MATIERE

METHODE ET PROGRAMMATION NUMERIQUES AVANCEES

PROGRAMMATION DE L'ALGORITHME D'ARNOLDI

Responsable de formation :
Mme. Nahid Imad

Etudiante
AICHA LAMMAMRA

2020/2021

Table des matières

Table des figures

Liste des tableaux

I. Projection d'Arnoldi

Cette méthode de projection d'une matrice *A* de taille *n*n* sur un sous-espace de Krylov. Elle est utilisée par certaines méthodes pour calculer de manière itérative une bonne approximation d'un certain nombre de valeurs propres.

Dans un sous-espace de Krylov, $K_m(A,v)$ permet d'obtenir une matrice de Hessenberg supérieure H_m de taille *m*. Ainsi Les valeurs propres d'H_m sont les approximations des valeurs propres correspondantes de *A* et la qualité de cette approximation augmente avec *m* mais l'augmentation de la taille de H_m amplifie en même temps le coût des calculs en (Om^2) et la taille mémoire nécessaire (m vecteurs de taille *n*, en plus de la matrice H_m)..

II. Mécanismes de l'itération d'Arnoldi

Une réduction complète de A en forme de Hessenberg par une transformation de similarité orthogonale pourrait s'écrire $A = QHQ^*$, ou $AQ = HQ$. Cependant, en traitant des méthodes itératives, nous considérons que *m* est énorme ou infini, de sorte qu'il est hors de question de calculer la réduction complète. À la place, nous considérons les n premières colonnes de $AQ = HQ$. Soit Q_n la matrice *m* x *n* dont les colonnes sont les *n* premières colonnes de *Q* :

$$Q_n = \left[\begin{array}{c|c|c|c} q_1 & q_2 & \cdots & q_n \end{array}\right]. \tag{1}$$

Soit $\tilde{H}_n$ ((*n* + 1) x *n*) être la partie supérieure gauche de *H*, qui est aussi une matrice de Hessenberg:

$$\tilde{H}_n = \begin{bmatrix} h_{11} & \cdots & h_{1n} \\ h_{21} & h_{22} & \\ & \ddots & \ddots & \vdots \\ & & h_{n,n-1} & h_{nn} \\ & & & h_{n+1,n} \end{bmatrix}. \tag{2}$$

Ensuite nous avons :

$$AQ_n = Q_{n+1}\tilde{H}_n \tag{3}$$

$$\begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix} \left[\begin{array}{c|c|c} q_1 & \cdots & q_n \end{array}\right] = \left[\begin{array}{c|c|c} q_1 & \cdots & q_{n+1} \end{array}\right] \begin{bmatrix} h_{11} & \cdots & h_{1n} \\ h_{21} & & \vdots \\ & \ddots & \\ & & h_{n+1,n} \end{bmatrix}.$$

La nième colonne de cette équation peut s'écrire comme suit :

$$Aq_n = h_{1n}q_1 + \cdots + h_{nn}q_n + h_{n+1,n}q_{n+1} \qquad (4)$$

L'itération d'Arnoldi est simplement l'itération de Gram-Schmidt modifié qui implémenté (4) :

Reduction d'Arnoldi ARB:

- Choisi un vecteur $v_1$de norme 1 ;
- Pour $K = 1, \ldots\ldots, n$
  - Calculer $h_{ik} = (Av_k, v_i)$ pour $i = 1, \ldots\ldots, n$
  - Calculer $w_k = Av_k - \sum_{i-1}^{k} h_{ik}\ v_i$
  - $h_{k+1,K} = \|W_k\|\ .Si\ \ h_{k+1,K} = 0, Arrêt.$
  - $v_{k+1} = \frac{w_k}{h_{k+1,k}}$
- Fin Pour

A chaque itération, l'algorithme sauvegarde le vecteur résultant v, dans la matrice globale Q_{m+1} = [v1, v2…….vm+1].

III. Complexité en mémoire

- Matrice *A* : $O(n^2)$
- Matrice *Q* : $O(mn)$
- Matrice H_m : $O(m^2)$
- Vecteur stocké :$O(2m)$

IV. Évaluation de performance

Pour l'évaluation de performance j'ai choisi de calculer le temps d'exécution du programme en utilisant la bibliothèque «time.h».

Résultat séquentiel

La courbe suivante représente l'évolution du temps de calcul avec l'augmentation de la taille des données (matrice A).

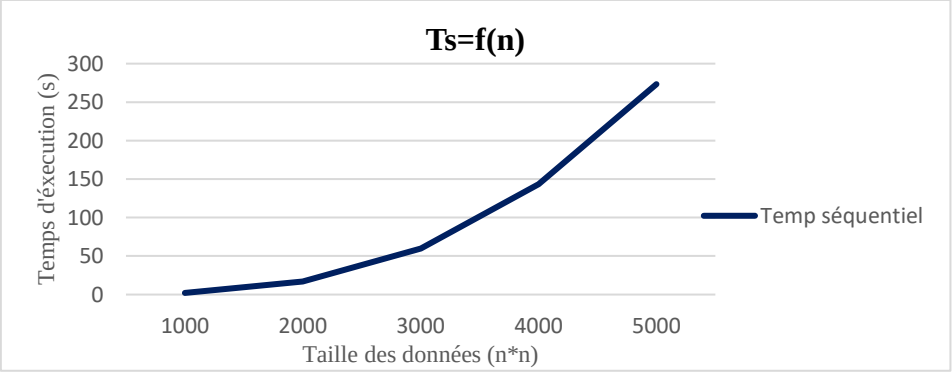

Figure 1.Temps d'exécution de d la méthode d'Arnoldi en fonction de la taille de la matrice

J'ai remarqué l'évolution du temps de calcul avec l'augmentation de la taille de la matrice, je peux aussi constater qu'il y a une forte augmentation du temps de calcul puisqu'à des tailles 5000*5000. je commence déjà à avoir des temps de calcul aux alentours de 4 min.

Résultat parallèle

Afin de passer à la version parallèle, nous devons d'abord connaitre les fonctions les plus coûteuses en termes du temps de calcul et cela par la mesure de temps d'exécution de ces dernières comme représenté dans le tableau ci-dessous.

Tableau 1.Temps d'exécution des différentes fonctions dans la fonction Arnoldi

Taille des données (2000*2000)	Méthode d'Arnoldi	dot	prod_scalair	norm
Temps d'exécution (s)	16.82	13,34	1,15	0 .83
Temps d'exécution total		79% du temps d'exécution de la fonction d'Arnoldi	6% du temps d'exécution de la fonction d'Arnoldi	4% du temps d'exécution de la fonction d'Arnoldi

Comme vous pouvez le voir, ces trois fonctions représentent environ 98% de tous les travaux du programme. Dans ce qui suit, je me focaliserai sur la parallélisation de ces trois fonctions. Le profilage a été réalisé avec de taille 2000*2000 .

Concernant la version parallèle de la méthode, j'ai essayé d'implémenter une architecture de mémoire partagée, cette implémentation n'utilise que des threads (OpenMp), elle est basée principalement sur l'implémentation séquentielle (présentée précédemment). La seule différence c'est que les boucles principales ont été adaptées pour tirer les bénéfices du multithreading, et augmenter ainsi la performance de mon programme.

Pour calculer le temps d'exécution de programme parallèle, j'ai réalisé mes tests de performances sur une machine d'un 1 nœud et de 16 cœurs.

La figure ci-dessous représente les résultats du temps d'exécution de mon programme en variant la taille de la matrice A entre 1000 et 5000 éléments par dimension.

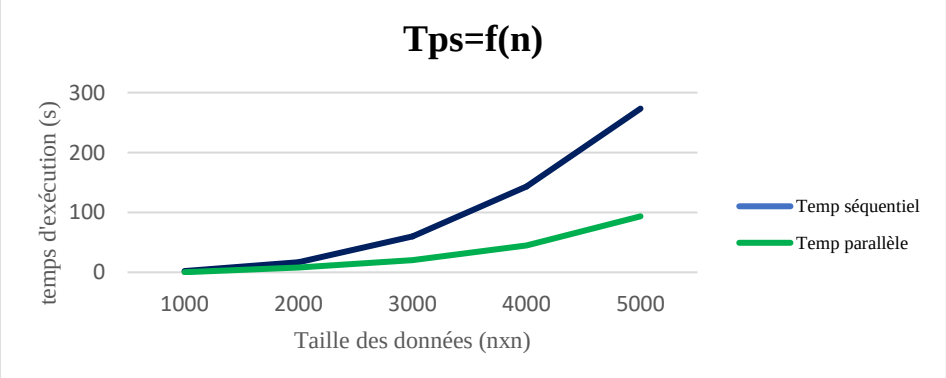

Figure 2. Temps d'exécution de deux versions en fonction de taille de la matrice

Je remarque que la version parallèle est plus rapide par rapport à la version séquentielle que j'ai réalisée. Ainsi, plus la taille de la matrice augmente plus elle va plus vite.

Speed up

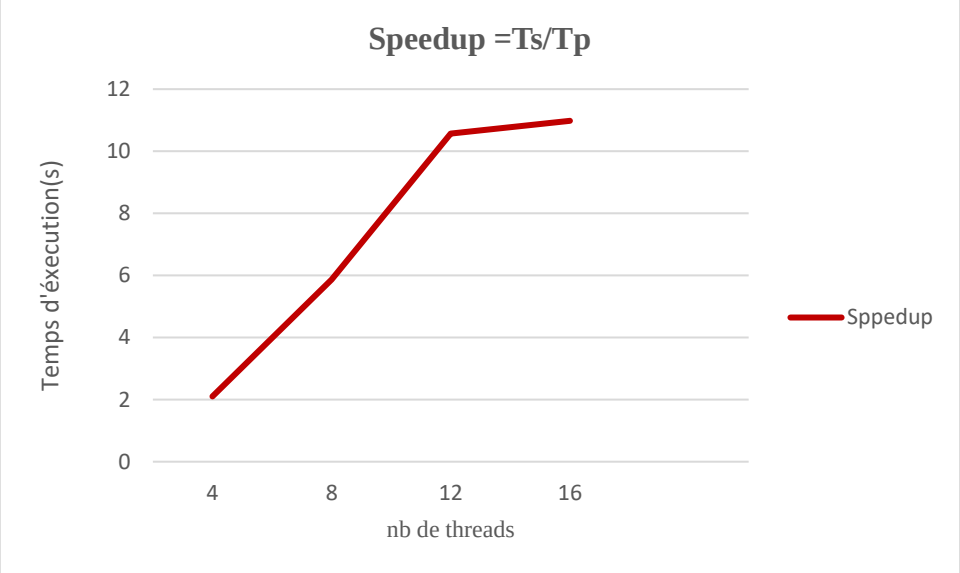

Figure 3. Speedup en fonction du nombre de threads

D'après la courbe du speedup j'ai remarqué qu'il y'a un gain important, cette augmentation se stabilise à partir de 12 cœurs de calculs, à partir de ce résultat on aura la valeur maximale du speed up qui vaut 5.

V. Conclusion

L'implémentation de versions séquentielles, m'a permis de comprendre le fonctionnement et d'avoir une idée sur la complexité de la fonction. Cela me permettra de choisir la plus apte à la parallélisation dans le but de réduire le temps d'exécution et traiter des problèmes de taille importante.

La version parallèle nous a permis d'avoir des temps d'exécutions réduits, nous pouvons encore les améliorer en parallélisant encore plus le code.