Projet 3
Compression d'image à travers la décomposition SVD

Groupe 3 - Equipe 2

Responsable : Arthur Fourcadet
Secrétaire : Imad Boudroua
Codeurs : Mohamed Faycal Boullit, Salim Bekkari , Mohammed Boudali

Résumé : Le but de ce projet consiste à programmer un algorithme permettant de faire de la compression d'images en utilisant des techniques matricielles basées sur la factorisation SVD. Ce type d'algorithme est à relier aux algorithmes de compression avec pertes, dont le plus connu est certainement l'algorithme de compression JPEG, lui-même basé usuellement sur la Discrete Cosine Transform (DCT), une transformation voisine de la transformée de Fourier discrète.

1 Transformations de Householder

1.1 Matrice de Householder

La matrice de householder présente une symétrie hyperplan donc il existe U un vecteur colonne de taille n et de norme euclidienne 1, tel que $H = Id - 2 \times U^t U$
Soient X et Y deux vecteurs colonnes de taille n et de même norme pour obtenir la matrice de householder qui envoie X sur Y (ie $HX = Y$) On cherche d'abord le vecteur U
Nous allons montrer que $U = \frac{X-Y}{\|X-Y\|}$ est une condition suffisante pour réaliser ce qui précède.

En effet, soit U défini ainsi. On a alors $HX = (Id - 2U^tU)X = X - 2U^tUX = X - 2U \underbrace{({}^tUX)}_{\in \mathbb{R}}$

Donc, $HX = X - 2({}^tUX)U = X - 2\frac{{}^t(X-Y)X(X-Y)}{\|X-Y\|^2} = X - \frac{2\langle X-Y,X\rangle}{\|X-Y\|^2}(X-Y) = X - \frac{1}{\|X-Y\|^2}\langle X - Y, 2X\rangle(X-Y)$ Or, $\langle X-Y, 2X\rangle = \langle X-Y, X+Y\rangle + \langle X-Y, X-Y\rangle = \|X\|^2 - \|Y\|^2 + \|X-Y\|^2$
$= \|X-Y\|^2$
car X et Y ont même norme.
Donc, $HX = X - \frac{1}{\|X-Y\|^2}\|X-Y\|^2 (X-Y) = X - (X-Y) = Y$
Finalement, H envoie bien X sur Y et U est bien de norme 1, donc U convient.

On écrit donc un algorithme qui détermine U puis H en fonction de deux vecteurs X et Y passés en arguments.
On peut également appliquer householder sur de des vecteur de norme différent de 1 pour faire cela il suffit de passer $\frac{X}{\|X\|}$ et $\frac{Y}{\|Y\|}$ en paramètre a la fonction householder et après pour vérifier l'équation de la symétrie on peut multiplier la matrice par $\frac{\|Y\|}{\|X\|}$
En effet, la complexité temporelle de la multiplication matricielle classique (matrice-matrice) est en $\Theta(n^3)$ où n est le nombre de lignes de la matrice.
En procédant comme décrit précédemment, on obtient une complexité en $\Theta(n^2)$, ce qui accélère considérablement les calculs.
Succinctement, on ramène le produit matrice-matrice à un produit matrice-vecteur, que l'on calcule grâce à l'expression $Hx = x - 2 \times (U^tU)x$.

2 Mise sous forme bidiagonale

La mise sous forme bidiagonale selon cet algorithme consiste à projeter les zéros nécessaires sur chaque ligne et chaque colonne une par une. L'algorithme effectue alors un produit matriciel pour chaque ligne et colonne, lui attribuant une complexité de $n*(n*m) + m*(n*m) = (n*m)*(n+m)$ pour une image de résolution n*m. L'algorithme présente donc une complexité de l'ordre de $O(n^2)$.
L'algorithme tel qu'implémenté dans le code est plutôt lent mais produit des résultats satisfaisants.
Les termes censés être nuls sont de l'ordre de 10^{-15}.

Nous devons construire Q_{left}, Q_{right} et BD tels que $Q_{left}.BD.Q_{right} = A$ où A est la matrice de départ. Q_{left} et Q_{right} sont deux matrices orthogonales et BD est une matrice bidiagonale. Nous initialisons Q_{left} et Q_{right} en tant que matrices identités (qui sont bien des matrices orthogonales) et

$BD = A$. A chaque itération, nous vérifions que Q_{left} et Q_{right} restent bien des matrices orthogonales, que $Q_{left}.BD.Q_{right} = A$ et que les zéros ont bien été placés sur les i premières lignes et i premières colonnes de BD.

Les changements effectués à chaque itération sont :

— $Q_{left} \leftarrow Q_{left}.H_1^t$

— $BD \leftarrow H_1.BD.H_2$

— $Q_{right} \leftarrow H_2^t.Q_{right}$

Les matrices H_1 et H_2 sont les matrices de householder qui placent correctement les 0 sur la ième ligne et la ième colonne de BD. Ces matrices sont orthogonales car si $H = Id - 2.U^tU$, on a :

$H^tH = H^2 = (Id - 2UtU)^2 = Id - 4U^tU + 4U^tU = Id$

L'ensemble des matrices orthogonales est un groupe, donc Q_{left} et Q_{right} restent des matrices orthogonales lorsqu'on les multiplie par des matrices orthogonales. L'égalité $Q_{left}.BD.Q_{right} = A$ est conservée car $H_1^tH_1 = Id$ et $H_2H_2^t = Id$.

3 Transformations QR

3.0.1 Description de l'algorithme :

Une fois la matrice A factorisée sous la forme $A = Q_{left}.BD.Q_{right}$, il convient désormais de factoriser la matrice BD ssous la forme $BD = U.S.V$, où S est une matrice diagonale dont les éléments sont positifs et ordonnés de manière décroissante. Pour cela, nous devons appliquer un certain nombre de fois l'algorithme de la transformation QR. Celui-ci consiste à chaque itération à déterminer la décomposition QR de la transposée de S (initialisée à BD) et de R_1 qui est donné par la première décomposition. Dans un premier temps, cette décomposition est réalisée à l'aide de la fonction **numpy.linalg.qr**.

La traduction de cet algorithme sous python permet de tracer la courbe ci dessous (figure 1) La figure

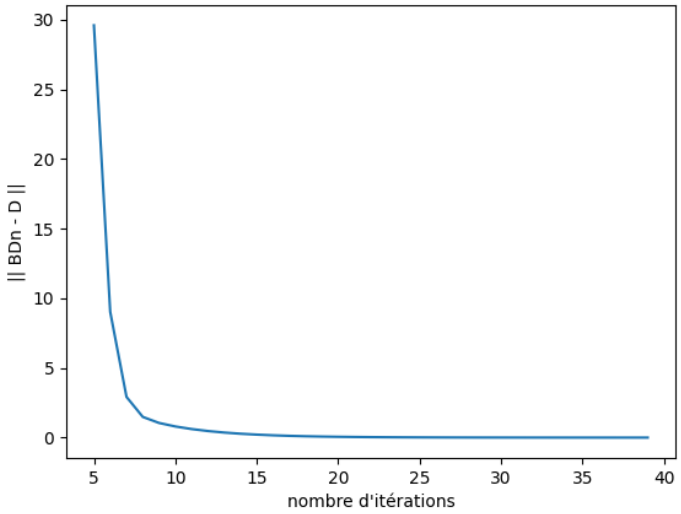

FIGURE 1 – convergence de la matrice BD vers D

1 montre la convergence de BD qu'on peut noter BD_n avec n le nombre d'itération qu'on souhaite itérer l'algorithme pour fournir les 3 matrices U,BD_n,V , on remarque bien que lorsque n tend vers $+\infty$ la matrice BD_n tend vers une matrice diagonale D en effet, cela est juste et on remarque que la convergence a lieu dès $n > 7$ et donc on a $\lim_{n \to +\infty} ||BD_n - D|| = 0$ et alors on obtient bien une matrice diagonale .L'invariant que les 3 matrices vérifie reste vrai après chaque itération et on peut le vérifier simplement en multipliant les 3 matrices de façon à avoir $U \times BD_n \times V = BD$

3.1 Invariant

Il est demandé de fournir une preuve de l'invariant suivant : "S, R_1 et R_2 sont des matrices bidiagonales". Pour démontrer cela, les données initiales montrent que S est dès le départ une matrice bidiagonale. Or, d'après le cours, la factorisation QR conserve la forme d'une matrice tridiagonale, donc S est une matrice bidiagonale supérieure.
Donc, ${}^tS$ est bidiagonale inférieure. En particulier, ${}^tS$ est une matrice tridiagonale. R_1 est donc une matrice tridiagonale (car la factorisation QR conserve la forme tridiagonale), et, puisqu'il s'agit d'une matrice triangulaire supérieure, R_1 est donc une matrice bidiagonale supérieure.
Il en est de même pour la matrice R_2.

3.2 Optimisation de la transformation QR

Pour ce qui est de l'optimisation de la transformation QR, nous avons implémenter une version de l'algorithme qui fournit des résultats correctes mais qui n'est pas optimisée par rapport à celle de la bibliothèque , même en remarquant que nos matrices sont bidiagonales et que le calcul de householder est simple dans ce cas (maximum deux itérations pour trouver la marice de householder) mais cela n'optimise non plus la décomposition. En effet, en appliquant la décomposition de QR sur une matrice bidiagonale de taille 9×8, notre algorithme passe 0.000615 seconde à calculer alors que la fonction prédéfinie dans la bibliothèque **numpy** ne passe que 0.000298 dans le calcul, cela est bien dû de la non utilisation du fait que la matrice est bidiagonale pour que le calcul soit restreint sur quelques valeurs.

3.3 Conservation des propriétées de la décomposition SVD

Dans la décpmposition SVD, les éléments de la matrice S sont positifs et ordonnés de manière décroissante. Pour assurer cette propriétée, nous allons modifier les matrices U et S par le billet de la fonction **svd_rectifie** qui prend les deux premières matrices U et S générées par la fonction qui réalise la décomposition svd, et transforme la matrices diagonale S en une matrice SP diagonale d'élements positifs ordonnés de manière décroissante en prenant la valeur absolue de toutes les valeurs puis les ordonnées, pour U on prend chaque vecteur colonne d'indice i et on multiplie ses valeurs par $\frac{SP_{i,i}}{S_{i,i}}$ pour qu'il correspond bien à la multiplication définie par l'invariant qui reste valide avec les matrices U et S rectifiées.

4 Application à la compression d'image

4.1 Algorithme

Dans cette partie nous allons appliquer la transformation SVD à la compression d'images. En effet, les différentes étapes décrites précédemment permettent, à partir d'une image représentée par une matrice de triplets RGB, de compresser cette image selon un rang de compression k : plus k est grand et plus l'image compressée se rapproche de l'image originale, en qualité et en poids. Au contraire, plus k est petit et plus l'image compressée devient de piètre qualité et dispose d'une taille légère.

En pratique, considérons la matrice A représentant une image. Nous allons d'abord décomposer cette matrice en trois matrices correspondant à chacune des coordonnées RGB. Nous appliquons ensuite la décomposition `SVD` (c'est à dire la mise sous forme bidiagonale puis les transformations QR sur la matrice BD obtenue) sur chacune des sous matrices. On obtient donc une décomposition de la forme $A = Q_{left} \times U \times S \times V \times Q_{right}$, que l'on peut réécrire sous la forme $A = U' \times S \times V'$ où S est une matrice diagonale.

Compresser une image au rang k consiste à annuler, pour chaque sous matrice, les termes diagonaux de S dont l'indice est strictement plus grand que k. On effectue ensuite le produit avec les matrices U' et V' pour obtenir la sous matrice "compressée". On rassemble ensuite les trois sous-matrices en une matrice de triplets RGB.

On effectue un post-traitement, qui remplace les valeurs incorrectes par des valeurs limites : les éléments supérieurs strictement à 1 sont remplacés par 1 et les éléments inférieurs strictement à 0 sont remplacés par 0.

Notons que les transformations de Householder, la mise sous forme bidiagonale, les transformations `SVD` et le post-traitement cité plus haut ont été réalisés par nos soins. Seule la transformation `QR` est obtenue grâce à `Numpy`.

On affiche ensuite le résultat obtenu pour des valeurs différentes de k, avec deux images différentes :

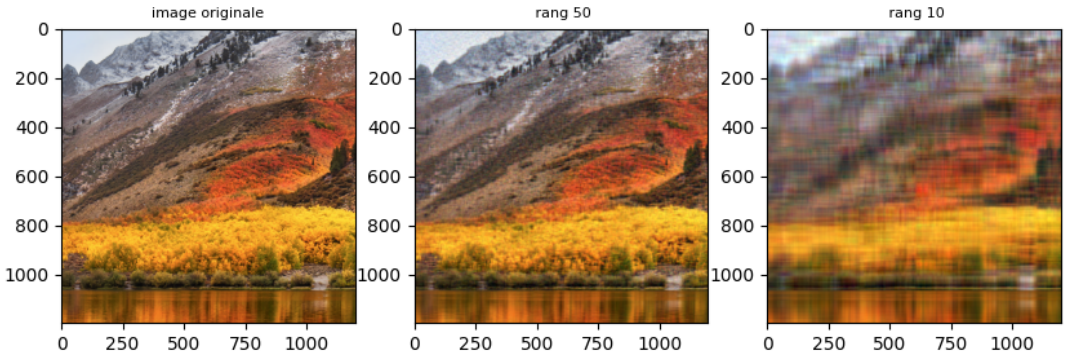

FIGURE 2 – Compression d'une image pour k=50 et k=10

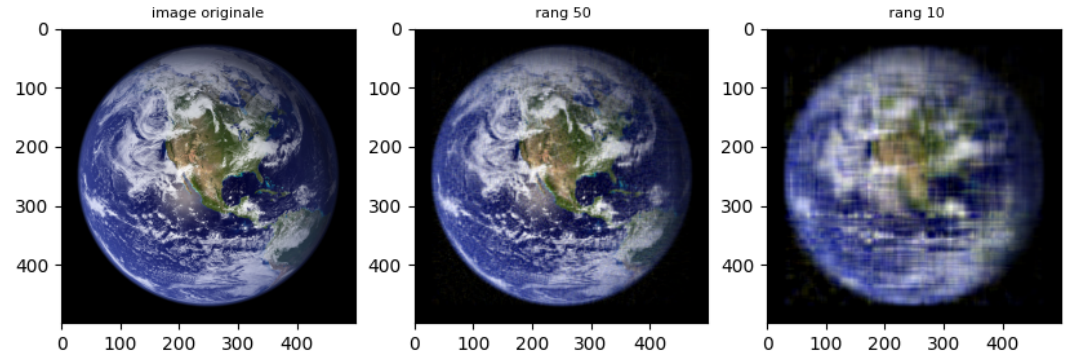

FIGURE 3 – Compression d'une image pour k=50 et k=10

Ainsi avec la méthode SVD on peut compresser des images de type png quel soit de petite taille comme l'image de la figure 500 représentée par une matrice de taille $(500, 500)$ ou bien de grande taille comme la figure 2 représenté par une matrice de taille $(1200, 1200)$ avec différent rang .

4.2 Gain de place pour la compression

Nous remarqons que pour la matrice B résultante de l'image compressée se met sous la forme :

$$B = \begin{pmatrix} U1 & U2 \\ U3 & U4 \end{pmatrix} \begin{pmatrix} S1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} V1 & V2 \\ V3 & V4 \end{pmatrix} = \begin{pmatrix} U1.S1.V1 & U1.S1.V2 \\ U3.S1.V1 & U3.S1.V2 \end{pmatrix}$$

Donc, $U2$, $U4$, $V3$ et $V4$ n'interviennent pas dans le résultat final. Pour stocker la matrice B, il suffit donc de mémoriser les blocs restants soit : $2k.n+k$ valeurs. Le gain de place pour une compression au rang k vaut, alors, $n^2 - 2k.n - k$.

4.3 Mesure de la distance entre l'image réelle et l'image compressée

Le comparatif visuel des images est intéressant, mais il ne permet pas de quantifier les différences réelles entre les deux images. Nous avons donc réalisé un graphique représentant l'erreur relative (telle que définie plus haut) entre l'image compressée et l'image originale, en fonction de la valeur de k. Cette fonction de k représente la qualité de la com- pression. Nous constatons que la décroissance de l'erreur relative s'effectue en $\frac{1}{k}$.

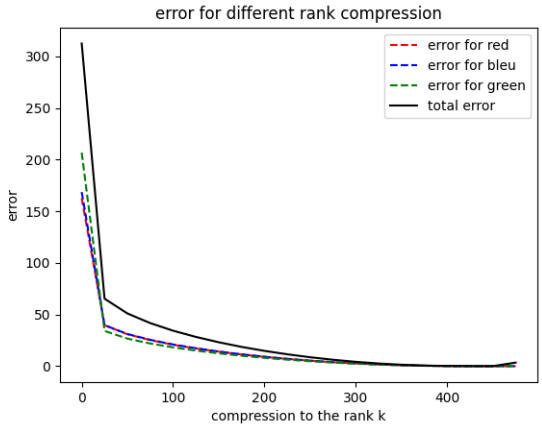

FIGURE 4 – erreur entre l'image réelle et l'image compressée

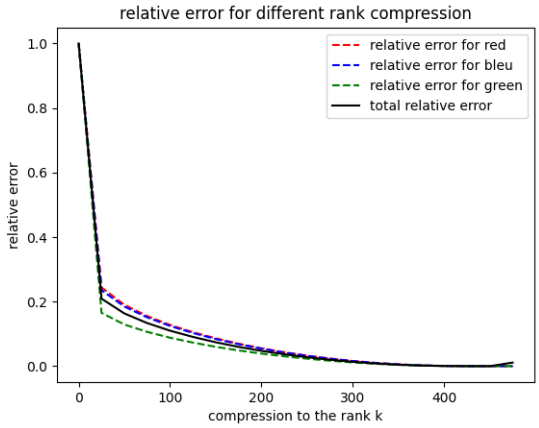

FIGURE 5 – erreur relative entre l'image réelle et l'image compressée

On remarque que les deux figures 4 et 5 illustrent la convergence des erreurs de toutes les couleurs vers 0 lorsque le rang de la compression est assez grand, généralement supérieur à 30, où on remarque une chute remarquable avant que la courbe s'approche de 0 montrant l'effet de la compression qui demeure invisible à partir d'un certain rang, ce qui donne à l'image la clarté et la taille optimale qu'on s'intéresse bien sur à la comparer dans la figure suivante pour différents rangs.6.

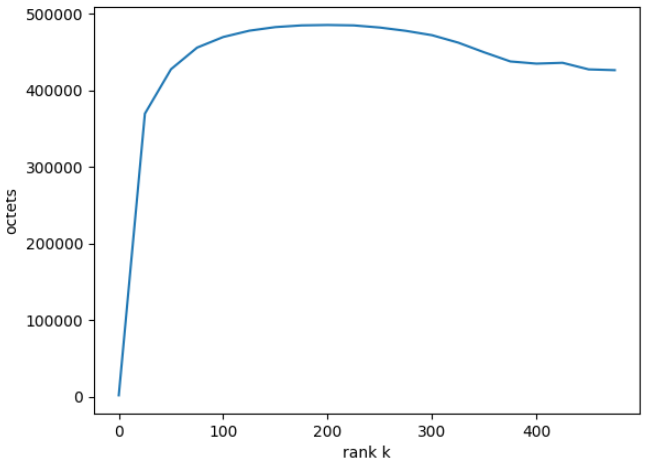

FIGURE 6 – Stockage en fonction du rang k

on remaque dans la figure 6 que plus on diminue le rang de la compression plus le poids de fichier diminue d'où vient l'intérêt de la compression. En revanche, pour que la compression soit efficace on doit avoir un gain positif donc un rang qui ne dépasse pas $\frac{n^2}{2n+1}$ Cette valeur est presque égale a 250 dans le cas de l'image de la terre , ainsi on peur remarquer que le graphe de la figure 6 commence présente des valeur inattendu dés que K dépasse 250.