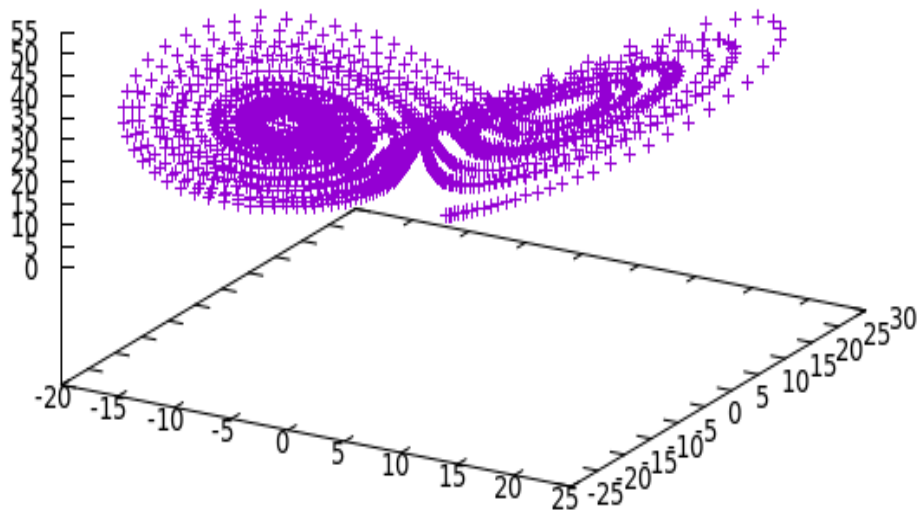


Projet en langage C :

MODELISATION DE LA TRAJECTOIRE D'UN POINT

"lorenz.dat" u 2:3:4 +



MOUDDENE Hamza
SANZ -- SOUHAIT Corina

L2 CUPGE Année 2018–2019

Table des matières :

A. Introduction :	3
B. Guide pour l'architecture du répertoire :	3
I. ACCÈS AU DÉPÔT PRIVÉ (Bitbucket) :	3
II. STRUCTURE HIERARCHIQUE DU RÉPERTOIRE :	3
C. Code source :	6
I. Initialisation de variables :	6
II. Implémentation de fonctions :	6
1) Fonctions de bases :	7
2) Fonctions avancées :	9
III. Implémentation de bibliothèques :	9
D. Compilation et exécution :	10
E. Synthèse :	11

A. Introduction :

Dans ce projet, nous allons modéliser le phénomène de l'attracteur de Lorenz en langage C. L'attracteur de Lorenz est une structure fractale correspondant au comportement à long terme de l'oscillateur de Lorenz. L'attracteur montre comment les différentes variables du système dynamique évoluent dans le temps en une trajectoire non périodique.

L'oscillateur de Lorenz, est une modélisation simplifiée de phénomènes météorologiques basée sur la mécanique des fluides. Ce modèle est un système dynamique tridimensionnel qui engendre un comportement chaotique dans certaines conditions.

B. Guide pour l'architecture du répertoire :

I. ACCÈS AU DÉPÔT PRIVÉ (BITBUCKET) :

Afin de faciliter le travail en groupe, le projet s'effectue en binôme. Nous avons premièrement créé un dépôt privé sur Bitbucket. Nous avons ensuite relié notre projet avec l'enseignant « veronique.Gaildrat@irit.fr » avec la permission « Read ».

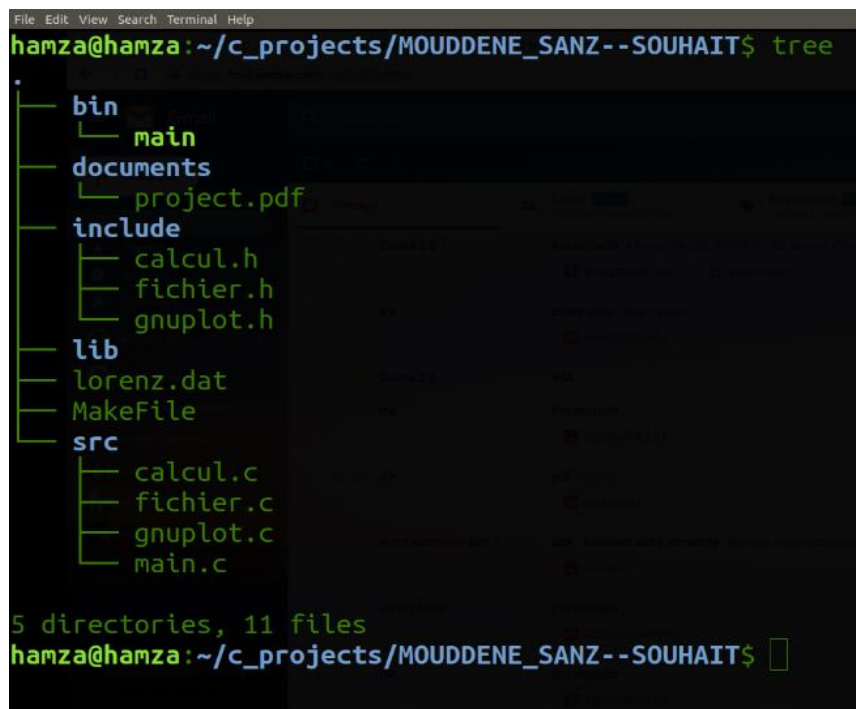
II. STRUCTURE HIERARCHIQUE DU RÉPERTOIRE :

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 :

- **src** : il s'agit du répertoire contenant les fichiers.c où se trouve le code source ainsi que main.c : le fichier principal de ce projet qui relie toutes les bibliothèques avec les autres fichiers et qui permet d'exécuter le code.
- **bin** : il s'agit du répertoire qui contient les fichiers exécutables d'où on peut exécuter le programme.
- **lib** : il s'agit du répertoire contenant les fichiers.o (fichiers objets) contenant la compilation des fichiers.c (langage-computer), ainsi que des bibliothèques de langage C qui seront utilisées lors de la conception du code.
- **documents** : il s'agit du répertoire contenant l'énoncé ainsi que le rapport de la première phase. Plus tard, il contiendra le rapport de la deuxième phase.
- **include** : il s'agit du répertoire contenant l'ensemble des fichiers.h, ces derniers ne contiennent pas de code mais seulement les déclarations de fonctions et les objets de type « **struct** ».

- **MakeFile** : il s'agit d'un fichier ayant pour fonction d'organiser et d'automatiser le processus de la compilation. Il suffit de taper la commande « **make** » afin de l'exploiter. Cette commande peut aussi exécuter le programme.
- **lorenz.dat** : il s'agit du fichier qui stockera les coordonnées de la position entre l'instant $t = 0$ et $t = T_{\max}$, sous formes :
temps x y z



```
File Edit View Search Terminal Help
hamza@hamza:~/c_projects/MOUDDENE_SANZ--SOUHAIT$ tree
.
├── bin
│   └── main
├── documents
│   └── project.pdf
├── include
│   ├── calcul.h
│   ├── fichier.h
│   └── gnuplot.h
├── lib
├── lorenz.dat
├── MakeFile
└── src
    ├── calcul.c
    ├── fichier.c
    ├── gnuplot.c
    └── main.c

5 directories, 11 files
hamza@hamza:~/c_projects/MOUDDENE_SANZ--SOUHAIT$
```

C.Code source :

I. INITIALISATION DE VARIABLES :

- Après avoir compilé et exécuté le programme, ce dernier aura besoin de quelques entrées pour fonctionner, on va alors demander à l'utilisateur d'entrer des variables telles que :

- la position initiale : **double** x, y, z ;
- la vitesse : **double** dx, dy, dz ;
- les paramètres : **double** sigma, ro, beta ;
- l'incrément dt: **double** dt ;
- le temps d'arrêt : **double** Tmax ;

II. IMPLEMENTATION DE FONCTIONS :

Dans un premier temps, les variables dont nous aurons besoin ont été initialisées. Maintenant, 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 allons donc utiliser ce concept afin d'avoir une idée claire d'un plan bien détaillé de ce projet.

Les fonctions utilisées sont les suivantes :

1) Fonctions de bases :

```
int main(int argc, char *argv[]){  
    /*c'est la fonction générique de tout le  
    programme, les autres fonctions y seront  
    appelées */  
}
```

```
void vitesse(double x, double y, double z, double dt,  
double dx, double dy, double dz){  
    /*cette fonction va calculer la vitesse à travers les  
    positions initiales et les paramètres  $\sigma$ ,  $\rho$  et  $\beta$  en  
    utilisant les équations de différentielles de
```

$$\text{Lorenz : } \left\{ \begin{array}{l} \frac{dx}{dt} = \sigma(y - x) \\ \frac{dy}{dt} = x(\rho - z) - y \quad */ \\ \frac{dz}{dt} = xy - \beta z \end{array} \right.$$

```
}
```



```
void position_t(double x, double y, double z, double
dt, double dx, double dy, double dz)
```

```
/*cette fonction calcule la position du point entre
t = 0 et t = Tmax avec la formule suivante
donnée dans l'énoncé :
```

$$\begin{cases} x(t + dt) = x(t) + \frac{dx}{dt} \cdot dt \\ y(t + dt) = y(t) + \frac{dy}{dt} \cdot dt \\ z(t + dt) = z(t) + \frac{dz}{dt} dt \end{cases}$$

```
*/
```

```
void ouvrir_fichier(){
```

```
/*cette fonction initialise et ouvre le fichier,
avec l'option w+ qui le crée s'il n'existe
pas*/
```

```
}
```

```
void ecrire_fichier(){
```

```
/*cette fonction stockera dans le fichier
« lorenz.dat » les coordonnées entre t = 0 et
t = Tmax*/
```

```
}
```

```
void fermer_fichier(){
```

```
/*cette fonction ferme le fichier avant d'arrêter le
programme*/
```

```
}
```

2) Fonctions avancées :

```
void gnuplot(){  
    /*cette fonction lance les commandes et trace le  
    graphique à partir du programme*/  
}
```

III. IMPLEMENTATION DE BIBLIOTHEQUES :

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). On va alors définir les différentes bibliothèques qui seront utilisées dans ce projet :

- **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.

- **calcul.h** : il s'agit de la bibliothèque qui calcule la position et la vitesse du point à un instant t . Pour faire ce calcul, on fournit l'expression des équations différentielles de Lorenz.

- **fichier.h** : il s'agit de la bibliothèque qui relie le fichier « lorenz.dat » avec le langage C, elle contient différentes fonctions qui gèrent ce fichier.

- **gnuplot.h** : il s'agit de la bibliothèque qui permet de lancer les commandes et tracer le graphique à partir du programme. Elle permet aussi d'afficher les vecteurs vitesse par une couleur et le temps par un changement de couleur.

- **input.h** : il s'agit de la bibliothèque contenant la fonction vérifiant le type des entrées afin de pouvoir détecter une possible erreur de typage et le signaler à l'utilisateur.

D. Compilation et exécution :

- Lorsque le code sera fini, nous passerons à l'étape finale qui consiste à créer et à configurer le fichier Makefile qui s'occupera de la compilation et de l'exécution en reliant le code source (fichiers.c) et les fichiers.h (header files). Ce processus est fortement recommandé car plus rapide et plus efficace, surtout avec la compilation séparée.

Il suffit de taper la commande « run » pour l'utiliser.

Exemple du fichier MakeFile :

```
#Makefile
OBJ=main.o test1.o test2.o
CC =gcc
test:${OBJ}
    $(CC) -o $@ $^
main.o:main.c test1.h test2.h
    $(CC) -c $<
test1.o:test1.c test1.h
    $(CC) -c $<
test2.o:test2.c test2.h
    $(CC) -c $<
.PHONY:clean
clean:
    rm -f $(OBJ) test
```

E. Synthèse :

Ce rapport constitue seulement un premier plan de notre futur projet, celui-ci sera développé dans la suite. Nous avons et continuerons à rencontrer des difficultés lors de la réalisation du projet. Actuellement, nous avons réussi à faire un premier programme basique qui fonctionne. Nous continuerons d'améliorer la structure et la logique du code en optimisant au maximum.

