Programmation pour le calcul scientifique
Année : 2019-2020
Formation : L3 Ingénierie Mathématique

TP1 : Rappels

Conseils pratiques

Avant de commencer, voici quelques règles de « bonne conduite » qu'il vaut mieux suivre lorsque vous vous lancez dans l'écriture d'un code. Certains de ces conseils ne sont pas forcément pertinents pour des petits exercices de TP, mais il n'est jamais trop tôt pour prendre de bonnes habitudes.

- Prenez le temps de concevoir dans votre tête ou sur le papier ce que vous allez implémenter. Dans le cadre d'un TP ou d'un examen, cela commence par lire l'intégralité d'un exercice. Vous ne serez pas surpris par une question qui vous obligerait à tout réécrire...
- Testez régulièrement les fonctionnalités que vous implémentez. N'attendez pas la dernière question pour procéder au débogage du code. Celui-ci est d'autant plus complexe qu'un code est long !
- Codez de préférence en anglais. Sauriez-vous maintenir un code écrit et commenté en polonais ou en swahili ?
- Choisissez des noms de fichiers, de variables et de fonctions qui soient compréhensibles et permettent d'identifier facilement ce que vous manipulez.

```
double dn(double d, double e);
// Wrong: what is this function even doing ?

double deltaTNext(double deltaT, double error);
// Better. Now we get the idea.

double computeNextTimeStepFromError(double currentTimeStep, double truncationError);
// No possible doubt on what the function does, but the names are too long.
// It becomes tedious to read and write such names.
```

- Choisissez une convention d'écriture pour vos noms de variables et tenez vous-y. Par exemple la Camel Case comme ci-dessus ou la Snake Case.
- Commentez votre code de manière complète, mais concise. Il s'agit de trouver un bon équilibre entre les noms de variables et les commentaires afin de donner une quantité d'information nécessaire et suffisante à un relecteur.
- Indentez votre code pour mieux repérer les blocs. Choisissez votre façon d'indenter préférée, et là aussi tenez-vous y. Votre code gagnera en cohérence et en lisibilité. Le C++ est un langage qui autorise les retours à la ligne et les espaces additionnels. Utilisez cela à votre avantage en formatant le code pour améliorer sa lisibilité.
- Évitez la duplication de code tant que faire se peut. Dites vous que chaque copier/coller fait au sein d'un même code est source d'erreur lorsqu'il faut modifier toutes les parties qui ont été copiées. Privilégiez l'usage de fonctions pour les éléments semblables de votre programme.

- Enfin n'oubliez pas :

> "*Always code as if the guy who ends up maintaining your code will be a violent psycopath who knows where you live.* "
>
> John Woods

  Cette personne, c'est votre futur collègue avec qui vous allez passer des mois ou des années, c'est vous dans plusieurs années, ou c'est le correcteur de votre TP !

1 Aire d'un triangle

Écrire un programme demandant à l'utilisateur d'entrer les coordonnées 2D de trois points A, B et C et et affichant l'aire du triangle formé par ces points. On pourra utiliser la formule suivante :

$$\mathcal{A} = \frac{1}{2}\left|\overrightarrow{AB} \times \overrightarrow{AC}\right|.$$

2 Arc tangente

Écrire un programme qui prend en entrée deux valeurs réelles x et y données par l'utilisateur et affiche la valeur de l'angle polaire $\theta \in [0, 2\pi)$ du point M de coordonnées (x, y) en utilisant la fonction `atan`. π peut-être obtenu avec la variable `M_PI` définie dans la bibliothèque `cmath`. Vous aurez aussi besoin de la fonction `fmod` qui est l'équivalent de l'opérateur modulo `%` pour les réels.

Note : il s'agit de déterminer dans quel quadrant se trouve M. À l'avenir, il vaudra mieux utiliser la fonction `atan2` (qui est cependant plus adaptée à un usage en coordonnées sphériques).

3 Calcul des termes d'une suite

On définit la suite $(u_n)_{n\in\mathbb{N}}$ par

$$u_1 = 1, \quad u_2 = 3, \quad u_n = \frac{1}{u_{n-2}} + u_{n-1}$$

Écrire un programme qui, au choix de l'utilisateur :
- calcule la somme des n premiers termes de la suite (n donné par l'utilisateur),
- affiche les n premiers termes de la suite (n donné par l'utilisateur),
- détermine la valeur de n pour laquelle la somme des n premiers termes est supérieure à une valeur v donnée par l'utilisateur.

Le programme devra comporter une structure `switch`.

4 Opérations sur des matrices creuses

Soit A une matrice carrée, mais qui contient de nombreux éléments nuls, par exemple une matrice tridiagonale. On décide de stocker uniquement les éléments non nuls de A avec ce qu'on appelle un stockage creux. Le but de l'exercice est d'effectuer des opérations sur A en utilisant uniquement ce stockage creux.

1. Créez un type structuré qui contient
   - un entier décrivant la taille de la matrice,
   - un entier décrivant le nombre de termes non nuls,
   - deux tableaux d'entiers (en utilisant la classe `std::vector`), contenant les indices des lignes et colonnes des termes non nuls de la matrice,
   - et enfin un tableau de réels contenant ces termes.

   Il existe bien d'autres manières de stocker les coefficients d'une matrice creuse, manière dont le choix dépend des algorithmes utilisés ensuite. Ici nous utiliserons une représentation « naïve »de la matrice, mais qui est peu efficace.
2. Écrire une fonction qui prend en entrée un nom du fichier et retourne une structure de matrice creuse, qui sera remplie avec le contenu du fichier. Ce dernier aura sur sa première ligne le nombre n de coefficients à lire, et sur chacune des n lignes suivantes un indice de ligne, un indice de colonne et la valeur du coefficient correspondant. La taille de la matrice sera considérée comme étant le maximum des indices fournis.
3. Écrire une fonction pour calculer la trace d'une matrice creuse.
4. Écrire une fonction permettant de calculer la norme infinie d'une matrice creuse.
5. Écrire une fonction permettant de faire le produit d'une matrice creuse par un vecteur `x` (de la classe `vector`). Un conseil : écrire préalablement la formule sur un papier pour avoir les idées claires (une seule boucle est nécessaire).
6. Écrire un programme permettant, à l'aide des fonctions ainsi écrites, d'appliquer l'algorithme de la puissance itérée à une matrice creuse pour en déterminer la valeur propre de plus grande valeur absolue. Pour initialiser l'algorithme, on pourra utiliser la fonction `rand()` de la bibliothèque standard.

5 Pivot de Gauss

Ici nous considèrerons des matrices pleines, qui peuvent être représentées à l'aide de vecteurs de vecteurs :

```
std::vector< std::vector<double>> mat;
```

Implémentez et testez une fonction qui résoud un système linéaire $Ax = b$ avec la méthode du pivot de Gauss.

```
Gauss-Jordan
   r = 0                                    (r est l'indice de ligne du dernier pivot trouvé)
   Pour j de 1 jusqu'à m                    (j décrit tous les indices de colonnes)
   |   Rechercher max(|A[i,j]|, r+1 ≤ i ≤ n). Noter k l'indice de ligne du maximum
   |                                        (A[k,j] est le pivot)
   |   Si A[k,j]≠0 alors                    (A[k,j] désigne la valeur de la ligne k et de la colonne j)
   |   |   r=r+1                            (r désigne l'indice de la future ligne servant de pivot)
   |   |   Diviser la ligne k par A[k,j]    (On normalise la ligne de pivot de façon que le pivot prenne la valeur 1)
   |   |   Échanger les lignes k et r       (On place la ligne du pivot en position r)
   |   |   Pour i de 1 jusqu'à n            (On simplifie les autres lignes)
   |   |   |   Si i≠r alors
   |   |   |   |   Soustraire à la ligne i la ligne r multipliée par A[i,j] (de façon à annuler A[i,j])
   |   |   |   Fin Si
   |   |   Fin Pour
   |   Fin Si
   Fin Pour
Fin Gauss-Jordan
```

 Attention les indices commencent à 1 dans ce pseudo-code, et le second membre du système n'est pas pris en considération.