Compression de données

L'objectif de ce projet est de présenter une méthode de compression de données (utilisée par exemple pour la compression d'images dans le format jpeg).

Des données sont stockées dans une liste ou un vecteur (voire une matrice s'il s'agit d'une image par exemple). Si la taille de la liste est très grande, le stockage en mémoire est lourd. La méthode que nous développerons repose sur l'idée suivante : pour des raisons informatiques, le stockage d'une valeur nulle ne coûte rien en mémoire. On peut ainsi réduire le stockage de données en réduisant certaines valeurs à 0.

Nous travaillerons avec des listes générées en Python de la manière suivante :

donnees=[f(a+(b-a)*k/n) for k in range(n)]

où f est une fonction usuelle définie sur l'intervalle $[a, b]$. Pour simplifier l'écriture des algorithme, on choisira pour n une **puissance de 2** assez élevée.

On pourra représenter les données à l'aide de *plot*, ce qui permettra de comparer la liste initiale avec sa version compressée.

1 Méthode naïve

On propose de choisir une valeur seuil ε et de réduire à 0 tous les termes de la liste qui sont inférieurs à ε en valeur absolue.

1. Écrire un programme *naif(u,ε)* qui prend une liste u en entrée et effectue la suppression des petites valeurs.
2. Tester votre programme sur plusieurs listes. Représenter à chaque fois la liste et sa version allégée. Préciser à chaque fois le pourcentage de données qui ont été réduites.
3. Que penser de cette méthode ?

2 Méthode de Haar

L'idée reste la même : on souhaite supprimer des coefficients faibles. Mais pour que cela soit plus efficace, on commence par modifier la suite de données. On note u la liste des données (elle démarre avec u_1). On considère les paire de termes consécutifs de la liste (de la forme (u_{2i-1}, u_{2i}) et on crée deux nouvelles listes : la liste des valeurs de la forme $u_{2i-1} + u_{2i}$ et celle des valeurs de $u_{2i-1} - u_{2i}$. On concatène enfin ces deux listes et on obtient une liste de même longueur que u. On appelle T cette transformation.

1. Écrire un programme qui prend une liste en entrée et lui applique la transformation T.
2. Écrire un programme qui prend une liste en entrée et effectue la transformation inverse T^{-1}.
3. Écrire un programme qui prend une liste, la transforme, puis supprime les petites valeurs et enfin applique la transformation inverse.
4. Tester votre programme sur vos listes précédentes. Les figures obtenues sont-elles satisfaisantes ? A-t-on plus réduit le nombre de valeurs stockées ?

Il est facile d'améliorer cette méthode : il suffit d'appliquer plusieurs fois la transformation T. En général, ce sont des valeurs $u_{2i-1} - u_{2i}$ qui ont été supprimées et il est douteux de continuer à les traiter. En revanche, les valeurs $u_{2i-1} + u_{2i}$ sont souvent élevées et on peut espérer améliorer le stockage en les traitant.

Voici la méthode : on transforme la liste, on supprime les petits coefficients, puis on ne garde que la première moitié de la liste obtenue. On applique alors notre méthode à cette sous-liste, et on continue ensuite avec des sous-listes de plus en plus petites.

1. Écrire un programme *Haar(u,k,ε)* qui applique k fois cette idée à des sous-listes de plus en plus petites (prendre $k \leqslant \log_2(n)$ où n est la longueur de u).
2. Écrire un programme effectuant le chemin inverse pour retrouver la liste initiale (modifiée à cause des termes supprimés).
3. Tester vos programmes pour différentes valeurs de ε. Commenter à chaque fois la qualité des figures et la qualité de la compression des données.

3 Interprétation et amélioration

Notre méthode consiste en fait juste à effectuer un changement de base judicieux. On peut en effet considérer que nos listes sont des vecteurs et les transformations qu'on leur fait subir sont linéaires. La base choisie pour le changement de base est appelée base de Haar.

1. Donner la matrice de la transformation T. Quelle est la base correspondante ?
2. Pourquoi cette nouvelle base permet-elle en général de supprimer plus de données ?
3. Déterminer également la base associée à la seconde transformation de la liste. Généraliser aux étapes suivantes de l'algorithme.

   La base obtenue présente un défaut : elle est orthogonale mais n'est pas orthonormée.
4. Expliquer en quoi ce défaut rend l'utilisation de notre valeur seuil peu rigoureuse.
5. Afin d'éviter ce défaut, il suffit d'utiliser une base orthonormée. Comment modifier très légèrement l'opération T afin que la base de Haar correspondante soit orthonormée ? Modifier vos programmes en conséquence.
   Il peut également être intéressant de récrire ces programmes sous forme matricielle.

4 Expériences et conclusions

On souhaite conclure quant à l'efficacité de cette méthode et ses limites. On n'attend pas ici d'étude théorique de la méthode mais de nombreux tests bien choisis et bien commentés. Tester ainsi vos nouveaux programmes sur de nombreuses fonctions f avec différentes valeurs seuil.

Quels taux de compression arrivez-vous à atteindre ? Quelles propriétés des fonctions f interviennent dans la qualité de la compression ? Comment bien choisir la valeur seuil ε ?