Projet 1
Compression d'image à travers la factorisation SVD

Groupe 3 - Equipe 8625

Responsable : DECOU Nathan
Secrétaire : DELPEUCH Sébastien
Codeurs : BAUDE Clément, PIERRE Hugo, PRINGALLE Antoine

Résumé : L'idée de ce projet est de réaliser la compression SVD en mettant en place une transformation QR. La question sous jacente est de nous sensibiliser à la réflexion d'algorithmes permettant une mise en pratique numérique stable.

1 Transformations de Householder

Nous allons commencer par définir une transformation permettant de réaliser plusieurs points cruciaux de la compression SVG : la transformation de HouseHolder.

1.1 Elements théorique sur la matrice de HouseHolder

Une matrice de HouseHolder associée à un vecteur non nul U de taille n peut s'écrire sous la forme suivante où Id est la matrice identité de dimension n :

$$H = Id - 2 \times UU^t \tag{H1}$$

avec $||U|| = 1$. Nous voulons trouver une méthode pour avoir l'expression du vecteur U de taille n pour que la matrice de HouseHolder H envoie le vecteur X (de taille n) sur le vecteur Y de même taille et norme que X. L'expression mathématique du vecteur U en fonction de X et Y est donc

$$U = \frac{X - Y}{||X - Y||} \tag{1}$$

L'enjeu de cette partie est de calculer le vecteur U pour déterminer H, dans un premier temps sur un vecteur puis sur une matrice.

Pour obtenir un produit de matrice optimisé, il est préférable de distribuer le produit de matrice directement à l'intérieur du calcul de la matrice de HouseHolder, cela permet de diminuer les sources d'erreurs. Ainsi le produit de la matrice de HouseHolder par un vecteur V se calculera de la manière suivante :

$$H \times V = V - 2 \times UU^tV \tag{H2}$$

Ensuite, pour transformer cela en produit de matrice-matrice, nous réutilisons le produit matrice-vecteur en considérant qu'une matrice est équivalente à n vecteurs.

Niveau complexité, le produit de matrice-matrice se décomposait en deux étapes, tout d'abord le calcul de H puis la multiplication entre la matrice H et l'autre matrice.

Soit le produit de matrice $U \times V$, on note n le nombre de lignes de U, m le nombre de colonnes de U et p le nombre de colonnes de V.

Le calcul de H se fait en $O(n \times m \times p)$ et la multiplication de matrice $U \times V$ se fait en $O(n \times m \times p)$. Nous avons un algorithme en complexité cubique. Lorsque nous effectuons le produit matriciel avec la méthode présentée à l'équation (H2), nous obtenons comme complexité pour le produit matrice - vecteur une complexité linéaire. Lorsque nous généralisons aux produits matrice-matrice nous avons donc une complexité quadratique. Ainsi, la deuxième méthode est moins energivore que la première.

1.2 Mise en place de tests

Maintenant que nous avons la méthode, nous allons tester si nos algorithmes nous fournissent des résultats cohérents, nous allons regarder au fur et à mesure les différentes potentielles sources d'erreurs. Nous pouvons obtenir des erreurs à deux endroits : tout d'abord dans le calcul de U puis dans le calcul de H. Nous estimons que la simplicité des calculs nécessaires pour construire U et H font que nous n'allons pas tester l'erreur relative pour ces cas-là. Cependant il peut être intéressant de confirmer ce que nous avons déduit théoriquement c'est à dire la différence de complexité entre la méthode "directe" et la méthode que nous avons présentée basée sur l'équation (H2). Pour cela, nous réalisons une batterie de tests pour mettre en évidence la différence de temps de calcul, l'idée est de faire la réalisation des deux méthodes sur 500 matrices de taille 1 à 500.

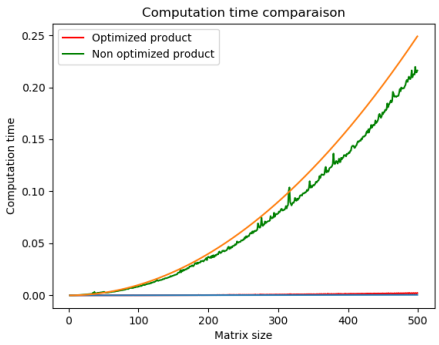

FIGURE 1 – Mise en évidence de la différence de temps de calcul entre les deux méthodes pour le produit matrice-vecteur

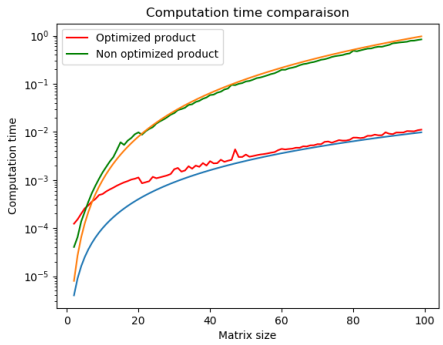

FIGURE 2 – Mise en évidence de la différence de temps de calcul entre les deux méthodes pour le produit matrice-matrice

La figure 1 confirme la théorie, le calcul "optimisé" (courbe rouge) prend moins de temps que le calcul classique (courbe verte), la théorie est d'autant plus confirmée que lorsque nous traçons la courbe quadratique (courbe orange) elle suit parfaitement la courbe verte. Il est de même avec la courbe linéaire (courbe bleue) qui suit parfaitement la courbe du calcul optimisé. Cela nous prouve donc empiriquement que la complexité de calcul de la méthode optimisée est en $O(n)$ et la complexité de la méthode classique est en $O(n^2)$.

La figure 2 possède les mêmes caractéristiques. Regardons tout d'abord la courbe verte, cette dernière représente le temps de calcul pour la méthode non optimisée. Nous pouvons remarquer qu'elle suit la courbe orange qui représente la courbe $f : x \mapsto x^3$, cela confirme notre étude théorique la méthode non optimisée est en $O(n^3)$. A contrario, si nous observons la courbe rouge, cette dernière utilise moins de temps, elle est donc bien optimisée, de plus elle suit la courbe bleue ($f : x \mapsto x^2$), la méthode optimisée est donc de complexité quadratique.

En somme, la transformation de Householder est implémentée de manière optimisée (H2), c'est à dire de complexité quadratique, cela est crucial pour le reste du travail puisque nous allons constamment réutiliser cette transformation.

2 Mise sous forme bidiagonale

Nous allons maintenant nous intéresser à la seconde partie nécessaire pour la compression d'image à travers la factorisation SVD : la transformation d'une matrice A en une matrice bidiagonale BD par une méthode directe en utilisant la méthode de HouseHolder définie par H2.

2.1 Principe de la méthode

Pour réaliser cette méthode, l'idée est de décomposer la matrice en vecteur ligne et en vecteur colonne (pour pouvoir utiliser les algorithmes de la première partie en toutes circonstances, nous transformons les vecteurs lignes en vecteurs colones). Une fois cela effectué, des vecteurs sont générés de même norme que les vecteurs extraits de la matrice mais possédant une unique composante non nulle. A partir de là, l'algorithme peut être implémenté, l'idée est qu'à chaque itération, l'extraction d'un vecteur, la génération d'un vecteur de même norme que ce dernier, le calcul du Householder et l'actualisation des bases et de la matrice BD sont effectués.

Parlons succintement de la complexité de cette méthode, à chaque itération nous devons réaliser l'extraction d'un vecteur (opération supposée constante), la génération d'un vecteur de même norme que ce dernier (opération linéaire), le calcul du Householder (opération quadratique avec H2) et l'actualisation des bases et de la matrice BD (trois opérations cubiques). Cela nous donne donc un algorithme de complexité polynomiale de degré quatre.

2.2 Mise en place de test

Nous devons contrôler la résolution du problème pour s'assurer de deux choses. Tout d'abord, il est évident que le résultat doit être une matrice bidiagonale. De plus à tout momdent de la résolution nous devons avoir

$$Q_{\text{left}} \times BD \times Q_{\text{right}} = A \tag{2}$$

Nous n'allons pas uniquement tester l'égalité, pour s'assurer d'un fonctionnement optimal, nous allons regarder au fur et à mesure la distance entre $Q_{\text{left}} \times BD \times Q_{\text{right}}$ et A pour cela nous définissons le calcul de l'erreur relative suivant :

$$\delta = \frac{||Q_{\text{left}} \times BD \times Q_{\text{right}} - A||_2}{||A||_2} \tag{3}$$

Nous réalisons alors la bidiagonalisation sur une matrice de taille 100×100 et nous calculons δ à chaque itération. En addition nous réalisons une bidiagonalisation de cette matrice et nous observons la densité de termes non nuls en dehors de la bidiagonale. Nous obtenons les graphes suivants :

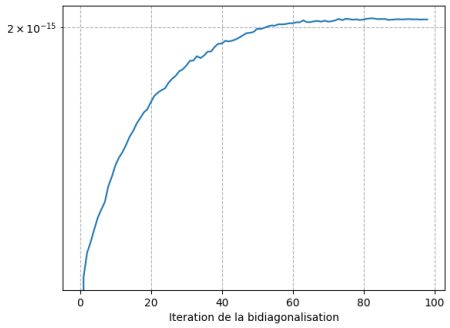

FIGURE 3 – Valeur de δ en fonction du nombre d'itérations

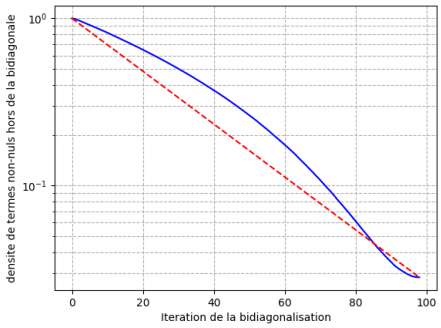

FIGURE 4 – Densité de termes non nuls en dehors de la bidiagonale

Premièrement regardons la figure 3. Cette dernière représente la distance entre $Q_{\text{left}} \times BD \times Q_{\text{right}}$ et A pour une matrice aléatoire 100×100. Au début elle est nulle (puisque les deux matrices sont identiques) et au fur et à mesure des itérations, la distance se rapproche de 2×10^{-15}, nous nous rapprochons donc de $\epsilon_{\text{machine}} = 2.2 \times 10^{-16}$ pour numpy. En somme, notre erreur relative est relativement faible : l'assertion est donc largement vérifiée.

Sachant que l'assertion est valide, nous devons maintenant vérifier que l'algorithme fait réellement la bidiagonalisation, nous avons donc tracé la figure 4 pour vérifier la densité de termes non bidiagonaux non nuls. Une bidiagonalisation a été réalisée sur une matrice 100×100. Observons la courbe bleue, au début la densité vaut 1 ce qui est normal puisque nous n'avons pas commencé la bidiagonalisation. Lorsque nous multiplions les itérations de bidiagonalisation, nous obtenons à la fin une densité de termes non bidiagonaux non nuls très proche de 0. Pour qualifier la quantité de termes que la bidiagonalisation enlève à chaque itération, nous avons tracé la courbe rouge, ainsi nous pouvons quantifier la décroissance comme une décroissance exponentielle.

Nous avons donc désormais un algorithme permettant de mettre sous forme bidiagonale une matrice, nous pouvons maintenant passer à la prochaine étape pour réaliser notre compression SVD.

3 Transformations QR

La deuxième étape pour réaliser une mise sous forme SVD consiste à appliquer un certain nombre de fois la transformation QR sur une matrice bidiagonale. L'idée est de récupérer la matrice bidiagonale BD et la transformer en une matrice S par une méthode itérative.

3.1 Principe de la méthode

L'idée générale de cette méthode est de récupérer une matrice bidiagonale BD et un nombre de tours maximal. Le but est de modifier U, S et V tels que deux assertions soient vérifiées. Tout d'abord la solution S doit converger vers une matrice diagonale, pour vérifier cette assertion nous définissons $\sigma = \sum_{i=1}^{n} \sum_{j=1}^{m} x_{i,j} \; \forall i, j, i \neq j$. De plus à tout moment nous devons avoir :

$$U \times S \times V = BD \tag{4}$$

Si nous regardons au niveau des coûts, nous pouvons voir que pour une matrice $n \times n$ le nombre d'opérations sera de $\frac{4}{3}n^3$. Ce coût est relativement élévé si nous comparons avec des algorithmes que nous avons étudiés précédemment (comme celui de l'algorithme LR qui comportait un nombre d'opérations en $\frac{1}{3}n^3$). Cependant le principal avantage théorique de cette méthode est sa stabilité numérique car elle limite la division par des nombres petits.

3.2 Mise en place de test

Pour contrôler la résolution correcte de la transformation QR, nous calculons tout d'abord la distance δ entre $U \times S \times V$ et BD. Ensuite nous vérifions que la matrice résultat S tend bien vers une matrice diagonale.

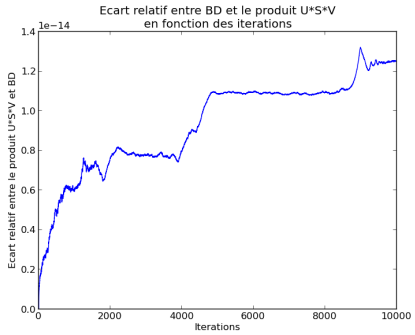

FIGURE 5 – Distance δ entre $U \times S \times V$ et BD pour 10 000 itérations

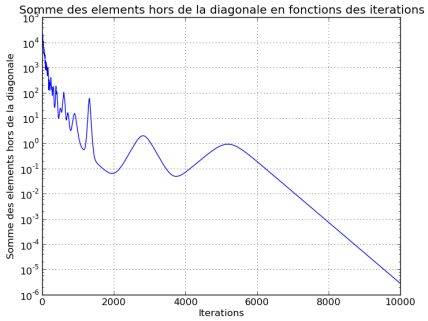

FIGURE 6 – Evaluation de σ pour 10 00 itérations de l'algorithme

Tout d'abord, nous pouvons remarquer sur la figure 5 que au fur et à mesure des itérations, la distance δ augmente, ce qui est normal puisque nous modifions de plus en plus les matrices. Cependant, le pic d'erreur relative δ est atteint en 1.2×10^{-14}. Cette valeur est cohérente puisque inférieur à $\epsilon_{\text{machine}}$ et nous indique que l'algorithme vérifie l'invariant de l'équation 4. Il faut cependant vérifier que l'algorithme réalise une décomposition correcte c'est à dire que la matrice S résultante est une matrice diagonale. Ainsi la figure 6 représente la somme σ des éléments hors diagonaux, cette dernière met en lumière la transformation de S en une matrice diagonale, au bout de 10 000 itérations, σ vaut 10^{-7} ce qui est un terme tout à fait négligeable. Ainsi nous pouvons être sur que notre algorithme respecte bien l'assertion.

3.3 Optimisation de la décomposition QR

L'algorithme que nous avons présenté nous fournit une transformation QR pour n'importe quelle matrice, cependant dans le cadre de la factorisation SVD nous ne traitons que des matrices bidiagonales. Considérer cette propriété pour réaliser la transformation QR nous permet de ne réaliser Householder que sur la bidiagonale et donc de faire une succession d'opérations constantes. Cette méthode n'a pas été implémentée et nous n'avons qu'un algorithme permettant de réaliser une transformation sur une matrice $n \times m$ en temps cubique.

4 Application à la compression d'image

Nous allons maintenant réutiliser tous les algorithmes implémentés pour réaliser une compression SVD c'est à dire mettre une matrice A (c'est à dire une image) sous la forme $U \times S \times V$. Pour ce faire, nous réalisons tout d'abord une bidiagonalisation présentée en section 2 utilisant l'implémentation H2 de HouseHolder. Ensuite, nous réalisons la transformation QR présentée en section 3. Cependant étant donné que notre implémentation de la transformation QR n'exploite pas le fait que la matrice est bidiagonale, l'étape de bidiagonalisation perd de son intérêt dans notre cas. Une fois cela effectué, nous obtenons $U \times S \times V$. Finalement, pour obtenir une factorisation au rang k il faut supprimer toutes les termes diagonaux de S qui sont supérieurs à k.

4.1 Mise en place de tests visuels

Nous pouvons rapidement effectuer quelques tests succins pour contrôler la compression SVD, c'est à dire vérifier que tous les éléments diagonaux de S sont inférieurs à k.

Ensuite nous tentons de refaire les mêmes images que dans le sujet et nous obtenons les images suivantes :

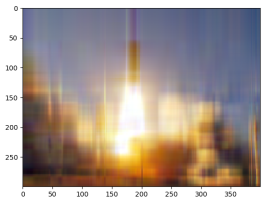

FIGURE 7 – Rang 5

FIGURE 8 – Rang 50

FIGURE 9 – rang 130

Nous pouvons effectivement voir à l'oeil nu que les images obtenues avec nos algorithmes se confondent de celles du sujet. Cependant, nous estimons que cela n'est pas suffisant pour s'assurer de la fiabilité de notre algorithme, nous allons donc mettre en place des test algébriques sur cet algorithme.

4.2 Mise en place de tests algébriques

Nous allons maintenant rajouter une caractérisation algébrique du résultat de la compression, pour cela, nous allons utiliser deux méthodes. Tout d'abord une carte colorée de l'erreur relative de la distance pour chaque point entre l'image du sujet et l'image obtenue par nos algorithmes de compression.

$$\delta_{\text{composantes}} = \frac{||x_{i,j}^{\text{original}} - x_{i,j}^{\text{rangk}}||_2}{||x_{i,j}^{\text{original}}||_2} \tag{5}$$

Nous pouvons alors visualiser l'erreur relative pour chaque point de l'image via un mesh (plus le point est coloré plus l'erreur relative est grande), cela nous permet de visualiser les zones de compression dans l'image et de voir l'information « perdue ». Nous traçons alors la carte pour l'image de la fusée au rang de compression 5. Nous obtenons la figure 10

Nous remarquons alors que l'information perdue réside dans les changements brusques de couleurs. Lorsque nous avons des zones « unies » aucune compression n'est effectuée. A contrario lorsque nous avons des zones présentant un fort changement de couleur nous pouvons remarquer que cela implique une grande erreur relative. Cela est normal puisque lorsque nous compressons une image fortement (au rang 5 par exemple) nous ne pouvons plus dicerner avec exactitude les bordures entre les couleurs.

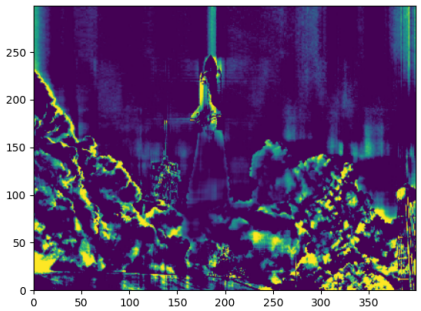

FIGURE 10 – Carte de l'erreur relative

De plus, nous allons utiliser un autre indicateur algébrique, le coefficient de compression

$$\mu = \frac{n \times k + k + m \times k}{m \times n} \tag{6}$$

Nous traçons alors la figure représentant le coefficient de compression en fonction du rang k. La figure 11 nous montre que le taux de compression est linéaire, ainsi, lors d'une compression SVD la quantité d'information retenue augmente linéairement avec le rang.

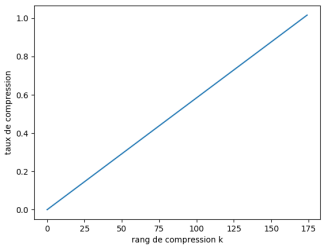

FIGURE 11 – Coefficient de compression μ en fonction du rang k