Ecole d'ingénieurs de CY Cergy Paris Université

Compressive Sensing

Mini projet

Auteurs :
Mme Manon Cassagne
M. Valentin Labat

Encadrante :
Mme Nisrine Fortin

Table des matières

1 Informations préalables au projet

Nos fichiers s'organisent de la façon suivante. A la racine de notre dossier se trouvent les codes Scilab, Ainsi, vous n'avez qu'à placer votre environnement Scilab au niveau d'où se trouve ce dossier sur votre ordinateur, en faisant "cd '...../Groupe4CassagneLabatMiniProjet'", et vous pourrez ensuite exécuter les fichiers Scilab. Dans le dossier "Resultat" se trouvent les sorties des fonctions sous forme de CSV, dans le dossier "Sujet" il y a le sujet du Mini Projet, et dans le dossier "Donnees", il y a les données fournies transformées comme expliqué ci-dessous. Nous vous conseillons de faire un "clear()" dans votre terminal Scilab entre chaque exécution de fichier, pour qu'il n'y ait pas de problème de variables. De plus, les fichiers sont faits pour ne pas executer leur fonction au départ. Cela permet de les appeler dans d'autres fichiers sans qu'ils ne posent de problème de variables. Veuillez donc décommenter les dernières lignes des fichiers pour les exécuter séparemment. Cette démarche sera rappelée au début de chaque chapitre.

1.1 Technologie utilisée

Pour ce projet nous avions le choix entre coder les algorithmes en ***Python*** ou en ***Scilab***. Etant tous deux plus à l'aise en Scilab, nous avons décidé d'utiliser ce langage. Ainsi, les algorithmes demandés seront codées en Scilab, et vous les trouverez dans l'archive associée à ce rapport.

1.2 Transformation des fichiers de données

Comme la lecture de fichiers *xls* pose des problèmes en Scilab, nous avons donc transformer le fichier "**Données projet.xlsx**" en trois fichiers csv, qui correspondent au trois fenêtres du fichier xls. Ainsi, nous nous retrouvons avec les fichiers "**x.csv**", "**z.csv**" et "**xVal.csv**". Attention tout de même, lors de la conversion, nous avons utilisé le séparateur " ;". En effet, le séparateur "," aurait posé un problème avec la détection des nombres à virgule. Ainsi, il semblerait qu'en ouvrant un fichier csv avec séparateur " ;", seul **Libre Office** arrive a faire un bon affichage des données. Si on veut ouvrir ce type de fichier avec **Word**, il doit être nécessaire de modifier les paramètres d'affichage de l'éditeur. Pour ne pas rentrer dans cette tâche compliquée, nous vous conseillons, si vous voulez ouvrir les fichiers csv, de les ouvrir avec **Libre Office**. Lors de l'ouverture du fichier, dans la partie **Separator**, cochez "**Other**" (ou "**Autre**"), décochez "**Comma**" (ou "**Virgule**") et écrivez "**;**". Ainsi, les données apparaitront sous la bonne forme.

1.3 Notations et dimensions

Pour tout le rapport ci dessous, on note $\mathbf{X}$ la matrice de vecteurs d'apprentissage, et $\mathbf{D}$ le dictionnaire. En fonction des dimensions des fichiers tests, nous avons donc considéré le vecteur x de taille **98x108**. De plus, pour le KSVD,

nous utilisons un nombre de répétitons **K** égal à 10, et le nombre de lignes de la matrice **A** égal à 100.

2 Apprentissage d'un dictionnaire

2.1 L'algorithme du KSVD

Ci-dessous (en page 4), vous retrouverez l'algorithme du KSVD par apprentissage. Pour transformer cet aglorithme en algorithme du KSVD simple, il suffit de supprimer la boucle Pour qui démarre à la ligne **7**, et qui permet d'éxecuter le KSVD L fois.

2.2 Implémentation en Scilab

L'algorithme du KSVD, ainsi que les autres fonctions nécessaires à son fonctionnement, ont donc été implémentées dans le fichier "KSVD.sce". Merci de décommenter la ligne 245 du fichier afin que l'exécution du fichier se fasse comme il faut. Ce fichier contient donc les fonctions de l'OMP et ses sous fonctions, ainsi que celles liées au KSVD et ses sous fonctions. Leur fonctionnement est expliqué dans les cartouches de fonction, et leurs noms sont souvent explicites. En ce qui concerne la valeur des variables, nous lisons la valeur de X dans le fichier "x.csv", et nous avons K=100 et L=10, comme spécifié dans la consigne. Cependant, il est facile de modifier les valeurs dans la fonction "initialisationVariables()" dans le fichier Scilab.

2.3 Problèmes rencontrés

Pendant plusieurs semaines, nos fonctions étaient extrèmement lentes. Par exemple, pour seulement deux répétitions de l'apprentissage par KSVD, le fichier prenait plus de deux minutes à s'éxécuter. C'est en se plongeant dans le code que nous avons repensé la façon dont il était fait, et nous avons pu remarqué que certaines fonctions que nous avions créées existaient déjà en **Scilab**. Par exemple, nous avions créés une fonction qui permet d'ajouter une colonne à une matrice, en parcourant la matrice, et en la mettant à la fin, alors qu'une simple instruction **Scilab** permet de le faire. Cela venait du fait que certains de nos codes étaient en Python, et que nous les avons "traduits" en Scilab, mais Scilab permet de faire des choses comme je l'ai dit en exemple que ne permet pas Python. Au final, en modifiant tous ces aspects là, nous nous retrouvons avec un temps d'execution de moins d'une minute pour 10 répétitions du KSVD, ce qui est raisonnable.

2.4 Résultats et interprétation

Les résultats du code KSVD sont les fichiers "Dico.csv" et "A.csv", et comme leurs noms l'indiquent, ils sont constitués respectivement du dictionnaire et de la matrice A. Il semble que les résultats soient cohérents. En effet, quand on regarde

Algorithme 1 Algorithme du KSVD : **KSVD()**

Données : X, K, L
Début
$[N, l] \leftarrow taille(X)$ ▷ X : N lignes, l colonnes
$Dico \leftarrow []$
$A \leftarrow zeros(K, l)$
Pour i **allant de** 1 **à** K
  $Dico \leftarrow [Dico, \frac{X(:,i)}{\|X(:,i)\|}]$ ▷ On ajoute $\frac{X(:,i)}{\|X(:,i)\|}$ à Dico
Fin Pour
Pour j **allant de** 1 **à** L
  $A \leftarrow OMP(X, Dico)$ ▷ Calcul de A par la méthode de l'OMP
  Pour i **allant de** 1 **à** K ▷ Parcours des colonnes de A
    $wi \leftarrow []$
    Pour k **allant de** 1 **à** l
      Si $A[i, k] \neq 0$ ▷ Si A(i,k) non nul
        $wi \leftarrow wi + [j]$ ▷ Le "+" représente la concaténation de tableaux
      Fin Si
    Fin Pour
    $card_{wi} \leftarrow taille(wi)$
    Si $card_{wi} \neq 0$ ▷ Si w_i n'est pas vide
      $somme \leftarrow 0$
      Pour k **allant de** 1 **à** K
        Si $k \neq i$
          $somme \leftarrow somme + D[:, k] * A[k, :]$ ▷ col. k de D * ligne k de A
        Fin Si
      Fin Pour
      $E_i \leftarrow X - somme$ ▷ Calcul de l'erreur
      $[valeurs, card] \leftarrow taille(w_i)$
      $\Omega_i \leftarrow zeros[l, card]$ ▷ Matrice de 0 de taille l*card
      Pour k **allant de** 1 **à** card
        $\Omega_i[w_i[k], k] \leftarrow 1$ ▷ Ω vaut 1 si k est entre 1 et card
      Fin Pour
      $Ei_r \leftarrow Ei * \Omega_i$ ▷ Calcul de l'erreur rectifiée
      $[U, \Delta, V] \leftarrow svd(Ei_r)$ ▷ Décomposition en valeurs singulières de l'erreur rectifiée
      $Dico(:, i) \leftarrow U(:, 1)$
      $[M, N] \leftarrow taille(A)$
      $A_{maj} \leftarrow A$
      Pour k **allant de** 1 **à** N
        Si $(A[i, k] \neq 0)$ ▷ Si la valeur de la composante en i*k est non nulle
          $A_{maj}[i, k] \leftarrow (Delta(1, 1) * V(1, :))[1]$ ▷ On met A à jour
        Fin Si
      Fin Pour
      $A \leftarrow A_{maj}$ ▷ Mise à jour de A pour que X soit parcimonieux dans D
    Fin Si
  Fin Pour
Fin Pour
Fin

la matrice A, elle est constituée en grande partie de 0. Ainsi, le dictionnaire D adapté au KSVD pour ce X se trouve dans le fichier "Dico.csv".

3 Codage parcimonieux

3.1 Ordre de parcimonie

Dans la fonction parcimonie() qui se trouve dans notre fichier KSVD, nous avons calculé la parcimonie de chaque vecteur de A. Ainsi, nous trouvons que l'ordre de parcimonie pour ce vecteur est 39. Il y a donc dans chaque vecteur de A, 39 composantes non nulles. Nous utiliserons donc s = 39 comme base pour le codage du COSAMP.

3.2 Implémentation du COSAMP

3.2.1 Organisation du code

Vous trouverez dans le fichier "COSAMP.sci" notre implémentation du COSAMP. Là encore, notre fichier s'articule comme le précédent. Il y a la fonction COSAMP(), la fonction d'initialisation des données, et la fonction d'exécution. N'oubliez pas de décommenter la ligne 107 afin d'exécuter le fichier seul, et à faire un clear() afin d'avoir un environnement de variables propre. De plus, vous pouvez modifier les données dans la fonction initialisationVariables(). L'affichage des résultats se fait "en dur", dans le terminal de Scilab, avec l'affichage de alpha, du nombre d'itérations et de la norme du résiduel.

3.2.2 Observations et résultats de l'algorithme

Pour ce qui est des observations autour de notre algorithme, il semblerait que celui-ci "stagne" au bout d'un moment. En effet, quelque soit le nombre d'itérations choisi, il semblerait qu'il parvienne toujours au bout, et que la norme du résiduel soit toujours égale à $1.068D+10$ pour un s=39. En effet, le problème se passe quelque soit la valeur de s choisie. Cependant, comme l'algorithme fonctionne comme il faut pour les premières itérations, c'est certainement la matrice choisie, la matrice D crée par le KSVD, qui fait cet effet là.

3.3 Relaxation l_p et optimisation convexe

Soit le problème $(P_p) : min \left\|\alpha\right\|_p$ tel que $x = D\alpha$ avec

$$l_p(\alpha) = \left\|\alpha\right\|_p = (\sum_{i=1}^{K} |\alpha_i|^p)^{\frac{1}{p}}$$

Montrons que le problème (P_p) s'écrit sous forme du problème de moindres carrés pondérés suivant :

$$(P_2) : min \left\|W\alpha\right\|_2 \text{ tel que } x = D\alpha$$

$$\|\alpha\|_p = (\sum_{i=1}^{K} |\alpha_i|^p)^{\frac{1}{p}} = (\sum_{i=1}^{K} (|\alpha_i|^{\frac{p}{2}-1} \times |\alpha_i|)^2)^{\frac{1}{p}} = (\sum_{i=1}^{K} (w_i \times |\alpha_i|)^2)^{\frac{1}{p}} = (\sum_{i=1}^{K} |w_i \alpha_i|^2)^{\frac{1}{p}}$$

Or,

$$(\sum_{i=1}^{K} |w_i \alpha_i|^2)^{\frac{1}{p}} = (\|W\alpha\|_2)^{\frac{1}{p}}$$

On a donc écrit le problème (P_p) sous la forme d'un problème de moindres carrés pondérés.
Le problème (P_2) possède une solution si le produit $W \times \alpha$ existe. Il faut donc que le nombre de colonnes de W soit égal à la taille de α. De plus,

$$\sum_{i=1}^{K} w_i^2 \alpha_i^2 = \sum_{i=1}^{K} (|\alpha_i|^{\frac{p}{2}-1})^2 \times |\alpha_i|^2 = \sum_{i=1}^{K} |\alpha_i|^{p-2} \times |\alpha_i|^2 = \sum_{i=1}^{K} |\alpha_i|^p$$

Donc le problème (P_2) possède une solution si le problème (P_p) possède une solution, donc si le vecteur est parcimonieux. Comme X est en fonction de α, le problème n'est pas linéaire. De plus, si α devient parcimonieux, la matrice W devient de ce fait singulière.

Dans le problème (P_2), on a W en fonction de α. En procédant de manière itérative et en fixant le W à partir du α précédent. On a donc, $\alpha^{(k-1)} = argmin \left\| x - D\alpha^{(k)} \right\|$ (car on a la formule de base $x = D\alpha$). Avec les moindres carrés, on obtient donc $\alpha^{(k-1)} = (W'D'DW)^{-1}W'D'x = W'WD'(W'DD'W)^{-1}x$. Si on note $Q = W'W$, on a bien Q matrice diagonale et $\alpha^{(k-1)} = QD'(DQD')^{-1}x$.

3.4 L'algorithme IRLS

Vous trouverez l'algorithme IRLS dans le fichier "irls.sce" avec le même fonctionnement que tous les autres fichiers.

4 Procédé du compressive sensing

4.1 Vecteurs de mesures

Les différents vecteurs de mesures ont été calculé à l'aide du code se trouvant dans "Procede.sce". Veuillez décommenter la ligne 185 pour exécuter ce fichier seul, et la recommenter pour utiliser la reconstruction. Ils ont été calculés en fonction des x1, x2 et x3 fournis. Ainsi, vous retrouverez les résultats des y dans le dossier "Resultats/y" et les résultats des phi dans le dossier "Resultats/phi". Ainsi, le fichier du nom de "$x1\_phi1\_y1.csv$" est constitué du vecteur de mesure

y1, calculé avec la fonction phi1, et avec le vecteur x1 ; et ainsi de suite. Et le fichier "$x1\_y1\_phi1.csv$" est constitué de la fonction ph1, calculée à partir du vecteur x1 et du vecteur y1.
Nous avons ensuite calculé la cohérence mutuelle entre le dictionnaire D et les différentes matrices de mesure pour chacun des pourcentages du vecteur P. Les voici dans le tableau ci-dessous. Il est important de noter tout du moins que les valeurs du tableaux varient à chaque exécution du code. En effet, les phi sont construits à l'aide de tirages aléatoires. De plus, si vous voulez avoir l'affichage de cette matrice dans notre code Scilab, elle correspond à la variable **matriceCM**.

Cohérence mutuelle	P=15	P = 20	P = 25	P = 30	P = 50	P = 75
Φ_1	8,6472	8,5825	8,5942	8,7295	8,8134	8,5811
Φ_2	2,294	1,8982	2,3966	2,1902	1,9356	1,7167
Φ_3	7,3279	7,608	7,8901	7,3766	7,7127	7,3899
Φ_4	2,7673	1,7475	2,8718	2,7008	2,529	2,5279
Φ_5	6,4715	6,2992	6,6451	6,457	6,6521	6,6944

Les meilleurs matrices de mesure sont celles qui ont la cohérence mutuelle avec le D la plus petite. Ainsi, on suppose que phi2 et phi4 sont les meilleurs matrices de mesures. Pour la suite, on ne considèrera que ces deux là.

4.2 Reconstruction des vecteurs d'origine

Après reconstruction des vecteurs d'origine en utilisant tous les algorithmes étudiés, nous calculons les erreurs relative de reconstitution. Nous obtenons les tableau suivant :

Erreur relative	P = 15	P = 20	P = 25	P = 30	P = 50	P = 75
MP	0,4765	0,4107	0,3861	0,3885	0,4180	0.5228
OMP	$1,12.10^{-10}$	$1,59.10^{-10}$	79755,728	16112,515	177428,7	178054
StOMP	$4,64.10^{-11}$	$3,38.10^{-10}$	$1,06.10^{-10}$	$4,19.10^{-10}$	0,9934	0,9934
CoSaMP	$4,64.10^{-11}$	$3,38.10^{-10}$	$1,06.10^{-10}$	$4,19.10^{-10}$	$3,05.10^{-9}$	$1,29.10^{-8}$
IRLS	0,0186	0,0799	0,1033	0,1857	0,2287	0,2988

Tout d'abord, nous remarquons une erreur lors du calcul de l'erreur relative pour la reconstitution par la méthode OMP. En effet, pour les deux premières valeurs de P, les erreurs semblent cohérentes puisqu'elles sont très petites. En revanche, à partir de P=25, les erreurs sont énormes. Nous avons essayé de comprendre d'où ce changement brutal venait mais nous n'avons pas réussi à résoudre ce problème.
Nous remarquons un problème du même style, mais moins gênant, pour la méthode StOMP. Les erreurs relatives pour P=50 et P=75 sont grandes mais restent en dessous de 1.

Outre ces problèmes, nous remarquons que l'algorithme le plus efficace est CoSaMP : toutes les erreurs sont très petites.

En prenant en compte le fait que nous avons des erreurs dans les méthodes OMP et StOMP, par précaution, nous ne les sélectionnons pas pour la suite du projet. Nous allons utiliser les méthodes CoSaMP et IRLS pour travailler sur la partie suivante : reconstituer les données manquantes du vecteur z.

5 Données manquantes

Le code associé à ce chapitre se trouve dans le fichier "valeursManquantes.sce". Le vecteur z_r que l'on considère pour cette partie contient 86 lignes. On choisit donc d'utiliser l'algorithme COSAMP, le meilleur des cinq méthodes proposées dans ce projet. A l'aide des matrices de mesure sélectionnées précédemment (Φ_2 et Φ_4), on reconstruit donc le vecteur z. Les vecteurs obtenus sont très proches du vecteur z à trous, en ce qui concerne les valeurs connues. Les vecteurs reconstitués se trouvent dans le dossier "Resultat/z".

Pour la dernière partie de ce projet, nous considérons le vecteur z_0. Celui-ci est constitué de 0 à la place des données manquantes. Nous utilisons ensuite les matrices de mesure Φ_2 et Φ_4 pour acquérir des données sur z_r. On remarque alors que ce procédé suit exactement la même trame que celui effectué précédemment. Autrement dit, nous avons déjà fait ce travail ci-dessus.

6 Conclusion

Ce projet nous a permis de parcourir les aspects généraux du Compressive Sensing en appliquant les concepts et méthodes vues en cours et en découvrant de nouvelles méthodes de codage parcimonieux (COSAMP et IRLS).