Projet en langage C:
Modélisation de la trajectoire d'un point

L2 CUPGE

Année 2018–2019

1 Introduction

Le but de ce projet est de permettre de représenter graphiquement la trajectoire d'un point étant données les équations aux dérivées partielles définissant cette trajectoire. Lorsque le point est soumis à des forces dépendant uniquement de sa position et de sa vitesse, le Principe Fondamental de la dynamique implique qu'il est possible de calculer sa trajectoire en fonction de ces deux paramètres. Dans ce projet, on part des équations définissant la vitesse en fonction de la vitesse et de la position, et on met à jour la position du point en fonction de la vitesse et d'un petit incrément de temps paramétrable dt. Par exemple, pour les équations de Lorenz :

$$\begin{cases} \frac{dx}{dt} &= \sigma(y-x) \\ \frac{dy}{dt} &= x(\rho-z)-y \\ \frac{dz}{dt} &= xy-\beta z \end{cases}$$

et avec $dt = 0.01$, un point dont la position est $\vec{u} = (x, y, z)$ et la vitesse est $\vec{v} = (\frac{dx}{dt}, \frac{dy}{dt}, \frac{dy}{dt})$ à l'instant t aura, à l'instant $t + dt$, la position $(x + \frac{dx}{dt} \cdot dt, y + \frac{dy}{dt} \cdot dt, z + \frac{dz}{dt} \cdot dt)$ et la vitesse $(\sigma(y-x), x(\rho-z)-y, xy-\beta z)$. Les paramètres σ, ρ, et β sont fixés pour chaque trajectoire calculée.

Le programme demandera à l'utilisateur des paramètres pour le calcul de la trajectoire, l'incrément dt, la position initiale (x, y, z) (on suppose que le point est initialement stationnaire), et le temps d'arrêt $T_{\max}$. Il calculera ensuite toutes les coordonnées des points entre $t = 0$ et $t = T_{\max}$ (attention, il y en a $\frac{T_{\max}}{dt}$) et les stockera dans un fichier dans le format :

temps `x` `y` `z`

La trajectoire sera tracée avec `gnuplot`, en utilisant les commandes :

```
> set parametric
> splot "lorenz.dat" u 2:3:4
```

si le fichier de sortie s'appelle `lorenz.dat`.

2 Objectifs

L'objectif de base du projet est de parvenir à afficher la courbe correspondant aux coordonnées initiales $(1, 2, 3)$ pour les équations de Lorenz avec les paramètres $\sigma = 10, \rho = 28, \beta = \frac{8}{3}$. Au-delà de cet objectif de base, plusieurs extensions sont possibles, donc (liste non-exclusive) :

- (facile) Demander les paramètres et la position initiale à l'utilisation ;
- (facile) Créer plusieurs fonctions pour donner à l'utilisateur le choix entre différents systèmes dynamiques ;
- (modéré) Écrire des modules lisant les paramètres et effectuant la mise à jour d'un point pour différents systèmes. Il faut pour cela utiliser une structure sera commune à ces modules et dont les membres sont les fonctions d'initialisation, de mise à jour des coordonnées, ainsi que les paramètres eux-mêmes.
- (modéré–difficile) Demander à l'utilisateur de définir les paramètres et la mise à jour de la vitesse, et tracer ensuite la courbe.
- (difficile) Utiliser une bibliothèque d'interfaçage entre C et Gnuplot pour lancer les commandes et tracer la graphique à partir du programme.
- (difficile) Lire la documentation de Gnuplot pour réussir à afficher (au choix) :
  - la vitesse par des vecteurs ;
  - la vitesse par une couleur ;
  - le temps par un changement de couleur.
- (au-delà du cours) Lancer le programme gnuplot à partir du programme créé pour qu'il affiche la courbe sans avoir besoin de repasser par la ligne de commande.

Le premier point est "requis" pour avoir la moyenne, mais pas pour pouvoir rendre le projet. Le second est un bonus facile à obtenir une fois que les fonctions pour le cas des équations de Lorenz ont été écrites. Le troisième point est plus difficile, mais ne va pas au-delà de ce qui a été vu en cours/TP. La difficulté principale est dans la définition de la structure qui sera commune à tous les modules.

On rentre dans le C avancé à partir du quatrième point. Une possibilité est de demander à l'utilisateur de rentrer les formules en notation polonaise inversée, avec des paramètres sur une lettre, et ensuite, pour la mise à jour de la vitesse, de parcourir la chaîne de caractère entrée pour l'utilisateur pour construire une liste des valeurs. Par exemple, avec la chaîne :

```
"s x y - *"
```

on part de la liste vite, on ajoute s, puis x, puis y (pour obtenir la liste des valeurs de $[y; x; s]$. Lorsqu'on arrive au caractère $-$, on lit les deux premiers éléments et on ajoute leur différence pour obtenir la liste $[y - x; s]$. Lorsqu'on arrive au caractère $-$, on lit les deux premiers éléments et on ajoute leur différence pour obtenir la liste $[(y - x) * s]$. Arrivé à la fin de la chaîne de caractères, on rend la valeur de son premier élément.

Une bibliothèque d'interfaçage entre C et `Gnuplot` a été écrite par Nicolas

Devillard et est téléchargeable sur Internet. La difficulté ici consiste surtout à comprendre, à partir de la documentation, comment elle fonctionne pour pouvoir l'utiliser. De même, la difficulté du 6ème point est la difficulté de la lecture de la documentation de `Gnuplot`. Pour ces 2 problèmes, il est important de réaliser que la lecture et la compréhension d'une documentation sont des parties très importantes de la programmation (et plus généralement des métiers d'ingénierie).

Le dernier point est difficilement classable. Il est relativement simple une fois que la manière dont les programmes sont créés sous Unix est bien connue, mais nous ne les aurons pas vus dans ce cours. Ce point consiste donc en une auto-formation à la programmation système.

Important. Les points listés comme difficiles peuvent prendre beaucoup de temps. Il est vivement recommandé de ne les aborder qu'une fois les points précédents écrits, et s'il reste un temps raisonnable avant de rendre le projet.

3 Modalités

Le projet doit être fait par binômes. Chaque binôme doit :

— mettre en place un dépôt privé auquel seuls les membres du groupe et l'enseignant de TP peut accéder ;

— faire des *commit* réguliers pour qu'il soit possible d'évaluer qui a contribué à quels aspects du projet.

Le projet est en 2 phases : à la fin de la première phase le groupe doit avoir un plan pour pouvoir guider l'écriture du code, et le code source du programme doit être rendu à la fin de la seconde phase.

3.1 Première phase : Description du projet

Date de retour : 16 novembre 2018

On demande au groupe de fournir un plan détaillé pour l'architecture du code source du programme. Même si ça n'a pas encore été vu au moment de la conception, vous serez capable, au moment de l'écriture :

— de lire et écrire des données dans des fichiers ;

— de définir des types de données regroupant plusieurs valeurs (comme un triplet de `float`, etc.) ;

— d'écrire des tests permettant de savoir si les fonctions que vous écrirez sont correctes ;

— d'écrire une documentation.

Guide pour l'architecture du répertoire. Idéalement, le nom de votre dépôt sera de type `Dupont-Dupond` (les noms des membres). Il contiendra au moins des sous-répertoires `src`, `lib`, `include`, `bin`, et `doc`, et un fichier `Makefile`[1] permettant de compiler, tester, et exécuter votre programme.

1. L'écriture de ce type de fichiers sera vue en TP

Rapport de fin de première phase. Le rapport doit contenir un plan très détaillé de ce que vous allez faire. **Une liste n'est pas suffisante**. Vous devez organiser votre projet autour de plusieurs librairies (que vous définirez), et ce plan doit comporter une section par librairie. Chaque section doit contenir un court paragraphe de description générale sur le thème commun aux fonctions et types de cette librairie. Ensuite, pour chaque type et fonction déclarée, d'une description avec :

— la déclaration de la fonction ;
— Une description de ce que fera la fonction,
— Si la fonction a besoin d'autres fonctions que vous aurez à écrire pour ses calculs, une description des données que vous passerez à ces fonctions, et la liste des fonctions que vous utiliserez.

Il faut veiller à ce que les données passées aux fonctions aient un type compatible avec la déclaration de cette fonction !

Une section supplémentaire sera utilisée pour décrire les dépendances entre librairies, et l'organisation du programme principal.

Ce rapport sera noté en fonction de la profondeur de votre réflexion sur les difficultés que vous pourrez avoir et les solutions que vous proposez.

3.2 Seconde phase : le programme

Date de retour : 7 décembre 2018

Le projet final **doit** contenir :

— un fichier Makefile permettant de compiler le programme ;
— un sous-répertoire `doc` contenant :
  — la description initiale du projet (le rapport de la phase 1)
  — un fichier indiquant comment utiliser le programme (entre un paragraphe et une page, sauf si le programme est extrèmement compliqué) ;
  — un second rapport indiquant les changements par rapport à l'architecture que vous aviez décrite dans le premier rapport ;
— les fichiers sources, organisés comme il vous plaira.

Il est normal, et encore plus pour un premier projet de programmation, d'avoir des différences entre ce qu'on pensait faire et ce qu'on a vraiment fait. Le second rapport sert à justifier les changements que vous avez fait !

3.3 Barême

Le barême suivant est indicatif...

Documentation :

Dépôt : Mise en place correcte et partage 3, utilisation fréquente de git 2 ;

Pré-rapport : Présentation du rapport 2, complétude (de la description des fonctions, des modules, etc.) 2, clarté des explication sur les choix

(de découpage, d'options choisies, de structures de données choisies, etc.) 2, richesse de la réflexion 2 ;

Rapport final : explication des corrections par rapport aux sources du programme final 5, manuel d'utilisation 2, présentation/clarté 2 ;

Programme : documentation du code 2, Makefile 3, compilation 2, tests 2 qualité de la décomposition en librairies 5, contenu 2× nombre d'options au-delà de la deuxième, clarté du code 2 ;

La facilité d'extension est fonction de la modularité et du découpage du code en petites unités (libraries, fichiers .c, fonctions) qu'il est facile de changer si on veut ajouter des options supplémentaires. Un extrême, où tout le code est dans la fonction `main`, correspond à une note de 0. La clarté du code est jugée sur la non répétition de morceaux de code similaires qui pourraient être remplacé par un morceau de code commun (avec des paramètres supplémentaires, en général).

Le barême est normalisé à 20 en divisant par 2 (ce qui correspond à 2 options).