UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

Groupe N° 3

SUJET : Méthode de Lanczos

ENSEIGNANTS :

Mme NAHID EMAD

M. THOMAS DUFAUD

Participants :

AMINA Glenn Stevy

TCHABOU Vanessa

Année 2019/2020

SOMMAIRE

Liste des tableaux

Liste des figures

PREMEIRE PARTIE

I. ETUDE DE L'ALGORITME D'Arnoldi

1. *ALGORITHME D'ARNOLDI*

Initialisation : $q_1 = q / ||q||$

For j = 1,…, m do

W = Aq_j

For i = 1,…,j do

$h_{ij} = (w,q_i)$

$w = w - h_{ij}q_i$

end for

$h_{j+1.j} = ||w||_2$. If $h_{j+1.j} = 0$ then stop.

$q_{j+1} = w / h_{j+1.j}$

end for

2. *DEFINITION ET DESCRIPTION*

L'algorithme d'Arnoldi consiste à calculer récursivement la mise de A sous la représentation unitairement semblable W, W de type Heisenberg, soit $A = QWQ^T$. Il est mathématiquement équivalent à la factorisation QR de la matrice.

3. COMPLEXITE THEORIQUE

- Complexité en temps : c'est le temps que prends l'exécution de toutes les opérations arithmétiques. Elle est sensiblement égale à $0(n^2)$.
- Complexité en mémoire : c'est le coût de stockage des différentes variables. Elle est de $8(n*n + m*n + m^2)$.

4. *COMPLEXITE PRATIQUE*

- Les mesures ont été faites en séquentiel sur un processeur Intel(R) Core(TM) i5-8265u CPU @ 1.60GHz 1.80 GHz.
- Les mesures de temps sont répétées 5 fois et on prend la moyenne.
- Le temps de calcul mesuré est la durée de l'exécution de la méthode de d'Arnoldi pour le calcul des valeurs propres.

a. TEMPS D'EXECUTION EN FONCTION DE LA TAILLE DE LA MATRICE

Ici, l'augmentation du temps d'exécution en de l'algorithme avec la taille de la matrice est encore vu comme la **scalabilité faible.** Nous l'étudierons en profondeur à la troisième partie du document

Taille de la matrice	2	4	8	16	32	64	128	256
Tps d'exécution en us	6	30	68	943	6068	55614	58568	61392

Tableau 1: temps d'exécution en fonction de la taille de la matrice

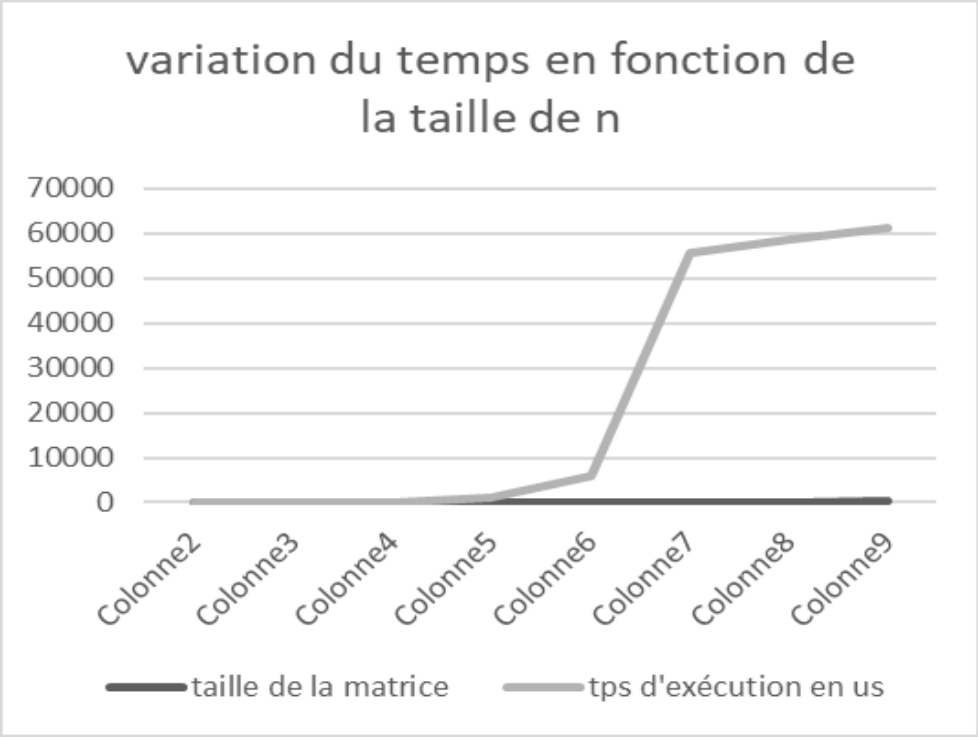

Figure 1: scalabilité faible de l'algorithme d'Arnoldi

b. INTERPRETATION

Nous constatons que le temps d'exécution augmente avec le nombre d'itération. Ce qui est normal car l'exécution s'effectue sur un seul processus.

DEUXIEME PARTIE

I. INTRODUCTION

L'algorithme de Lanczos est un algorithme itératif permettent de calculer les valeurs et vecteurs propres d'une matrice creuse, symétrique, carré. Il est très utilisé pour la décomposition de grandes matrices en météorologie et en traitement des langues naturelles car dans ces domaines les matrices traitées possèdent de milliers de termes. L'objectif de notre projet est d'implémenter la version séquentielle de cet algorithme et de le paralléliser par la suite. Nous espérions que notre implémentation de l'algorithme de Lanczos prendra au moins une matrice de 10000 termes.

II. PROBLEMATIQUE

Le calcul des valeurs et vecteurs propres d'une matrice en mathématique est très simple lorsque la matrice est de petite taille. Même à la main lorsque la matrice est de 4 termes, 9 termes par exemples. Cependant lorsqu'il s'agit d'une matrice avec une taille allant à de centaine de milliers de termes il devient très difficile. Le problème est donc énorme.

III. APPROCHE UTILISEE

Lanczos permet de réduire la dimension du problème avant de poursuivre par une méthode générale, QR proposes de types sous-espace appliquées au cas de matrices symétrique réelles. Elle permet de construire itérativement, colonne par colonne, une matrice tri diagonale de plus petite taille (matrice de Rayleigh) dont les éléments propres sont des approximations d'un sous-ensemble d'éléments propres de la matrice A. On obtient les valeurs propres de plus grand module par itération directe, et comme pour la méthode de la puissance, celles de plus petit module par itération inverse. Elle s'interprète en fait comme une méthode de projection.

IV. CAS SEQUENTIEL

1. DESCRIPTION DE L'ALGORITHME PROPOSE

On projette le problème $AV = \lambda V$ dans un sous-espace de dimension $m << N$, selon ($AV - \lambda V$, qi) = 0 pour $i = 1...m$. Voici l'algorithme de Lanczos pour la recherche des m plus grandes valeurs propres et des vecteurs propres associés d'une matrice A symétrique réelle $N \times N$:

Soit $q_0 = 0$ et $q_1 \in IR_N$ tel que (q_1(2 = 1 on calcul successivement pour k = 1, 2… m-1

- $W_k = Aq_k - \beta_{k-1}q_{k-1}$: itération directe ;
- $A_k = (w_k, q_k)$: produit scalaire ;
- $V_k = w_k - \alpha_k q_k$: orthogonalisation ;
- $\beta k = ||v_k||_2$: Calcul de la norme euclidienne ;
- $q_{k+1} = v_k / = v_k/\beta_k$: Normalisation .

On obtient ainsi m vecteurs orthonormés q_i et une matrice T tridiagonale symétrique m x m (matrice de Rayleigh) constituée d'éléments diagonaux $T_{i,i} = \alpha_i$ et extra diagonaux $T_{i,i+1} = T_{i+1} = T_{i,i+1,i} = \beta_i$. Les valeurs propres et vecteurs propres de T se calculent par la méthode QR. On obtient les m plus grandes valeurs propres de A et les composants des vecteurs propres de A dans l'espace q_i. Les erreurs d'arrondis produisent, comme pour le gradient conjugué, des défauts d'orthogonalité. Il est nécessaire de réorthogonaliser le temps en temps les q_i.

2. VALIDATION DE L'IMPLEMENTATION

Pour la validation des implémentations, nous avons vérifie que les vecteurs Q générés sont bien orthogonaux pour cela, on fait : Q*Qt = Id.

3. EDP THORIQUE

a. NOTION DE COMPLEXITE

Les algorithmes de calcul sont caractérisés par leur complexité arithmétique, mesurée par le nombre d'opérations (additions, multiplications…) sur des réels, ainsi que par leur coût de stockage, mesure par le nombre de variables réelles.

- Complexité en temps : $O(nm^2)$
- Complexité en mémoire : O(n)

b. NOTION DE PERFORMANCE

Les performances d'un algorithme se mesurent par sa vitesse de calcul. Celle-ci dépend de la complexité mais aussi de l'exploitation de l'architecture de l'ordinateur, notamment du parallélisme interne et de la hiérarchie des mémoires.

4. EDP PRATIQUE

a. METHODE DE MESURE

- Les mesures ont été faites en séquentiel sur un processeur Intel(R) Core(TM) i5-8265u CPU @ 1.60GHz 1.80 GHz.
- Les mesures de temps sont répétées 5 fois et on prend la moyenne.
- Le temps de calcul mesuré est la durée de l'exécution de la méthode de Lanczos pour le calcul des valeurs propres.

TEMPS DE MESURE

Nbre d'itération	Temps d'exécution en US
2	2.4
4	6.4
8	14.8
16	82
32	522.6
64	634,4
128	6344

Tableau 2: temps d'exécution de l'algorithme d'Arnoldi en fonction de la taille de la matrice

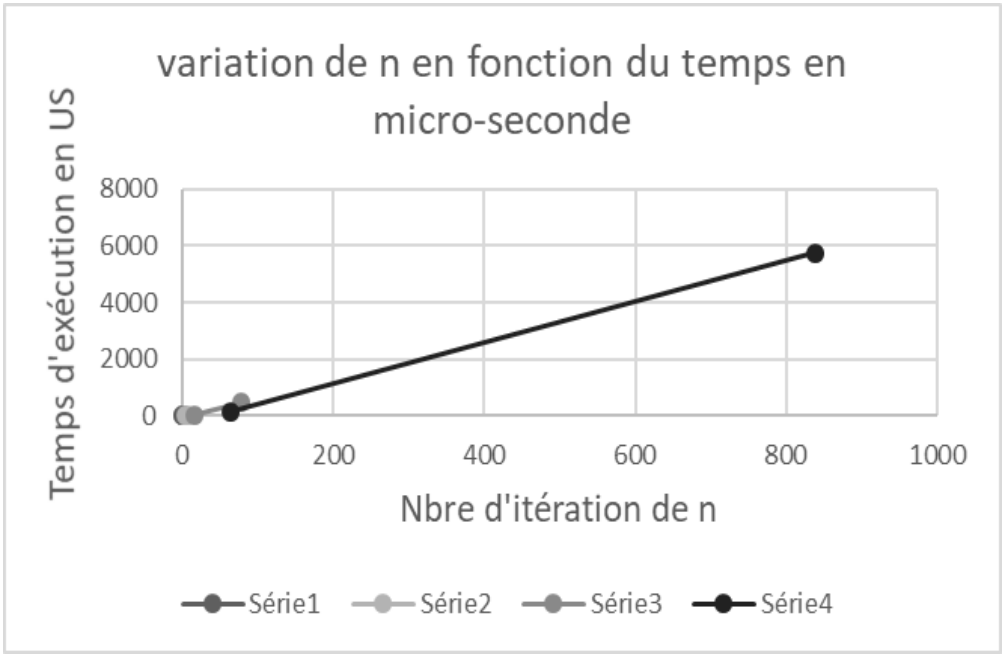

Figure 2: scalabilité faible de l'algorithme de Lanczos

b. INTERPRETATION

Nous constatons que le temps d'exécution augmente avec le nombre d'itération. Ce qui est normal car l'exécution s'effectue sur un seul processus. Et nous souhaiterons améliorer celui-ci lors de la Parallélisation.

5. CONCLUSION

En somme il était question dans cette partie d'étudier la version séquentielle de l'algorithme de Lanczos. Après l'avoir fait nous constatons que le temps d'exécution est énorme et nous ne pouvons pas aller à plus de 128 itérations ce qui est limité. Nous proposons d'étudier maintenant la version parallèle afin de palier à tous ces manquements.

```
amina@amina:/mnt/c/Users/hp/Desktop/hpcs/M2/Methode_de_programmation_numq/Projet/seq$ gcc lanczos.c -lm
amina@amina:/mnt/c/Users/hp/Desktop/hpcs/M2/Methode_de_programmation_numq/Projet/seq$ ./a.out
Méthode de Lanczos
Matrice A
  1.000000000e+01  1.300000000e+01  1.600000000e+01  1.900000000e+01
  1.300000000e+01  2.300000000e+01  2.600000000e+01  2.900000000e+01
  1.600000000e+01  2.600000000e+01  3.600000000e+01  3.900000000e+01
  1.900000000e+01  2.900000000e+01  3.900000000e+01  4.900000000e+01
Matrice Q
  2.777777778e-01 -1.988229726e-01  1.975522382e-01 -6.413300924e-03
  4.166666667e-01  1.242643579e-01  5.005748603e-01  4.976745250e-01
  5.555555556e-01 -7.455861472e-02  5.492546299e-01  3.161738436e-01
  6.944444444e-01 -9.692619914e-01  6.393099984e-01  8.076589858e-01
Matrice T
  1.136959877e+02  5.588444316e+00  0.000000000e+00  0.000000000e+00
  5.588444316e+00  2.982295167e+00  1.801994819e+01  0.000000000e+00
  0.000000000e+00  1.801994819e+01  1.168399011e+01  3.691444882e+01
  0.000000000e+00  0.000000000e+00  3.691444882e+01  2.082178201e+01
```

6. COMPARAISON DES VERSION SEQUENTIELLES L'ALGORITHME DE LANCZOS ET ARNOLDI

Cas particulier de la méthode d'Arnoldi quand la matrice est symétrique. La complexité d'ARNOLDI est de $O(nm^3)$ et pour Lanczos $O(nm^2)$. La méthode Lanczos utilise moins de mémoire que celle d'Arnoldi. Similaire aux méthodes classique de Gram-Schmidt, nous pouvons avoir une perte d'orthogonalité : La méthode de Lanczos est plus sensible aux erreurs arrondis que la méthode d'Arnoldi.

TROISIEME PARTIE

I. CAS PARALLELE

1. DESCRIPTION

Dans cette partie, nous avons parallélisé le code séquentiel. Chaque processus possède la grande matrice initiale mais lit une partie des colonnes selon le rang.

Nous avons parallélisé le produit matrice vecteur. Chaque processus prend deux colonnes de la grande matrice et fait un produit scalaire ; Puis envoie le résultat au processus 0. Le processus 0 continue le reste de calculs et fait une diffusion pour partager les résultats nécessaires aux autres processus pour faire les calculs.

Nous travaillons avec un nombre de processus égal à la taille de la matrice divisée par 2.

2. EDP

Pour cette étude de performance, nous travaillons dans les mêmes conditions opératoires que la partie séquentielle. Nous allons ainsi étudier la scalabilité et le speed up.

a. Etude de la scalabilité

Il existe deux types de scalabilité : la ***scalabilité forte*** qui consiste à fixer la taille du problème (ici nous fixons la taille de la matrice) et varier le nombre de processus et la ***scalabilité faible*** qui consiste à augmenter la taille du problème proportionnellement au nombre de processus.

Dans notre cas, Pour la scalabilité, nous augmentons la taille du problème avec le nombre de processus, et on mesure le temps d'exécution.

Nbre de processus	**2**	**4**	**8**	**16**	**32**
Taille de la	2	4	8	16	32

matrice					
Temps d’	258	509	570	5451	6924

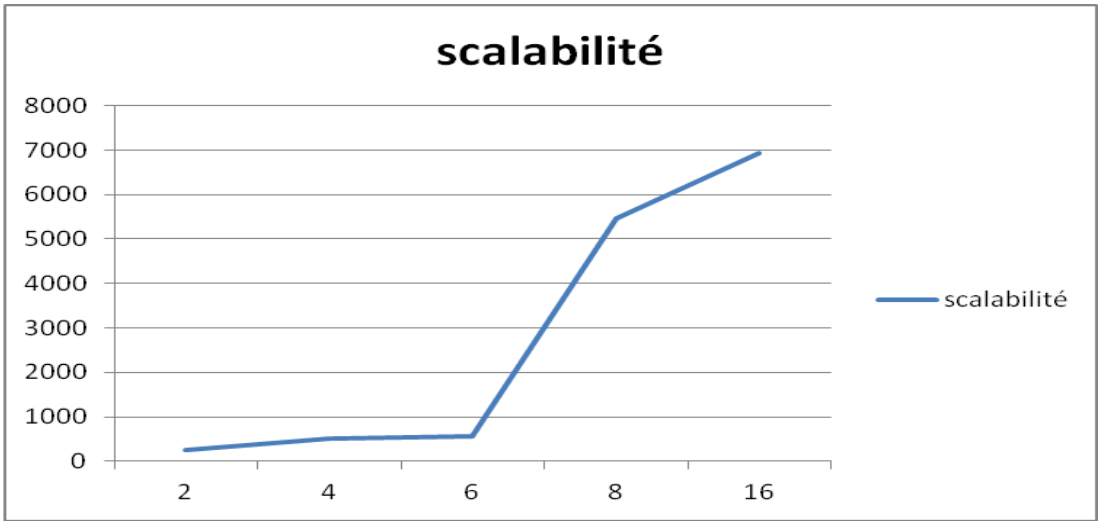

Figure 3: scalabilité de l'implémentation parallèle de la méthode de Lanczos

- Sur la figure ci-dessus, en l’axe des abscisses représente le nombre de processus
- L’axe des ordonnées représentes le temps d’exécution

b. Interprétation

La courbe croit avec le nombre de processus, on peut dire que la scalabilité semble être faible.

a. Etude du speed up

Le **speed up** est défini par la formule suivante : **Sp= *T* 1/*Tp***
où **p** est le nombre de processus, **T1** le temps d’exécution séquentiel et **Tp** le temps d’exécution parallèle avec p processus. Le tableau suivant est obtenu en calculant le speed up à partir des données recueillies.

Processus	Taille	Temps en us	Temps seq/temps parallèle (speed_up)	Temps seq
2	4	258	0,04263566	11
4	8	509	0,03536346	18
6	12	570	0,64035088	365
8	16	5451	0,01999633	109
16	32	6924	0,07293472	505

Tableau 5: speed up

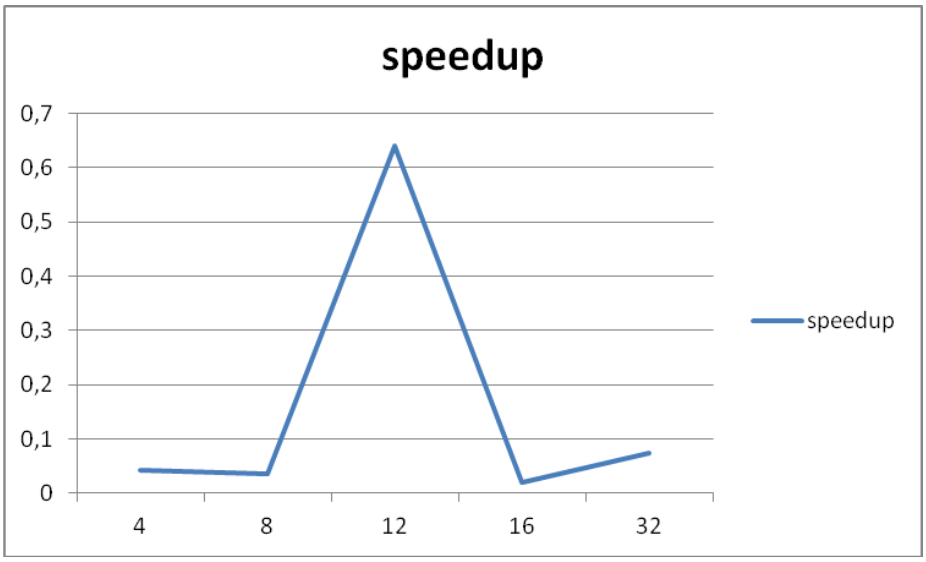

Figure 4: speed up

- Sur la figure ci-dessus, nous avons le nombre de processus en abscisse
- En ordonnées nous avons le speed-up

b. Interprétation

On remarque ici que la courbe du speedup à plusieurs variations, sur l'interval 8-12, elle est croissante. C'est peut être du au fait que nos matrices testées ne sont pas très grandes pour bien voir l'apport du code parallèle.

Dans notre cas, il serait peut être mieux de travailler avec une matrice allant de 8 à 12 si on veut un gain en performance.

3. CONCLUSION

Dans cette seconde étude partie, nous avons proposé une implémentation parallèle du code séquentiel. Et nous avons étudié son efficacité. Un compromis reste à faire entre l'espace et le temps de communication. Nos tests ont été limités car nous avons travaillé avec des matrices relativement faibles et le nombre de processus aussi limité.

BIBLIOGRAPHIE

- Rapport de recherche sur les Solveurs numériques pour l'étude des états frustrés de la matière. ENS d'Informatique et de mathématique appliqués GRENOBLE ;
- Wikipédia ;
- Pdf Chapitre III : Méthodes numériques de base.

Dépôt git : https://github.com/GLENNAMINEM/M-thode-de-Lanczos