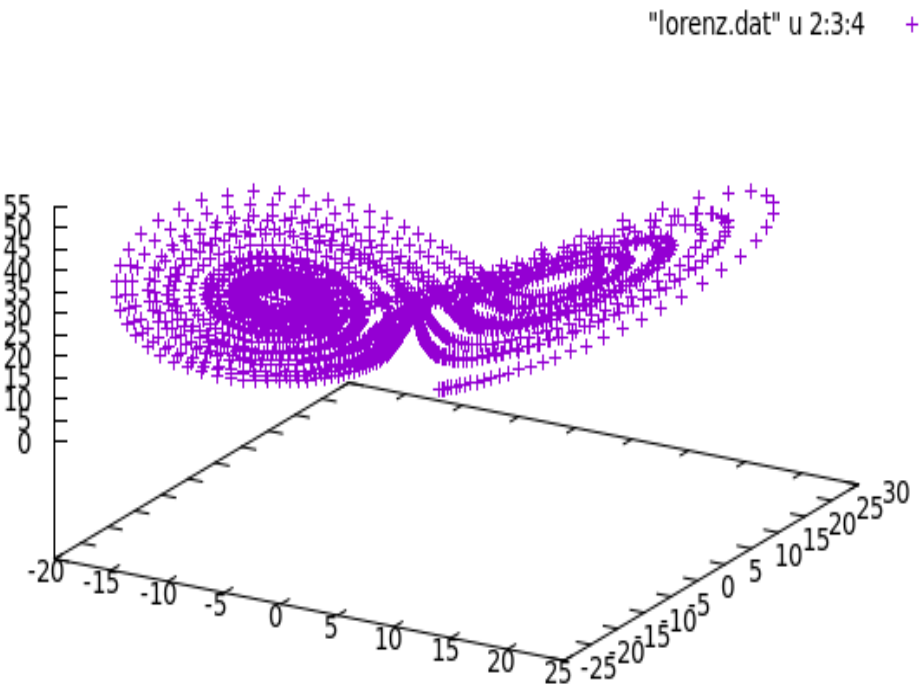
UNIVERSITÉ TOULOUSE III PAUL SABATIER

Projet en langage C :

MODELISATION DE LA TRAJECTOIRE D'UN POINT



MOUDDENE Hamza
SANZ -- SOUHAIT Corina

L2 CUPGE Année 2018–2019

Table des matières :

I. Introduction

Le projet consiste à calculer et à afficher la trajectoire d'un point pour différents systèmes dynamiques grâce à des paramètres choisis par l'utilisateur.
Notre programme implémenté en langage C propose donc à l'utilisateur le choix entre dix systèmes dynamiques qui se basent sur la théorie des chaos (les attracteurs de : Lorenz, Euler, Aizawa, Anishchenko, Hoover, Rossler, Coullet, Hadley, Rayleigh et Bouali) L'utilisateur à ensuite le choix d'utiliser les paramètres entrés par défaut ou de les rentrer par lui-même.

II. Les changements par rapport à la première phase :

Les changements de notre programme par rapport à la première phase et à nos attentes auraient pu être des difficultés et donc des abandons d'idées ; mais nous avons finalement rempli tous les objectifs que nous nous étions fixés et même plus encore.

Nous pensions seulement afficher la courbe de Lorenz (et un peu plus tard la courbe d'Euler) tout en implémentant des structures dans le code et en affichant non seulement la courbe des points mais aussi les vecteurs vitesse de chacun d'entre eux. Nous avions donc deux sous-dossiers pour Lorenz et de même pour Euler (un pour les .h l'autre pour les .c). Nous n'étions pas très organisés dans la structure du programme puisqu'il il fallait se rendre dans le dossier de Lorenz où se trouvait une fonctions main_lorenz.c pour exécuter Lorenz ; et dans celui d'Euler où se trouvait une fonction main_euler.c pour exécuter Euler ; ce n'était pas pratique pour l'utilisateur. Cependant, nous avons préféré organiser, structurer mais surtout optimiser le code afin qu'à partir d'une seule fonction main.c contenu dans un dossier à part des systèmes (main_src) nous puissions lancer les deux programmes. Nous avons donc créé des modules communs à chaque système dynamique.

Finalement nous avons rempli tous nos objectifs de bases et sommes allé encore plus loin en implémentant dix systèmes dynamiques, en structurant et optimisant le code. Il a fallu créer plusieurs Makefile du fait d'avoir des modules, afin de compiler séparant chacun d'entre eux. Nous avons aussi ajouté les **mallocs** dans nos structures.

III. Guide pour l'architecture du répertoire :

La première étape nécessaire afin de réaliser un projet informatique est de modéliser une architecture optimale. Le projet est un répertoire nommé MOUDDENE_SANZ--SOUHAIT (les noms des membres) qui représente une structure hiérarchique contenant plusieurs sous-répertoires et fichiers :

```
apple@apples-macbook-10:~/Desktop/mouddene_sanz-souhait$ tree
.
├── README.md
├── bin
├── documents
│   ├── project.pdf
│   └── rapport.pdf
├── include
│   ├── calcul_lib
│   │   ├── calcul.h
│   │   └── dynamicSystems.h
│   ├── displayShell_lib
│   │   └── displayShell.h
│   ├── file_lib
│   │   └── file.h
│   ├── gnuplot_lib
│   │   └── gnuplot.h
│   ├── input_lib
│   │   ├── input.h
│   │   └── inputSystems.h
│   └── struct_lib
│       └── struct.h
├── lib
└── src
    ├── calcul_src
    │   ├── Makefile
    │   ├── calcul.c
    │   └── dynamicSystems.c
    ├── displayShell_src
    │   ├── Makefile
    │   └── displayShell.c
    ├── file_src
    │   ├── Makefile
    │   └── file.c
    ├── gnuplot_src
    │   ├── Makefile
    │   └── gnuplot.c
    ├── input_src
    │   ├── Makefile
    │   ├── input.c
    │   └── inputSystems.c
    ├── main_src
    │   ├── Makefile
    │   └── main.c
    └── struct_src
        ├── Makefile
        └── struct.c

18 directories, 27 files
```

- **Readme.md** : le fichier qui explique à un utilisateur les packages qu'il faut installer et la notice d'utilisation de notre programme.

- **bin :** il s'agit du répertoire qui contient le fichier exécutable d'où l'on peut exécuter le programme.

- **documents :** il s'agit du répertoire contenant l'énoncé du projet ainsi que le rapport associé à ce dernier sous format PDF.

- **include :** il s'agit du répertoire contenant l'ensemble des fichiers .h organisés dans des librairies modules. Ces fichiers ne contiennent pas de code mais seulement les prototypes de fonctions et la déclaration des objets de type « **struct** ».

- **lib :** il s'agit du répertoire contenant les fichiers .o qui sont fichiers objets contenant la compilation des fichiers .c (langage-computer).

- **src :** il s'agit du répertoire contenant les modules qui contiennent les fichiers .c où se trouve le code source. Il s'y trouvera aussi les fichiers **Makefile** qui ont pour fonction d'organiser et d'automatiser le processus de la compilation et de l'exécution où il suffit de taper la commande « **make all**» afin d'exécuter le programme. (nous verrons cela plus bas). Chaque module comporte son propre Makefile qui permet de compiler séparément chaque fichier. Les modules présents sont les suivants :

- **calcul_src** : Il s'agit du module contenant les fichiers dynamicsSystems.c (contenant les fonctions calculant la trajectoire d'un point pour un des dix systèmes dynamiques possibles), ainsi que calcul.c (contenant les fonctions qui font appel aux fonctions décrites précédemment suivant le système dynamique choisi par l' utilisateur. Ces fonctions font la liaison entre le fichier principale du code et le ficher où se trouve l'implémentation des systèmes dynamiques).
- **displayShell_src :** Il s'agit du module contenant le fichier displayShell.c qui se charge de l'affichage des différentes étapes d'exécution dans le Shell.
- **file_src :** Il s'agit du module contenant le fichier file.c qui affecte au fichier du stockage le nom du système dynamique choisi afin de créer deux fichiers : « système »_coordinates.dat et « système »_vectors.dat où l'on stocke les coordonnées de chaque point de la trajectoire.
- **gnuplot_src :** Il s'agit du module contenant le fichier gnuplot.c qui récupère les fichiers où les coordonnées de chaque point sont stockées puis en permet l'affichage.
- **input_src :** Il s'agit du module contenant les fichiers inputSystems.c (qui contient l'input de chaque système dynamique avec l'option d'input automatique qui permet à l'utilisateur d'exécuter le programme avec des

paramètres par défaut) et input.c (qui contient les fonctions génériques du programme qui permettent d'assurer le bon fonctionnement du module input_src).

- **struct_src :** il s'agit du module contenant le fichier struct.c où se trouve l'algorithme associe à l'objet struct.
- **main_src :** le module générique de tout le projet contenant le fichier main.c qui permet de relier tous les modules entre eux et construire l'intégralité du programme.

IV. Code source :

1. INITIALISATION DE VARIABLES (POUR UN SYSTEME DYNAMIQUE DONNE) :

```
Typedef struct position {

        double *position; //position est un tableau de doubles contenant les degrés de libertés.

        double *speed_t; //speed_t est un tableau de doubles contenant les coordonnées de chacun des points de la trajectoire

        double * speed; //speed est un tableau de doubles contenant la vitesse initiale pour des coordonnées données

        double dt; //dt est un double représentant l'écart de temps entre deux points.

        double tmax; // tmax est un double représentant le temps au bout duquel on souhaite que la trajectoire se termine.

} ;


Typedef struct parameter{

        double *parameter; //parameter est un tableau de doubles contenant les paramètres pour un système dynamique donné.

} ;

char *flag; //Les systèmes dynamiques sont numérotés de 1 à 10 ; le flag est une chaine de caractère où l'on va stocker le numéro du système dynamique associé.

char *file_name; // file_name est une variable où l'on stocke le nom du système dynamique choisit.
```

```
char *point_file; //C'est la variable où l'on fait la concaténation du nom du système et de « _coordinates.dat». Exemple pour Lorenz : lorenz_coordinates.dat »

char *vector_file;// C'est la variable où l'on fait la concaténation du nom du système et de « _vectors.dat ». Exemple pour Lorenz : lorenz_vectors.dat »
```

2. IMPLEMENTATION DE FONCTIONS :

L'objectif est d'organiser toutes ces données dans différentes **fonctions** ; cette méthode est très utile surtout dans les grands projets ou les projets professionnels, elle permet aux développeurs d'avoir une structure cohérente et sophistiquée. Nous avons donc utilisé ce concept afin d'avoir une idée claire d'un plan bien détaillé de ce projet. Les fonctions sont créées dans des fichiers .c et sont déclarées dans des fichiers .h, elles sont ensuite appelées dans d'autres fichiers .c. Les fonctions sont réparties dans les modules suivants :

1) calcul_src :

a) calcul_src/dynamicSystems.c

```
/*Dans ce fichier se trouve toutes les fonctions de calcul pour chaque système dynamique ; deux fonctions par système : une pour calculer la vitesse initiale et l'autre pour calculer les coordonnées en chaque point de la trajectoire. En voici un exemple général : */

-----------------------------------------------------------------------------------------

/*fonction qui calcule les coordonnées en chaque point de la trajectoire de l'attracteur d'un système */
void « système »_t(double speed_t[], double position[], double parameter[], double *dt) ;

/*fonction qui calcule la vitesse initiale de l'attracteur d'un système*/
void « système »_init(double speed[], double position[], double parameter[]) ;
```

b) calcul_src/calcul.c

```
/*Cette fonction appelle la fonction précédente qui calcule la vitesse initiale par rapport au système choisit par l'utilisateur ; grâce au flag. */
void initial_speed(double speed[], double position[], double parameter[], char flag[] ;
```

```
/*Cette fonction appelle la fonction précédente qui calcule la vitesse en chaque point de la trajectoire par rapport au système choisit par l'utilisateur ; grâce au flag. */
void instant_speed(double speed_t[], double position[], double parameter[], double *dt, char flag[]) ;
```

2) displayShell_src/displayShell.c :

```
/*Cette fonction est une fonction d'affichage ; elle affiche les différents systèmes dynamiques que le programme est en capacité d'afficher*/
void launch_program() ;

/*display_init est seulement une fonction d'affichage qui s'adapte au nom du système choisit par l'utilisateur*/
void display_init(char file_name[]) ;

/*Cette fonction affiche sur le Shell la vitesse initiale en fonction du système dynamique choisit*/
void display_speed(double speed[], char flag[]) ;

/*Cette fonction affiche chaque coordonnées - ainsi que le temps auxquelles elles sont associées – dans le Shell*/
void display_coordinates(double position[], double *i, char flag[]) ;

/*display_end est seulement une fonction d'affichage de fin de programme*/
void display_end() ;
```

3) file_src/file.c :

```
/*Cette fonction fixe le nom du système dynamique choisit grâce au flag pour pouvoir faire des fonctions générales et communes à tous les systèmes dynamiques. */
char f_name(char file_name[], char flag[]) ;

/*Cette fonction concatène le nom du système dynamique choisit avec « _coordinates.dat »*/
void p_file(char file_name[], char point_file[], char flag[]) ;

/*Cette fonction concatène le nom du système dynamique choisit avec « _vectors.dat »*/
void v_file(char file_name[], char vector_file[], char flag[]) ;
```

```
/*Cette fonction initialise et ouvre les fichiers « système »_coordinates.dat et
« système »_vectors.dat, écrit les coordonnées dedans puis le ferme*/
void file(double position[], double speed_t[], double *i, char flag[], char point_file[],
char vector_file[]) ;
```

4) gnuplot_src/gnuplot.c :

```
/*Cette fonction trace la courbe de la trajectoire du système dynamique choisit par
l'utilisateur*/
void gnuplot_point(char flag[], char point_file[]) ;

/*Cette fonction trace les vecteurs vitesse de la trajectoire du système dynamique
choisit par l'utilisateur */
void gnuplot _vector(char flag[], char vector_file[]) ;
```

5) input_src :

a) input_src/inputSystems.c

```
/*Cette fonction est une fonction qui permet à l'utilisateur, de taper un nombre
seulement entre 1 et 10*/
void type_flag(char flag[]) ;

/*Cette fonction demande à l'utilisateur s'il souhait exécuter le programme avec
ou sans les paramètres par défaut. Elle détecte aussi les fautes de frappe dans le
terminal */
void default_parameters(char by_default[]) ;

/*Cette fonction prend en compte de choix de l'utilisateur afin d'obtenir les inputs
du bon système dynamique*/
void coordinates(double position[], char flag[], char by_default[]) ;

/*Ces trois dernières fonctions, suivant le choix de l'utilisateur, utilisent les
paramètres par défaut du système choisit ou laisse l'utilisateur le faire
manuellement*/
void constants(double parameter[], char flag[], char by_default[]) ;
void increment(double *dt, char by_default[]) ;
void break_time(double *tmax, double *dt, char by_default[]) ;
```

b) input_src/input.c

```
/*Dans ce fichier se trouve toutes les fonctions d'input de chaque système dynamique ; deux fonctions par système : une pour les coordonnées initiales et une pour les paramètres. L'input peut soit être automatique, soit manuel suivant le choix de l'utilisateur ; cette différence est faite au sein de ces fonctions. Voici un exemple généralisé : */
void « système »_coordinates(double position[]) ;
void  « système »_constants(double parameter[], char by_default[]) ;
```

3. IMPLEMENTATION DE BIBLIOTHEQUES DE BASES OU CREEES :

Pour raccorder les parties du projet, on utilise souvent des bibliothèques déjà faites ou que nous construisons nous-même (ce qui est le cas dans ce projet).

1- Bibliothèque de bases

- stdio.h (standard input output) : il s'agit de la bibliothèque système du langage C contenant les fonctions de bases (printf, scanf, etc…). Ce fichier contient seulement les déclarations des fonctions, le code de ces fonctions est écrit ailleurs. Elle est présente dans tous les fichiers de notre programme.

- stdlib.h : il s'agit de la bibliothèque qui nous permet d'utiliser la fonction malloc() et l'implémenter dans l'objet struct. De plus nous avons utilisé la fonction snprintf() dans le fichier src/input_src/input.c qui convertit un double en chaîne de caractère.

- string.h : il s'agit de la bibliothèque système qui nous permet d'utiliser les fonctions **strcmp**() (c'est la fonction qui permet de comparer deux chaîne de caractère), **strcat**() (c'est la fonction qui permet de faire la concaténation de deux chaîne de caractère) et **strcpy**() (c'est la fonction qui permet de copier une chaîne de caractère dans une autre). Elles sont toutes les trois présentes dans le fichier file_src/file.c. strcmp() est aussi présente dans les fichiers input_src/inputSystems, input_src/input.c, calcul_src/calcul.c, displayShell_src/displayShell.c, gnuplot_src/gnuplot.c.

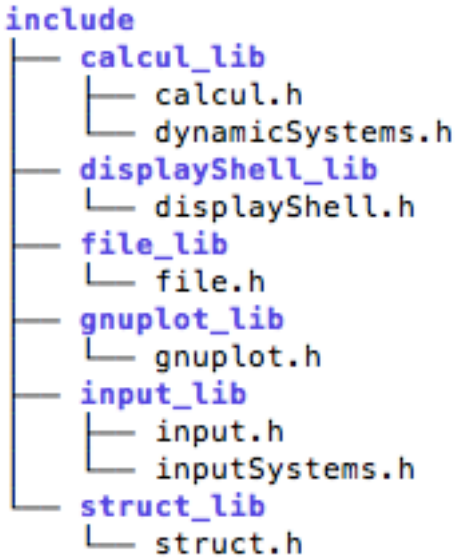
- math.h : il s'agit de la bibliothèque contenant toutes les fonctions mathématiques. On s'en est servi pour utiliser les fonctions sinusoïdales ainsi que pour les puissances dans le fichier calcul_src/dynamicSystems.c

2- Bibliothèques créées

Chaque fichier .c à son propre fichier .h (à l'exception du fichier main.c) où seules les déclarations des fonctions s'y trouvent.

- **calcul.h**
- **dynamicSystems.h**
- **displayShell.h**
- **file.h**
- **gnuplot.h**
- **input.h**
- **inputSystems.h**
- **struct.h** : il s'agit du fichier où les structures sont définies.

Tous les fichiers .h sont placés dans le dossier **include**, ils sont tous regroupés dans des modules exactement comme les fichiers .c.



Le fichier **main.c** regroupe tous les appels de fonction, toutes les bibliothèques du programme (bibliothèques crées : fichiers .h) y sont donc implémentées.

```c
#include <stdio.h>
#include "input.h"
#include "calcul.h"
#include "gnuplot.h"
#include "file.h"
#include "displayShell.h"
#include "struct.h"
#include "dynamicSystems.h"
#include "inputSystems.h"
```

V. Compilation et exécution :

Pour exécuter le programme facilement, nous allons utiliser la compilation séparé grâce à des Makefiles, ainsi, chaque module aura son propre Makefile qui compilera le fichier en lui attribuant un .o (avec la commande make compile) et le placera dans le dossier main_src (avec la commande make move). Chaque Makefile comprend une commande « make all » qui permet de faire directement les étapes « compile » et « move » à la suite.

Exemple d'un fichier Makefile (calcul_src) :

```
GCC = gcc
INCLUDE = -I ../../include/calcul_lib/ -I ../../include/struct_lib/
SRC = $(wildcard *.c)
OBJ = $(SRC:.c=.o)

all : compile move

compile : $(SRC)
	$(GCC) $(INCLUDE) -c $(SRC)

move : $(OBJ)
	mv $(OBJ) ../main_src/
```

Maintenant que tous les fichiers .c ont été compilés et ont un fichier .o placé dans main_src, il faut les regrouper afin de permettre la compilation du fichier général main.c en générant un fichier .o.

Le Makefile dans main_src est, comme les précédents Makefile, composé de plusieurs commandes (à taper dans le Shell) :

- make build : exécute chaque Makefile de chaque fichier avec la commande « make all » afin de générer tous les .o et de les placer dans main_src (donc dans le dossier où l'on se trouve lorsqu'on exécute le Makefile du programme)
- make compile : génère le .o du fichier main.c à partir de tous les .o de tous les fichiers .c.
- make move : place le fichier main.o dans le dossier lib.
- make run : exécute le fichier main.o afin de lancer le programme.
- make clean : cette commande permet d'effacer tous les fichiers .o ainsi que le fichier .dat qui ont été créés afin de pouvoir ré exécuter le programme sans problèmes. Cette commande est à faire à chaque fin d'exécution du programme.
- **make all** : Cette commande exécute toutes les commandes énumérées précédemment (à l'exception de make clean). Elle permet à l'utilisateur de compiler et exécuter le programme d'un seul coup, sans écrire plusieurs étapes dans le Shell. Cependant il faut tout de même taper la commande « make clean » en fin de programme car « make all » ne peut pas le faire.

Le Makefile de main_src

```
GCC = gcc
INCLUDE = -I ../../include/struct_lib -I ../../include/calcul_lib -I ../../include/displaySh
ell_lib -I ../../include/file_lib -I ../../include/gnuplot_lib -I ../../include/input_lib
FLAG = -lm
SRC = $(wildcard *.c)
OBJ = calcul.o dynamicSystems.o displayShell.o file.o gnuplot.o input.o main.o inputSystems.
o struct.o
EXEC = ../../bin/main
LIB = ../../lib/
MAKE = make all

all :
        make build -s
        make compile -s
        make move -s
        make run -s

build :

        cd ../calcul_src/ && $(MAKE)
        cd ../displayShell_src && $(MAKE)
        cd ../file_src/ && $(MAKE)
        cd ../gnuplot_src/ && $(MAKE)
        cd ../input_src/ && $(MAKE)
        cd ../struct_src/ && $(MAKE)
        cd ../main_src

compile : $(SRC)
        $(GCC) $(INCLUDE) -c $(SRC)


move : $(OBJ)
        mv $(OBJ) $(LIB)

run : $(SRC)
        $(GCC) -o $(EXEC) $(LIB)*.o $(FLAG)
        $(EXEC)

clean:
        rm $(EXEC) $(LIB)* ./*.dat
```

VI. Les problèmes rencontrés :

Les problèmes rencontrés n'ont pas été nombreux mais nous avons mis du temps à les résoudre.

1) Problème de fonction

Nous avons tout d'abord eu un problème pour relier les .c avec main.c (et même dans d'autres fichiers lors d'appel de fonction) ; nous ne savions pas s'il fallait envoyer les adresses ou directement la variable, ou pointer sur la variable. Nous avons finalement compris qu'il fallait envoyer l'adresse lors de l'appelle de la fonction dans le fichier main.c et ensuite pointer sur les variables à l'intérieur de la fonction.

Exemple de fonction dans un fichier .c (calcul_src/dynamicSystems.c)

```c
//calculation of instant speed of lorenz system
void lorenz_t(double speed_t[], double position[], double parameter[], double *dt){

        speed_t[0] = position[0] + (parameter[0] * (position[1] - position[0])) * (*dt);
        speed_t[1] = position[1] + (position[0] * (parameter[1] - position[2]) - position[1]) * (*dt);
        speed_t[2] = position[2] + (position[0] * position[1] - parameter[2] * position[2]) * (*dt);

        return;
}
```

*On peut voir que les arguments sont des doubles *, soit l'intérieur de la variable. Ainsi il faut envoyer l'adresse lors de l'appel de la fonction.*

```c
/*main loop of calculation of instant speed according to choosen dynamic system */
void instant_speed(double speed_t[], double position[],double parameter[], double *dt, char flag[]){

        if (strcmp(flag,"1") == 0){//lorenz system : instant speed
                lorenz_t(speed_t, position, parameter, dt);
        }
```

On peut voir que lors de l'appel de la fonction dans une autre fonction, on envoie l'adresse des variables

---

2) Problème de structure

Nous avons d'abord construit notre code sans utiliser les structures, lorsque nous avons voulu les implémenter nous étions un peu perdus et ne savions pas comment faire. Lorsque ce fût fait, il a fallu implémenter les mallocs, nous ne savions pas du tout comment faire mais nous y sommes finalement parvenus.

Exemple d'une fonction utilisant la fonction malloc() :

```c
/*this function will handle the point struct by creating a dynamic allocation*/
void handle_p(void){
        p = malloc(sizeof(struct point));
        p->position = malloc(3 * sizeof(double *));
        p->speed = malloc(3 * sizeof(double *));
        p->speed_t = malloc(3 * sizeof(double *));
        return;
}
```

Dans cette fonction, la fonction malloc() a été utilisé pour allouer dynamiquement la structure.

3) Problème de module

Lorsque nous avions implémenté deux systèmes dynamiques, notre code n'était toujours pas optimisé (comme dit plus haut dans l'introduction). Nous voulions implémenter d'autres systèmes dynamiques mais nous ne pouvions pas continuer à créer un dossier pour chaque système, l'utilisateur allait se perdre et cela allait rendre notre programme difficile d'utilisation. Nous avons donc dû créer des

modules. Cela ne fût pas une tâche facile car il fallait entièrement réorganiser notre code ; tout restructurer. Il fallait créer des fonctions générales et ce fût compliquer de trouver des idées pour optimiser le programme. En effet, le coder n'était pas compliqué mais on se perdait souvent dans nos idées parfois un peu floues. Nous sommes finalement parvenus à optimiser notre programme de sorte à avoir des modules généraux et faciles d'application. Une fois cela fait, l'implémentation de d'autres systèmes dynamiques aurait dû être simple.

4) Problème de systèmes dynamiques

En effet, implémenter des systèmes dynamiques n'était plus un problème une fois les modules créer. Cependant les trouver ne fût pas une mince affaire. Après de longues recherches nous avons finalement trouvé plusieurs systèmes dynamiques ; mais nous ne trouvions que les équations et les paramètres. L'absence des coordonnées initiales nous a forcé à essayer plusieurs combinaisons pour au final abandonner certains systèmes car nous ne trouvions pas. Nous avons tout de même réussi à trouver dix systèmes qui fonctionnent.

5) Problème de Makefile

Une fois tous nos modules créés, il a fallu créer des Makefiles pour chaque module, ainsi que Makefile regroupant tous les autres Makefiles. Nous ne savions pas comment procéder et avons dû faire de longues recherches afin de comprendre le fonctionnement d'un Makefile.

Finalement aucun problème n'a conduit à un abandon d'idée ; on a dû prendre du temps mais nous avons finalement réussi à tous les résoudre.

VII. Évolutions personnelles

Suite à réalisation de ce projet, nous avons énormément progressé en terme de codage informatique. Nous savons maintenant utiliser l'interface gnuplot afin de tracer des courbes, utiliser les structures et les mallocs. Nous avons appris à organiser les fichiers en créant des .h associés aux .c ainsi qu'en créant plusieurs fichiers séparés. Grâce à la création de modules, nous savons maintenant créer un programme général qui peut s'adapter à certains paramètres. Et nous savons maintenant utiliser l'administration du système d'exploitation Linux et les softwares comme git ou BitBucket.

VIII. Manuel d'utilisation de notre programme

Après avoir téléchargé le projet sur BitBucket les étapes à suivre afin d'utiliser notre programme sont les suivantes :

1- Il faut tout d'abord se rendre dans le dossier main_src qui se trouve lui-même dans le dossier src. Pour cela il faut taper dans le terminal :

cd src/main_src

2- Il faut ensuite exécuter le Makefile, une seule commande suffit, il faut taper dans le terminal :

make all -s

Dans le terminal va s'afficher :

```
**********************************************************

        MODEL THE TRAJECTORY OF A POINT

**********************************************************

there are ten dynamic systems, to execute :

        the Lorenz attractor enter 1.
        the Euler attractor enter 2.
        the Aizawa attractor enter 3.
        the Anishchenko - Astakhov attractor enter 4.
        the Nose - Hoover atractor enter 5.
        the Rossler atractor enter 6.
        the coullet atractor enter 7.
        the Hadley atractor enter 8.
        the Rayleigh - Benard atractor enter 9.
        the Bouali atractor enter 10

you choose :
```

Ce message invite l'utilisateur à choisir un système dynamique parmi les dix proposés.

3- Il faut taper le nombre correspondant au système dynamique que l'on souhait exécuter. Par exemple si l'utilisateur souhaite exécuter le système dynamique d'Aizawa, il doit taper dans le terminal :

3

Remarque : Nous traiterons seulement un exemple (le cas 3) mais les étapes sont les mêmes pour chaque système dynamique.

En cas d'erreur : *Le programme a été conçu en prenant en compte une erreur de frappe de la part de l'utilisateur, en cas d'un numéro ne se situant pas entre 1 et 10 ou une autre touche du clavier pressé, le programme renverra un message d'erreur et demandera à l'utilisateur de recommencer :*

```
there are ten dynamic systems, to execute :

        the Lorenz attractor enter 1.
        the Euler attractor enter 2.
        the Aizawa attractor enter 3.
        the Anishchenko - Astakhov attractor enter 4.
        the Nose - Hoover atractor enter 5.
        the Rossler atractor enter 6.
        the coullet atractor enter 7.
        the Hadley atractor enter 8.
        the Rayleigh - Benard atractor enter 9.
        the Bouali atractor enter 10

you choose : pkl

ERROR : you can only type an integer between 1 and 10, please try again.

you choose : c

ERROR : you can only type an integer between 1 and 10, please try again.

you choose :
```

Le programme invite ensuite l'utilisateur à choisir s'il souhaite utiliser les paramètres par défauts (conseillé si l'utilisateur ne connaît pas les paramètres et/ou les coordonnées initiales)

```
**********************************************************

        MODEL THE TRAJECTORY OF A POINT

**********************************************************

there are ten dynamic systems, to execute :

        the Lorenz attractor enter 1.
        the Euler attractor enter 2.
        the Aizawa attractor enter 3.
        the Anishchenko - Astakhov attractor enter 4.
        the Nose - Hoover atractor enter 5.
        the Rossler atractor enter 6.
        the coullet atractor enter 7.
        the Hadley atractor enter 8.
        the Rayleigh - Benard atractor enter 9.
        the Bouali atractor enter 10

you choose : 3

**********INIT : AIZAWA SYSTEM**********

if you want to execute the programm with the default parameters type 'yes' otherwise type 'no'.

you choose :
```

4- Il faut taper dans le terminal la réponse correspondant au souhait de l'utilisateur, par exemple s'il ne souhaite pas utiliser les paramètres par défaut, il doit taper :

no

Remarque : nous traiterons seulement le cas où l'utilisateur souhaite rentrer lui-même les paramètres car en cas contraire le programme s'exécute directement sans aucune étape supplémentaire de la part de l'utilisateur.

<u>En cas d'erreur :</u> Le programme a été conçu en prenant en compte une erreur de frappe de la part de l'utilisateur, si l'utilisateur se trompe en écrivant le nom du système dynamique qu'il souhaite, le programme renverra un message d'erreur et demandera à l'utilisateur de recommencer :

```
**********INIT : AIZAWA SYSTEM**********

if you want to execute the programm with the default parameters type 'yes' otherwise type 'no'.

you choose : 3

ERROR : you can only type 'yes' or 'no', please try again.

if you want to execute the programm with the default parameters type 'yes' otherwise type 'no'.

you choose : ldfv

ERROR : you can only type 'yes' or 'no', please try again.

if you want to execute the programm with the default parameters type 'yes' otherwise type 'no'.

you choose :
```

Le programme affiche donc :

```
--------------------COORDINATES----------------------
enter the coordinate x :
```

Qui invite l'utilisateur à rentrer les coordonnées initiales : x, y, z.

5- L'utilisateur doit rentrer les valeurs de x puis de y puis de z en faisant « entrer » entre chaque coordonnée:

```
--------------------COORDINATES----------------------
enter the coordinate x : 0.1
enter the coordinate y : 0
enter the coordinate z : 0
-----------------------------------------------------

---------------------CONSTANTS-----------------------
enter a :
```

Attention à bien utiliser le point du clavier et non pas la virgule pour les nombres décimaux !

6- L'utilisateur doit ensuite rentrer les paramètres. Comme pour les coordonnées initiales, il les rentre une par une en faisant « entrer » entre chacune d'elles :

```
----------------------CONSTANTS-----------------------
enter a : 0.95
enter b : 0.7
enter c : 0.6
enter d : 3.5
enter e : 0.25
enter f : 0.1
------------------------------------------------------
```

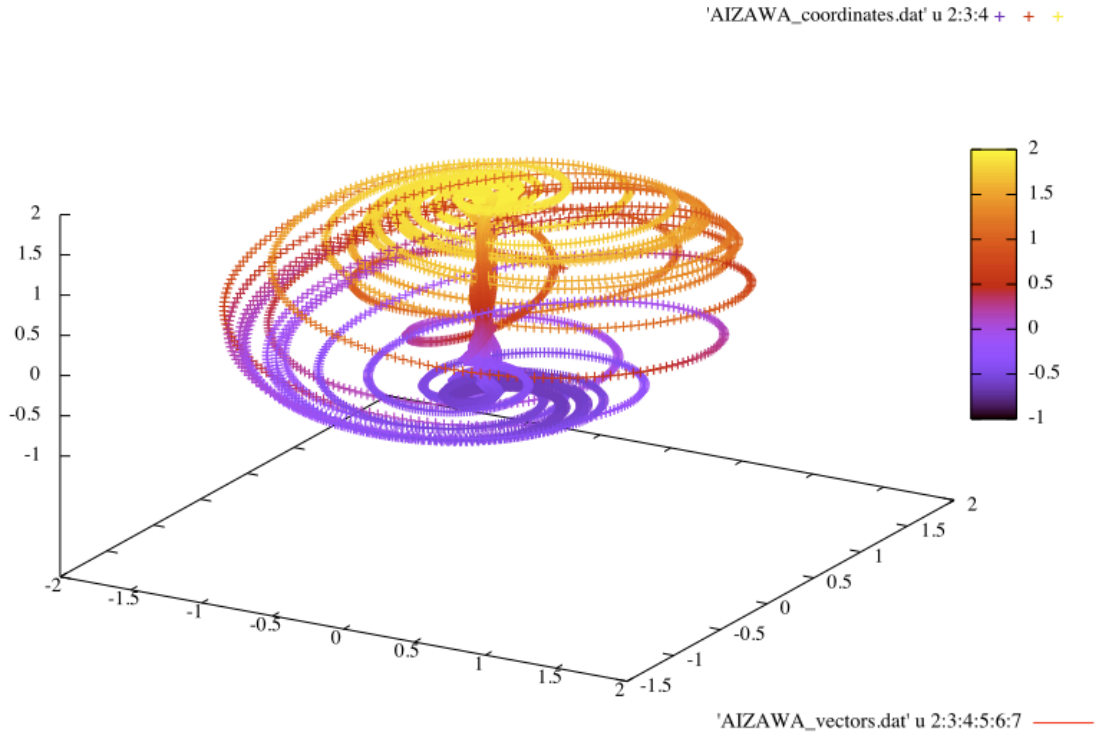
Attention à bien utiliser le point du clavier et non pas la virgule pour les nombres décimaux !

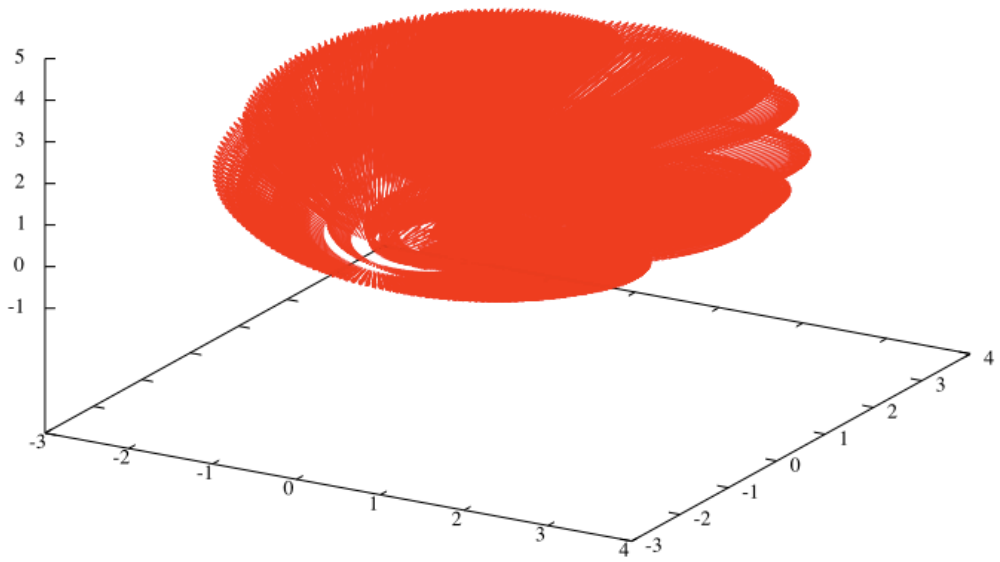
Le programme affiche les vitesses initiales dx, dy, dz et invite l'utilisateur à rentrer les paramètres de temps.

```
----------------------CONSTANTS-----------------------
enter a : 0.95
enter b : 0.7
enter c : 0.6
enter d : 3.5
enter e : 0.25
enter f : 0.1
------------------------------------------------------

-----------------------SPEED------------------------
dx = -0.070000
dy = 0.350000
dz = 0.590000
------------------------------------------------------

-----------------------TIME-------------------------
type the increment dt :
```

7- L'utilisateur doit finalement rentrer les paramètres de temps : l'incrément de temps « dt » et le temps maximal de la trajectoire « tmax » en faisant « entrer » entre les deux. (*Nous conseillons dt = 0.01 et tmax = 100).*

Finalement le programme se lance et affiche dans le terminal les valeurs de x, y et z en fonction du temps. Il affiche ensuite automatiquement la courbe en couleur ainsi que les vecteurs vitesses sur un deuxième graphe.





8- L'utilisateur doit maintenant taper :

make clean

afin d'effacer tous les fichiers .o ainsi que le fichier .dat créés. Après cela il peut de nouveau utiliser le programme. (Il est conseillé de faire une make clean avant de faire « make all » en cas d'oubli lors de la dernière utilisation du programme)