ECOLE NORMALE SUPÉRIEURE DE PARIS-SACLAY

Compte-rendu

Projet MPNA : algorithme de Lanczos pour les matrices symétriques

étudiant	Katia MOUALI, Aïcha MAAOUI et Raphaël ATANACKOVIC
UE	Méthodes de programmation numérique avancée
enseignant	Sebastien GOUGEAUD
Date de l'annonce du devoir	25 octobre 2022
Date butoir pour le rendu	04 novembre 2022
Date d'édition du document	04 novembre 2022
adresse de l'organisme	9 Bd d'Alembert, 78280 Guyancourt

Table des matières

Table des figures

Morceaux de code

Liste des tableaux

1 Introduction

Le calcul de valeurs propres est, et ce depuis les débuts du HPC, un problème fondamental. Pour résoudre ce type de problèmes, de nombreuses méthodes itératives ont été développées. Parmis ces méthodes, il y a celle de Lanczos : s'adressant aux matrices symétriques, elle constitue une amélioration de la méthode des puissances et permet de trouver les valeurs propres de plus grande valeur absolue. L'objectif du présent travail est d'implémenter cet algorithme avec deux contraintes : utliser des fonctions optimisées (BLAS/LAPACK) et paralléliser le code avec du MPI.

2 Version séquentielle de l'implémentation

Dans un premier temps l'algorithme a été implémenté de manière séquentielle dans le but d'arriver rapidement à un algorithme fonctionnel et d'avoir un comparant pour la future version parallèle.

2.1 Algorithme implémenté

Il existe deux versions très répandues de l'algorithme de Lanczos (Ref. [1]) : une première version réservée aux cas symétriques (Ref.1a) et une version généralisée aux cas quelconques (Ref.1b). Seule la première sera implémentée. Pour tout le reste de ce travail, $n \in \mathbb{N}$ et $m \in \mathbb{N}$ désigneront respectivement la taille de la matrice carrée A et des vecteurs v_j et w_j ainsi que le nombre d'étapes de la projection. On notera également que tous les réels sont $64 - bits$ et que les matrices sont stockées en *rowmajor*.

Algorithme 8 – Lanczos

1. Choisir un vecteur initial v_1 de norme unité. Poser $\beta_1 \leftarrow 0$, $v_0 \leftarrow 0$
2. For $j = 1,2,\ldots,m$ Do:
3. $\quad w_j := Av_j - \beta_j v_{j-1}$
4. $\quad \alpha_j := (w_j, v_j)$
5. $\quad w_j := w_j - \alpha_j v_j$
6. $\quad \beta_{j+1} := \|w_j\|_2$. Si $\beta_{j+1} == 0$ Arrêt
7. $\quad v_{j+1} := w_j/\beta_{j+1}$
8. EndDo

(a) Algorithme de Lanczos classique

Algorithme 9 – Procédure de biorthogonalisation de Lanczos

1. Choisir v_1, w_1 tel que $(v_1, w_1) = 1$. Set $\beta_1 = \delta_1 \leftarrow 0$, $w_0 = v_0 \leftarrow 0$
2. For $j = 1,2,\ldots,m$ Do:
3. $\quad \alpha_j = (Av_j, w_j)$
4. $\quad \hat{v}_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}$
5. $\quad \hat{w}_{j+1} = A^H w_j - \overline{\alpha}_j w_j - \delta_j w_{j-1}$
6. $\quad \delta_{j+1} = \left|(\hat{v}_{j+1}, \hat{w}_{j+1})\right|^{1/2}$. Si $\delta_{j+1} = 0$ Stop
7. $\quad \beta_{j+1} = (\hat{v}_{j+1}, \hat{w}_{j+1})/\delta_{j+1}$
8. $\quad w_{j+1} = \hat{w}_{j+1}/\overline{\beta}_{j+1}$
9. $\quad v_{j+1} = \hat{v}_{j+1}/\delta_{j+1}$
10. EndDo

(b) algorithme de Lanczos généralisé

FIGURE 1 – différents algorithmes de Lanczos

On initialise les variables comme indiqué dans (Ref.1a). On stock les valeurs des α_j et des β_j dans des vecteurs de taille m. Il en va de même pour les vecteurs w_j et v_j qui sont stockés dans

des vecteurs (matrices linéarisées) de taille $(m+1) \times n$ soit les m vecteurs des étapes 1 à m et le vecteur de l'étape 0. La matrice A est également stockée dans un vecteur de taille $n \times n$.

2.2 Implementation d'une itération de l'algorithme de Lanczos

On se place à l'itération j. Pour écrire la première opération on utilise *cblas_ dgemv*. Cependant il vient un problème : on doit stocker le résultat dans w_j, cependant w_j est nul. Pour respecter le prototype de notre fonction, il faut donc copier v_j dans w_j. Pour ce faire on utilise la fonction *cblas_ dcopy*, on copie donc les n éléments de v_{j-1} vers w_j. Une fois cela fait, il n'y a plus qu'a procéder à l'opération d'axpy en prenant bien garde à ce que les pointeurs demandés pour nos deux vecteurs soient à l'adresse de l'élément n° $j \times n$. Il faut ensuite calculer la valeur de α_j à l'aide d'un produit scalaire entre w_j et v_j qui est réalisé dans le code à l'aide de *cblas_ ddot*. La valeur α_j est ensuite réutilisée pour recalculer w_j à l'aide de *cblas_ daxpy*. Enfin, β_j, la norme de w_j est calculée à l'aide de *cblas_ dnorm2*. Pour la dernière opération, réalisée uniquement si $\beta \neq 0$, on stock $1/\beta$ dans une variable et on copie w_j dans v_{j+1}. Une fois cela fait, on réalise le produit scalaire-vecteur via l'opération *cblas_ dscal*. On note que la variable stockant l'inverse de β sert exclusivement à respecter le prototype de la fonction

2.3 Autres fonctions

Différentes fonctions ont été réalisées, elles sont divisées en deux types : les fonctions d'affichage qui servent à vérifier les résultats et les fonctions d'initialisation qui permettent la génération de matrices et de vecteurs pour tester l'algorithme.

fonctions d'affichage	
printvec	affiche le vecteur de taille n
printmat	affiche la matrice de taille $m \times n$ si elle est stockée en *rowmajor*
fonctions d'initialisation	
A1, *A3*, *A9*, *AMn*	génère une matrice de tests parmis celles du document fourni de taille n stockée en *rowmajor*
randsym	génère une matrice symétrique aléatoire de taille n

TABLE 1 – Tableau récapitulant les fonctions utilisées

Références

[1] Yousef SAAD Bernard PHILIPPE. Calcul des valeurs propres. *Techniques de l'ingénieur*, 2008.