \frac{512 \times 512 \times 3}{512 + 3 \times 512 + 1} = 383, 81$.



(a) $k = 10$

(b) $k = 100$

(c) $k = 383$

Nous obtenons, pour l'image colorée, les mêmes perturbations que pour l'image grise. En effet prendre une petite valeur pour k n'est pas suffisant. De même prendre $\frac{1}{5}$ des données de l'image originale est largement suffisant pour permettre d'approcher au mieux celle-ci.

Pour $k=500$ voici ce que l'on obtient :

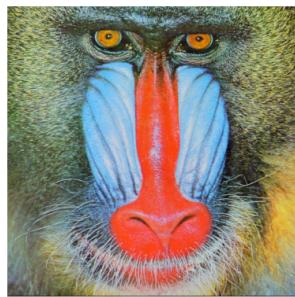


FIGURE 9 – $k = 500$

On remarque que contrairement à l'image en niveau de gris, l'image colorée est nette même pour un k supérieur à la condition. Cela peut s'expliquer par le fait que les images en couleur contiennent généralement plus d'informations que les images en niveaux de gris, car chaque pixel doit contenir des valeurs pour les canaux de couleur (rouge, vert, bleu), tandis que les images en niveaux de gris n'ont qu'un seul canal de luminance. Ainsi les images ayant qu'un canal ont plus de chances d'être soumis aux erreurs d'arrondis due aux calculs de recherche de valeur propres.

3.4.3 Image 3 - Tigre

Comme nous avons vu que l'algorithme de SVD fonctionne aussi pour les matrices rectangulaires, nous avons voulu le tester en prenant une image qui en plus d'être colorée, est rectangulaire.

Pour appliquer notre algorithme, nous avons procéder de la même façon que pour l'image 2 (mandrill) à savoir appliquer la SVD pour chaque canaux Rouge, Vert, Bleu.

L'image est de taille (441,660,3) donc la condition pour le k est : $k < \frac{441 \times 660}{441 + 3 \times 660 + 1} = 360,52$



(a) $k = 10$

(b) $k = 100$

(c) $k = 360$

De la même manière que les exemples précédents, cette image présente des perturbations pour un petit k et est nette pour un k satisfaisant la condition.

Pour $k=600$ nous obtenons ceci :



FIGURE 11 – k = 600

On remarque ici que, contrairement à l'image du Mandrill, cette image est très perturbée par le fait que l'on ait augmenté la valeur du k.

En effet, il faut souligner que cette images possède énormément de teintes de bleues et elle joue sur les ombres et les lumières. Il faut aussi savoir que l'oeil humain perçoit plus les perturbation dues à la lumière que ceux dues aux couleurs.

Si on compare cette image avec l'image 2, on remarquera que celle-ci possède plus de détail lié à la luminance (la luminance fait référence à la composante de luminosité) contrairement à l'image du mandrill qui, elle, possède plus de détail lié à la chrominance (la chrominance fait référence aux composantes de couleur).

Nous observons une teinte de bleue prépondérante, comme l'image de lena , nous pouvons imaginer que des erreurs d'arrondis ont perturbé la décomposition SVD de la matrice gérant le canal bleu. Ceci est confirmé dans l'analyse de performances de l'algorithme de compression.³

Revenons à notre image obtenue pour un k=600. On observe que malgré la différence de ton de bleu, l'image est nette. C'est donc, en effet, la luminance qui est dégradée ici.

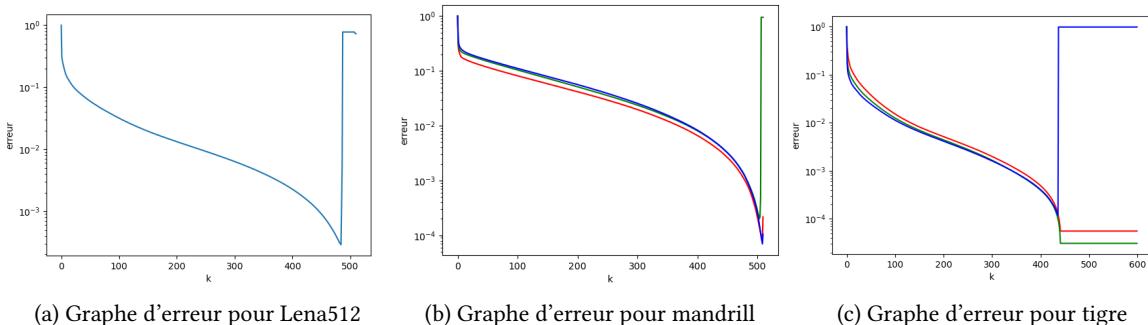
3.5 Performances de l'algorithme de compression

Pour étudier la performance de l'algorithme de compression, nous avons tracer le graphe d'erreur de reconstruction pour chacune de ces images. Nous choisissons de tracer nos graphes en lin-log (échelle linéaire pour k et logarithmique pour l'erreur). En effet si les valeurs de l'erreur diminuent rapidement avec k, une échelle logarithmique pour l'erreur permet de mieux visualiser les petites valeurs et de distinguer les variations. Pour cela, nous allons utiliser la norme de Frobenius.

L'erreur de reconstruction après une décomposition en valeurs singulières (SVD) est mesurée en comparant la matrice originale A à sa version reconstruite A_k . Pour ce faire, on calcule la différence entre A et A_k , puis on divise cette différence par la norme de Frobenius de A. Cela normalise l'erreur par rapport à la taille de la matrice originale, ce qui permet une comparaison de l'erreur de reconstruction indépendamment de la taille de la matrice.

Ainsi pour plusieurs k, nous allons calculer l'erreur : $e = \frac{\|A - A_k\|_F}{\|A\|_F}$.

Voici ce que l'on obtient pour les 2 images :



1. Commentaire Graphe 1 :

Pour le 1er graphe associé à l'erreur de reconstruction pour l'image en niveau de gris, on voit que plus le k augmente, plus la matrice compressée se rapproche de l'image originale. Comme remarqué ci-dessus, on observe un "pic" lorsque k s'approche de 500. Cela est due au manque de précision lors des calculs comme expliqué ci-dessus.

2. Commentaire Graphe 2 :

Pour le 2ème graphe, nous avons tracé pour chaque canal sa courbe correspondante, ainsi la courbe bleue correspond au canal bleu, etc. On observe ici la même décroissance que le premier graphe. De plus, lors de notre test de compression avec un $k=500$, nous avions remarqué que l'image n'avait pas de perturbation malgré le fait que l'on ait dépassé le seuil de notre condition. Avec le graphe, nous le voyons clairement car pour la courbe bleu et rouge, nous n'avons pas ce pic comme observé sur le 1er graphe. Cependant, pour le graphe vert il est présent. Comme vu plus haut ces perturbations sont dues à des erreurs d'arrondis.

3. Commentaire Graphe 3 :

Idem que pour le graphe 2, nous avons choisi de tracer, pour chaque canal, sa courbe associée à son erreur. En effet, cela nous permet de mieux comprendre les différentes perturbations liées aux différents matrices de couleurs que l'on pourrait rencontrer lors de la compression de notre image. Pour ce 3ème graphe, on remarque aussi ce fameux pic présent sur tout les différentes courbes de couleurs. Cependant, ici, il ne concerne que la courbe bleue. Ainsi la matrice correspondant au canal de la couleur bleue est plus propice aux erreurs d'arrondis et de calculs.

4 Conclusion

Pour conclure, ce projet nous a permis de revoir des notions d'algèbre linéaires telles que le théorème spectrale et certaines propriétés sur les matrices. Nous avons pu également voir comment la notion de valeurs propres et valeurs singulières sont interprétées de manière concrète selon les contextes (imagerie, finance..). Grâce à toutes ces notions, nous avons pu comprendre en profondeur la décomposition en valeurs singulières ainsi que son application dans le domaine de la compression d'images.

Puis nous avons implémenté numériquement la méthode. Pour cela, nous nous sommes aidé de plusieurs algorithmes (que nous avons étudié lors du premier semestre) tels que l'algorithme de puissance itérée ou encore celui de déflation. L'algorithme de déflation, utilise la méthode de la puissance itérée, pour calculer successivement les valeurs propres et les vecteurs propres . L'algorithme de puissance itérée est reconnue pour son efficacité sur de grandes matrices. Effectivement, celui-ci recherche de manière itérative la plus grande valeur propre et son vecteur propre associé, . Néanmoins, comme toutes méthodes itératives, la méthode de puissance itérée est soumise à quelques hypothèses de convergence, elle demande que le premier vecteur de notre itération ne soit pas orthogonale à notre matrice ce qui est totalement aléatoire. Ainsi, la méthode de déflation ne fonctionne que lorsque les valeurs propres sont distinctes. De plus, celle-ci est d'autant plus efficace que les valeurs propres que les valeurs propres sont éloignées les unes des autres.

Ensuite, nous avons constaté que la SVD peut être soumis à divers problèmes. En effet, la compression d'image reposant sur la SVD utilise comme principe qu'il existe sur la matrice à étudier des valeurs singulières négligeables c'est à dire que nous pouvons supprimer. En pratique cela se vérifie pour la plupart des matrices. Cependant, il existe des cas où des matrices n'ont pas de valeurs singulières négligeables. Par exemple la matrice identité qui possède des valeurs singulières toutes égales.

Il est important de souligner que de nos jours, ce n'est pas la méthode de Décomposition en Valeurs Singulières (SVD) qui est la plus utilisée, d'autres méthodes de compression d'images sont utilisées comme le format JPEG.

Pour finir, ce projet nous a beaucoup plus car là où en classe nous étudions des concepts très théoriques, nous avons pu explorer , via ce projet une application concrète des concepts mathématiques étudiés.

5 Sources

On the Early History of the Singular Value Decomposition (Ce document nous a été utile pour la revue historique de la SVD)

Polycopié INSA ROUEN GM3 (Ce document nous a été utile pour implémenter les algorithmes de puissance itérée et déflation)

La décomposition en valeur singulière (Ce document nous a été utile pour la démonstration de la SVD)

SVD appliquée à la compression d'image (Ce document nous a été utile pour comprendre l'intérêt du théorème d'Eckart-Young et son utilité à la compression d'image)

Document sur la SVD (Ce document nous a été utile pour la démonstration du théorème d'Eckart-Young)