Retour à IS104 – Algorithmique Numérique (http://mfaverge.vvv.enseirb-matmeca.fr/wordpress/?page_id=159)

Projet 3 : Compression d'image à travers la factorisation SVD

Le but de ce projet consiste à programmer un algorithme permettant de faire de la compression d'images en utilisant des techniques matricielles basée 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.

Tout au long du sujet, nous allons manipuler des images sous forme de matrices. Avec `matplotlib` , il est possible de charger de telles matrices en utilisant le code suivant :

```
img_full = mp.imread("essai.png")
```

Un exemple d'image en couleur est accessible à l'adresse suivante (http://mfaverge.vvv.enseirb-matmeca.fr/wordpress/wp-content/cours/IS104/images/p3_takeoff_base.png) (image du domaine public, dûe à la NASA). Les images sont en couleurs correspondent à des matrices de triplets `RGB` . Il est alors nécessaire d'en extraire les composantes rouge, verte et bleue avant de pouvoir chercher à les compresser.

Tout au long de ce projet, notre but va consister à mettre cette matrice (notée A , de dimensions $n \times m$) sous la forme suivante :

$$A = U \times S \times V$$

… où U et V sont des matrices orthogonales carrées (respectivement de dimensions « m x k » et « k x n » avec « k = min(m, n) », et S est une matrice de dimensions « k x k », diagonale réelle, pour laquelle les éléments diagonaux sont positifs et forment une suite décroissante. Le sujet se décompose en quatre parties :

- la première partie met en place les fonctions utilitaires pour manipuler les matrices de Householder, nécessaires à la réalisation des deux parties suivantes;
- la seconde partie transforme la matrice A en une matrice bidiagonale BD par une méthode directe;
- la troisième partie transformer une matrice bidiagonale BD en une matrice diagonale S par une méthode itérative
- la dernière partie applique ces transformations à l'algorithme de compression d'image par transformation SVD.

Transformations de Householder

Rappelons que toute symétrie hyperplane peut s'écrire sous la forme suivante : $H = Id - 2 \times U^t U$, où Id est la matrice identité en dimension n et U est un vecteur de taille n et de norme euclidienne 1. Une telle matrice est appelée matrice de Householder.

1. Étant donné deux vecteurs X et Y de même norme, trouver comment choisir le vecteur U pour que la matrice de symétrie H alors obtenue envoie X sur Y. Écrire l'algorithme correspondant. L'exemple suivant peut servir de test de fonctionnement :

$$\text{Pour envoyer } \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix} \text{ sur } \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}, \text{ la matrice de Householder est } \begin{bmatrix} 0.64 & -0.48 & 0.6 \\ -0.48 & 0.36 & 0.8 \\ 0.6 & 0.8 & 0 \end{bmatrix}$$

1. Écrire une fonction optimisée qui calcule le produit d'une matrice de Householder par un vecteur. On fera attention d'écrire une fonction en complexité linéaire en exploitant le produit scalaire. Étendre votre fonction pour qu'elle calcule le produit d'une matrice de Householder par un ensemble de vecteurs (ou de manière équivalente une matrice).

Comparer la complexité obtenue avec la complexité d'un produit matrice-matrice usuel.

Remarque : commencer par écrire la fonction non optimisée qui effectue le même calcul, puis écrire la fonction optimisée qui doit avoir une complexité d'ordre strictement inférieur.

Mise sous forme bidiagonale

Supposons maintenant disposer d'une matrice initiale A de dimension $n \times m$ censée représenter une image. Nous allons commencer par transformer cette matrice en la mettant sous forme bidiagonale :

$$\begin{bmatrix} d_1 & e_1 & 0 & & \\ 0 & d_2 & e_2 & \ddots & \\ & \ddots & \ddots & \ddots & 0 \\ & & \ddots & d_{n-1} & e_{n-1} \\ & & & 0 & d_n \end{bmatrix}$$

Rappelons que la factorisation QR est une méthode permettant de mettre une matrice donnée sous la forme d'un produit d'une matrice orthogonale et d'une matrice triangulaire supérieure. Ici, nous allons nous autoriser à multiplier à gauche et à droite par des matrices orthogonales pour obtenir cette forme bidiagonale. Considérons l'algorithme suivant (dans lequel les indices des tableaux commencent à `0`) :

```
Qleft = Id; Qright = Id; BD = A;
For i from 0 to (n-1)
    Let Q1 be a HH matrix mapping BD[i:n,i] on a vector with a single non-zero el
    Qleft = Qleft * Q1;
    BD    = Q1 * BD,
    If (not(i==(m-2)))
        Let Q2 be a HH matrix mapping BD[i,(i+1):m] on a vector with a single nor
        Qright = Q2 * Qright
        BD     = BD * Q2
    End if
End for
Return(Qleft, BD, Qright)
```

Concrètement, on place les zéros nécessaires sur la première colonne, puis sur la première ligne, puis sur la seconde colonne … À chaque tour de boucle de l'algorithme, on peut vérifier que $Q_{left} \times BD \times Q_{right} = A$.

- Mettre en place l'algorithme de mise sous forme bidiagonale, de manière à renvoyer la matrice transformée, ainsi que les changements de base à gauche et à droite.

Remarque : vérifier que l'invariant est conservé à chaque étape de votre algorithme, et que vous faites bien apparaître les zéros dans les bonnes lignes et colonnes.

Transformations QR

La deuxième étape de la mise sous forme SVD consiste à appliquer un certain nombre de fois la transformation QR sur une matrice bidiagonale. Concrètement :

```
U = Id; V = Id; S = BD;
For i from 0 to NMax
    (Q1, R1) = decomp_qr(matrix_transpose(S))
    (Q2, R2) = decomp_qr(matrix_transpose(R1))
    S = R2;
    U = U * Q2;
    V = matrix_transpose(Q1) * V
    End for
Return (U,S,V)
```

Dans cet algorithme, la matrice S converge vers une matrice diagonale, et les matrices (U, S, V) correspondent aux matrices de la décomposition SVD.

1. Écrire une version préliminaire de cet algorithme en utilisant la fonction `numpy.linalg.qr` .
2. Tester et dessiner la convergence de la matrice S vers une matrice diagonale, et s'assurer que l'invariant $U \times S \times V = BD$ est toujours vérifié.

Les appels à la fonction de décomposition QR dans cet algorithme sont en fait disproportionnés : les matrices sur lesquelles on applique cet algorithme sont beaucoup plus simples que des matrices pleines.

3. Montrer l'invariant suivant du calcul : « les matrices S , R_1 et R_2 sont toujours bidiagonales ».
4. Réécrire une version simplifiée de la transformation QR en utilisant l'algorithme vu en cours et l'invariant précédent, puis utiliser cette optimisation pour votre algorithme de décomposition SVD. Quelles sont les complexités des deux algorithmes obtenus ?
5. Usuellement, la décomposition SVD demande à ce que les éléments de la matrice S soient positifs, ordonnés de manière décroissante. Modifier les matrices U et S de manière à assurer cette

propriété.

Application à la compression d'image

Supposons vouloir compresser une image sous la forme d'une matrice carrée A. La première étape consiste à mettre la matrice A sous forme $A = U \times S \times V$. La seconde étape consiste à considérer les valeurs de la diagonale de S :

$$s_{1,1} \geq s_{2,2} \geq \ldots \geq s_{n,n}$$

La compression au rang k consiste à annuler les termes diagonaux de S d'indice strictement plus grand que k.

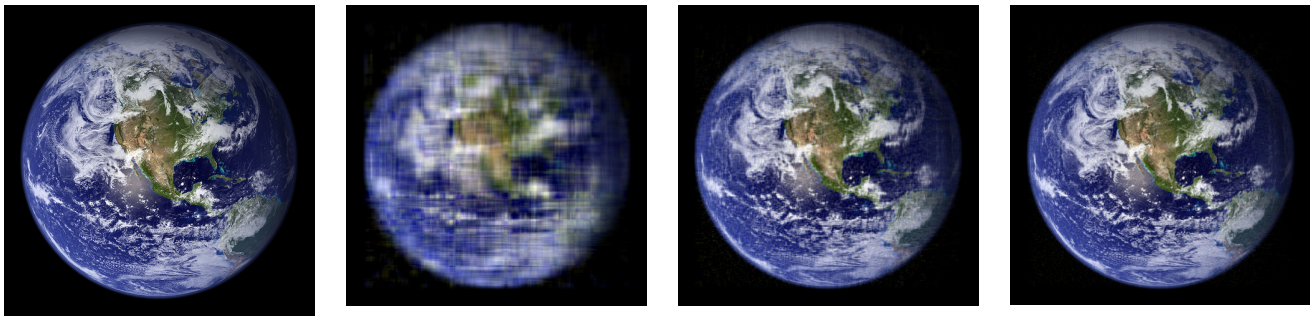

Image originale | Rang 10 | Rang 50 | Rang 100

de Reto Stöckli (NASA)

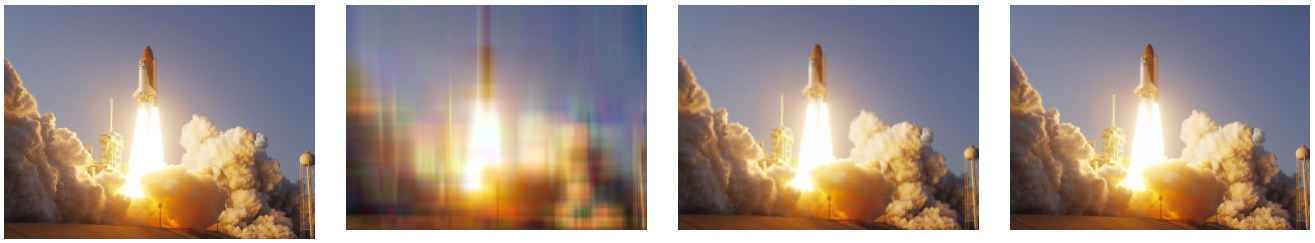

Image originale (NASA) | Rang 5 | Rang 50 | Rang 130

1. Écrire un algorithme permettant d'obtenir la compression au rang k d'une matrice carrée donnée.
2. Quel est le gain de place pour une compression au rang k par rapport à la matrice initiale ? Quelle est le rang maximal au delà duquel la transformation est de taille supérieure à l'image initiale ?
3. Tracer l'efficacité de cette compression en fonction de k, (par exemple en mesurant la distance entre l'image réelle et l'image compressée).

Dans cette section

IS104 – Algorithmique Numérique (http://mfaverge.vvv.enseirb-matmeca.fr/wordpress/?page_id=159)

Construit avec ♥ par Thèmes Graphene (https://www.graphene-theme.com/).