TE 1-A Introduction

Objectif Il s'agit de préparer les TE portant sur des thèmes numériques en introduisant la mise en œuvre conjointe des deux outils utilisés par la suite :

— un programme de calcul en C++, en fichiers source séparés, avec compilation gérée par l'outil `make` : ce code écrit ses résultats dans un fichier ;

— une application graphique sous `python` qui relit ce fichier afin de tracer les résultats.

À titre d'introduction, on se propose de tabuler quelques fonctions avec un pas fixe en C, pour les représenter ensuite graphiquement.

Dans un premier temps, il est conseillé de traiter l'ensemble (calcul et visualisation) dans une version simplifiée ne comportant pas les éléments signalés par le signe **(+)** dans l'énoncé qui seront implémentés ultérieurement.

1 Partie calcul en C++

1.1 Organisation des codes

La structure générale, présentée en cours, doit permettre la compilation séparée des différents fichiers. Elle comporte quatre fichiers :

1. Un programme principal `main.cpp` auquel est confié le dialogue avec l'utilisateur pour choisir **(+)** la fonction à étudier, l'abscisse de début, le pas d'échantillonnage et l'abscisse de fin. Ce programme crée alors le tableau des abscisses et le transmet à la procédure `tabule` qui calcule les ordonnées. Il fournit enfin les tableaux de résultats à la procédure d'écriture sur fichier.
2. Un fichier `fcts.cpp` définissant les différentes fonctions à tester (voir section 2).
3. Un fichier `methode.cpp` hébergeant la fonction `tabule` qui évalue la fonction passée en argument aux points de la grille fournie par le programme appelant. `tabule` sera conçue de façon à pouvoir traiter une fonction quelconque : on doit pouvoir changer de fonction sans recompiler le fichier.
4. Un fichier `util.cpp` hébergeant une fonction `ecrit` chargée d'écrire le fichier formaté des résultats.

Fichier des fonctions à tester
`fcts.cpp`

Définition des différentes fonctions à tester (voir section 2).
Un seul argument pour chacune des fonctions ;
(+) les paramètres éventuels des fonctions sont des variables globales du fichier.

Fichier de la méthode à appliquer
`methode.cpp`

Arguments de la méthode `tabule` :

— la fonction à tester
— le vecteur des abscisses rempli par le programme principal
— le vecteur des ordonnées à remplir

Programme principal
`main.cpp`

— Saisie des paramètres : abscisse de début, abscisse de fin, pas
— Construction du vecteur `x` comportant la grille des abscisses
— **(+)** Choix de la fonction à tabuler
— Appel de la méthode `tabule` pour remplir le vecteur des ordonnées
— Appel de la fonction d'écriture des résultats sur fichier

Fichier d'écriture sur fichier de sortie
`util.cpp`

Rôle de la fonction `ecrit` :

— Ouverture du fichier
— **(+)** Écriture de l'entête avec les valeurs extrêmes en abscisse et ordonnée
— Boucle d'écriture des couples $x, f(x)$ à raison d'un couple par ligne
— Fermeture du fichier

1.2 Structure du fichier de résultats

1.2.1 Entête du fichier de résultats

(+) Le fichier de résultats comportera trois lignes d'**en-tête** indiquant respectivement :

— les bornes des abscisses ;
— les bornes des ordonnées (valeurs minimale et maximale) ;
— le nombre de points et le pas d'échantillonnage.

Chacune des lignes d'entête débutera par le caractère `#` qui permet à `python` de considérer que ces lignes ne représentent pas les données à tracer.
Cet entête ne sera implémenté qu'après un premier test et une visualisation.

1.2.2 Corps du fichier de résultats

Le **corps du fichier** sera constitué des couples abscisse, ordonnée des points de la grille, à raison d'un point par ligne. On spécifiera un format compatible avec la dynamique des valeurs et assurant une précision relative de l'ordre de 10^{-7}.

1.3 Aspects techniques du C++

Pour simplifier, on choisira d'utiliser des objets `VectorXf` de la librairie `Eigen`. Ces objets sont des vecteurs (`Vector`) de `float` (`f`) de taille définie à l'exécution du code (`X`). Ils pourront être déclarés tardivement dans le programme principal, une fois leur taille calculée avec les données fournies par l'utilisateur. La fonction à tabuler sera vue par `tabule` sous la forme d'une référence de fonction à valeur de type `float` et à un argument de type `float`. On respectera donc les prototypes suivants :

fcts.h
```
float carre(float);
float logistique(float);
```

methode.h
```
#include <functional>
void tabule(std::function<float(float)> f,
            const Eigen::VectorXf &x, Eigen::VectorXf &y);
```

util.h
```
void ecrit(const std::string &fichname,
           const Eigen::VectorXf &x, const Eigen::VectorXf &y);
```

1.4 Mise au point du `makefile`

On associera à ces codes un fichier `makefile` pour prendre en charge la compilation séparée et l'édition de liens. Le fichier sera construit progressivement et testé avant d'être paramétré.
Une fois la compilation effectuée à la main, on pourra utiliser l'option `-MM` du compilateur `g++` pour extraire les dépendances des fichiers objets, fournissant un embryon de fichier `makefile` :
`gcc -MM fichier.cpp` affiche les dépendances de `fichier.o` (nécessite les `.h`)
Les règles implicites suffiront alors à générer les fichiers objets. Placer la cible de l'exécutable en premier pour que la simple commande `make` permette de lancer sa construction. Introduire ensuite les options de compilation (variables `CPPFLAGS` en C++ sous `make`).
On adjoindra enfin une cible `clean` qui supprime les fichiers objets et l'exécutable. Attention, `make clean` ne doit pas être systématiquement lancé pendant la mise au point des codes !

2 Fonctions proposées

2.1 Fonction simple

On pourra tout d'abord tester le code avec une fonction élémentaire facile à vérifier comme $y(t) = t^2$.

2.2 Équation logistique

On échantillonnera ensuite entre $t = 0$ et $t = 20$ la fonction solution de l'équation différentielle abordée dans le TE 2 avec pour valeur initiale $y(t_0) = y_0$ [1].

1. Attention : en C++, `y0`, `y1` et `yn` sont des fonctions de Bessel, ainsi que `j0`, `j1` et `jn`. On évitera donc de nommer ainsi des variables.

L'équation différentielle (1) régit l'évolution d'une population (en fait, la variable $y = N/N_0$ représente le rapport du nombre d'individus N à une population de référence N_0) avec un taux de croissance $a > 0$ en présence d'un mécanisme la limitant à une valeur maximale $k > 0$.

$$\frac{\mathrm{d}y}{\mathrm{d}t} = ay\left(1 - \frac{y}{k}\right) \tag{1}$$

On suppose qu'à l'instant initial, $0 < y(0) < k$. On pourra montrer que la solution analytique se met sous la forme (2).
On prendra $t_0 = 0.$, $y_0 = 0.5$, $a = 1.$ et $k = 1$.

$$y(t) = \frac{k}{1 + \dfrac{k - y_0}{y_0}\exp\left(-a(t - t_0)\right)} \tag{2}$$

3 Partage des paramètres des fonctions avec le programme principal

Les fonctions tests traitées par la méthode sont des fonctions d'une seule variable. Mais elles peuvent comporter des paramètres que la méthode ne doit pas connaître.
Dans un premier temps, on peut considérer comme fixés les paramètres a, k, τ, y_0 et t_0 des fonctions à tester. Ils peuvent alors être locaux à ces fonctions.

(+) Dans un second temps, on souhaite que le programme principal puisse choisir les paramètres a, k, τ, y_0 et t_0 des fonctions. À cet effet, il faut leur accorder une visibilité dans le programme principal au lieu de les considérer comme locaux à chaque fonction : ce seront donc des **variables globales**.

Ces paramètres doivent être déclarés comme variables globales partagées entre le programme principal et les fonctions à tester. À cet effet, on doit préciser le qualificatif `extern` lors de leur déclaration.

4 Lecture des données et visualisation en `python`

On utilisera le script `affiche.py` fourni qui permet de produire un fichier PDF affichant la fonction tabulée par le programme en C++. Ce script sera exécuté par l'instruction `python affiche.py`.

4.1 Commentaires sur le script python

4.1.1 Librairies utilisées

On utilise la librairie `numpy` pour les aspects numériques et les structures de tableaux multidimensionnels de type `array`. La visualisation fera appel à la librairie `matplotlib`. Ces librairies sont chargés en utilisant les raccourcis habituels `np` et `plt` pour les espaces de noms via :

```
import numpy as np
import matplotlib.pyplot as plt
```

4.1.2 Lecture du fichier

La lecture des données numériques depuis le fichier texte (à sélectionner au début du code python), dont le nom est enregistré dans la variable `nomfichier`, s'effectue avec la fonction `loadtxt` de `numpy` :
`mat = np.loadtxt(nomfichier, dtype='float')`,
qui rend une matrice dont les éléments sont par défaut des flottants. L'extraction de la colonne `k` se fait par `xk = mat[:,k-1]`.

La fonction `loadtxt` possède un paramètre optionnel `skiprows` qui permet de spécifier le nombre de lignes d'entête à sauter avant de lire la matrice. Mais, par défaut, elle ignore les lignes commençant par le caractère `#`.

La lecture de l'entête se fait avec la primitive `readline` :

```
fich = open(nomfichier,'r')
...
ligne = file.readline()
```

```
...
file.close()
```

Chaque ligne est stockée dans une chaîne qui comporte le changement de ligne `\n` en fin. Si on les concatène avec l'opérateur `+`, on obtiendra une chaîne de deux lignes terminée par un changement de ligne. On remplace les passages à la lignes inutiles par un blanc avec `ligne = texte.replace("\n", " ")`
Les informations d'entête sont utilisées pour documenter le graphique.

4.1.3 Tracé graphique

La fonction principale d'affichage de courbes $y = f(x)$ est `plot`. Elle admet comme arguments le vecteur **colonne** des abscisses et celui des ordonnées pour tracer une courbe. Le paramètre optionnel `label` permet de lui associer une légende.

```
# une courbe où x et y1 sont des tableaux 1D de même taille
plt.plot(x, y1, label="y1(x)") // affiche y1(x)
```

D'autres fonctions permettent notamment d'ajouter un titre, des libellés d'axes, de choisir des échelles logarithmiques...

```
plt.grid(True) # ajout d'une grille
# libelles d'axes
plt.xlabel("x")
plt.ylabel("y")
# une ou des lignes de titre
plt.title(titre, fontsize=8)
# active la légende du graphique (textes fixés auparavant via l'option label)
plt.legend()
```

La production de fichiers graphiques pour sauvegarder la figure affichée s'effectue avec `savefig`, par exemple au format postscript : `plt.savefig("f1.pdf")`