UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

Méthodes et Programmation Numériques Avancées
Rendu Tps
Master 2 : Calcul Haute Performance, Simulation

Nom: Lougani
Prénom: Faouzi
Numero etudiant:22003152

Avant propos : *Dans le cadre de ce module les différentes fonctions ainsi que leurs codes C,scripts pythons ,graphes sont disponible sur mon dépôt github :*https://github.com/lougani-faouzi/mpna

Avant de commencer nos implémentations de d algorithmes des fonctions basiques ont été mises en place comme :

- La lecture de matrice à partir d'un fichier mtx
- Création de matrice et vecteur
- Affichage de matrice et vecteurs
- Création des différentes fonctions blas 1,2
- Calcul de la norme et de la norme frobinus

```
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Téléch

 ***TP01 *****

lecture matrice normale
2.000000 1.000000 0.000000 0.000000 0.000000
1.000000 2.000000 1.000000 0.000000 0.000000
0.000000 1.000000 2.000000 1.000000 0.000000
0.000000 0.000000 1.000000 2.000000 1.000000
0.000000 0.000000 0.000000 1.000000 2.000000


lecture vecteur
1.000000
1.000000
1.000000
1.000000

 simple dot prod de vecteur
0.000000
 norme d un vecteur
0.000000
 norme frobenieus
4.123106

lecture et affchage d'une matrice format mtx
0.300111 0.000000 0.000000 0.000000 0.000000
0.000000 0.200074 0.000000 0.000000 0.000000
0.000000 0.000000 0.244371 0.000000 0.000000
0.000000 0.000000 0.000000 0.183278 0.000000
0.000000 0.000000 0.000000 0.000000 0.272241


 ***TP01 *****
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Téléch
```

```c
void scal_fois_vect(int n, double scal, double *vect) {
  for (int i = 0; i < n; i++) {
    vect[i]= scal * vect[i];
  }
}

void copier_vect_vect(int n, double *vect_a, double *vect_b) {
  for (int i = 0; i < n; i++) {
    vect_b[i]=vect_a[i];
  }
}

void scal_fois_vect_plus_vect(int n, double scal, double *vect_a, double *vect_b) {
  for (int i = 0; i < n; i++) {
    vect_b[i] = scal * vect_a[i] + vect_b[i];
  }
}

double somme_mul_vect_vect(int n, double *vect_a, double *vect_b) {

  double somme = 0.0;
  for (int i = 0; i < n; i++) {
    somme = somme + vect_a[i] * vect_b[i];
  }
  return somme;
}
```

l’algorithme de projection d’Arnoldi:

La méthode d’arnoldi est une méthode de projection de type Krylov qui permet de résoudre un problème important dans le calcule scientifique.

L'algorithme de projection produit une séquence de vecteurs orthonormés, appelés vecteurs d’arnoldi et prend en paramètre :

void arnoldi (float **A, float *v, int n, int m, float **H, float **V) :

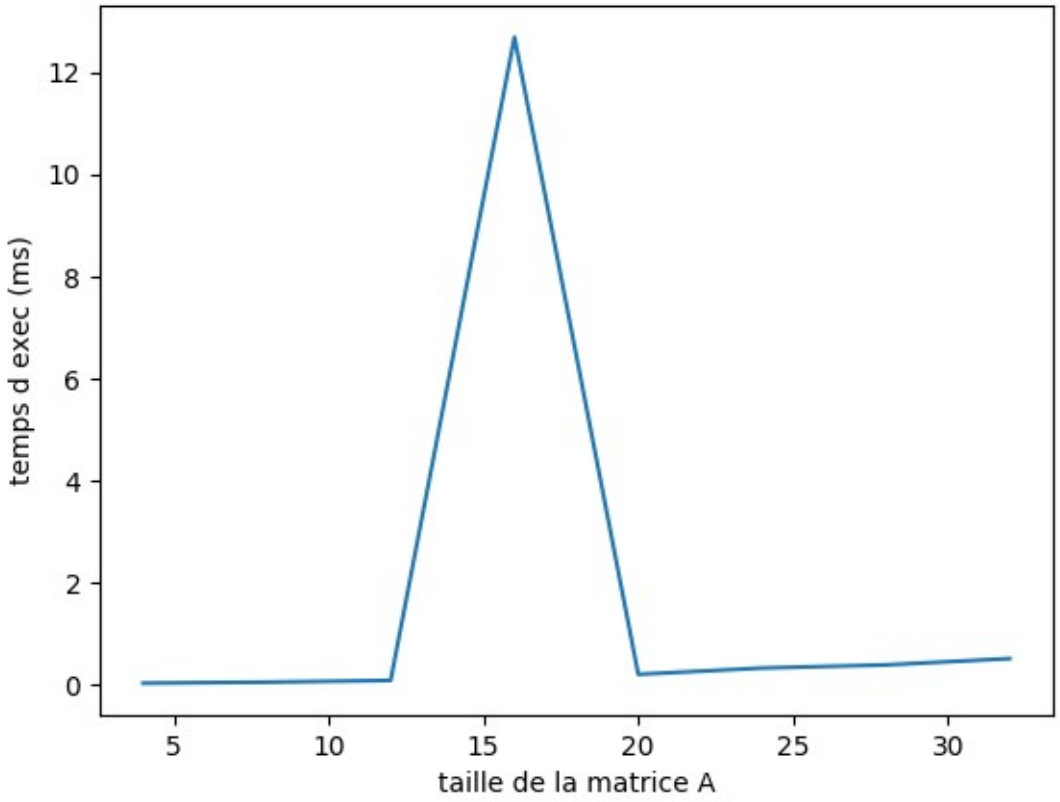
A la matrice
H la matrice hessenberg
v le vecteur initial
n la taille de la matrice A
m la taille du sous espace de projection
V la matrice unitaire

```
 ***TP02 arnoldi *****
La matrice A :
1.000000        1.000000        1.000000        1.000000
1.000000        2.000000        3.000000        4.000000
1.000000        3.000000        5.000000        7.000000
1.000000        4.000000        7.000000        10.000000

Le vecteur v :
1.000000        2.000000        3.000000

Si n = 4 et m = 3

La matrice resultante H :
0.000000        0.857143        0.000000
2.138090        0.000000        0.000000
0.000000        3.090473        0.000000


La matrice des vecteurs propres :
0.267261        0.534522        0.801784
1.000000        0.000000        0.000000
0.963624        -0.148250       -0.222375

le temps d'execution est = 5 ms

 ***TP02 arnoldi *****
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Téléchargements/M2IHPS/mpna/
```

Évaluation des performances :

Pour mesurer les performance on a utilisé la fonction ***clock_t clock(void)*** *de la librairie* ***time.h***

Afin de faire notre évaluation on augmente la taille de la matrice A et on mesure le temps d'exécution (en ms) de la fonction arnoldi a chaque fois .



On voit bien que le temps est bon pour les matrice de taille allant de [5,11] et même pour [20,30]. Le temps en général est minimale mème en augmentant la taille de la matrice A. Ce qui explique l'efficacité du procédé de la projection d'arnoldi sur des ***problèmes de grandes taille****.*

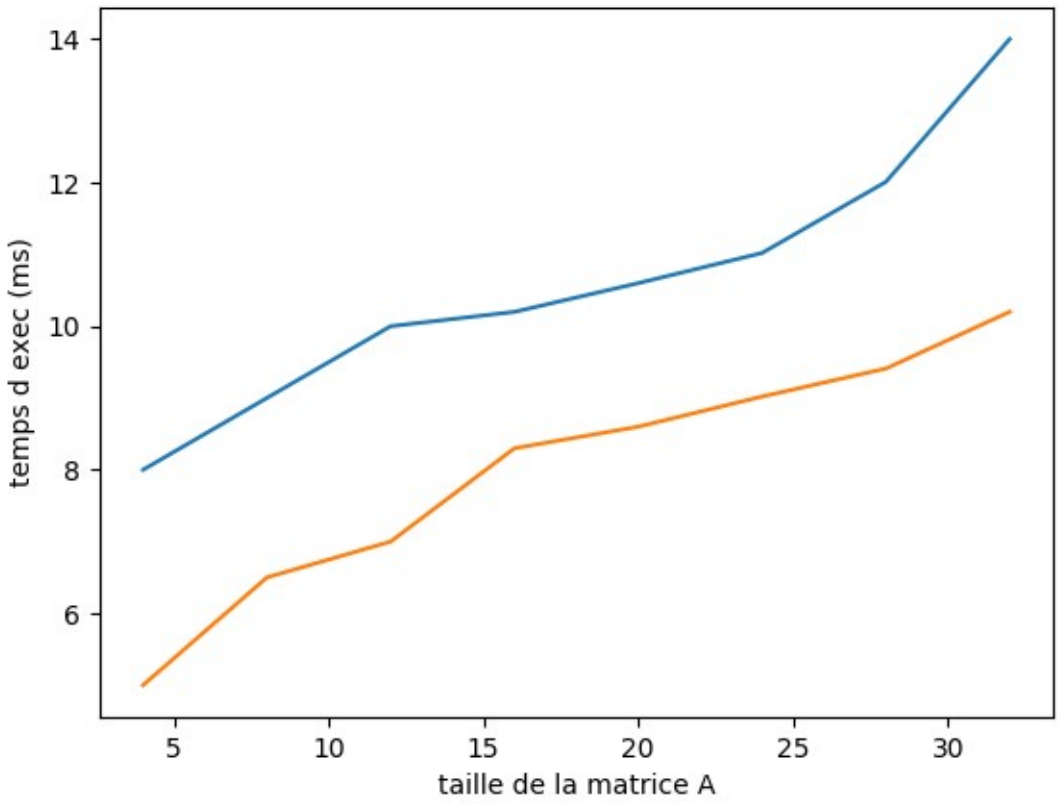
Gram-Schmidt classique (CGS) et Gram-Schmidt modifée (MGS):

On exécute nos implémentations des 2 le procédé pour une mème matrice carrée de taille 3 et on aura le résultat suivant qui justifie déjà l'importance du procédé de MGS.
Le temps d exécution ***CGS =8ms***
Le temps d'exécution ***MGS=5ms***

Le temps d'exécution a presque été devisé sur 2 avec MGS, mais pour voir le vrai impact on fera comme la question précédente, on varie la taille de la matrice A et on évalue le temps d exécution.

```
./main.exe

 ***TP03 gramshmid *****
La matrice en entree m1 :
1.000000 1.000000 -1.000000
0.000000 1.000000 -1.000000
1.000000 1.000000 0.000000

La matrice resultante en classical gs :
0.707107 0.000000 -0.707107
0.000000 1.000000 0.000000
0.707107 0.000000 0.707107

le temps d'execution classical gs est = 8 ms

La matrice resultante en Modified gs :
0.707107 0.000000 -0.707107
0.000000 1.000000 0.000000
0.707107 0.000000 0.707107

le temps d'execution Modified gs est = 5 ms
louganifaouzi@louganifaouzi-ThinkPad-L520:~/Télé
```

On aura le graphe suivant en variant la taille de la matrice A :



*La courbe en orange c'est celle de **MGS**, celle en bleu c'est celle de **CGS.***

Le temps d exécution a diminué en utilisant le procédé MGS.

Version parallèle :

On a essayé d appliquer la parallélisation ***OpenMp*** *sur les boucles de nos algorithmes mais ça ne rapporte pas trop de gains en terme de performances*
Par contre avec ***la vectorisation*** *on gagne encore mieux.*

Le tableau suivant montre le gains sur chaque algorithme en ms:

	O1	O2	O3	Ofast	Vect optim
arnoldi	1.4	1.5	3	4.22	5
CGS	1.62	1.8	3.31	4.6	5.45
MGS	2.01	2.24	4.01	5.5	6.87

On voit bien que le gain en MGS est mieux que les autres .

Conclusion Version parallele:

Nous avons donc montrer qu'un parallélisme de taches peut donner des résultats surtout pour MGS ,mais un parallélisme de données donnera encore de résultats plus intéressants , la version modifiée du processus de Gram-Schmidt est plus parallèle que sa version classique. Ce résultat est vrai seulement dans le cas où dès le début on a les vecteurs à orthonormaliser, ce n'est pas le cas pour le processus d' Arnoldi.