Ecole d'ingénieur ENSEEIHT

Algèbre Linéaire Numérique - Rapport de projet - G29

Noël DUPUIS
Florian GARIBAL
Guillaume HOTTIN

3 juin 2016

Sommaire

Chapitre 1

Travail effectué pendant ce projet

Le but de ce projet était d'illustrer l'utilité des valeurs propres dans la résolution des équations dérivées partielles. Ce problème étant appliqué à deux cas concrêts : la prévision d'évolution en espace/temps d'une vague atmosphérique sans l'intégration d'un modèle numérique et l'estimation de la composante de vitesse des conditions initiales en partant des observations.

Afin de résoudre ce problème, le projet était divisé en deux parties. La première consistait en la compréhension de l'utilisation des techniques de réduction afin d'utiliser des espaces de plus petites dimensions. Pour cela il a été utile de savoir réduire puis reconstruire un espace à l'aide de sous espace. La deuxième partie concerne l'optimisation du calcul des valeurs singulières et des vecteurs singuliers droit et gauche associés grâce à l'amélioration de la technique des *power method*.

1.1 Première partie : Réduction et reconstruction d'espaces

Dans cette première partie, nous avons travaillé sur les inconvénient d'utiliser les méthodes déjà implémentée dans *Matlab* concernant la décomposition en valeurs singulière. En effet cette méthode calcule l'ensemble des valeurs singulières alors que, vu la taille très importante des espaces, nous souhaitions seulement les valeurs singulières dominantes.

Suite à cela, on a remarqué que l'étude du critère $\frac{\|UU^T Z - Z\|_F}{\|Z\|_F}$ pour évaluer la qualité des vecteurs de la base du sous espace U n'était pas très **optimale**.

Ensuite on a cherché comment reconstruire l'ensemble original à partir des données compressées puis on a essayé de justifier le choix de *nd*. On a trouvé que :

$$Z = UD^T V$$

Concernant le choix de *nd*, ce dernier doit être suffisamment petit pour occuper le moins d'espace mémoire possible, mais assez grand pour obtenir des résultats satisfaisant, en fonction des besoins que l'on a et des ressources dont l'on dispose.

De plus, on a déduit que pour **alpha** fixé, l'approximation peut ne pas convenir car, lors de comportements chaotiques (fortes variations sur de courtes périodes de temps), la résolution s'écarte de la solution.

Nous avons, par la suite, cherché à justifier le choix du paramètre **GF** en fonction de la distance π_i entre Z_{obs} et les 3 espaces calculés. On a donc décidé de prendre la plus petite des valeurs car cela permet de choisir le modèle qui se rapproche le plus des observations sur le laps de temps de simulation.

Pour finir la première partie nous avons démontré plusieurs propriétés à propos des valeurs propres de A etses vecteurs propres associés puis nous avons codé les différentes parties en Matlab.

1.2 Deuxième partie : Optimisation du calcul des valeurs propres et vecteurs propres associés

Afin d'optimiser le calcul des valeurs propres et vecteurs propres associés, nous avons développé une fonction Fortran qui calcule ces valeurs et vecteurs selon plusieurs méthodes.
Les quatres méthodes sont les suivantes :

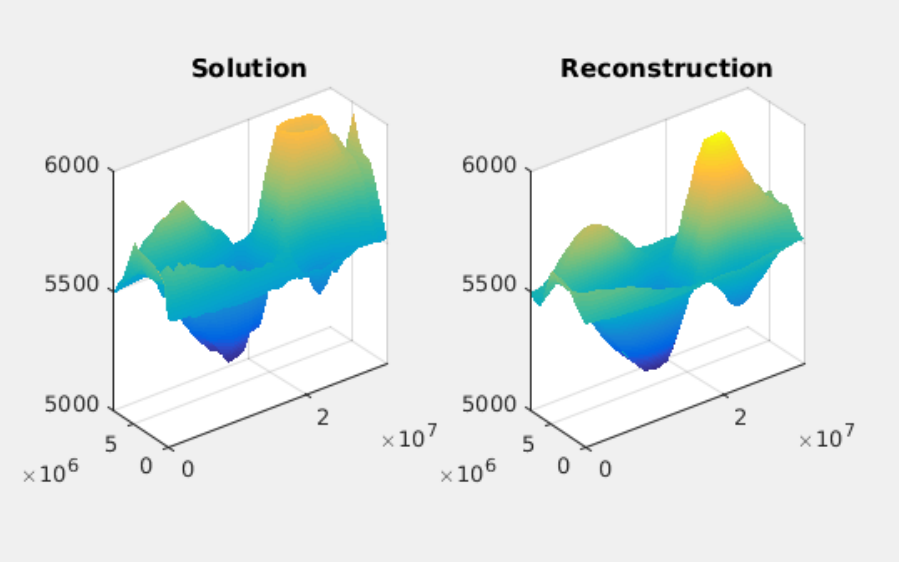
- Calcul de ZZ^T :
  - Implicitement : Utiliser la méthode *subspace_iter_ev* avec Z en paramètre qui itère sur ZZ^T sans la calculer.
  - Explicitement : Calcule explicitement ZZ^T avant de trouver ses valeurs propres et ses vecteurs singuliers gauche puis droit.
- Calcul de Z^TZ
  - Implicitement : Utiliser la méthode *subspace_iter_ev* avec Z^T en paramètre qui itère sur Z^TZ sans la calculer.
  - Explicitement : Calcule explicitement Z^TZ avant de trouver ses valeurs propres et ses vecteurs singuliers gauche puis droit.

Chapitre 2

Tests effectués

2.1 Explicit $Z^T Z$

Résultat Matlab :
IT : 1 – Found 0 eigenvalues (0.00000 percent achieved)
IT : 2 – Found 0 eigenvalues (0.00000 percent achieved)
IT : 3 – Found 0 eigenvalues (0.00000 percent achieved)
IT : 4 – Found 0 eigenvalues (0.00000 percent achieved)
IT : 5 – Found 0 eigenvalues (0.00000 percent achieved)
IT : 6 – Found 0 eigenvalues (0.00000 percent achieved)
IT : 7 – Found 0 eigenvalues (0.00000 percent achieved)
IT : 8 – Found 8 eigenvalues (0.98607 percent achieved)
8 sing. values found in 0.284 seconds ; they provide 0.99error = 0.019922

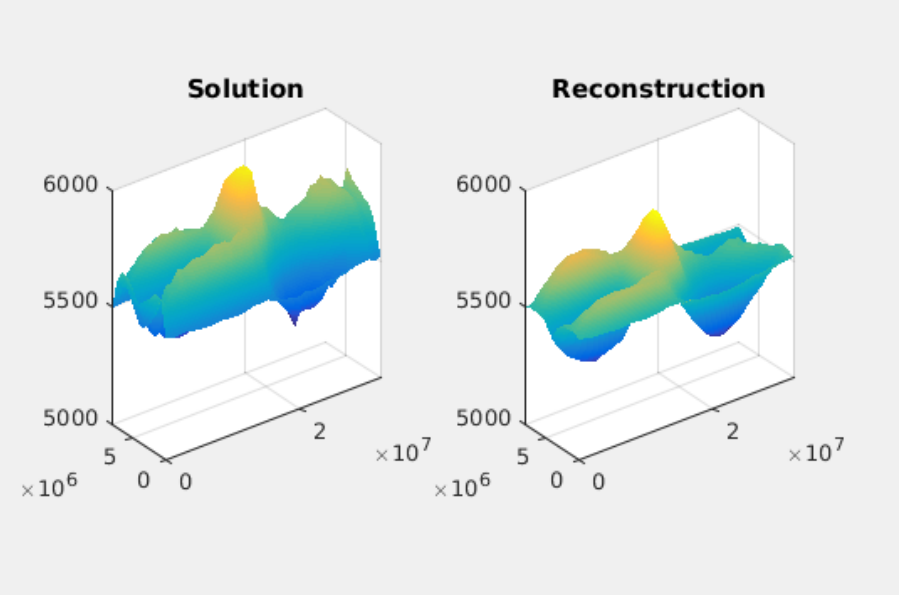


2.2 Implicit ZZ^T

Cette méthode ne fournit pas de valeurs singulières après 300 itérations.

2.3 Implicit Z^TZ

Résultat Matlab :
IT : 1 – Found 0 singular values (0.00000percent achieved)
IT : 2 – Found 0 singular values (0.00000percent achieved)
IT : 3 – Found 0 singular values (0.00000percent achieved)
IT : 4 – Found 0 singular values (0.00000percent achieved)
IT : 5 – Found 0 singular values (0.00000percent achieved)
IT : 6 – Found 0 singular values (0.00000percent achieved)
IT : 7 – Found 0 singular values (0.00000percent achieved)
IT : 8 – Found 9 singular values (0.98725percent achieved)
9 sing. values found in 3.927 seconds ; they provide 0.99error = 0.020104



2.3.1 Différences de performances

4 méthodes étaient à notre disposition pour la décomposition en valeurs singulières :
— Implicit ZZ^T
— Implicit Z^TZ
— Explicit ZZ^T
— Explicit Z^TZ

Parmis ces méthodes,la méthode Implicit ZZ^T ne convergeait pas vers une solution et l'éxecution de la méthode Explicit ZZ^T était impossible.

Parmis les deux méthodes restantes, c'est la méthode Explicit Z^TZ qui est la plus rapide, tout en restant aussi efficace que la méthode Implicit Z^TZ.

Chapitre 3

Analyse des algorithmes matlab

Dans cette partie on choisit l'algorithme explicite t $Z \times Z^T$

3.1 Vitesse d'execution

on peut remarquer que l'algorithme en phase 2 est plus rapide, (si l'on choisit la bonne méthode). En effet, l'option check permet de comparer les deux temps de reconstruction. Avec l'option à *true* on lance l'algorithme de la phase 1. Sur l'exemple initial du fichier $Reconstruction_fortran$ le temps d'execution est de quasiment 1 seconde pile. Avec le nouvel algorithme, on est de l'ordre du quart de seconde sur un PC de salle de TP. C'est un gain considérable si l'on souhaite agrandir le modèle.

3.2 Approximation du modèle

Lorsqu'on choisit le nouvel algorithme, on trouve un modèle un peu plus éloigné de la solution, de l'ordre de 0.5% par rapport au premier algorithme si l'on commence a pousser le nombre d'itérations. Ici, les valeurs sont les valeurs initiales du fichier.

3.3 Modification du nombre d'itération

Le modèle reconstruit grâce au second algorithme est moins éloigné de la solution lors de la répétition des itération. La reconstruction du modèle est beaucoup plus fidèle à la solution. Ici, en répetant l'opération sur 250 itérations :

3.4 Modification du paramètre $percentInfo$

On ne reconnait pas de grandes diffféences à part en temps d'execution si l'on ne fait varier que ce paramètre entre 0 et 1 :

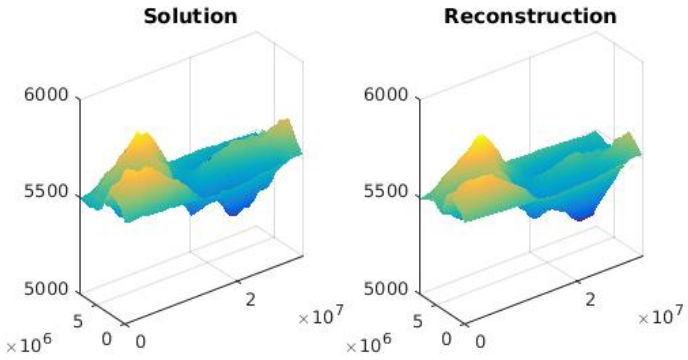


FIGURE 3.1 – Données initiales (50 iter/ percentInfo = 0.99/ nouvel algorithme)

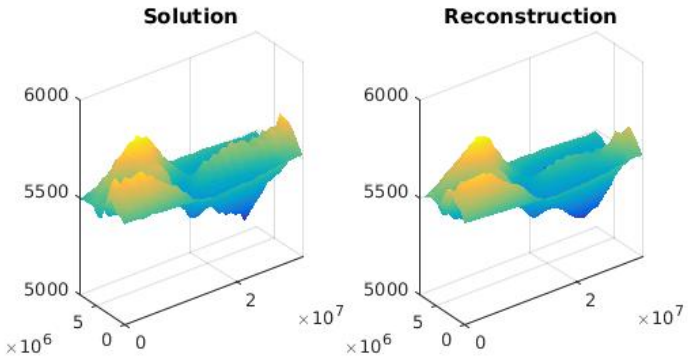


FIGURE 3.2 – Données initiales (50 iter/ percentInfo = 0.99/ ancien algorithme)

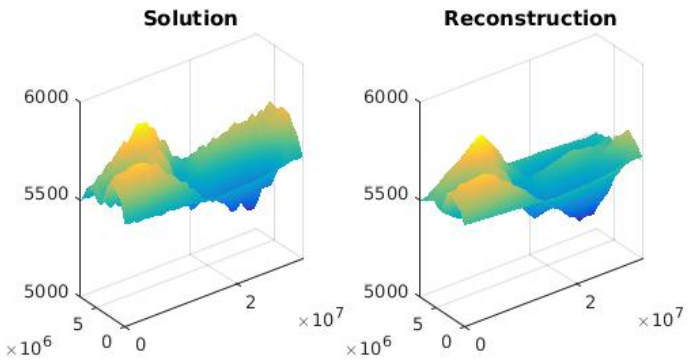


FIGURE 3.3 – Données : 250 iter/ percentInfo = 0.99/ nouvel algorithme

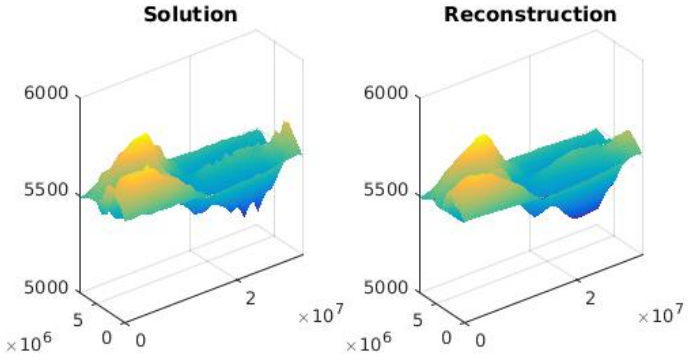


FIGURE 3.4 – Données 250 iter/ percentInfo = 0.99/ ancien algorithme

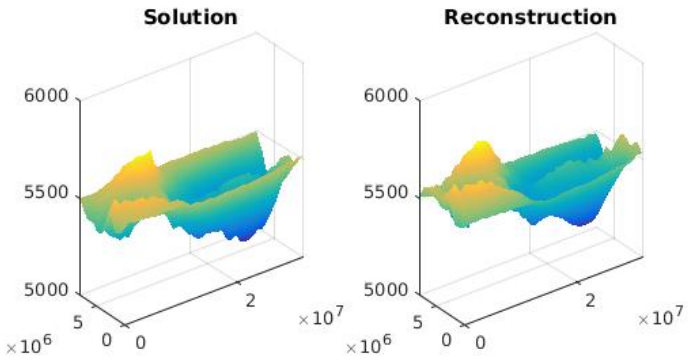


FIGURE 3.5 – Données : 50 iter/ percentInfo = 0.10/ nouvel algorithme

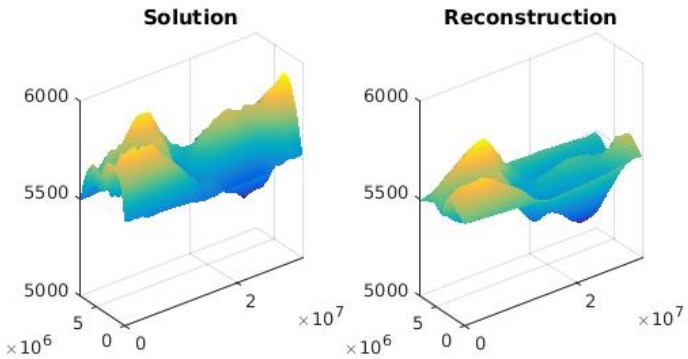


FIGURE 3.6 – Données : 50 iter/ percentInfo = 0.10/ ancien algorithme

Chapitre 4

Conclusion

Pour reconstruire les sous-espaces du modèle, plusieurs méthodes étaient à notre disposition.

Les méthodes de la phase 2 en Fortran se révèlent plus rapide que la méthode en Matlab de la phase 1, mais fait entrer une erreur dans le résultat.

Cependant, en utilisant des échantillons de données plus grand, les calculs restent relativement rapide et proches de la méthode Matlab.

On peut alors décemment sacrifier un peu de précision pour de la vitesse de calcul avec les nouveaux algorithmes, en particulier l'algorithme Explicit $Z^T Z$, qui est le plus rapide.

Pour des grands modèles, il sera plus judicieux de choisir l'algorithme le plus rapide, tandis que sur un petit modèle, la précision des algorithmes de la phase 1 sera valorisée.