Programmation pour le calcul scientifique
Année : 2017-2018
Formation : L3 Ingénieurie Mathématique

TP7 : DS

A lire attentivement avant de commencer :

Ce TP est noté. L'utilisation de documents tels que des notes de cours est autorisé. Le recopiage de programmes et de manière générale la triche sont interdits et seront sanctionnés.

A la fin de la séance vous devez m'envoyez les programmes que vous aurez écrits par email avec accusé de réception.

Pour garantir une bonne lisibilité, commentez systématiquement vos programmes. Une question sera considérée comme traitée si les intructions correspondantes, mais également l'affichage des résultats de ces instructions, ont été programmés, sont compilables et fonctionnent à l'exécution. Cela signifie que tous les programmes doivent être validés par un test même si ce n'est pas demandé explicitement dans la question.

Si vous avez traité certaines questions mais que le code résultant ne produit pas les résultats escomptés, rajoutez à côté des lignes de code correspondantes des commentaires pour décrire le problème rencontré.

Chaque exercice sera traité dans un programme indépendant. L'exercice 1 sera traité dans un seul fichier. Les deux exercices suivants seront structuré en trois fichiers : un fichier contenant le programme principal, un autre les fonctions associées et un troisième (header) les éventuelles classes et prototypes des fonctions.

1 Pointeurs, références, allocation de mémoire

1. Définir un entier k et un pointeur sur un entier p. Initialiser la valeur de k.
2. Modifier la valeur de k en utilisant le pointeur p.
3. Définir q un autre pointeur sur un entier en lui allouant directement une place mémoire.
4. Attribuer une valeur à cette place mémoire.
5. Faire pointer p vers cette place mémoire, et modifier la valeur contenue dans cette place mémoire en utilisant p.
6. Détruire la place mémoire allouée à q.
7. Rajouter le code ci après à votre programme. Créer un pointeur (de type string) qui pointera sur "Euler" ou "Runge-Kutta" suivant la réponse donnée au clavier.

```cpp
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string repA("Euler"), repB("Runge-Kutta");
  cout << "Quel schéma en temps souhaitez-vous utiliser ? " << endl;
  cout << "A) " << repA << endl;
  cout << "B) " << repB << endl;

  char votre_reponse;
  cout << "Votre reponse (A ou B) : ";
  cin >> votre_reponse; //Récupère la réponse de l'utilisateur
```

2 Multiplication matrice-vecteur

On considère la matrice A_N de la discrétisation du Laplacien en une dimension, avec un pas d'espace $h = 1/N$, pour un nombre total d'inconnues dans le domaine égal à $N - 1$.

1. Ecrire une fonction qui prend en argument un vecteur x de taille $N - 1$ et renvoie un nouveau vecteur qui est le résultat de la multiplication de x par la matrice A. Pour simplifier le programme et économiser de la mémoire, on pourra tirer parti de la structure particulière de la matrice A.
2. Ecrire une fonction qui prend en argument un vecteur x, un entier *nite*, et calcule les *nite* premières itérations de l'algorithme de la puissance itérée appliqué à la matrice A_N et au vecteur x. Cette fonction renvoie un réel λ qui représente l'approximation à la *nite*-ième itération de la valeur propre dominante de la matrice, et modifie le vecteur x afin qu'il prenne comme valeur l'approximation correspondante du vecteur propre associé.
3. Ecrire une fonction qui prend en argument un vecteur x et un réel λ et renvoie la norme max de $A_N x - \lambda x$.
4. Ecrire un programme qui demande à l'utilisateur la valeur de N et un entier *nite* et affiche à l'écran le résultat de l'algorithme de la puissance itérée appliqué à la matrice A_N au bout de *nite* itérations et une estimation de l'erreur correspondante.

Théorème — Soit A une matrice carrée d'ordre n et $(\lambda_1,\lambda_2,...,\lambda_n)$ ses valeurs propres. On suppose :

$$|\lambda_1| > \max(|\lambda_2|, \ldots, |\lambda_n|).$$

On considère la somme directe $\mathbb{C}^n = E \oplus F$ où E est le sous-espace caractéristique de A associé à la valeur propre λ_1, et F est le sous-espace caractéristique de A associé aux autres valeurs propres.

Alors si $w^{(0)} \notin F$, la suite de vecteurs $(w^{(n)})$ définie par la relation de récurrence

$$\forall n \in \mathbb{N},\ w^{(n+1)} = \frac{1}{||Aw^{(n)}||} Aw^{(n)}$$

vérifie

- $w^{(n)} \neq 0$ pour tout $n \in \mathbb{N}$.
- $||Aw^{(n)}|| \longrightarrow |\lambda_1|$ lorsque $n \to \infty$.
- $\left(\frac{\overline{\lambda_1}}{|\lambda_1|}\right)^n w^{(n)} \longrightarrow v$ lorsque $n \to \infty$, où v est un vecteur unitaire de A associé à la valeur propre λ_1.
- Si j est une composante non nulle du vecteur v, alors $\frac{(Aw^{(n)})_j}{w_j^{(n)}} \longrightarrow \lambda_1$ lorsque $n \to \infty$.

FIGURE 1 – *Rappel : Algorithme de la puissance itérée (source : Wikipedia)*

3 Une classe de nombres complexes

1. Dans un fichier header, déclarer une classe `complexe` avec comme données privées deux réels représentant la partie réelle et la partie imaginaire d'un nombre complexe.
2. Définissez le constructeur par défaut pour cette classe.
3. Définissez le constructeur par copie pour cette classe.
4. Définissez un constructeur initialisant une instance de la classe en lui donnant des valeurs reçues en paramètres du constructeur.
5. Définissez le destructeur de la classe.
6. Définissez une fonction membre public de la classe qui prend en argument une autre instance de la classe et qui renvoie le produit des deux instances considérées.
7. Définissez une fonction amie de la classe qui prend en argument deux instances de la classe et qui renvoie leur produit.

4 Makefile

1. Créer un makefile basique pour compiler les programmes des exercices précédent.
2. Créer un makefile avancé (utilisant les variables vues en cours) pour compiler les programmes des exercices précédent. Lors de l'envoi des programmes, vous pourrez sauvegarder ces deux makefiles différents sous les noms 'Makefile1' et 'Makefile2'.