Projet d'Informatique et Mathématiques Appliquées
Algèbre Linéaire Numérique

BOUKRAICHI Hamza, DEMERY Cyril, NGUYEN Teo

1 Deliverables phase 1

1.1 Analysis of the data matrix Z, reconstruction and classification of solutions of a simplified atmospheric model

1. L'utilisation de la fonction matlab *svd* permet de calculer tous les couples singuliers de la matrice, puis nécessite un filtrage afin de récupérer le couple dominant, ce qui implique des calculs inutiles et un travail supplémentaire. À l'inverse, la méthode des espaces propres, inspirée de la méthode de la puissance successive, calcule directement le couple dominant et déduit les vecteurs singuliers gauches et droites les uns des autres.

2. On a : $Z = U\times$ diag$(D) \times V^T$ où diag(D) est la matrice carrée de dimension $n_d \times n_d$, diagonale, et ayant sur sa diagonale les valeurs successives de D.

   n_d étant la taille du vecteur D, n_d représente le nombre de valeurs singulières. Il faut donc prendre n_d grand pour avoir une meilleure précision.

3. α dépend directement de la condition initiale. Dans un système où on a un très grand changement du système par rapport à l'état initial, alors le α trouvé à l'état initial ne sera plus valide.

4. Après l'exécution du fichier *construction.m* on remarque que l'erreur relative est de l'ordre de 10^{-2}. Malgré cela, il existe des différences entre la courbe reconstituée et la courbe réelle : on retrouve certes les mêmes intervalles de croissance et de décroissance mais la valeur des pics est différente.

   Quand on exécute une nouvelle fois, on remarque que l'erreur relative est de l'ordre de 10^{-3}et dans ce cas la courbe reconstruite approche de façon convaincante la courbe solution, néanmoins le nombre de valeurs singulières calculées est constant, et la vitesse de calcul aussi (au centième de seconde près).

   Les exécutions successives montrent la variation de l'ordre de grandeur entre 10^{-2} et 10^{-3} avec les différences que cela implique sur les courbes ; on garde le même nombre de valeurs singulières et le même temps de calcul.

   Ces différences découlent du fait que les perturbations suivent une loi normale, d'où leur variation, et cela a été modélisé par l'utilisation de la fonction matlab *random*.

   Donc pour comparer proprement l'efficacité des deux méthodes, le programme devra comparer les erreurs relatives résultantes des deux méthodes et le temps d'exécution pour les mêmes valeurs de perturbation, mais plusieurs fois. En effet, comme nous l'avons remarqué, il existe des cas favorables et non favorables pour cette méthode.

5. On a : $\frac{D_{n_d+1}}{D_1} \leq 1 - PercentInfo$ d'après la partie 3 de l'algorithme d'analyse de données. Donc si $PercentInfo \approx 1$, on aura $D_{n_d+1} \ll D_1$ et on aura donc un très grand nombre de valeurs singulières, donc une meilleure précision.

   À l'inverse, si $PercentInfo \approx 0$, on aura un petit nombre de valeurs singulières (puisqu'on aura immédiatement $\frac{D_2}{D_1} \leq 1$), donc une moins bonne précision.)

6. D'après la partie 4 de l'algorithme d'analyse de données, on cherche à minimiser le quotient :

$$\frac{||UU^T Z - Z||_2}{||Z||_2}$$

Or ce quotient est égal à :

$$\frac{||Z(I - UU^T)||_2}{||Z||_2} = \frac{\pi_i}{||Z||_2}$$

On a donc tout intérêt à prendre π_i le plus petit possible.

7. Selon le graphe en barre issu de l'exécution du fichier *classification.m* on remarque que la plus basse valeur de $||(I - U_i^T U_i)Z_{obs})||$ correspond à $i = 1$, donc on peut en conclure que nous sommes dans le cas d'un champ de vitese geostrophique.

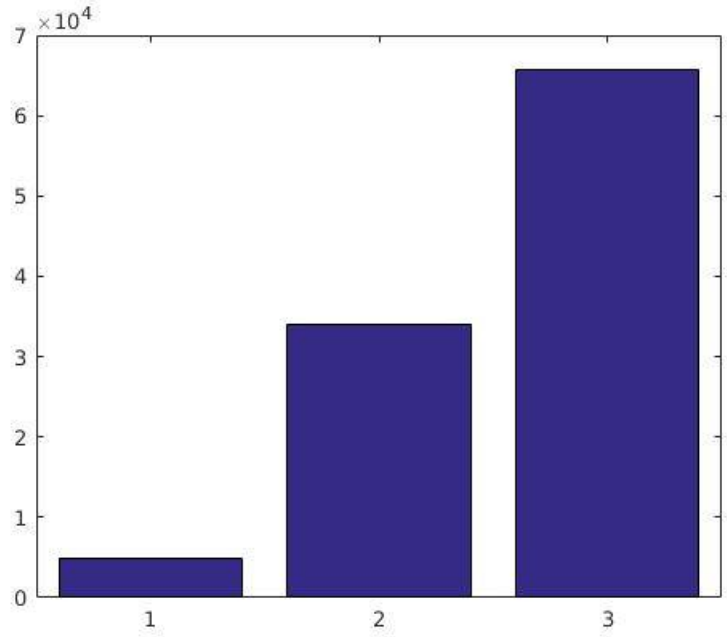

Figure 1 – Exécution de *classification.m*

1.2 Building Information-dense subspaces : a preliminary strategy

1. - Explicite : ZZ^T de dimension m^2, complexité de n^2 pour le calculer. Dans l'algorithme on aura donc ensuite une complexité de $niter \times p \times m^2$. Donc une complexité spatiale de m^2 et complexité temporelle de $niter \times p \times m^2 + n^2$

     Z^TZ de dimension n^2, complexité de m^2 pour le calculer. Dans l'algorithme on aura donc ensuite une complexité de $niter \times p \times n^2$. Donc une complexité spatiale de n^2 et complexité temporelle de $niter \times p \times n^2 + m^2$

     Donc complexité spatiale et temporelle mieux pour Z^TZ que pour ZZ^T.

   - Implicite : On doit stocker Z donc une complexité spatiale de $n \times m$.

     Pour calculer $(ZZ^T)^p$ on effectue $ZZ^TZZ^T...ZZ^T$, soit $pn^2 + (p-1)m^2$ opérations, donc l'algorithme aura une complexité de $niter \times ((p-1)m^2 + pn^2)$

     Idem pour $(Z^TZ)^p$ la complexité sera de $niter \times (pm^2 + (p-1)n^2)$.

2. Le plus intéressant, pour $m \geq n$, est donc de calculer Z^TZ en explicite.

1.3 Building Information-dense subspaces : an alternative algorithm

initialization
repeat
  if niter pair **then**
    $Y \leftarrow V$
    for i = 1 **to** p **do**
      if i pair **then**
        $Y \leftarrow Z^TY$
      else
        $Y \leftarrow ZY$
      end if
    end for
  else
    $Y \leftarrow U$
    for i = 1 **to** p **do**
      if i pair **then**
        $Y \leftarrow ZY$
      else
        $Y \leftarrow Z^TY$
      end if
    end for
  end if
  if p impair **then**
    if niter pair **then**
      $Y \leftarrow ZY$
    else
      $Y \leftarrow Z^TY$
    end if
  end if
  if niter pair **then**
    $Y \leftarrow$ orthonormalisation des colonnes de Y
    Calculer le quotient de Rayleigh $H = V^TAV$
    Calculer la décomposition spectrale $H = X\Lambda X^T$
    $V \leftarrow VX$
    $U \leftarrow Z^TV$
  else
    $Y \leftarrow$ orthonormalisation des colonnes de U
    Calculer le quotient de Rayleigh $H = U^TA^TU$
    Calculer la décomposition spectrale $H = X\Lambda X^T$
    $U \leftarrow UX$
    $V \leftarrow ZU$
  end if
  for $i = converged + 1, m$ **do**
    if $||Av_i - \lambda_i v_i||/||A|| <$ **then**
      $converged = converged + 1$
    else
      break
    end if
  end for
  $niter \leftarrow niter + 1$
until $converged > m$ or $niter > MaxIter$

Explication de l'algorithme : Selon la parité de *niter*, on calcule soit le vecteur singulier droit soit le gauche.

À chaque iteration, on met à jour U et V à partir des informations obtenues à partir d'un seul des deux. On profite ainsi d'une éventuelle convergence plus rapide de l'un.

2. p représente le nombre de matrices qu'on va multiplier. Si on a p pair il suffit de faire la multiplication par Z^TZ un certain nombre de fois. En revanche, si p est impair, alors on doit multiplier une fois de plus par Z (ou Z^T).

3. L'écriture de *reconstruction_alt.m* nous donne :

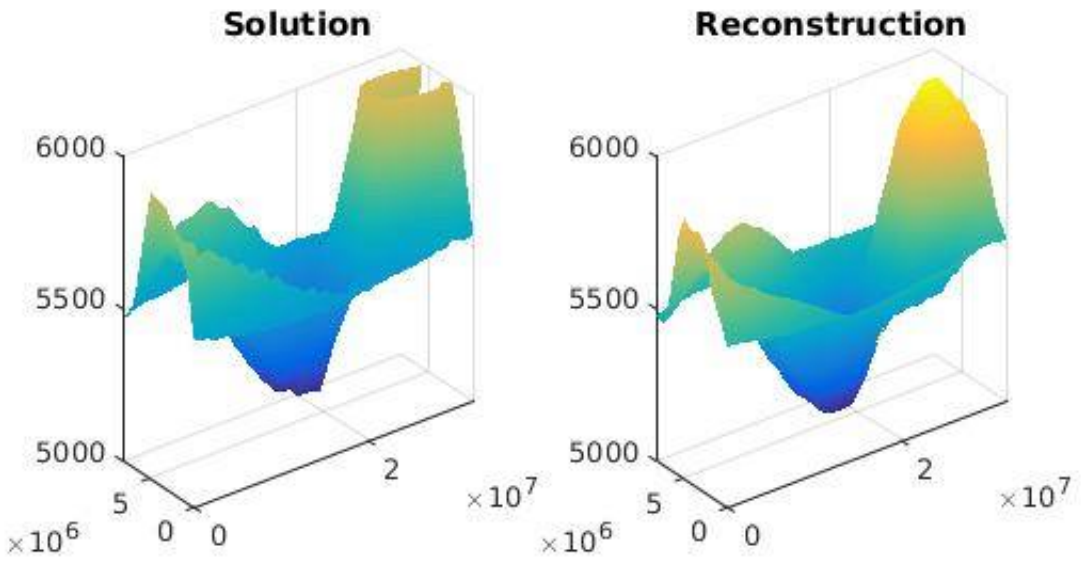

Figure 2 – Exécution de *reconstruction_alt.m*

On trouve 45 valeurs singulières contre 7 pour l'algorithme initial. Pour $p = 2$, la vitesse de convergence est beaucoup plus lente, de l'ordre de 200s, contre une dizaine pour l'algorithme initial. En revanche, pour un p plus grand (par exemple $p = 4$), la vitesse de convergence est beaucoup plus rapide : elle se fait en environ 5s.