Projet n°3
Compression d'image à travers la factorisation SVD

Groupe n°1 - Equipe n°5

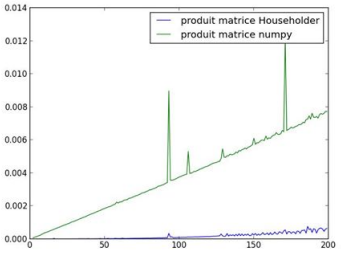
Responsable : aauzemberger
Secrétaire : vjeanjean
Codeurs : jarnault, mbenkirane, apatry

Résumé : Ce projet consiste à compresser des images à partir de la méthode SVD. Afin d'effectuer la compression de manière optimale, il sera nécessaire de passer par des étapes de tridiagonalisation puis bidiagonalisation et enfin, diagonalisation de toute matrice. Pour ce faire, nous utiliserons les matrices de Householder et la décomposition QR. Le but sera de trouver le compromis entre une compression une taille réduite et une image de bonne qualité.

1 Transformations de Householder

Les matrices de Householder correspondent à une représentation des symétries hyperplanes dans l'espace. Par conséquent, l'image d'un vecteur donné par cette fonction ne contiendra plus qu'une coordonnée non nulle. Cette propriété sera par ailleurs utilisée plus tard, lors de la factorisation QR.
De plus, ces matrices possèdent deux propriétés particulières et importantes : ce sont des matrices symétriques orthogonales (et par conséquent, des matrices carrées). Dans un premier temps, il est nécessaire de se demander comment la matrice de Householder sera crée à partir d'un vecteur X donné et de son image Y (qui est également connu), qui plus tard, ne contiendra plus qu'une seule coordonnée non nulle.
Afin de déterminer la matrice H, on utilise le fait que $HX = Y$. Par ailleurs, H étant la représentation matricielle d'une symétrie hyperplane, elle peut se mettre sous la forme $H = Id - 2 \times U.{}^tU$ où Id est la matrice identitée et où U est la représentation matricielle du vecteur selon lequel est faite la symétrie hyperplane (projection le long de la droite vectorielle générée par ce vecteur)
Par conséquent, à partir des deux équations précédentes, on obtient l'expression du vecteur U : $U = \frac{(X-Y)}{\|X-Y\|}$. On en déduit alors la matrice de Householder H associée à U.

Le produit d'une matrice de Householder H par une matrice colonne V (représentant un vecteur) de taille n peut alors s'écrire sous la forme suivante : $HV = V - 2 \times (U.{}^tU).V$. Cependant, on remarque qu'il existe une autre manière possible de raisonner. En effet, on remarque également que le produit peut s'écrire : $HV = V - 2 \times U.({}^tU.V)$. La différence ici se place au niveau des priorités de calculs, qui finit par influencer la complexité en calculs du produit matriciel. En effet, le premier cas la complexité est celle d'un produit d'une matrice carrée $U.{}^tU$ de taille n avec une matrice colonne V de taille n , ce qui correspond à $\Theta(n^2)$. Pour la seconde expression, il s'agit simplement d'un produit de matrice ligne de taille n et d'une matrice colonne de taille n, ce qui permet d'obtenir une complexité linéaire $\Theta(n)$.
Par conséquent, dans le cas d'un produit de matrices carrées de taille n, on obtiendra dans le premier cas, une complexité de l'ordre de $\Theta(n^3)$ alors que dans le second cas, on aura une complexité de l'ordre de $\Theta(n^2)$, la matrice quelconque M multipliée par la matrice de Householder étant traitée en réalité par colonne. Le graphique qui suit montre une comparaison du temps d'exécution des deux algorithmes.



Par la suite, le produit matriciel par une matrice de Householder sera très utilisé. Par conséquent, il est nécessaire d'optimiser autant que possible l'algorithme afin que la complexité par la suite soit grandement améliorée. De plus, un cas a du être ajouté car il ne pouvait pas être ajouté lors du calcul de U : le cas où $X = Y$. En effet, dans ce cas, U n'est pas défini et $H = Id$.

2 Mise sous forme bidiagonale

Afin de bidiagonaliser une matrice, on applique l'algorithme donné sur la page du projet. Nous expliquons ci après le choix des matrices de Householder :
Soit A une matrice carrée de taille m × n $tq : A = (a_{i,j})_{\substack{1\leq i\leq m\\1\leq j\leq n}}$
On note $(C_k)_{1\leq k\leq n}$ et $(L_k)_{1\leq k\leq m}$ respectivement les vecteurs colonnes, lignes de A.
On cherche à déterminer pour chaque vecteur colonne C'_k (on note C'_k car la matrice A est modifiée à chaque tour de boucle) une matrice de Householder noté Q_k permettant de transformer C'_k comme suit :

$$Q_k.C'_K = Q_k.\begin{pmatrix} a'_{1,k}\\ a'_{2,k}\\ \vdots\\ a'_{m-1,k}\\ a'm,k \end{pmatrix} = \begin{pmatrix} a'_{1,k}\\ \vdots\\ a'_{k-1,k}\\ \alpha\\ 0\\ \vdots\\ 0 \end{pmatrix} \text{ tel que } \alpha = \pm\sqrt{\sum_{i=k}^{m} {a'_{i,k}}^2} \text{ tq } \begin{cases} \alpha \leq 0 & \text{si } a'_{k,k} \geq 0\\ \alpha \geq 0 & \text{sinon} \end{cases}$$

On prend U le vecteur définissant Q_k comme suit : $U = \begin{pmatrix} 0\\ \vdots\\ 0\\ \frac{a'_{k,k}-\alpha}{\sqrt{2*\alpha(\alpha-a'_{k,k})}}\\ \frac{a'_{k+1,k}}{\sqrt{2*\alpha(\alpha-a'_{k,k})}}\\ \vdots\\ \frac{a'_{m,k}}{\sqrt{2*\alpha(\alpha-a'_{k,k})}} \end{pmatrix}$

Par la même méthode on détermine pour chaque vecteur ligne L'_k $(k < n-1)$ une matrice de Householder noté P_k permettant de transformer L'_k comme suit :
$L'_k.P_K = (a'_{k,1}a'_{k,2}\dots a'_{k,n}).P_K = (a'_{k,1}\dots a'_{k,k}\alpha.0\dots 0)$
tel que $\alpha = \pm\sqrt{\sum_{i=k+1}^{n} {a'_{i,k}}^2}$ où $\begin{cases} \alpha \leq 0 & \text{si } a'_{k,k+1} \geq 0\\ \alpha \geq 0 & \text{sinon} \end{cases}$

On prend U le vecteur définissant P_k comme suit : $U = \begin{pmatrix} 0\\ \vdots\\ 0\\ \frac{a'_{k,k+1}-\alpha}{\sqrt{2*\alpha(\alpha-a'_{k,k+1})}}\\ \frac{a'_{k,k+2}}{\sqrt{2*\alpha(\alpha-a'_{k,k+1})}}\\ \vdots\\ \frac{a'_{k,n}}{\sqrt{2*\alpha(\alpha-a'_{k,k+1})}} \end{pmatrix}$

La complexité de l'algorithme est en $\Theta(n^3)$ puisqu'on utilise la version optimisée du produit matriciel. En effet, on effectue une boucle linéaire en n sur un produit matriciel de complexité $\Theta(n^2)$ (comme vu précédemment) et sur la fonction qui détermine le vecteur U qui est de complexité $\Theta(n)$. On a ainsi une complexité totale pour cet algorithme de $\Theta(n^3)$.

3 Transformation QR

Cet algorithme va itérer un certain nombre de fois une double décomposition QR sur les matrices U, S et V. Nous partons de U et V des matrices identité et S une matrice bidiagonale. Suite aux décompositions QR successives, les matrices U, S et V vont être modifiées telles que S tende vers une matrice diagonale tout en conservant le produit U*S*V au cours des itérations. Les matrices U et V seront donc modifiées en conséquence de cet invariant.
On s'assure à chaque itération de cet invariant par une fonction dédiée, et on le prouve également par le calcul :

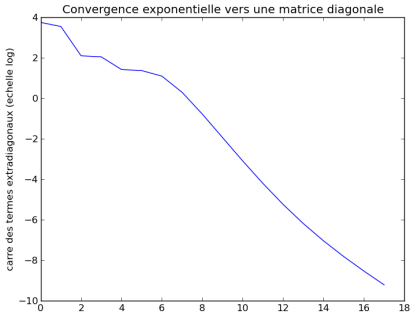
$^tS_k = Q_1.R_1 \quad , {}^tR_1 = Q_2.R_2 \quad , S_{k+1} = R_2 \quad , U_{k+1} = U_k * Q_2 \quad \text{et} \quad V_{k+1} =^t Q_1 * V_k.$
Donc :

$$\begin{aligned} U_{k+1} * S_{k+1} * V_{k+1} &= U_k * \underbrace{Q_2 * R_2}_{} *^t Q_1 * V_k \\ &= U_k * \underbrace{^tR_1 *^t Q_1}_{} * V_k \\ &= U_k *^t (^tS_k) * V_k \\ U_{k+1} * S_{k+1} * V_{k+1} &= U_k * S_k * V_k \end{aligned}$$

Pour déterminer le nombre d'itérations, il faut déclarer un seuil à partir duquel on peut dire que la matrice S est considérée comme diagonale. Pour cela, on étudie la somme des carrés des termes extra-diagonaux. On la compare avec cette même somme avant l'itération, puis on compare cette différence avec la somme des carrés des termes extra-diagonaux. On fixe alors un ε à partir duquel on considère que la matrice S est diagonal telle que :

$$\frac{|(\sum termes\ extra-diagonaux^2)_{etape\ k-1} - (\sum termes\ extra-diagonaux^2)_{etape\ k}|}{(\sum termes\ diagonaux^2)_{etape\ k}} < \varepsilon$$

En conservant la somme des carrés des termes extra-diagonaux après chaque itération, cela nous permet de tracer la convergence de la matrice S vers une matrice diagonale. En analysant la courbe obtenue sur une échelle logarithmique (voir figure ci-dessous), nous nous apercevons qu'elle admet comme asymptote oblique une droite d'équation $g(x) = -x + C$. Or $g(x) = ln(f(x))$. Donc $f(x) = e^{-x+C}$. On peut donc dire que la matrice S converge vers une matrice diagonale de manière exponentielle et qu'il y aura donc $log(\varepsilon)$ itérations dans l'algorithme.



Cependant, les appels effectués lors de l'algorithme QR sont disproportionnés du fait de la présence de matrices bidiagonales. En effet, étant donné que les matrices S, $R1$ et $R2$ sont toutes les trois issues de la décomposition QR d'une matrice bidiagonale, il suffit de montrer que la matrice R issue d'une décomposition QR d'une matrice bidiagonale est elle même bidiagonale. (par un raisonnement analogue à R_1, on prouve que R_2 est elle aussi bidiagonale)
On sait que la matrice carrée S envoyé à l'agorithme de factorisation SVD est bidiagonale. Or, l'algorithme de factorisation QR conserve la forme tridiagonale : la matrice R issue de la factorisation QR est elle-même tridiagonale. Par conséquent, nous allons démontrer cette propriété pour R_1 par le raisonnement suivant :
Si S est une matrice bidiagonale supérieure, alors ^tS est bidiagonale inférieure et par conséquent

tridiagonale. (car l'ensemble des matrices bidiagonales est inclus dans l'ensemble des matrices tridiagonales)
Sachant que la forme tridiagonale est conservée lors de l'application de l'algorithme QR, R_1 est alors tridiagonale. Or, celle-ci étant aussi triangulaire supérieure, elle est donc bidiagonale supérieure. Par conséquent, les matrices S, R_1 et R_2 sont toujours bidiagonales.

En connaissance de cet invariant, nous avons donc implémenté un algorithme simplifié de la version QR. Celui-ci tient compte du fait que nous opérons systématiquement la transformation QR sur des matrices bidiagonales inférieures (et non supérieures car nous travaillons sur les transposées). Celle-ci peuvent donc être caractérisées comme des matrices de Hessenberg. Pour toute factorisation de matrice de Hessenberg M, on a :
$M = QR$ avec $Q = Q_1.Q_2.\ ...\ .Q_{n-1}$ et $R = {}^tQ.M = Q_{n-1}.\ ...\ .Q_2.Q_1.M$
Avec Q_1 de la forme : $\begin{pmatrix} \hat{\mathrm{Q}}_1 & 0 \\ 0 & I_{n-2} \end{pmatrix}, Q_2 : \begin{pmatrix} 1 & 0 & 0 \\ 0 & \hat{\mathrm{Q}}_2 & 0 \\ 0 & 0 & I_{n-3} \end{pmatrix}$ etc ...
Toutes les matrices Q_i sont orthogonales et les $\hat{\mathrm{Q}}_i$ sont des matrices également orthogonales de taille $2 * 2$ telles que : $\hat{\mathrm{Q}}_i.\begin{pmatrix} \times & \times \\ \times & \times \end{pmatrix} = \begin{pmatrix} \times & \times \\ 0 & \times \end{pmatrix}$

On obtient ainsi une factorisation QR simplifiée pour les matrices bidiagonales.
Pour une matrice bidiagonale de taille $n \times n$, l'algorithme est composé de $\Theta(n)$ recherches de matrices orthogonales. Cette recherche est optimisée et est linéaire en n. Ainsi la complexité de cette factorisation QR simplifiée est en $\Theta(n^2)$ comparé à celle implémentée dans python qui est en $\Theta(n^3)$.
Comme nous l'avons vu précédemment, la décomposition SVD opère $\log(\varepsilon)$ itérarions. Les étapes de chaque itérations qui sont les plus fortes en complexité sont les 2 factorisations QR. Grâce à notre algorithme simplifié, nous obtenons donc une complexité en $\Theta(n^2.log(\varepsilon))$ pour la méthode SVD avec une matrice de taille $n \times n$ et une précision ε fixée arbitrairement.

Suite à la décomposition SVD, nous obtenons 3 matrices U, S et V avec S une matrice diagonale. Nous opérons des modifications sur les matrices U et S afin que les valeurs de la diagonale de S soient strictement positives et ordonnées de manière décroissante.
On commence par rendre toutes les valeurs strictement positives. Pour cela, on regarde si la valeur est négative. Si c'est le cas, alors on transforme $S[i,i] = -S[i,i]$ et pour toutes les lignes j de U, on transforme $U[j,i] = -U[j,i]$.
Pour l'étape du tri, on l'effectue tout d'abord sur les valeurs de la diagonal de S. Une fois S trié, on modifie U afin que le produit $U.S$ reste le même pour cela, pour tous les termes de U, on opère :

$$U[j,i] = \frac{ancienne valeur de S[i,i]}{nouvelle valeur de S[i,i]} \times U[j,i].$$

Ainsi, nous obtenons une matrice S diagonale, à termes positifs et triés de manière décroissant tout en conservant l'invariant $U * S$.
Exemple : $U.S = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}.\begin{pmatrix} -2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -3 \end{pmatrix} = \begin{pmatrix} -\frac{2}{3}a & \frac{1}{2}b & -3c \\ -\frac{2}{3}d & \frac{1}{2}e & -3f \\ -\frac{2}{3}g & \frac{1}{2}h & -3i \end{pmatrix}.\begin{pmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = U'.S'$

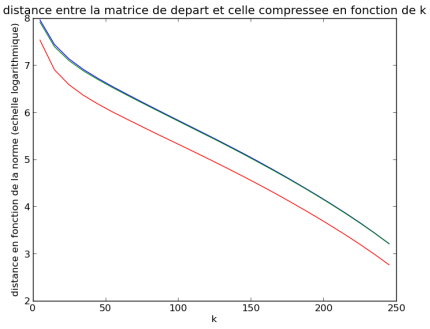
4 Application à la transformation d'image

La compression d'image au rang k consiste à réduire une matrice diagonale donc les valeurs sont strictement positives et ordonnées de manière décroissante en une matrice de rang k (donc suppression des termes d'indice supérieur à k).
Initialement, l'image est représentée par une matrice de taille $n \times n$. Par la méthode SVD, nous nous retrouvons avec 3 matrices U, S et V de taille $n \times n$. Si l'on compresse la matrice diagonale S dont les conditions sont optimales, elle devient de taille k*k. On supprime alors les termes sur U et V qui correspondaient au termes supprimés de S. Ainsi U est de taille $n \times k$ et V de taille $k \times n$. Ainsi le gain de place est : $n^2 - 2kn - k^2$.

Il faut que ce gain soit strictement positif pour que la taille de la transformation soit inférieure à la matrice de départ. Donc :
$n^2 - 2kn - k^2 > 0 \iff k < n(\sqrt{2} - 1)$
Le rang maximal au delà duquel la transformation est de taille supérieure à l'image initiale est donc $k = n(\sqrt{2} - 1)$ pour une matrice de taille $n \times n$

$$\begin{pmatrix} . & . & . & . & n \\ . & & & & \\ . & & & & \\ . & & & & \\ n & & & & \end{pmatrix} \Rightarrow \begin{pmatrix} . & . & k \\ . & & \\ . & & \\ . & & \\ n & & \end{pmatrix} . \begin{pmatrix} . & . & k \\ . & & \\ k & & \end{pmatrix} . \begin{pmatrix} . & . & . & . & n \\ . & & & & \\ k & & & & \end{pmatrix}$$

Pour déterminer l'efficacité d'une compression au rang k, la première méthode est celle de la vérification à l'oeil nu. Il est clair de voir qu'une image compressée au rang 5 est nettement plus floue qu'au rang 50. Afin de rendre cette efficacité plus précise et plus scientifique, le choix a été de calculer la "distance" entre la matrice de l'image réelle et la matrice de l'image compressée. Pour cela, on regarde la différence des normes de ces 2 matrices. On voit donc que plus le rang de la compression est faible, plus les matrices ont une norme éloignée et donc plus la "distance" entre les 2 images est élevée. En traçant l'évolution de cette distance en fonction de k à l'échelle logarithmique (figure ci-dessous), on obtient une courbe linéaire. De même qu'au chapitre précédent, on voit donc que la distance croit de manière exponentielle lorsque l'on diminue le rang de compression de manière linéaire.



Grâce à de nombreux tests effectués sur différentes images, nous nous sommes aperçus qu'en-dessous de $k = \frac{n}{3}$, la qualité de l'image est fortement déteriorée. Il faut donc bien choisir le rang de compression afin de trouver un bon compromis entre gain de taille et non-déterioration de la qualité graphique. Idéalement, nous sommes donc venus à la conclusion de toujours choisir un rang de compression k tel que : $\frac{n}{3} < k < n(\sqrt{2} - 1)$ avec $n = min(m, m')$ pour une matrice de taille $m \times m'$.
Voici des exemples de compression sur l'image du robot martien Curiosity :

. Image réelle de Curiosity | Compression k = 5 | k= 20 | k = 50 | k = 100