UNIVERSITE PAUL SABATIER

TOULOUSE III

M1 EEA

Techniques et Implémentations de Méthodes Numériques

Cahier de TP

Patrick DANES - Sylvain DUROLA - Frédéric GOUAISBAUT - Nicolas RIVIERE

2023/2024

Présentation de l'environnement de travail

Ces TP de programmation en langage C se déroulent sur PC sous le système d'exploitation Linux. Ici sont réunies des remarques d'ordre pratique concernant l'utilisation des machines de la salle mises à votre disposition.

Login et mot de passe

Les machines sont reliées en réseau [1] à un serveur qui stocke en particulier tous les fichiers utilisateurs. Chaque utilisateur (chaque binôme en l'occurrence) doit posséder un compte pour pouvoir se connecter à une machine. Le nom de compte correspond à votre identifiant UPS (ex : abc1234a) qui vous a été attribué à votre première inscription ; c'est le même identifiant que celui utilisé pour vous connecter à la plateforme Moodle de l'université.

Vos données de travail seront stockées sur un serveur. Pour travailler, nous utiliserons l'environnement Linux avec un terminal pour exécuter des commandes et l'éditeur de texte gedit pour saisir le code.

Il faudra toujours travailler dessus pour avoir accès à vos données en réseau.

Arrêt des machines

Il n'y a pas de sauvegarde automatique de votre travail. Par conséquent, si vous ne souhaitez pas qu'une mauvaise manipulation ou un arrêt inopiné de la machine entraîne la perte de tous vos efforts, pensez à sauvegarder régulièrement votre travail ! Si vous voulez (ou devez) arrêter la machine sur laquelle vous travaillez de manière propre, fermez la session en passant par la barre des tâches ! En cas de blocage complet n'hésitez pas à faire appel à un enseignant.

1. Il s'agit d'un réseau local non relié à l'extérieur.

Outils pour la programmation en C

L'écriture, la compilation, la mise au point et l'exécution d'un programme C fait appel à différents outils :

- l'éditeur de texte pour saisir les instructions du programme comme du texte normal dans un fichier qui est le programme *source*.
- le compilateur pour créer un exécutable (fichier binaire sans extension) ou un fichier objet (extension `.o`) à partir du programme source.
- le debugger qui aide l'utilisateur à trouver des erreurs en exécutant le programme de différentes manières et notamment pas à pas.

Dans le cadre de ces TP, l'environnement de développement sera celui de Linux avec l'éditeur Gedit et la chaine de compilation **GNU gcc**.

Il sera aussi possible d'utiliser **Dev C++** si et seulement si l'accès au serveur azteca était fermé. L'éditeur de Dev C++ reconnait les types de fichier en fonction de leur extension et mettra en surbrillance les mots-clefs du langage de programmation.

Le fichier contenant le programme source doit toujours comporter l'extension `.c` (Ex : `prog.c`). Il est important de préciser cela lors de la sauvegarde du fichier pour obtenir la bonne compilation.

Pour compiler le programme, le compilateur GNU gcc sera utilisé de manière transparente depuis les menus de l'éditeur. Le compilateur est appelé au moyen de boutons de la barre d'outils d'Eclipse Luna.

S'il n'y a pas d'erreur, un exécutable est créé (`prog`) et il pourra être exécuté depuis Dev C++ ou depuis l'explorateur de fichiers de Windows ou un terminal MS-DOS (en tapant `prog.exe`).

Impression des listings

Ne pas utiliser la commande d'impression des éditeurs de texte ! L'impression de listings en salle de TP n'est pas indispensable. L'impression des listings se fait sur une imprimante connectée au réseau.

Il est rappelé que l'imprimante est une ressource collective. Le respect des autres utilisateurs impose de vérifier que l'impression que l'on a lancée s'est bien déroulée et ne bloque pas l'imprimante pour une raison ou pour une autre. Le papier est par ailleurs une ressource rare qui doit donc être utilisée à bon escient.

Programmation modulaire et Compilation séparée

Il a été vu en cours comment faire de la programmation modulaire en utilisant des fonctions et des bibliothèques.

Les sous-programmes étant supposés pouvoir être utilisées à partir de différents programmes, il est souhaitable de les regrouper dans une bibliothèque. Une bibliothèque est composée d'un fichier d'en-tête (d'extension .h) et d'un fichier source (d'extension .c).

Le fichier d'en-tête (d'extension .h) doit contenir la définition des structures de données et les déclarations (en-têtes) des sous-programmes externes pouvant être appelés depuis un autre programme.

Le fichier source associé (d'extension .c) doit, lui, contenir les définitions des sous-programmes.

Nous avons besoin pour cela :

— du fichier contenant les définitions des fonctions.

— du fichier d'en-têtes contenant les déclarations des sous-programmes et les définitions des structures de données.

— du fichier contenant la fonction principale `main` (ou programme principal).

Comment faire ? (commandes en ligne depuis un terminal)

Prenons l'exemple où nous avons un programme `essai.c` qui utilise des sous-programmes de la bibliothèque `une_biblio.h`.

Avant de compiler une bibliothèque, il faut qu'il y ait les deux fichiers `une_biblio.c` et `une_biblio.h`. Ensuite il faut inclure au début du fichier `une_biblio.c` le nom de la bibliothèque `une_biblio.h` comme ceci : `#include "une_biblio.h"`

Ensuite, il faut compiler en tapant la ligne de commande :

```
gcc -Wall -c une_biblio.c -lm
```

Cela crée le fichier `une_biblio.o`.

Pour compiler un programme utilisant cette bibliothèque, il suffit d'inclure la bibliothèque `"une_biblio.h"`dans le programme `essai.c` à l'aide de la directive de préprocesseur et taper la ligne de commande :

```
gcc -Wall -g -o essai essai.c une_biblio.o -lm
```

Et le Makefile ?

Une alternative à ceci est d'utiliser un `Makefile`, c'est-à-dire un fichier contenant toutes les règles de compilation. Du coup, cela simplifie la compilation car il suffit juste d'exécuter ce fichier pour que toutes les étapes de compilation soient exécutées.

Le `Makefile` équivalent à ce qui est écrit plus haut serait le suivant :

```
#Makefile de essai.c et de une_biblio

all: une_biblio.o essai.c
	gcc -Wall -g -o essai essai.c une_biblio.o -lm

une_biblio.o: une_biblio.c une_biblio.h
	gcc -Wall -c une_biblio.c -lm
```

Il n'y a plus qu'à compiler avec `make all`

Tout ça dans Dev C++ !

Pour nos TP, nous travaillons avec un environnement de développement intégré (IDE). Le principe de la programmation modulaire avec Dev C++ est le même sauf que vous n'aurez ni à faire le Makefile, ni à taper les lignes de commande pour compiler vos programmes pendant les TP...c'est l'avantage de ces environnements de programmation.

Travailler avec Dev C++ consiste à travailler dans un environnement où tous les outils sont regroupés et accessibles à partir d'une même fenêtre : programmer, compiler, exécuter et débugger.

Pour faire de la programmation modulaire dans Dev C++, il est important de créer un "Nouveau projet", de le nommer dans "Nom" et de choisir :

— "Console Application"
— "Projet C"
— "Langage par défaut"

Travaux Pratiques 1

L'objectif de ce TP est de :
— prendre en main l'environnement de développement,
— réviser les bases en langage C sur des exemples divers,
— apprendre à programmer de façon modulaire (bibliothèques, makefile).

---

1 - Equation du 2nd degré (sans sous-programme)

Le calcul des racines d'une équation du second degré du type $ax^2 + bx + c = 0$ se fait en calculant d'abord son discriminant $\Delta = b^2 - 4ac$ lorsque $a \neq 0$. Les valeurs sont du domaine des réels.

1.1 Ecrire un programme permettant de calculer Δ et d'afficher sa valeur à l'écran à partir des valeurs réelles a, b, c qui seront saisies au clavier par l'utilisateur.

⇒ *Compiler et tester.*

1.2 Compléter le programme précédent afin de calculer et d'afficher les solutions de l'équation pour toute valeur de Δ.

Pour les racines complexes, les parties réelles et imaginaires seront affichées séparément.

⇒ *Compiler et tester.*

1.3 Compléter le programme précédent afin de calculer et d'afficher la (ou les) solution(s), s'il y en a, et d'afficher les valeurs à l'écran pour toute valeur de a, b et c.

⇒ *Compiler et tester.*

2 - Calcul de valeurs et affichage graphique

Soit une équation du second degré du type $y = ax^2 + bx + c$ où a, b et c sont des valeurs réelles saisies au clavier par l'utilisateur. Soit x, un nombre réel, variant entre x_{min} et x_{max} avec un pas δx fixé par l'utilisateur.

2.1 Ecrire, sans sous-programme, un programme qui permet de calculer et afficher les valeurs de x et y correspondantes sans utiliser de tableau. Le programme doit vérifier et empêcher qu'il y ait plus de 100 valeurs, ce qui peut amener l'utilisateur à changer δx.

⇒ *Compiler et tester.*

2.2 Modifier le programme précédent en utilisant des tableaux pour stocker les valeurs de x et de y. La taille des tableaux sera statique et fixée à $N = 100$.

Il faut rajouter 2 sous-programmes :

— un pour l'affichage d'un tableau de valeurs réelles de taille N,

— un pour le calcul des points x et y avec les deux tableaux en paramètre.

Il faudra aussi afficher graphiquement les points calculés.

⇒ *Compiler et tester.*

2.3 Modifier le programme précédent afin d'afficher le minimum et le maximum de y avec la valeur de x correspondante.

Pour cela, il faut rajouter 2 sous-programmes qui prennent les 2 tableaux en paramètres :

— un qui recherche et renvoie y_{min} avec la valeur x associée,

— un un qui recherche et renvoie y_{max} avec la valeur x associée.

⇒ *Compiler et tester.*

3 - Moyenne, écart type et fichiers

3.1 Ecrire un programme qui, à partir d'une liste de N nombres entiers positifs ou nuls saisis par l'utilisateur, calcule : la moyenne, l'écart type, la note minimale, la note maximale.

Pour cela, il faudra utiliser 5 sous-programmes : un pour calculer la somme de N entiers, un pour calculer la moyenne de N entiers à partir du calcul de la somme, un pour calculer l'écart type à partir de la somme et de la moyenne, un pour le calcul du minimum de N entiers et un pour le calcul du maximum de N entiers. L'utilisateur devra d'abord saisir le nombre N avant de saisir les N nombres qui seront stockés dans un tableau statique de dimension 30.

⇒ *Compiler et tester.*

3.2 Modifier le programme précédent de manière à ce que l'utilisateur effectue la saisie des nombres sans connaître le nombre de notes N. La saisie se termine lorsque l'utilisateur saisit la valeur -1.

$\Rightarrow$ *Compiler et tester.*

3.2 Modifier le programme précédent pour que désormais les notes soient lues à partir d'un fichier `notes.dat` que vous devrez créer manuellement avec un éditeur de texte.

$\Rightarrow$ *Compiler et tester.*

L'écart type est défini par :

$\sigma = \sqrt{(\sum_{i=0}^{N-1}(y_i - \overline{y})^2/N)}$ avec $\overline{y}$ la moyenne des y_i

Travaux Pratiques 2

L'objectif de ce TP est d'apprendre à manipuler les tableaux de dimension variable, les pointeurs et les structures. Ce sera illustré par le calcul des coefficients de la droite des moindres carrés et la création d'une bibliothèque de fonctions C permettant de manipuler des polynômes à coefficients réels.

---

1 - Droite des moindres carrés

La droite des moindres carrés est une droite qui passe au mieux à travers un ensemble de points x_i, y_i. L'équation de cette droite est donnée par : $y = ax + b$ avec :

$$a = \frac{\overline{xy} - \overline{x}.\overline{y}}{\overline{x^2} - \overline{x}^2} \qquad b = \frac{\overline{y}.\overline{x^2} - \overline{x}.\overline{xy}}{\overline{x^2} - \overline{x}^2}$$

L'objectif est d'écrire un programme permettant de calculer et d'afficher la valeur de $y = ax + b$ en fonction d'une valeur de x rentrée au clavier.

On suppose que les couples de points x_i, y_i sont contenus dans un fichier et nous ne connaissons pas leur nombre qui est inférieur à 100. Chaque ligne du fichier contient un couple x_i, y_i. La taille des tableaux sera statique et fixée à $N = 100$.

Préparation

Le problème est décomposé en sous-problèmes. Donner les en-têtes des fonctions des questions 1.2 et 1.3.

1.1 - Lecture du fichier Ecrire une fonction permettant de :
- lire les valeurs des couples x_i, y_i contenus dans le fichier,
- les stocker dans deux tableaux X et Y,
- renvoyer le nombre de couples.

L'interface de la fonction sera la suivante :

```
int lecture(float X[], float Y[])
```

1.2 - Valeur moyenne Soit Z un tableau contenant n valeurs réelles notées z_0 à z_{n-1}.
Ecrire une fonction renvoyant la valeur moyenne des valeurs z_i définie par :
$\overline{z} = (\sum_{i=0}^{n-1} z_i)/n$
Pour cela, il faut :
— préciser les entrées et les sorties de la fonction
— en déduire une interface possible
— écrire le code de la fonction.

1.3 - Moyenne d'un produit Soit deux tableaux X et Y contenant chacun n valeurs réelles, écrire une fonction renvoyant la moyenne des produits $x_i y_i$ définie par $\overline{xy} = (\sum_{i=0}^{n-1} x_i y_i)/n$
Comme précédemment, on définira d'abord l'interface de la fonction.

1.4 - Calcul des coefficients Utiliser les deux fonctions précédentes pour calculer les quantités $\overline{x}$, $\overline{y}$, $\overline{xy}$, $\overline{x^2}$.
En déduire les expressions permettant de calculer a et b.
Intégrer cela dans une fonction dont l'interface sera la suivante :

```
void calcul_coefs(float X[], float Y[], int N, float* a, float* b)
```

1.4 - Programme principal Ecrire le code de la fonction `main()` permettant d'afficher $y = ax+b$ avec les coefficients calculés ainsi que la valeur de y pour une valeur de x rentrée au clavier.
⇒ *Compiler et tester.*

2 - Manipulation de polynômes

On souhaite écrire des fonctions permettant de :
— saisir un polynôme (degré et valeurs de coefficients),
— l'afficher à l'écran,
— calculer sa valeur en un point,
— calculer le polynôme dérivé,
— additionner deux polynômes,
— multiplier deux polynômes,

Préparation : Analyse et algorithme

Le problème est décomposé en sous-problèmes.
Donner pour chacun des sous problèmes :
— Les données nécessaires à ce traitement

— Les données générées
— Les en-têtes de fonctions

2.1 - Représentation d'un polynômes Un polynôme est caractérisé par son degré N et par N+1 coefficients. On choisira pour le représenter d'utiliser une structure composée de deux membres, :

— un entier pour son degré,
— un pointeur vers un tableau de réels pour ses coefficients.

Le tableau sera alloué dynamiquement en fonction du degré du polynôme.

Ecrire la structure dans le fichier `bibliotheque.h` permettant de représenter un polynôme. Dans ce fichier `bibliotheque.h`, vous écrirez tous les en-têtes de fonction sauf `main()`.

Ecrire un premier programme `polynome.c` avec seulement la fonction `int main()` utilisant la structure de données et initialisant le polynôme :

$3x^2 - 4x + 2$.

⇒ *Compiler et tester* le programme principal `polynome.c` que vous ferez évoluer tout au long de ce TP pour tester les fonctions qui seront écrites dans le fichier `bibliotheque.c`.

2.2 - Affichage d'un polynôme À partir de maintenant, vous travaillerez dans le fichier `bibliotheque.c` dans lequel il y aura toutes les fonctions sauf `main()`. Ecrire une fonction prenant en paramètre un polynôme (voir question précédente) et l'affichant à l'écran sous une forme du style :

```
3.00 x^2 -4.00 x^1 2.00 x^0
```

⇒ *Compiler et tester.*

2.3 - Calcul de la valeur d'un polynôme en x Ecrire une fonction renvoyant la valeur du polynôme $P(x)$ à partir d'un polynôme et d'une valeur réelle x.

⇒ *Compiler et tester* le bon fonctionnement de cette fonction dans le programme principal en utilisant le premier polynôme pris en exemple et en calculant et affichant $P(x)$ en différents points.

2.4 - Création d'un polynôme par allocation Ecrire une fonction qui modifie un polynôme passé en paramètre, *i.e.* une structure définissant le polynôme. On affectera au membre contenant le degré du polynôme une valeur passée en paramètre (degré). On effecuera l'allocation dynamique du tabelau de coefficents de la structure en paramètre. On ne fera pas la saisie des coefficients du polynôme ici, ce sera fait à la question 2.5.

⇒ *Compiler et tester* cette fonction en remplaçant l'allocation dynamique de la première question par l'appel de la fonction.

2.5 - Saisie d'un polynôme au clavier On souhaite disposer d'une fonction permettant de saisir au clavier les coefficients d'un polynôme.

La fonction à écrire doit modifier un polynôme en paramètre.

⇒ *Compiler et tester.*

2.6 - Calcul du polynôme dérivé Ecrire une fonction prenant en paramètre deux polynômes, le polynôme initial et le polynôme dérivé résultant.

⇒ *Compiler et tester.*

2.7 - Somme de deux polynômes La somme est un polynôme dont le degré est celui du polynôme de plus haut degré.

Ecrire une fonction prenant en paramètre trois polynômes, les polynômes initiaux et le polynôme somme résultant.

⇒ *Compiler et tester.*

2.8 - Produit de deux polynômes Le produit de deux polynômes est un polynôme dont le degré est égal à la somme des degrés des deux polynômes sommés.

Ecrire une fonction prenant en paramètre trois polynômes, les polynômes initiaux et le polynôme produit résultant.

⇒ *Compiler et tester.*

TP Subsidiaire : Les listes chaînées

Ne pas faire sauf si vous êtes insomniaque !

L'objectif de ce TP est de trier des informations contenues dans un fichier afin de pouvoir y rechercher des informations particulières. Ce sera illustré par une gestion dynamique des données au moyen de listes chaînées .

On souhaite écrire des fonctions permettant de :

— lire les données contenues dans un fichier en renvoyant l'adresse de la liste,
— insérer un élément en tête de la liste,
— afficher la liste,
— supprimer un élément de la liste,
— trier la liste,
— rechercher un élément dans la liste,
— sauvegarder la liste dans un fichier,

On suppose que les valeurs stockées dans le fichier sont sous un format :

```
3.0 4.1
4.0 5.1
5.0 6.1
...
```

La lecture des couples de valeurs qui sont dans un fichier devra les stocker dans une liste simplement chaînée. En effet, nous ne connaissons pas a priori le nombre de couples à récupérer. Chaque élément de la liste est une structure composée de deux champs : les données et l'adresse de l'élément suivant. Le champ des données est une structure composée de deux champs de valeur réelle.

A chaque ligne du fichier lue, il faut insérer un élément dans la liste chaînée des valeurs.

Préparation

Ecrire la structure permettant de représenter un élément de la liste.

Pour chacune des fonctions, donner les en-têtes.

⇒ *Compiler et tester à chaque étape.*

TP 4 : Interpolation polynômiale, phénomène de Runge

L'objectif de ce TP est de mettre en oeuvre diverses techniques d'algorithmiques acquises pour résoudre le problème de l'interpolation polynômiale et traiter le phénomène de Runge.

---

1 - Principe de l'interpolation polynômiale

Dans de nombreux domaines de l'ingénierie, il est bien souvent nécessaire de simuler le comportement de systèmes dont nous n'avons qu'une connaissance limitée. Par exemple, sur une courbe $x \mapsto f(x)$ inconnue et suffisamment régulière, ayant mesuré $m+1$ valeurs de f en $\mathbf{x_i} \in [a,b], \mathbf{i} \in \{0,...,m\}$, nous voudrions simuler l'évolution de f en fonction de x sur tout l'intervalle $[a,b]$. Nous retrouvons par exemple cette problématique dans les jeux videos ou les films d'animation faisant appel à un système de *motion capture* : on vient simuler le mouvement de tout le corps d'un avatar à partir de la connaissance des déplacements de quelques points seulement.

Le problème de l'interpolation consiste à trouver une courbe la plus simple et la plus régulière possible passant par les points :

$$\mathbf{x_i}, f(\mathbf{x_i}), \mathbf{i} \in \{0,...,m\}$$

Dans ce TP, nous allons chercher à déterminer un polynôme passant par ces points. En effet, les polynômes constituent une famille de fonctions simple à manipuler, comme nous l'avons vu dans le TP 2. Ce problème s'appelle le problème de l'*interpolation polynomiale*. Avant d'exposer les différentes techniques que nous allons mettre en oeuvre, rappelons tout d'abord un certain nombre de résultats utiles pour la suite.

Théorème 1 *Un polynôme de degré m possède exactement m racines qui peuvent être réelles ou complexes, simples ou multiples.*

Un corrolaire de ce théorème est ainsi :

Théorème 2 *Il existe un et un seul polynôme de degré m passant par $m+1$ points $(\mathbf{x_i}, f(\mathbf{x_i})), \mathbf{i} \in \{0, ..., m\}$ avec $\mathbf{x_i} \neq \mathbf{x_j}$ pour $\mathbf{i} \neq \mathbf{j}$.*

Ce théorème est très important et à la base des résultats et techniques que nous allons étudier durant ce TP.

2 - Echantillonnage régulier d'une fonction sur un intervalle

Pour toute la suite de ce TP, on considérera la fonction $x \mapsto \frac{1}{1+25x^2}$ et l'intervalle $[-1, 1]$.

— Créer un programme permettant d'obtenir un fichier de données correspondant à n points de la courbe $x \mapsto f(x)$, régulièrement répartis sur l'intervalle demandé.

Remarque 1 *Pour réaliser un échantillonnage régulier sur un intervalle, on divise ce dernier en n intervalles de même taille. Chaque centre de ces intervalles sera une abscisse $\mathbf{x_i}$ d'échantillonnage. Par exemple, si l'on veut obtenir 4 points d'une courbe sur $[-1, 1]$, on l'évaluera en : -0.75, -0.25, 0.25 et 0.75.*

Remarque 2 *Dans le fichier, les données seront stockées en deux colonnes séparées par une tabulation. La première correspondra aux abscisses $\mathbf{x_i}$, la seconde aux ordonnées $f(\mathbf{x_i})$.*

Remarque 3 *Voici pour vous aider quelques exemples de prototypes de fonctions que vous serez amenés à créer :*

```
double f(double x)
void echantillon_regulier(double min, double max, int n, double * echantillon)
void calcul_fonction(int n, double * ech_x, double * ech_f)
void sauvegarde_fichier(char * nom_fichier, double * abscisses, double * ordonnees, int n)
```

Respectivement :

— *Une fonction calculant l'évaluation en x de f.*
— *Une fonction déterminant un échantillonnage régulier de l'intervalle $[min, max]$ sur n points.*
— *Une fonction calculant l'évaluation de f sur un échantillonnage donné.*
— *Une fonction sauvegardant dans un fichier un tableau d'abscisses et d'ordonnées à tracer.*

— Tracez la courbe $x \mapsto f(x)$ régulièrement échantillonnée sur 1000 points de $[-1, 1]$.

Remarque 4 *Pour tracer une courbe stockée dans un fichier "courbe.dat" dans* gnuplot*, vous pouvez utiliser la commande suivante :*

```
plot [-1:1] [0:1] "courbe.dat"
```

Cette commande affichera la fenêtre $[-1, 1]$ *en abscisses et* $[0, 1]$ *en ordonnées. Si vous souhaitez tracer plusieurs courbes "courbe1.dat" et "courbe2.dat" sur le même graphique, vous pouvez utiliser :*

```
plot [-1:1] [0:1] "courbe1.dat","courbe2.dat"
```

3 - Base de Lagrange : tracé d'un polynôme et changement de base

3.1 - Tracé d'un un polynôme exprimé dans la base canonique

Considérons un polynôme $P_m(x)$ de degré m écrit sur la base canonique (décomposé sur les monômes de degré entre 0 et m) :

$$P_m(x) = \sum_{\mathbf{i}=0}^{m} a_{\mathbf{i}} x^{\mathbf{i}}$$

— Créer un programme permettant d'obtenir un fichier de données correspondant à n points de la courbe $x \mapsto P_m(x)$, régulièrement répartis sur l'intervalle demandé, pour un polynôme exprimé dans la base canonique.

Remarque 5 *Voici pour vous aider quelques exemples de prototypes de fonctions que vous serez amenés à créer, en utilisant la structure de polynôme créée lors du TP 2 :*

```
double polynome(double x, polynome * poly)
void calcul_polynome(int n, double * ech_x, double * ech_poly, polynome * poly)
```

Respectivement :

— *Une fonction calculant l'évaluation en* x *d'un polynôme poly dont les coefficients sont donnés sur la base canonique.*

— *Une fonction calculant l'évaluation de ce même poly sur un échantillonnage donné.*

— Tracez la courbe $x \mapsto P_m(x)$ régulièrement échantillonnée sur 1000 points de $[-1, 1]$.

Remarque 6 *Pour ce tracé, vous pouvez prendre par exemple :* $m = 2$, $a_0 = 1$, $a_1 = -1$ *et* $a_2 = 1$.

3.2 - Tracé d'un polynôme exprimé dans la base de Lagrange

Considérons à présent la famille $m+1$ nombres réels quelconques $\mathbf{x_i}, \mathbf{i} \in \{0, ..., m\}$. On définit les polynômes de Lagrange comme étant :

$$l_{\mathbf{i}}(x) = \prod_{\mathbf{j}=0, \mathbf{j}\neq\mathbf{i}}^{m} \frac{x - \mathbf{x_j}}{\mathbf{x_i} - \mathbf{x_j}}, \qquad \mathbf{i} \in \{0, ..., m\}$$

On peut démontrer que la famille $l_{\mathbf{i}}(x), \mathbf{i} \in \{0, ..., m\}$ est une base des polynômes de degré m. De ce fait $P_m(x)$ s'écrit aussi :

$$P_m(x) = \sum_{\mathbf{i}=0}^{m} b_{\mathbf{i}} l_{\mathbf{i}}(x) = \sum_{\mathbf{i}=0}^{m} a_{\mathbf{i}} x^{\mathbf{i}}$$

— Créer un programme permettant, pour une famille $\mathbf{x_i}, \mathbf{i} \in \{0, ..., m\}$ donnée, d'obtenir des fichiers de données correspondant à n points des courbes $x \mapsto l_{\mathbf{i}}(x)$ (les $m+1$ polynômes de Lagrange, régulièrement répartis sur l'intervalle demandé.
— Créer un programme permettant, pour une famille $\mathbf{x_i}, \mathbf{i} \in \{0, ..., m\}$ donnée, d'obtenir un fichier de données correspondant à n points de la courbe $x \mapsto P_m(x)$, régulièrement répartis sur l'intervalle demandé, pour un polynôme exprimé dans la base de Lagrange.

Remarque 7 *Peut-on encore utiliser la structure de polynôme mise en place dans le TP2 ? Il est conseillé de créer deux nouvelles fonctions (à compléter) :*

```
double polynome_lagrange(???)
void calcul_polynome_lagrange(???)
```

Respectivement :
— *Une fonction calculant l'évaluation en x d'un polynôme poly dont les coefficients sont donnés sur la base de Lagrange sur les points $\mathbf{x_i}, \mathbf{i} \in \{0, ..., m\}$.*
— *Une fonction calculant l'évaluation de ce même poly sur un échantillonnage donné.*

3.3 - Changement de base d'un polynôme vers la base de Lagrange

Les polynômes de la base de Lagrange vérifient la propriété suivante :

$$l_{\mathbf{i}}(\mathbf{x_j}) = \delta_{\mathbf{ij}},$$

où $\delta_{\mathbf{ij}}$ est le symbole de Kronecker, c'est-à-dire :

$$\delta_{\mathbf{ij}} = \begin{cases} 1 & \text{si} \quad \mathbf{i} = \mathbf{j} \\ 0 & \text{si} \quad \mathbf{i} \neq \mathbf{j} \end{cases}$$

— Vérifiez cette propriété sur un exemple simple.

Remarque 8 *Pour cette vérification, vous pouvez prendre par exemple :* $m = 2$, $x_0 = -0.2$, $x_1 = 0.2$ *et* $x_2 = 0.6$.

Par la propriété précédente, il vient que :

$$P_m(\mathbf{x_j}) = \sum_{\mathbf{i}=0}^{m} b_\mathbf{i}\delta_\mathbf{ij} = b_\mathbf{j}$$

Ainsi, selon le théorème 2, le polynôme $P_m(x) = \sum_{\mathbf{i}=0}^{m} a_\mathbf{i}x^\mathbf{i}$ de degré m s'écrit de manière unique sous la forme :

$$P_m(x) = \sum_{\mathbf{i}=0}^{m} P_m(\mathbf{x_i})l_\mathbf{i}(x)$$

— Vérifiez cette propriété sur l'exemple précédent : $m = 2$, $a_0 = 1$, $a_1 = -1$, $a_2 = 1$, $x_0 = -0.2$, $x_1 = 0.2$ et $x_2 = 0.6$.

Remarque 9 *Le changement de base nécessite de "faire passer" le polynôme par les points* $(\mathbf{x_i}, P_m(\mathbf{x_i})), \mathbf{i} \in \{0, ..., m\}$. *Cela revient dont à réaliser l'interpolation polynomiale de* P_m *sur* $m + 1$ *points ! La méthode sera strictement identique lorsque l'on tâchera de réaliser l'interpolation polynomiale d'une fonction* f *: le polynôme interpolateur sur les points* $(\mathbf{x_i}, f(\mathbf{x_i})), \mathbf{i} \in \{0, ..., m\}$ *s'écrira :*

$$F_m(x) = \sum_{\mathbf{i}=0}^{m} f(\mathbf{x_i})l_\mathbf{i}(x)$$

4 - Phénomène de Runge

Nous allons à présent étudier l'interpolation polynômiale de f sur $m + 1$ points régulièrements répartis dans $[-1, 1]$. Les représentations graphiques seront effectués avec $n = 1000$ points régulièrements répartis dans $[-1, 1]$.

— Calculez l'interpolation polynômiale de f pour $m = 4$, $m = 8$, $m = 12$ et $m = 20$.

Remarque 10 *Pour comparer ces interpolations, vous pouvez les tracer sur une même courbe avec le tracé de* f. *On remarque que l'augmentation du nombre de points ne permet pas une meilleure approximation, au contraire. C'est ce que l'on appelle le phénomène de Runge du nom du mathématicien allemand du 19 ième siècle qui montra que la différence entre le polynôme interpolateur et la fonction n'était pas forcément bornée. Cette erreur peut diverger si l'on augmente le nombre de points d'interpolation.*

5 - Echantillonnage sur les nœuds de Tchebychev d'une fonction sur un intervalle et réduction du phénomène de Runge

Une méthode permettant de réduire le phénomène de Runge consiste à prendre d'autres points d'interpolation. On les choisit de telle sorte que le produit suivant soit minimal :

$$\min_{\mathbf{x_i}} \left\| \prod_{i=0}^{m} (x - \mathbf{x_i}) \right\|.$$

Cette minimisation fait appel aux racines d'une certaine classe de polynôme dit de Tchebychev. Ces racines sont appellées les nœuds de Tchebychev Pour un intervalle $[a, b]$, ces points sont donnés par :

$$x_i = \frac{b-a}{2} cos\left(\frac{\pi}{2}\frac{2i+1}{m+1}\right) + \frac{a+b}{2}, i \in \{0, \cdots, m\}$$

Remarque 11 *Pour réaliser un échantillonnage de Tchebychev sur un intervalle $[a, b]$, on divise le demi-cercle de diamètre $[a, b]$ en $m+1$ arcs de même angle au centre. Chaque milieu de ces arcs se projette sur l'axe $[a, b]$ en l'abscisse x_i d'échantillonnage.*

— Calculez l'interpolation polynômiale de f pour $m = 4$, $m = 8$, $m = 12$ et $m = 20$ en utilisant un échantillonnage de Tchebychev.

Remarque 12 *Voici pour vous aider un exemple de prototype de fonction que vous serez amenés à créer :*

```
void echantillon_tchebychef(double min, double max, int n, double * echantillon)
```

— Comparez les deux interpolation polynômiale de f pour $m = 12$ en utilisant un échantillonnage de Tchebychev ou un échantillonnage régulier.

6 - Conclusion

Pour obtenir une simulation de f, on a utilisé la connaissance de $m+1$ points et reconstruit un polynôme de degré m. Ce polynôme peut être déterminé sur la base canonique par le biais de la résolution du système suivant :

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^m \\ 1 & x_1 & x_1^2 & \dots & x_1^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^m \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_m) \end{bmatrix}$$

Cependant, cette méthode demande de mettre en place un algorithme dédié comme celui du pivot de Gauss ou bien d'appliquer des outils d'inversion matricielle très coûteux en temps de calcul et peu robustes aux erreurs numériques. En utilisant les théorèmes 2 et 1, on montre qu'il est bien plus facile d'utiliser les polynômes de la base de Lagrange.

Ensuite, le choix d'un échantillonnage régulier pour les points d'interpolation n'est pas toujours judicieux et fait apparaître le phénomène de Runge. Il est préférable de procéder à un échantillonnage de Tchebychef.

Enfin, il existe d'autres méthodes permettant d'approximer une fonction par un polynôme. Par exemple, on peut aussi citer le développement en série entière autour d'un point de référence. Dans notre exemple, le développement est :

$$f(x) = \frac{1}{1+25x^2} = \sum_{k=0}^{+\infty} (-25x^2)^k$$

Le rayon de convergence de cette série est de $\frac{1}{5}$.

— *(Facultatif)* Tracez le comportement des polynômes $\sum_{k=0}^{m/2}(-25x^2)^k$ pour $m = 4$, $m = 8$, $m = 12$ et $m = 20$.

TP 5 : Intégration numérique et Quadratures

Dans chaque partie de ce TP, vous allez être invités à étudier des structures spécifiques pour l'implémentation en langage C. Celles-ci ne sont pas indispensable à la réalisation d'un programme d'intégration numérique, mais nous vous conseillons fortement de respecter et de conserver la définition que nous vous indiquons. En effet, pour obtenir une visualisation graphique sur Gnuplot de la fonction choisie et de l'approximation de son intégrale, nous vous avons écrit une procédure utilisant explicitement les structures proposées.

Dans la bibliothèque *[TP5_library]* vous trouverez aussi les définitions des structures ainsi que quelques déclarations de fonctions. Ces dernières pourront grandement vous aider à répondre aux questions de ce TP : utilisez-les si vous êtes bloqués.

---

1 - Introduction à l'Intégration Numérique

On considère une fonction f, suffisamment régulière et définie sur l'intervalle $[a, b]$:

$$\begin{aligned} f : [a, b] &\to \mathbb{R} \\ x &\mapsto f(x). \end{aligned} \tag{1}$$

L'objectif est de calculer numériquement l'intégrale suivante :

$$I = \int_a^b f(x)dx.$$

À moins de disposer d'un environnement de travail autorisant les calculs symboliques, il n'est pas possible de déterminer exactement le résultat dans le cas général. En effet, nos outils numérique ne travaillent sur les fonctions qu'à travers leur échantillonnage sur leur

ensemble de définition : il faut donc trouver des moyens d'approximer ce calcul d'intégrale à partir de la connaissance de la fonction $f(x)$ en un nombre fini d'abscisses x.

Le calcul d'intégrale d'une fonction peut être requis dans de nombreuses applications :

— En Automatique linéaire, lorsque l'on implémente informatiquement un correcteur ayant une composante intégrale (PI, PID).

— En Statistique, pour le calcul de fonctions de répartition ou de l'espérance d'une variables aléatoires.

— En Electrotechnique pour la commande vectorielle, lorsque l'on dispose d'un capteur de vitesse angulaire mais pas de capteur de position.

Dans ce TP, nous allons étudier une méthode générique dite de *formules de quadrature*.

2 - Calcul d'une *subdivision* de l'intervalle $[a, b]$

Dans un premier temps, on va définir une *subdivision* de l'intervalle de calcul $[a, b]$:

$$a = \alpha_0 < \alpha_1 < \ldots < \alpha_n = b.$$

Le calcul de l'intégrale est donc réduit à

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \int_{\alpha_i}^{\alpha_{i+1}} f(x)dx.$$

Ainsi, notre problème initial est transformé en un calcul d'intégrales sur des intervalles plus petits.

— Ecrivez une fonction permettant de calculer la subdivision d'un intervalle donné en n intervalles de même longueur.

Remarque 13 *Pour vous aider à écrire et à tester cette fonction, vous êtes libres de définir des fonctions permettant : d'allouer de l'espace mémoire pour une subdivision, de libérer cet espace mémoire, d'afficher une subdivision à l'écran, de saisir un intervalle et un entier n... En revanche, nous vous imposons d'utiliser les structures suivantes :*

```
typedef struct{
      double debut;
      double fin;
} structIntervalle;
```

```
typedef struct{
      int nombre;
      structIntervalle* tabIntervalle;
} structSubdivision;
```

3 - Définition et lecture d'une *quadrature*

La démarche adoptée pour calculer une intégrale élémentaire est d'approximer le calcul de l'intégrale par une somme pondérée des valeurs de la fonction calculées en des abscisses

particulières $x_{i,j} = (1-\zeta_j)\alpha_i + \zeta_j\alpha_{i+1}$ appartenant à l'intervalle $[\alpha_i, \alpha_{i+1}]$.
Plus précisemment, on calcule l'intégrale par la formule suivante :

$$\int_{\alpha_i}^{\alpha_{i+1}} f(x)dx \simeq (\alpha_{i+1} - \alpha_i) \sum_{j=0}^{t-1} \omega_j f\left((1-\zeta_j)\alpha_i + \zeta_j\alpha_{i+1}\right)$$

— Les ζ_j appartiennent à l'intervalle $[0, 1]$.
— Le nombre $x_{i,j} = (1-\zeta_j)\alpha_i + \zeta_j\alpha_{i+1}$ est donc compris dans l'intervalle $[\alpha_i, \alpha_{i+1}]$ et est une abscisse de calcul de $f(x)$.
— Les ω_j sont de somme unitaire : $\sum_{j=0}^{t} \omega_j = 1$. Ce sont les intégrales entre 0 et 1 des polynômes de la base de Lagrange sur les nœuds $x_{i,j}$.

Cette somme pondérée traduit l'intégration du polynôme interpolateur en les points $(x_{i,j}, f(x_{i,j}))$.

Définition 1 *On appelle formule de* quadrature *le choix d'une taille $t \in \mathbb{N}^*$, de valeurs $\{\zeta_j \in [0,1]\}_{j\in\{0\ldots t-1\}}$ et de pondérations $\{\omega_j\}_{j\in\{0\ldots t-1\}}$ telles que $\sum_{j=0}^{t} \omega_j = 1$, le tout permettant l'approximation d'une intégrale.*

— Ecrivez une fonction permettant de lire un fichier "quadrature.txt" et de créer une formule de quadrature correspondante.

Remarque 14 *Pour vous aider à écrire et à tester cette fonction, vous êtes libres de définir des fonctions permettant : d'allouer de l'espace mémoire pour une formule de quadrature, de libérer cet espace mémoire, d'afficher une formule de quadrature à l'écran, de vérifier la validité d'un fichier de formule de quadrature... En revanche, nous vous imposons d'utiliser la structure suivante :*

```
typedef struct{
       int taille;
       double* tabZeta;
       double* tabOmega;
} structQuadrature;
```

Nous vous proposons aussi la syntaxe suivante pour le fichier "quadrature.txt" :

```
<entier t>
<r\'eel Zeta(0)>      <r\'eel Omega(0)>
   ...                  ...
<r\'eel Zeta(t-1)>    <r\'eel Omega(t-1)>
```

3.1 - Méthode des rectangles à gauche ou à droite

Pour cette méthode élémentaire, on considère un unique ζ_0. On approxime alors la fonction f, dans chaque intervalle $[\alpha_i, \alpha_{i+1}]$, par un polynôme d'ordre 0, c'est à dire une constante :

$$f(x) \simeq f\left((1-\zeta_0)\alpha_i + \zeta_0\alpha_{i+1}\right).$$

La quadrature élémentaire s'écrit alors :

$$\int_{\alpha_i}^{\alpha_{i+1}} f(x)dx \simeq (\alpha_{i+1} - \alpha_i) f\left((1-\zeta_0)\alpha_i + \zeta_0\alpha_{i+1}\right)$$

et la quadrature composée s'écrit :

$$\int_a^b f(x)dx \simeq \sum_{i=0}^{n-1} (\alpha_{i+1} - \alpha_i) f\left((1-\zeta_0)\alpha_i + \zeta_0\alpha_{i+1}\right).$$

La méthode des rectanges à gauche fixe $(1-\zeta_0)\alpha_i + \zeta_0\alpha_{i+1} = \alpha_i$. La méthode des rectanges à droite fixe $(1-\zeta_0)\alpha_i + \zeta_0\alpha_{i+1} = \alpha_{i+1}$.

— Ecrivez deux fichiers de quadrature correspondant aux méthodes des rectangles à gauche ou à droite.

3.2 - Méthode du point milieu

Pour cette méthode également, on approxime la fonction f, dans chaque intervalle $[\alpha_i, \alpha_{i+1}]$, par un polynôme d'ordre 0. Cette méthode impose de plus que l'abscisse de calcul de f soit au milieu de l'intervalle ;

$$(1-\zeta_0)\alpha_i + \zeta_0\alpha_{i+1} = \frac{\alpha_i + \alpha_{i+1}}{2}$$

— Ecrivez le fichier de quadrature correspondant à la méthode du point milieu.

3.3 - Méthode des trapèzes

On choisit d'effectuer une moyenne pondérée sur deux termes telle que la quadrature élémentaire s'écrive :

$$\int_{\alpha_i}^{\alpha_{i+1}} f(x)dx \simeq (\alpha_{i+1} - \alpha_i) \left(\frac{1}{2} f(\alpha_i) + \frac{1}{2} f(\alpha_{i+1})\right).$$

La quadrature composée s'écrit alors :

$$\int_a^b f(x)dx \simeq \sum_{i=0}^{k-1} (\alpha_{i+1} - \alpha_i) \left(\frac{1}{2} f(\alpha_i) + \frac{1}{2} f(\alpha_{i+1})\right).$$

— Ecrivez le fichier de quadrature correspondant à la méthode des trapèzes.

4 - Calcul de l'*échantillonnage d'une fonction* sur une subdivision, pour une formule de quadrature donnée

A présent que l'on dispose d'un moyen de réaliser une subdivision de $[a, b]$ et que nous pouvons choisir une formule de quadrature par l'intermédiaire de la lecture d'un fichier, nous avons besoin de récupérer l'échantillonnage de la fonction f sur les abscisses $(1-\zeta_j)\alpha_i + \zeta_j\alpha_{i+1}$. Comme i varie dans $\{0 \dots n-1\}$ et j dans $\{0 \dots t-1\}$ il convient de présenter ces échantillons dans une matrice :

$$\begin{bmatrix} f((1-\zeta_0)\alpha_0 + \zeta_0\alpha_1) & \cdots & f((1-\zeta_{t-1})\alpha_0 + \zeta_{t-1}\alpha_1) \\ \vdots & \ddots & \vdots \\ f((1-\zeta_0)\alpha_{n-1} + \zeta_0\alpha_n) & \cdots & f((1-\zeta_{t-1})\alpha_{n-1} + \zeta_{t-1}\alpha_n) \end{bmatrix}$$
$$= \begin{bmatrix} f(x_{0,0}) & \cdots & f(x_{0,t-1}) \\ \vdots & \ddots & \vdots \\ f(x_{n-1,0}) & \cdots & f(x_{n-1,t-1}) \end{bmatrix}$$

— Ecrivez une fonction permettant de calculer, et stocker dans une structure, l'échantillonnage de f nécessaire au calcul de l'intégrale I, pour un choix donné de subdivision et de formule de quadrature.

Remarque 15 *Pour vous aider à écrire et à tester cette fonction, vous êtes libres de définir des fonctions permettant : d'allouer de l'espace mémoire pour une échantillonnage, de libérer cet espace mémoire, d'afficher un échantillonnage à l'écran... En revanche, nous vous imposons d'utiliser la structure suivante :*

```
typedef struct{
      structQuadrature Quadrature_Echantillonnage;
      structSubdivision Subdivision_Echantillonnage;
      double** matriceFonctionEchantillonnee;
} structEchantillonnage_fonction;
```

La matrice contenant les valeurs échantillonnées doit être constituée de n lignes et t colonnes, stockez la dans un tableau de t pointeurs vers des tableaux de taille n.

5 - Calcul de l'intégrale d'une fonction f entre a et b

— Ecrivez une fonction qui, à partir du précédent échantillonnage, permet de calculer une approximation de l'intégrale I.

$$I \simeq \sum_{i=0}^{n-1} \left((\alpha_{i+1} - \alpha_i) \sum_{j=0}^{t-1} \omega_j f\left(x_{i,j}\right) \right)$$

Avec :

$$x_{i,j} = (1-\zeta_j)\alpha_i + \zeta_j\alpha_{i+1}.$$

Remarque 16 *Vous écrirez votre fonction* "main()" *de sorte qu'elle demande de rentrer un intervalle* $[a, b]$*, un nombre* n *de subdivisions, qu'elle lise le fichier de quadrature, calcule l'intégrale* I *et affiche la représentation graphique du résultat, pour la fonction implémentée dans :*

```
double CalculFonction(double x)
```

Remarque 17 *Voici quelques exemples de fonctions sur lesquelles vous pouvez tester votre calcul d'intégrale :*

- *Une fonction constante* $f(x) = a$*.*
- *Une fonction affine* $f(x) = ax + b$*.*
- *Une fonction non polynomiale facilement intégrable comme* $f(x) = e^{-x}$*.*
- *Une fonction plus compliquée dont l'étude de l'intégrale peut être intéressante...*

Pour l'affichage graphique, vous pouvez appeler la fonction suivante :

```
void AffichageGnuplot(structEchantillonnage_fonction Echantillonnage_fonction)
```

Elle génère deux fichiers "courbe_ligne.dat" *et* "courbe_integration_boxes.dat" *prêts à être affichés avec* gnuplot *grâce à la commande :*

```
plot "courbe_integration_boxes.dat" with boxes fs solid, "courbe_ligne.dat" with line
```

6 - Ordre d'une quadrature

Afin d'avoir une idée de la qualité et de la précision d'une méthode de quadrature, on définit son *ordre* de la manière suivante :

Définition 2 *Une méthode de quadrature est d'ordre* N *si la formule approchée est exacte pour toutes fonctions polynomiales d'ordre* N *et inexacte pour au moins un polynôme d'ordre* $N + 1$*.*

— Testez l'ordre des quadratures écrites jusqu'à présent.

Remarque 18 *Pour ce faire, vous pouvez choisir de calculer l'intégrale des monômes de base canonique, sur l'intervalle* $[0, 1]$*, avec un nombre de subdivisions égal à* 1 *et de comparer par rapport à la valeur théorique.*

7 - Quelques méthodes de quadrature plus évoluées

— Testez l'ordre des méthodes de quadrature suivantes :

7.1 - Méthode de Simpson

La méthode des trapèzes réalise l'interpolation polynomiale de degré 1 de f sur chaque intervalle $[\alpha_i, \alpha_{i+1}]$ (le polynôme de degré 1 ayant la même valeur que f en α_i et α_{i+1}).

Dans la même idée, la méthode de Simpson réalise l'interpolation polynomiale de degré 2 de f sur chaque intervalle $[\alpha_i, \alpha_{i+1}]$ (le polynôme de degré 2 ayant la même valeur que f en α_i, $\frac{1}{2}\alpha_i + \frac{1}{2}\alpha_{i+1}$ et α_{i+1}).

Remarque 19 *Après développement, on trouve le fichier de quadrature suivant :*

```
3
0       0.1666667
0.5     0.6666666
1       0.1666667
```

En augmentant encore l'ordre de l'interpolation polynomiale sur un échantillonnage régulier (voir le TP 4) de $[\alpha_i, \alpha_{i+1}]$, on crée ce que l'on appelle les méthodes de Newton-Cotes.

7.2 - Méthode de Gauss-Tchebychev

Dans la même idée que le Phénomène de Runge dans le TP4, on se rend compte que les méthodes de Newton-Cotes pour des degrés de polynômes élevés ont tendance à être numériquement peu stables et ne garantissent pas la convergence de l'estimation vers la véritable valeur de l'intégrale de f sur l'intervalle $[\alpha_i, \alpha_{i+1}]$. Pour contrer ce problème, il est possible de proposer un échantillonnage non régulier, mais faisant intervenir les nœuds de Tchebychev.

Après développement au degré t, la méthode de quadrature donne :

— $\zeta_j = \frac{1}{2}\left(1 + \cos\left(\frac{(2j+1)\pi}{2t}\right)\right)$ pour $j \in \{0 \dots t-1\}$.

— $\omega_j = \frac{\pi \sin\left(\frac{(2j+1)\pi}{2t}\right)}{2t}$ pour $j \in \{0 \dots t-1\}$.

8 - Application à l'étude de la densité de probabilité d'une loi normale

Soit la densité de probabilité d'une loi normale : $f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$.

Par définition : $\int_{-\infty}^{+\infty} f(x)dx = 1$.

— Vérifiez ce calcul.

— Quelle est la valeur de $\int_{-3}^{+3} f(x)dx$?

— Quelle méthode de quadrature utilisez-vous ? En combien d'intervalles subdivisez-vous $[-3, 3]$?

TP 6 : Résolution d'équations différentielles ordinaires

Ce TP se propose de revisiter l'une des méthodes « classiques » de résolution numérique d'équations différentielles ordinaires (EDO). Ses objectifs relativement aux outils mis en œuvre sont les suivants :

- implémentation de commandes/fonctions (M-files) sous MATLAB.
- exploitation de S-functions sous SIMULINK pour la simulation numérique de tout système dynamique.
- (*facultatif*) compilation et exécution de code C depuis l'interpréteur MATLAB (MEX-files)

---

1 - Résolution numérique des Équations Différentielles Ordinaires

a - Positionnement du problème

Soit t une variable indépendante, et y une fonction inconnue de t à valeurs dans $\mathbb{R}^{n_y}$. Une *équation différentielle ordinaire* (EDO) est une équation qui exprime le lien entre t, y, et les dérivées successives de y par rapport à t. En Sciences de l'Ingénieur, t désigne généralement le temps, et on note $\dot{y}(t) = \frac{\mathrm{d}y(t)}{\mathrm{d}t}$, $\ddot{y}(t) = \frac{\mathrm{d}^2y(t)}{\mathrm{d}t^2}$, ..., $y^{(n)}(t) = \frac{\mathrm{d}^n y(t)}{\mathrm{d}t^n}$, de sorte qu'une EDO s'écrit

$$\psi(t, y(t), \dot{y}(t), \ldots, y^{(n)}(t)) = 0. \tag{2}$$

En regroupant y et ses dérivées jusqu'à l'ordre $(n-1)$ dans un vecteur $Y(t) = \big(y(t), \dot{y}(t), \ldots, y^{(n-1)}(t)\big)^T \in \mathbb{R}^{n.n_y}$, (2) peut être exprimée comme une équation différentielle du premier ordre en Y, soit

$$\Psi(t, Y(t), \dot{Y}(t)) = 0. \tag{3}$$

On se restreint ici aux EDOs *explicites*, qui admettent la forme équivalente

$$\dot{Y}(t) = f(t, Y(t)). \tag{4}$$

Sous des conditions de « bien posé », la solution y de (2), ou, de manière équivalente, la solution Y de (4), existe et est unique dès lors qu'on se donne les valeurs de y et de ses dérivées successives jusqu'à l'ordre $(n-1)$ à l'instant initial t_0, *i.e.* si

$$Y(t_0) = (y(t_0), \dot{y}(t_0), \ldots, y^{(n-1)}(t_0))^T = (y_0, \dot{y}_0, \ldots, y_0^{(n-1)})^T \text{ avec } y_0, \dot{y}_0, \ldots, y_0^{(n-1)} \text{ connus.} \tag{5}$$

b - Résolution à un pas

Excepté dans de très rares cas, il est impossible d'obtenir l'expression analytique de $Y(t)$. On recourt alors à des méthodes de résolution numérique. Ces méthodes font naturellement partie des algorithmes de base des logiciels de simulation de systèmes dynamiques tels que MATLAB-SIMULINK.

Étant donnée une séquence d'instants régulièrement espacés $t_0, t_1, \ldots, t_f$, avec

$$t_{k+1} = t_k + h, \tag{6}$$

et la condition initiale (5), le but est de définir en tout instant t_k une approximation Y_k de la solution $Y(t_k)$ de (4). L'intégration de l'équation (4) entre t_k et t_{k+1} donne :

$$Y(t_{k+1}) = Y(t_k) + \int_{t_k}^{t_{k+1}} f(t, Y(t))dt. \tag{7}$$

Les *solveurs à un pas* proposent une solution numérique selon le schéma récursif

$$Y_{k+1} = Y_k + h\Phi(t_k, h, Y_k), \tag{8}$$

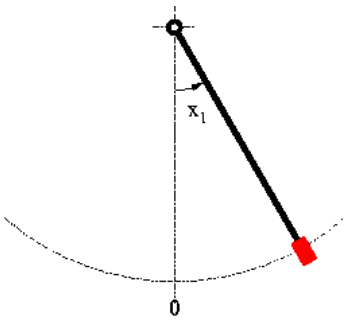
i.e., l'approximation Y_{k+1} de $Y(t_{k+1})$ ne dépend que de l'approximation Y_k de $Y(t_k)$, de t_k et de l'intervalle h séparant t_k et t_{k+1}.

L'initialisation de tels schémas suit naturellement l'équation $Y_0 = Y(t_0)$. Cependant, durant toute transition $t_k \mapsto t_{k+1}$, une erreur apparaît entre $Y(t_{k+1})$ et son approximation Y_{k+1} y compris si Y_k est initialisé avec la valeur exacte de $Y(t_k)$. Cette *erreur locale* est conditionnée par la forme de (8) et, plus accessoirement, par les effets d'arrondi dus à la précision finie des calculs. Les erreurs locales aux instants successifs $t_1, \ldots, t_k$ se cumulent à t_k en une *erreur globale*.

c - Exercice : comparaison de trois approches

i - Problème mécanique

L'EDO que l'on se propose de résoudre numériquement décrit le régime libre (*i.e.* sans excitation extérieure) d'un pendule évoluant dans le plan vertical sans frottement. Ce procédé est schématisé ci-contre. Il est constitué d'une masse m fixée au bout d'une tige rigide de longueur l sans masse. Sous les hypothèses considérées, les équations fondamentales de la dynamique permettent de définir la position $\theta(t)$ du pendule par rapport à la verticale en tant que solution de l'EDO



$$ml^2\ddot{\theta}(t) + mgl\sin\theta(t) = 0, \tag{9}$$

avec $g = 9.81\,\mathrm{m.s^{-2}}$ l'accélération gravitationnelle. L'équation (9) peut immédiatement être transformée sous la forme (4) suivante :

$$\begin{cases} x_1 = \theta \\ x_2 = \dot{\theta} \end{cases} \implies \begin{pmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{pmatrix} = \begin{pmatrix} x_2(t) \\ -\omega_0^2 \sin x_1(t) \end{pmatrix}, \text{ avec } \omega_0 = \sqrt{\frac{g}{l}}. \tag{10}$$

Si le pendule est lâché à vitesse nulle depuis un angle initial θ_0, alors il oscille indéfiniment. Pour de faibles valeurs de θ_0, la validité de l'approximation $\sin\theta \approx \theta$ permet d'établir que ses oscillations sont des sinusoides de période ω_0. Cependant, pour un angle initial θ_0 suffisamment élevé, cette approximation ne tient plus ; on constate alors que les oscillations ne sont plus isochrones, au sens où leur période dépend de leur amplitude.

On prend pour paramètres $m = 5\,\mathrm{kg}$ et $l = 1\,\mathrm{m}$. On rappelle qu'à l'instant initial, son état est caractérisé par la position $\theta(0) = \theta_0$ et la vitesse $\dot{\theta}(0) = 0$.

ii - Solution analytique du problème exact

La solution analytique générale de (9) pour $\theta(0) = \theta_0$ et $\dot{\theta}(0) = 0$ s'écrit

$$\theta(t) = 2\arcsin\Big(k_0\ \mathrm{sn}\Big(\big(\omega_0 t + \mathrm{K}(k_0)\big), k_0\Big)\Big), \text{ avec } k_0 = \sin\frac{\theta_0}{2}. \tag{11}$$

Ci-dessus, $\mathrm{K}(\kappa)$ et $\mathrm{sn}(\zeta,\kappa)$ désignent respectivement l'*intégrale elliptique complète de première espèce de Legendre* et la fonction *sinus de Jacobi* définies par, pour $\kappa \in [0;1]$,

$$\mathrm{K}(\kappa) = \int_0^{\frac{\pi}{2}} \frac{\mathrm{d}\theta}{\sqrt{1-\kappa^2\sin^2\theta}}, \qquad \mathrm{sn}(\zeta,\kappa) = \sin\phi, \text{ où } \zeta = \int_0^{\phi} \frac{\mathrm{d}\theta}{\sqrt{1-\kappa^2\sin^2\theta}}. \tag{12}$$

Sous MATLAB, leur évaluation s'effectue au moyen des commandes ellipke(κ^2) et ellipj(ζ,κ^2).

— Créer un script MATLAB permettant de tracer la solution analytique (11) sur l'horizon temporel $[t_0, t_f]$ = `[0:0.05:30]` et les quatre angles initiaux $\theta_0 \in \{0.1\,\mathrm{rad}, 1\,\mathrm{rad}, 3\,\mathrm{rad}, 3.13\,\mathrm{rad}\}$.

iii - Solution analytique du problème linéarisé

On considère à présent le problème linéarisé suivant :

$$ml^2\ddot{\theta}(t) + mgl\theta(t) = 0, \tag{13}$$

La solution analytique générale de (13) pour $\theta(0) = \theta_0$ et $\dot{\theta}(0) = 0$ s'écrit

$$\theta(t) = \theta(0)\cos(\omega_0 t)\,. \tag{14}$$

— Créer un script MATLAB permettant de tracer la solution analytique du problème linéarisé (14) sur l'horizon temporel $[t_0, t_f]$ = `[0:0.05:30]` et les quatre angles initiaux $\theta_0 \in \{0.1\,\text{rad}, 1\,\text{rad}, 3\,\text{rad}, 3.13\,\text{rad}\}$.
— Conclure sur la validité de la linéarisation.

iv - Solution approchée du problème exact

— Créer un script MATLAB permettant de tracer une solution approchée de (11) sur l'horizon temporel $[t_0, t_f]$ = `[0:0.05:30]` et les quatre angles initiaux $\theta_0 \in \{0.1\,\text{rad}, 1\,\text{rad}, 3\,\text{rad}, 3.13\,\text{rad}\}$.

On utilisera la fonction MATLAB `ode45`, qui constitue une méthode standard de résolution numérique d'EDOs. Son prototype est :

```
[Tout,Yout] = ode45(odefun, Tspan, Y0)
```

où

- `odefun` désigne un "handle" – sorte de "pointeur MATLAB" – vers la fonction $f(.,.)$ définie en (4) :
  - ▷ si $f(.,.)$ est implémentée sous la forme d'une fonction MATLAB `f` au moyen du mot-clé `function` – *e.g.* dans un fichier `f.m` –, alors `odefun = @f` ;
  - ▷ si $f(.,.)$ est implémentée en tant que fonction anonyme selon le modèle `f = @(t,y) expression_mathematique_du_vecteur_dy/dt`, alors `odefun = f`.
- `Tspan` désigne le vecteur ligne constitué de tous les instants de simulation $t_0, t_1, \ldots, t_f$ précédemment définis ;
- `Y0` est le vecteur colonne exprimant la condition initiale (5) ;
- `Tout` et `Yout` contiennent la solution numérique de l'EDO considérée, où chaque ligne de la matrice `Yout` correspond à la ligne correspondante du vecteur colonne des instants `Tout`.

— Conclure sur la validité de la solution approchée par la méthode de résolution `ode45`.

2 - Implémentation d'une méthode RK4 en M-files

a - Méthodes Runge-Kutta explicites

Dans la classe des solveurs à un pas, les solutions les plus génériques et les plus puissantes sont sans doute les *méthodes de Runge-Kutta*. Bien qu'elles ne conviennent pas dans toutes les situations, elles s'avèrent souvent efficaces, précises et remarquablement robustes numériquement.

La forme générale d'une méthode de Runge-Kutta explicite à m termes – ou RK(m) – permet l'obtention de l'approximation Y_{k+1} de $Y(t_{k+1})$ à partir de l'approximation Y_k de $Y(t_k)$ selon la formule

$$Y_{k+1} = Y_k + h_k \sum_{i=1}^{m} b_i f(t_{i,k}, Y_{i,k}), \tag{15}$$

où $f(.,.)$ est définie en (4) et $t_{i,k} = t_k + c_i h_k$ avec $0 \leq c_i \leq 1$ donc $t_k \leq t_{i,k} \leq t_{k+1}$.

Remarque 20 *Cette approximation correspond à une intégration numérique par méthode de quadrature : on approxime la fonction $f(t, Y(t))$ par le polynôme passant par les m points $(t_{i,k}, f(t_{i,k}, Y_{i,k}))$. On l'intègre ensuite entre t_k et t_{k+1} par la quadrature (voir TP5) :*

```
m
c1        b1
c2        b2
c3        b3
...       ...
cm        bm
```

Les b_i sont de somme unitaire $\sum_{i=0}^{m} b_i = 1$.

Le calcul de Y_{k+1} se fait ainsi grâce à m étapes intermédiaires qui évaluent les approximation $Y_{i,k}$ de $Y(t_{i,k})$ en m instants $t_k \leq t_{i,k} \leq t_{k+1}$.

$$\begin{array}{lll}
[1] & t_{1,k} = t_k & Y_{1,k} = Y_k \\
[2] & t_{2,k} = t_k + c_2 h_k & Y_{2,k} = Y_k + h_k \left(a_{21} f(t_{1,k}, Y_{1,k})\right) \\
[3] & t_{3,k} = t_k + c_3 h_k & Y_{3,k} = Y_k + h_k \left(a_{31} f(t_{1,k}, Y_{1,k}) + a_{32} f(t_{2,k}, Y_{2,k})\right) \\
 & \vdots & \vdots \\
[i] & t_{i,k} = t_k + c_i h_k & Y_{i,k} = Y_k + h_k \sum_{j=1}^{i-1} a_{ij} f(t_{j,k}, Y_{j,k}) \\
 & \vdots & \vdots \\
[m] & t_{m,k} = t_k + c_m h_k & Y_{m,k} = Y_k + h_k \sum_{j=1}^{m-1} a_{mj} f(t_{j,k}, Y_{j,k}).
\end{array} \tag{16}$$

Remarque 21 *Chaque étape intermédiaire correspond aussi à une intégration numérique par méthode de quadrature pour trouver $Y_{i,k}$ à l'instant $t_{i,k} = t_k + c_i h_k$: on approxime la fonction $f(t, Y(t))$ par le polynôme passant par les i points $(t_{j,k}, f(t_{j,k}, Y_{j,k}))$ avec $1 \leq j \leq i-1$. On l'intègre ensuite entre t_k et t_{k+1} par la quadrature (voir TP5) :*

```
i-1
c1              ai1
c2              ai2
c3              ai3
...              ...
c(i-1)          ai(i-1)
```

Attention, comme on estime $Y_{i,k}$ à l'instant $t_{i,k} = t_k + c_i h_k$, les a_{ij} ne sont pas forcément de somme unitaire : $\sum_{j=0}^{i} a_{ij} = c_i$.

b - Méthode Runge-Kutta 4

Une augmentation du nombre de termes m d'une méthode de Runge-Kutta permet de viser une précision élevée *via* l'accroissement du nombre de paramètres libres (les nœuds normalisés c_i avec $1 \leq i \leq m$, les poids normalisés pour les étapes intermédiaires a_{ij} avec $1 \leq j < i \leq m$ et les poids de l'étape finale b_i avec $1 \leq i \leq m$). Un compromis intéressant entre complexité calculatoire et ordre est obtenu pour $m = 4$ termes avec la paramétrisation

$$\begin{array}{cccccc} c_1 = 0 & c_2 = \frac{1}{2} & c_3 = \frac{1}{2} & c_4 = 1 & & \\ a_{21} = \frac{1}{2} & a_{31} = 0 & a_{32} = \frac{1}{2} & a_{41} = 0 & a_{42} = a_{43} = \frac{1}{2} & (17) \\ b_1 = \frac{1}{6} & b_2 = \frac{1}{3} & b_3 = \frac{1}{3} & b_4 = \frac{1}{6}. & & \end{array}$$

Remarque 22 *Dans cette méthode, 4 points intermédiaires sont calculés.*

• *Le premier est le plus simple et correspond à la méthode de quadrature :*

```
0
```

De ce fait, le premier point intermédiaire est le point initial $Y_{1,k} = Y_k$ à l'instant initial t_k.

• *Le second point intermédiaire est calculé à partir du premier par la méthode de quadrature :*

```
1
0       0.5
```

De ce fait, le second point intermédiaire $Y_{2,k}$ est situé en $t_{2,k} = t_k + 0.5h_k$, au milieu de l'intervalle (car $\sum_{j=0}^{1} a_{2j} = c_2 = 0.5$) et est obtenu par une méthode rectangle à gauche.

• *Le troisième point intermédiaire est calculé à partir des deux premiers par la méthode de quadrature :*

```
2
0         0
0.5       0.5
```

De ce fait, le troisième point intermédiaire $Y_{3,k}$ est situé en $t_{3,k} = t_k + 0.5h_k$, au milieu de l'intervalle (car $\sum_{j=0}^{2} a_{3j} = c_3 = 0.5$) et est obtenu par une méthode rectangle à droite.

• *Le quatrième point intermédiaire est calculé à partir des trois premiers par la méthode de quadrature :*

```
3
0         0
0.5       0.5
0.5       0.5
```

De ce fait, le quatrième point intermédiaire $Y_{4,k}$ est situé en $t_{4,k} = t_k + h_k$, à la fin de l'intervalle (car $\sum_{j=0}^{3} a_{4j} = c_4 = 1$) et est obtenu par une méthode rectangle point milieu en pondérant d'autant les estimation en ce point milieu.

• *Le point final est calculé à partir des quatre intermédiaires par la méthode de quadrature :*

```
4
0         0.166667
0.5       0.333333
0.5       0.333333
1         0.166667
```

De ce fait, le point final Y_{k+1} est obtenu par une méthode de Simpson en pondérant d'autant les estimation au point milieu.

Remarque 23 *Il serait possible de regrouper toutes ces quadratures en un seul fichier :*

```
4
0         0.166667       0         0         0.5
0.5       0.333333       0.5       0.5
0.5       0.333333       0.5
1         0.166667
```

Ce fichier serait composé de la manière suivante :

```
<m, nombre de termes de RK>
<c1>          <b1>          <am1>          <a(m-1)1>       ...       <a31>       <a21>
<c2>          <b2>          <am2>          <a(m-1)2>       ...       <a32>
...           ...           ...            ...
<c(m-1)>      <b(m-1)>      <am(m-1)>
<cm>          <bm>
```

c - Exercice : un pas de RK4 de t_k à t_{k+1}

— Implémenter sous MATLAB une fonction `unPasRK4` selon le prototype
`[tnext,Ynext] = unPasRK4(odefun, tk, Yk, h)`
qui, lorsque le "handle" `odefun` pointe vers $f(.,.)$ définie en (4), et lorsque $t_k =$ `tk`, $Y_k =$ `Yk`, $h =$ `h`, génère $t_{k+1} =$ `tk_next` et $Y_{k+1} =$ `Yk_next` selon (6),(15),(16),(17). On rappelle que si `odefun = @f`, alors la commande MATLAB `feval(odefun,x1,x2,...,xN)` invoque `f(x1,x2,...,xN)`

d - Exercice : méthode RK4 de t_0 à t_f

— Implémenter sous MATLAB une fonction `RK4` qui permet la résolution numérique de l'EDO (10) selon le prototype
`[Tout,Yout] = RK4(odefun, T0, Tf, h, Y0)`
où `odefun` et `Y0` admettent la même signification que dans `ode45` (cf. ci-dessus), et `T0`,`Tf`,`h` désignent respectivement t_0, t_f, h.
— Tester l'algorithme ainsi implémenté sur l'horizon temporel compris entre 0 et 30 s, pour les conditions initiales $\theta_0 \in \{0.1\,\text{rad}, 1\,\text{rad}, 3\,\text{rad}, 3.13\,\text{rad}\}$ et pour $h \in \{0.1, 0.01, 0.001\}$.
— Conclure sur la validité de cette méthode de résolution.

3 - Simulation par S-fonction SIMULINK et méthode RK(4)

a - Les S-fonctions SIMULINK

L'installation de MATLAB mise à votre disposition est accompagnée de SIMULINK. Ce progiciel permet la modélisation par schéma-blocs de systèmes dynamiques ainsi que la simulation de leur réponse au moyen de divers algorithmes, dont des méthodes de résolution numérique d'EDOs. SIMULINK est invoqué depuis l'invite MATLAB par la commande `simulink`. La librairie standard de SIMULINK comporte un ensemble conséquent de blocs : opérations algébriques, fonctions de transfert, représentations d'état, éléments statiques non linéaires, blocs logiques, etc. Un système complexe peut se présenter comme l'interconnexion de tels éléments. Une fonctionnalité particulièrement puissante offerte par SIMULINK consiste à décrire virtuellement tout système – statique ou dynamique, à temps continu ou discret, décrit par un automate, hybride, etc. – au moyen d'un bloc générique unique, associé à une « S-fonction » (S-function = System function) écrite en MATLAB ou C.

Bien sûr, une S-fonction est intimement liée au moteur de simulation SIMULINK. Ce TP se limitera aux S-fonctions MATLAB de Niveau 1 (Level-1 MATLAB

S-function). Une telle S-fonction est héritée sous SIMULINK au moyen du bloc S-Function (rubrique User-Defined Functions). Vous trouverez davantage d'information sur `http://www.mathworks.fr/help/toolbox/simulink/`, onglet User's Guide, sous-onglet Developing S-Functions, et onglet Blocks, sous-onglet User-Defined Functions.

b - Exercice : S-fonction du système linéarisé

Considérons la modélisation par S-fonction du système dynamique linéarisé et de sa condition initiale :

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases}, \; A = \begin{pmatrix} 0 & 1 \\ -\frac{g}{l} & 0 \end{pmatrix}, \; B = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \; C = \begin{pmatrix} 1 & 0 \end{pmatrix}, \; D = 0, \; x(0) = \begin{pmatrix} 3.13 \\ 0 \end{pmatrix}. \tag{18}$$

Pour créer cette S-fonction, ouvrez un modèle SIMULINK et glissez un bloc S-Function Builder depuis la librairie Functions & Tables. Un double clic sur ce bloc vous permet d'éditer ses propriétés :

- Dans l'onglet Initialization vous pouvez définir la taille de ses ports (entrées et sorties), la dimension de son état continu (la taille du vecteur $Y(t)$) ainsi que les conditions initiales $Y(t_0)$.
- Dans l'onglet Outputs vous pouvez définir comment se calcule la sortie du bloc en fonction de son état et de ses entrées. Vous devez ainsi traduire l'expression $y(t) = Cx(t) + Du(t)$.
- Dans l'onglet Continuous Derivatives vous pouvez définir l'équation régissant la dynamique du système, c'est-à-dire le lien entre les entrées, l'état et sa dérivée. Vous devez ainsi traduire l'expression $\dot{x}(t) = Ax(t) + Bu(t)$.

Une fois ceci effectué, vous pouvez construire le bloc S-Function par le bouton Build. Si aucune erreur n'apparaît, le bloc est opérationnel. Si une erreur spécifie de préciser un compilateur, tapez mex -setup dans l'invite MATLAB et suivez les instructions pour en choisir un.

Le bloc opérationnel peut être relié à un générateur de signaux (par exemple un bloc Step) et à un appareil de mesure (par exemple un bloc Scope). La simulation est lancée par l'icone ▷, les propriétés de simulation sont quant à elles accessibles dans le menu Simulation/Simulation Parameters....

— Implémenter l'EDO (13),(18) et sa condition initiale dans une S-fonction MATLAB de Niveau 1 `mon_pendule.m`.
— Simuler cette S-fonction avec un schéma RK(4) permettant de reproduire les scenarii précédemment envisagés et conclure.

c - Exercice : S-fonction du système exact

Vous pouvez à présent regarder les deux fichiers .c générés par le bouton Build. Ils doivent normalement s'appeler mon_pendule.c et mon_pendule_wrapper.c. Ils correspondent à la description orientée objet du bloc mon_pendule.c.

— Rechercher dans ces fichiers où modifier la dynamique du système et la remplacer pour implémenter à present l'EDO (10).

— Simuler cette nouvelle S-fonction avec un schéma RK(4) permettant de reproduire les scenarii précédemment envisagés et conclure.

4. *FACULTATIF* Implémentation MEX d'un RK(4)

MATLAB permet d'invoquer des routines codées en C/C++ comme s'il s'agissait de fonctions propres. Ceci présente le double intérêt de réutiliser du code existant sans devoir le retranscrire en fichiers `.m`, et d'implémenter en C/C++ certaines routines préalablement prototypées en MATLAB dont le temps d'exécution est critique. On crée des fichiers MEX (MEX-files, MEX = MATLAB EXecutable), objets binaires dont l'interpréteur MATLAB prend en charge l'édition de liens dynamique et l'exécution. La compilation de sources C/C++ peut s'effectuer depuis l'invite de MATLAB par la commande `mex`. Les fichiers MEX obtenus sont affectés d'une extension dépendant de l'architecture, en l'occurence `.mexglx` pour Linux 32bits et `.mexa64` pour Linux 64bits, et sont invoqués par leur racine, de la même manière que tout fichier `.m`[‡].

On se limitera à la création de fichiers MEX binaires en C. Ainsi, le fichier `nom_routine.ext` –où `ext` désigne l'extension dépendante de l'architecture– est créé sur la base de plusieurs fichier source. Parmi eux, `nom_routine.c` comporte la fonction principale `mexfunction`, équivalent du `main()` en C. `nom_routine.ext` est par conséquent obtenu par la commande de compilation

```
mex nom_routine.c autre_fichier1.c autre_fichier2.c ...
```

Un fichier typique `nom_routine.c` admet la forme suivante :

```
#include mex.h

/*
 * commentaires, autres fonctions \'eventuelles, etc.
 */

/* fonction principale */
void mexFunction(
    int           nlhs,    /* nombre de sorties */
    mxArray       *plhs[], /* vecteur de pointeurs vers les
arguments de sortie */
    int           nrhs,    /* nombre d'entr\'ees */
    const mxArray *prhs[]  /* vecteur de pointeurs vers les
arguments d'entr\'ee */
);
```

Pour permettre qu'un même code source soit portable sur de multiples architectures, plusieurs types et fonctions doivent être utilisées en lieu et place de leur équivalent C. Ainsi, toute donnée MATLAB est une structure de type `mxarray`. La librairie MX Matrix Library offre un ensemble de fonctions pour leur manipulation. La librairie MEX Library permet d'effectuer des opérations dans l'environnement MATLAB. L'Annexe A recense

‡. D'ailleurs, si un fichier `.m` et un fichier MEX admettent la même racine, *i.e.* ne diffèrent que par leur extension, alors l'interpréteur MATLAB évalue le MEX.

quelques-un(e)s de ces types et fonctions. Vous trouverez davantage d'information sur `http://www.mathworks.fr/help/techdoc`, onglet User's Guide, sous-onglets External Interfaces et C/C++ and Fortran API Reference.

À titre d'exemple, considérons le problème du calcul du produit d'un vecteur ligne par un scalaire selon le prototype

```
outMatrix = arrayProduct(multiplier, inMatrix)
```

où l'invocation de `arrayProduct` se réfère au fichier MEX `arrayProduc.ext`. Un fichier `arrayProduct.c` qui, une fois compilé par la commande `mex`, conduirait à `arrayProduct.ext` est fourni par The MathWorks, Inc. dans la distribution de MATLAB. Nous en proposons une version condensée dans l'Annexe B.

- en C la routine interne `unPasRK4enC` qui réalise un pas du RK(4) pour la fonction $f(.,.)$ définie en (4) ; cette routine peut par exemple obéir au prototype
  ```
  void unPasRK4enC(double tk, double *Yk, double h, double *tnext,
                                                    double *Ynext)
  ```
- dans un fichier `RK4enC.c` la fonction principale permettant d'implémenter le RK(4) global pour $f(.,.)$, selon le prototype
  ```
  [Tout,Yout] = RK4enC(T0, TK, h, Y0)
  ```
- Vérifier le fonctionnement correct de `unPasRK4enC` par un test unitaire. Compiler [§] `RK4enC.ext`, puis vérifier son fonctionnement correct sur un scenario déjà étudié sous MATLAB.
- Comparer les temps d'exécution (fonctions `tic` et `toc`).

ANNEXES

Annexe A

Voici un sous-ensemble des types et fonctions offerts par les MX Matrix Library et MEX Library. Précéder éventuellement de `#include <stdlib.h>` et/ou `#include "matrix.h"`.

`mxArray` type (unique) de la structure contenant toute donnée sous MATLAB
`mwSize` type des dimensions de tableaux
`mwIndex` type des indices de tableaux
`mwSignedIndex` type des indices signés de tableaux
`typedef enum mxComplexity {mxREAL=0, mxCOMPLEX}`
caractère complexe ou non des valeurs d'un tableau
`typedef enum {mxUNKNOWN_CLASS, mxCELL_CLASS, mxSTRUCT_CLASS, mxLOGICAL_CLASS,`

§. Si besoin, configurer l'installation au moyen de `mex -setup`, cf. `help mex` pour davantage d'information.

```
mxCHAR_CLASS, mxVOID_CLASS, mxDOUBLE_CLASS, mxSINGLE_CLASS,
mxINT8_CLASS, mxUINT8_CLASS, mxINT16_CLASS,
mxUINT16_CLASS, mxINT32_CLASS, mxUINT32_CLASS, mxINT64_CLASS,
mxUINT64_CLASS, mxFUNCTION_CLASS} mxClassID
```
classes possibles d'un tableau

`void *mxMalloc(mwSize n)`
alloue dynamiquement un bloc d'octets
`void *mxCalloc(mwSize n, mwSize size)`
alloue dynamiquement un bloc d'éléments (*e.g.* en conjonction de `sizeof(k*mwSize)`)
`void *mxRealloc(void *ptr, mwSize size)`
ré-alloue un bloc mémoire alloué par `mxCalloc`, `mxMalloc`, `mxRealloc`
`void mxFree(void *ptr)`
libère un bloc mémoire alloué par `mxCalloc`, `mxMalloc`, `mxRealloc`

`mxArray *mxCreateDoubleMatrix(mwSize m, mwSize n, mxComplexity ComplexFlag)`
crée un tableau 2-D de doubles et l'initialise à 0
`mxArray *mxCreateDoubleScalar(double value)`
`pa = mxCreateDoubleScalar(value)`
est équivalent à
`pa = mxCreateDoubleMatrix(1,1,mxREAL); *mxGetPr(pa) = value;`
`mxArray *mxCreateNumericArray(mwSize ndim, const mwSize *dims,`
`mxClassID classid, mxComplexity ComplexFlag)`
crée un tableau de doubles multidimensionnel et l'initialise à 0
`mxArray *mxCreateNumericMatrix(mwSize m, mwSize n, mxClassID classid,`
`mxComplexity ComplexFlag)`
crée un tableau de doubles multidimensionnel et l'initialise à 0
`void mxDestroyArray(mxArray *pm)`
libère la mémoire allouée par `mxCreate*`
`mxArray *mxDuplicateArray(const mxArray *in)`
duplique un tableau

`xClassID mxGetClassID(const mxArray *pm)`
retourne la classe d'un tableau
`bool mxIsNumeric(const mxArray *pm)`
`bool mxIsComplex(const mxArray *pm)`
`bool mxIsDouble(const mxArray *pm)`
`bool mxIsEmpty(const mxArray *pm)`
`bool mxIsFinite(const mxArray *pm)`
retourne le caractère numérique / complexe / double / vide des valeurs d'un tableau `mwSize mxGet`
`size_t mxGetNumberOfElements(const mxArray *pm)`
retourne le nombre de dimensions / nombre total d'éléments d'un tableau
`size_t mxGetM(const mxArray *pm)`
`size_t mxGetN(const mxArray *pm)`
retourne le nombre de lignes / nombre de colonnes d'un tableau `double *mxGetPr(const mxArray`
`double *mxGetPi(const mxArray *pm)`
pointe vers les éléments réels / éléments complexes d'un tableau de type `DOUBLE`
`double mxGetScalar(const mxArray *pm)`
pointe vers la valeur du premier élément d'un tableau

`double mxGetEps(void)`
`double mxGetInf(void)`
`double mxGetNaN(void)`
valeurs de `eps`, `Inf`, `NaN`
`bool mxIsFinite(double value)`
`bool mxIsInf(double value)`
`bool mxIsNaN(double value)`
caractère fini / infini / NaN d'une valeur
`void mexErrMsgTxt(const char *errormsg)`
affiche un message d'erreur et retourne à l'invite MATLAB
`void mexErrMsgIdAndTxt(const char *errorid, const char *errormsg, ...);`
affiche un message d'erreur avec identifiant et retourne à l'invite MATLAB
`const char *mexFunctionName(void)`
retourne nom de la fonction MEX courante
`int mexPrintf(const char *message, ...)`
affiche un message a-la-`printf` de C ANSI

Annexe B

```
/*=================================================================================================
 * myarrayProduct.c : outMatrix = myarrayProduct(multiplier, inMatrix)
 * Réalise le produit d'un scalaire (multiplier) par une matrice 1xN (inMatrix), de faÁon à obtenir
 * une matrice 1xN (outMatrix)
 * Le fichier originel arrayProduct.c obéit au copyright suivant :
 * Copyright 2007-2008 The MathWorks, Inc.
 *================================================================================================*/

#include "mex.h"

/* Routine C interne, effectuant la multiplication de chaque élément de y par x */
void arrayProduct(double x, double *y, double *z, mwSize n)
{
    mwSize i; for (i=0; i<n; i++) { z[i] = x * y[i]; }
}

/* Fonction principale du fichier MEX */
void mexFunction( int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    double multiplier;              /* scalaire d'entrée */
    double *inMatrix;               /* matrice 1xN d'entrée */
    mwSize ncols;                   /* taille de la matrice */
    double *outMatrix;              /* matrice 1xN de sortie */

    /* quelques tests facultatifs */
    if(nrhs!=2) { mexErrMsgIdAndTxt("MyToolbox:arrayProduct:nrhs","Two inputs required."); }
    if(nlhs!=1) { mexErrMsgIdAndTxt("MyToolbox:arrayProduct:nlhs","One output required."); }
    if( !mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) || mxGetNumberOfElements(prhs[0])!=1 ) {
        mexErrMsgIdAndTxt("MyToolbox:arrayProduct:notScalar","Input multiplier must be a scalar.");
    }
    if(mxGetM(prhs[1])!=1) {
        mexErrMsgIdAndTxt("MyToolbox:arrayProduct:notRowVector","Input must be a row vector.");
    }

    /* obtention de la valeur de l'entrée scalaire  */
    multiplier = mxGetScalar(prhs[0]);
    /* création d'un pointeur vers les valeurs réelles dans la matrice d'entrée */
    inMatrix = mxGetPr(prhs[1]);
    /* obtention des dimensions de la matrice d'entrée */
    ncols = mxGetN(prhs[1]);
    /* création de la matrice de sortie */
    plhs[0] = mxCreateDoubleMatrix(1,ncols,mxREAL);
    /* obtention d'un pointeur vers les valeurs réelles dans la matrice de sortie */
    outMatrix = mxGetPr(plhs[0]);
    /* appel de la routine sous-jacente */
    arrayProduct(multiplier,inMatrix,outMatrix,ncols);
}
```

Annexe C

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [sys,x0,str,ts] = mycsfunc(t,x,u,flag)
%MYCSFUNC S-fonction MATLAB permettant de coder
%      x' = Ax + Bu
%      y  = Cx + Du
%      x(0) = x0.
%   Voir également le fichier sfuntmpl.m pour des S-fonctions plus générales.
%   Le fichier originel csfunc.m obéit au copyright suivant :
%   Copyright 1990-2009 The MathWorks, Inc.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
g = 9.81; l = 1; w0 = sqrt(g/l);
A = [0 1 ; -w0^2 0]; B = [0 ; 1]; C = [1 0]; D = 0;
x_init = [3 ; 0];

switch flag,
  case 0, % Initialisation
    [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D,x_init);
  case 1, % Calcul des dérivées
    sys=mdlDerivatives(t,x,u,A,B,C,D);
  case 3, % Calcul des sorties
    sys=mdlOutputs(t,x,u,A,B,C,D);
  case { 2, 4, 9 }, % Valeurs "hors sujet" de la variable flags
    sys = [];
  otherwise % Cas d'erreurs
    DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));
end
% end csfunc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% mdlInitializeSizes
% Retourne les tailles, conditions initiales, périodes de simulation/d'échantillonnage de la S-fonction.
function [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D,x_init)
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0; % car D = 0
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = x_init;
str = [];
ts  = [0 0]; % système à temps continu
% end mdlInitializeSizes

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% mdlDerivatives
% Calcul de la dérivée de x(t)
function sys=mdlDerivatives(t,x,u,A,B,C,D)
sys = A*x + B*u;
% end mdlDerivatives

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% mdlOutputs
% Calcul du vecteur de sortie y(t)
function sys=mdlOutputs(t,x,u,A,B,C,D)
sys = C*x + D*u;
% end mdlOutputs

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```