

# Projet C : Modélisation de la trajectoire d'un point

# Sommaire

---

<b>I. Introduction</b>	<b>3</b>
<b>II. Guide d'utilisation</b>	<b>4</b>
<b>III. Librairies</b>	<b>5</b>
1. Log	5
Créer le fichier .log	5
Ecrire dans le fichier .log	5
2. Entrées	6
Lire la fin de la ligne et vider le buffer	6
Lire le format	6
"On a vraiment eu de la chance !"	6
Choix du système dynamique	6
Choix de Tmax et dt	7
3. Position	7
Structure Coord	8
Setters et Getters	8
Calcul de la nouvelle position	8
4. Fichier	8
Initialiser le fichier	9
Ecrire dans le fichier	9
5. Gnuplot	10
Tracer la Courbe	10
Lancer Gnuplot	10
6. Param	11
Union Param	11
Setters	11
Getters	11
<b>IV. Dépendances</b>	<b>12</b>
1. Bibliothèques standard	12
2. Librairies	12
<b>V. Organisation du programme principal</b>	<b>13</b>
1. Initialisation	13
2. Calcul	13
3. Affichage	13
<b>VI. Tests</b>	<b>15</b>
<b>VII. Conclusion</b>	<b>16</b>

---

# I. 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  $dt$ .

Axé sur plusieurs objectifs, ont été réalisés ici:

- Demande des paramètres et la position initiale à l'utilisation,
- Création de plusieurs fonctions pour donner à l'utilisateur le choix entre différents systèmes dynamiques,
- Des modules lisant les paramètres et effectuant la mise à jour d'un point pour différents systèmes,
- Demande à l'utilisateur de définir les paramètres et la mise à jour de la vitesse,
- Traçage de la courbe sans passer par les commandes (pour l'utilisateur),
- Affichage du temps par un changement de couleur.

Ce projet se réalise par binôme qui est composé ici de Valentin Frydrychowski et de Clément Truillet

## II. Guide d'utilisation

L'affichage des courbes se faisant avec Gnuplot, il est primordial de le télécharger. Que ce soit pour Windows, Linux ou MacOS, vous pouvez télécharger Gnuplot à l'adresse suivante: <https://sourceforge.net/projects/gnuplot/files/gnuplot/> .

### Windows

Tout d'abord, téléchargez le fichier .zip du projet à cette adresse: [clementtruillet.github.io/Double-Git-Heroes/](https://clementtruillet.github.io/Double-Git-Heroes/) et décompressez le.

L'utilisation de bash étant plus que conseillée voici un excellent tutoriel afin de se le procurer : <https://korben.info/installer-shell-bash-linux-windows-10.html>

Afin d'afficher Gnuplot, il est obligatoire d'avoir un serveurX que vous pouvez télécharger à l'adresse suivante : <https://sourceforge.net/projects/vcxsrv/>

La compilation de l'exécutable se fait à la racine du projet en tapant "make all", vous pouvez maintenant lancer le projet en écrivant respectivement "cd bin" puis "./attracteur.exe".

### Linux & MacOS

Tout d'abord, téléchargez le fichier tar.gz du projet à cette adresse: [clementtruillet.github.io/Double-Git-Heroes/](https://clementtruillet.github.io/Double-Git-Heroes/) et décompressez le avec la commande "tar xzf <fichier.tar.gz>".

La compilation de l'exécutable se fait à la racine du projet en tapant "make all", vous pouvez maintenant lancer le projet en écrivant respectivement "cd bin" puis "./attracteur.exe".

## III. Librairies

### 1. Log

Notre objectif, avec cette librairie, était d'avoir un suivi de l'exécution (bonne ou non) du projet et ce fut un franc succès.

Cette librairie étant assez simple et non nécessaire au bon déroulement de l'exécution du projet, le seul changement notable a été au niveau du format des noms des fichiers afin de le rendre plus lisible (le fichier log nommé 20181011-105823.log devient 11-10-2018\_10.58.23.log).

#### Créer le fichier .log

Afin de pouvoir récupérer l'heure précise de l'exécution, il a été nécessaire d'utiliser la bibliothèque `<time.h>`, qui, aux premiers abords nous a semblé assez obscurs.

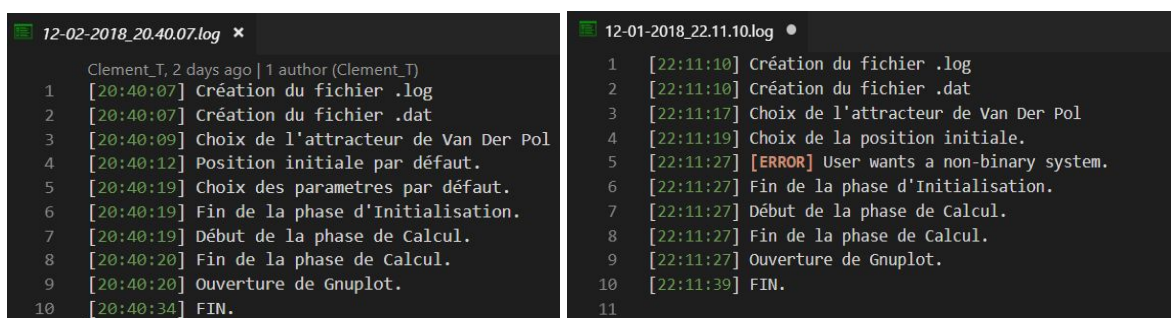
Toutefois, le fonctionnement inscrit dans le premier rapport est identique en tout point aujourd'hui (fonction void qui crée un fichier .log vide et qui stocke le nom de ce fichier dans la variable FICHER dont l'adresse a été prise en paramètres) .

#### Ecrire dans le fichier .log

De la même façon que la fonction précédente, cette fonction a le même fonctionnement que celui inscrit dans le premier rapport (prend en paramètre une chaîne de caractères et l'inscrit dans le fichier .log à la ligne).

Aussi, afin de suivre le temps entre chaque écriture (surtout entre chaque phase) dans le fichier .log, nous avons préféré ajouter l'heure (Heure:Minute:Seconde) de l'écriture avant chaque message.

Ainsi, nous avons pu nous rendre compte de la lenteur de l'exécution dans le cas d'une ouverture et fermeture systématique du fichier .dat à chaque écriture de la position (cf. *Fichier*).



figures 1 et 2 : différents fichiers log

## 2. Entrées

Afin de sécuriser la prise de données, nous avons dû créer des fonctions supplémentaires pour : vider la mémoire, vérifier l'entrée des nombres décimaux et des réponses textuelles.

Ces fonctions ont deux buts principaux.

Tout d'abord prévenir les erreurs de saisie de l'opérateur en détectant ces erreurs et lui signifier celles-ci, puis protéger le matériel informatique de l'opérateur en évitant le "buffer overflow" et qu'il aille écrire dans des parties de mémoire non prévues à cet effet.

### Lire la fin de la ligne et vider le buffer

Cette fonction, lireFinL, permet de récupérer la fin de la ligne et compte le nombre de caractères qu'elle a reçue jusqu'à ce qu'elle détecte le retour chariot. Afin de les compter, elle consomme tous les caractères du buffer.

Elle se substitue à la fonction videBuffer initialement prévue.

### Lire le format

Pour faciliter l'écriture de la librairie Entrées qui doit pouvoir traiter différents formats de variables, nous avons créé une fonction à l'image de scanf qui s'assure que la valeur entrée par l'utilisateur est dans le bon format ou non.

### "On a vraiment eu de la chance !"

Trouvant un peu handicapant de devoir relancer le programme pour avoir le résultat voulu en cas de fautes de frappes ou d'incompréhension de la consigne, nous avons décidé de créer une fonction qui donne des chances supplémentaires à l'utilisateur de se rattraper sans devoir recommencer toutes ses saisies.

### Choix du système dynamique

Pour choisir le plus simplement le type d'attracteur que souhaite l'utilisateur, il lui est demandé d'entrer un nombre associé à un système dynamique afin de le sélectionner.

Géré avec une instruction Switch, il est aisé de rajouter des systèmes dynamiques et de gérer le cas où aucun système n'est sélectionné (dans ce cas, on choisit Lorenz par défaut).

Sensibles à un simulacre d'interface graphique, cette fonction affiche aussi l'accueil de notre programme.

```
=====
===== Modélisation de Trajectoires =====
=====

Bienvenue sur ce projet de modélisation de trajectoire d'un point.

Sont proposés ici, trois attracteurs étranges :
0. Attracteur de Lorenz
1. Attracteur de Van Der Pol
2. Attracteur de Rössler

Selectionnez l'attracteur souhaité en entrant son numéro associé (Par défaut 0).
█
```

figure 3 : Interface "graphique" du projet

## Choix de Tmax et dt

Assez similaires par nature, ces deux fonctions sont indispensables dans une optique de bonne exécution du projet car elle permettent d'avoir un temps d'arrêt de calcul (éviter un calcul de position infini) et un incrément de temps entre chaque position. Soucieux de pouvoir de nuisance de l'utilisateur, nous avons pris soin de s'assurer que la valeur Tmax et dt sont dans les intervalles, respectivement, TMAX TMIN et DT\_MAX DT\_MIN, constantes définies dans le fichier entrees.c.

## 3. Position

Par la création d'une structure de la position et la nécessité de calculer les nouvelles positions en fonction du système dynamique, l'existence de cette librairie est plus que justifiée. Contrairement à ce qui est annoncé dans le premier rapport, la structure Coordonnées (renommée Coord) n'est plus définie dans le fichier .h mais dans le .c afin de rendre invisible son contenu au client.

Ce changement a amené un certain nombre d'ajouts tel que des fonctions dites `getter` et une dite `setter`. Par ailleurs, depuis le dernier rapport, nous avons choisi de proposer à l'utilisateur le choix entre plusieurs modèles dynamiques (au nombre de trois aujourd'hui) ce qui a conduit à la création de deux fonctions supplémentaires et similaires à la fonction de calcul de la position suivante en fonction de la position et des paramètres (entrés par l'utilisateur).

### Choix de la position initiale

Pour calculer la trajectoire du point, nous avons besoin de sa position initiale. La fonction `choix_position` permet de proposer à l'utilisateur de prendre des valeurs par défaut où de rentrer des valeurs personnalisées.

Bien entendu, nous avons pris soin de faire des recherches quant à la position initiale optimale pour chaque système dynamiques afin de proposer à l'utilisateur la meilleure expérience possible.

## Structure Coord

De la même manière que la structure Coordonnées du rapport premier, la structure `Coord` regroupe quatre variables `t`, `x`, `y` et `z` de type `double`. Elle est déclarée dans le fichier `position.c` afin que les manipulations s'opérant avec cette structure ne soit faites qu'avec les `getter` et `setter` décrit en suivant.

## Setters et Getters

C'est éplut de bons sens que nous avons choisi de créer ces quatre fonctions. Là où les trois fonctions `getter` permettent de récupérer la valeur en `x`, `y` ou `z` (selon la fonction) d'un point, la fonction `new_coord` nous permet de créer un point avec les coordonnées `t`, `x`, `y` et `z` sélectionnés.

## Calcul de la nouvelle position

Dans le premier rapport, nous faisons déjà mention d'une fonction de calcul de la nouvelle position, la structure de ces trois fonctions (une pour chaque système dynamiques) est identique, à la différence qu'elles usent de fonctions de la forme `get_Param` permettant de récupérer les valeurs des paramètres stockés dans un union de structures (une pour chaque système dynamiques, encore) comme demandé dans le retour du premier rapport.

## 4. Fichier

De la même manière que la librairie `Log`, cette librairie se décompose en deux fonctions similaires : l'initialisation et l'écriture dans le fichier.

Bien que conscients de la similarité entre ces deux librairies, c'est un choix affirmé et motivé par une utilisation totalement différente et dans des contextes différents des librairies qui nous ont poussé à les dissocier.

L'idée initiale étant de stocker les positions dans un fichier nommé `position.txt`, nous avons dévié et préféré le nommer `position.dat` car il concerne des données, c'est à sa juste valeur que nous pensons qu'il doit considérer comme tel.

Par ailleurs, notre première idée de besoin d'écrire une fonction de lecture du fichier (pour tracer la courbe sur `gnuplot`) a été rapidement détruite par la lecture de la documentation `gnuplot`, en effet, il faut et il suffit de donner en entrée un fichier de données formatées (ici "temps x y z") à `gnuplot` pour qu'il se charge de tracer l'ensemble des points.

Enfin, afin de pouvoir avoir des comparatifs pour nos tests, nous avons généré, pour chaque système dynamiques, des fichiers de données de références.



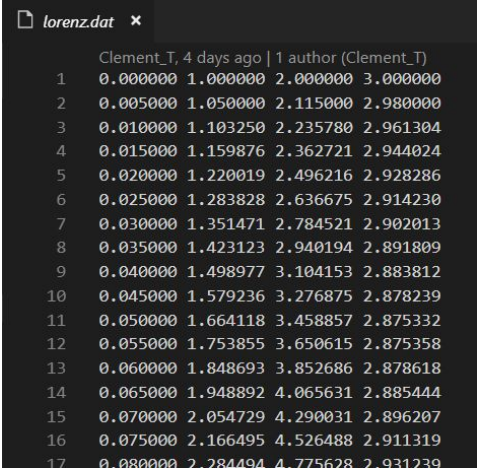
## Initialiser le fichier

Ici, la création et l'initialisation du fichier `.dat` est identique à celle pour les fichiers `.log`, à la différence près que le nom du fichier est déjà renseigné (et stocké dans une variable de type `const` afin de la protéger).

## Ecrire dans le fichier

Dans une volonté d'avoir un fichier avec des données toujours dans le bon format ("temps x y z"), nous avons usé de la fonction `fprintf()`, l'écriture est alors simple et toujours dans le même format (à condition de bien spécifier un retour chariot après chaque écriture).

Aussi, contrairement à l'idée première d'ouvrir et de fermer le fichier à chaque écriture, il a été décidé de ne l'ouvrir qu'une seule fois dans le fichier `main` (en début de phase de calcul) et de le refermer ensuite, en effet, les fichiers `.log` des premières exécutions (et notre bon sens de l'observation) ont révélés un très long temps de calcul (2min30s pour 500 000 positions) là où une ouverture et fermeture unique permettaient de diviser le temps par 150.



```
lorenz.dat x
Clement_T, 4 days ago | 1 author (Clement_T)
1 0.000000 1.000000 2.000000 3.000000
2 0.005000 1.050000 2.115000 2.980000
3 0.010000 1.103250 2.235780 2.961304
4 0.015000 1.159876 2.362721 2.944024
5 0.020000 1.220019 2.496216 2.928286
6 0.025000 1.283828 2.636675 2.914230
7 0.030000 1.351471 2.784521 2.902013
8 0.035000 1.423123 2.940194 2.891809
9 0.040000 1.498977 3.104153 2.883812
10 0.045000 1.579236 3.276875 2.878239
11 0.050000 1.664118 3.458857 2.875332
12 0.055000 1.753855 3.650615 2.875358
13 0.060000 1.848693 3.852686 2.878618
14 0.065000 1.948892 4.065631 2.885444
15 0.070000 2.054729 4.290031 2.896207
16 0.075000 2.166495 4.526488 2.911319
17 0.080000 2.284494 4.775628 2.931239
```

figure 4: exemple de fichier de position (équations de lorenz)

## 5. Gnuplot

Après lecture de la documentation de Gnuplot et l'étude des exemples d'utilisations présent çà et là sur internet, nous avons été surpris et heureux de découvrir que gnuplot n'avait besoin "que" d'un fichier comportant toutes les positions calculées (dans un format identique chaque ligne) ainsi que d'un fichier contenant tous les paramètres afin de "bien" lancer gnuplot (repère cartésien, titre, dégradé temporel, ...).

Par conséquent, l'écriture de cette librairie a été beaucoup plus simple que nous le pensions (un seul appel de l'unique fonction dans la fonction main).

### Tracer la Courbe

Cette fonction a pour but de tracer la courbe en lançant gnuplot avec les bons paramètres.

Pour cela, nous avons besoin de connaître le système dynamique choisie préalablement par l'utilisateur.

Ainsi, cette fonction lance le bon fichier .sh associé au système dynamique sélectionné et dont la valeur est stockée dans une variable mode (0 = Lorenz, 1 = Van Der Pol, 2 = Rossler).

Bien évidemment, une écriture dans le fichier Log est réalisé si on n'est dans aucun des modes (ce qui est normalement contrôlé dès le choix du mode).

### Lancer Gnuplot

Cette partie de la librairie, bien que n'ayant aucun rapport avec C, se décompose en deux fichiers (fois le nombre de systèmes dynamiques), un fichier shell (.sh) et un fichier de paramètres.

Le premier (figure 2) ne se compose seulement que de deux commandes, la première permettant au serveurX (nécessaire et obligatoire sous Windows pour lancer gnuplot) d'ouvrir des fenêtres.

La deuxième, quant à elle, lance gnuplot avec les paramètres stockés dans le second fichier (figure 3) dont la création d'une palette de couleurs afin d'afficher le temps par un changement de couleur sur la courbe.

L'existence de couple de ces fichier par systèmes dynamiques n'est lié qu'au changement de titre sur gnuplot et ont été d'une utilité forte durant les phases de test de cette librairie (exécution des fichiers shell avec les fichiers .dat de référence).

```
1 #!/bin/sh
2
3 #Permet de lancer un serveur X
4 export DISPLAY=:0
5
6 #On lance Gnuplot
7 gnuplot ../data/gnuplot_param/lorenz_param
8
9
1 set parametric
2 set title textcolor "red" "Attracteur de Lorenz"
3 set grid
4 set xlabel "x"
5 set ylabel "y"
6 set zlabel offset +5 +4 "z"
7
8 h1 = -50/360.0
9 h2 = 227/360.0
10 set palette model HSV functions (1-gray)*(h2-h1)+h1,1,0.68
11
12 splot "../data/position.dat" u 2:3:4:1 w l lc palette
13 pause -1 "Tapez \"Entrez\" pour fermer la fenetre"
14
15
```

figures 5 et 6 : contenus des fichiers lorenz.sh et lorenz\_param

## 6. Param

L'ajout de plusieurs systèmes dynamiques a conduit à une question qui aurait dû déjà être posé avant, "Comment stocker les paramètres ?". La réponse fut assez rapide et nous nous sommes dirigés vers le type union.

### Union Param

Pièce maîtresse de cette librairie, elle est assez similaire à la structure `Coord` à la différence près qu'elle comporte trois structures (une pour chaque systèmes) chacune contenant autant de `double` que de paramètres nécessaires. De la même façon que `Coord`, l'union et les structures sont déclarées dans le `.c` et les pointeurs associés dans le `.h`. Par conséquent, l'écriture d'assesseurs est plus qu'obligatoire.

### Setters

Devant l'empilement de pointeurs, nous avons préféré faire cette tâche en deux temps, la première stocke les paramètres dans la structure associée au système choisi, une fois ceci fait, la structure est stockée dans l'union. Cette décomposition en deux temps permet ainsi une bonne lisibilité du code.

### Getters

Une fois que les paramètres sont stockés dans notre union, il a bien fallu réussir à y accéder. Après courte réflexion, la méthode utilisée est strictement la même qu'avec les coordonnées à la différence que les valeurs sont stockées à l'adresse indiquée en paramètre réduisant ainsi fortement le nombre de fonction à écrire.

### Choix des paramètres

Encore une fois, cette fonction s'articule de la même manière que celle du choix de la position, c'est-à-dire que l'on propose d'abord de choisir entre paramètres par défaut ou non, puis, suivant ce choix, une entrée des paramètres est demandée. Ensuite, les paramètres sont stockés dans l'union à travers nos setters.

Bien évidemment, il est pris soin de ne pas avoir des paramètres dans un autre format que un `double` (format des paramètres).

## IV. Dépendances

### 1. Bibliothèques standard

C'est après trois minutes de recherches plus approfondies que nous nous sommes rendus compte que les fameuses bibliothèques `<xspv.h>`, `<xdef.h>` et `<splt.h>` étaient purement et simplement inutiles.

Par conséquent, notre programme ne nécessite que des bibliothèques `<stdlib.h>` et `<stdio.h>` ainsi que `<time.h>` pour la librairie Log seulement.

### 2. Librairies

Identiquement au premier rapport, l'intégralité des librairies dépendant de Log (afin de permettre l'écriture dans le fichier `.log`) et Position (et donc Param aussi) dépendent de Entrées (récupération de la position et des paramètres entrées par l'utilisateur).

Afin de satisfaire la compréhension et ce paragraphe trop court (mais néanmoins complet), voici un diagramme des dépendances des librairies.

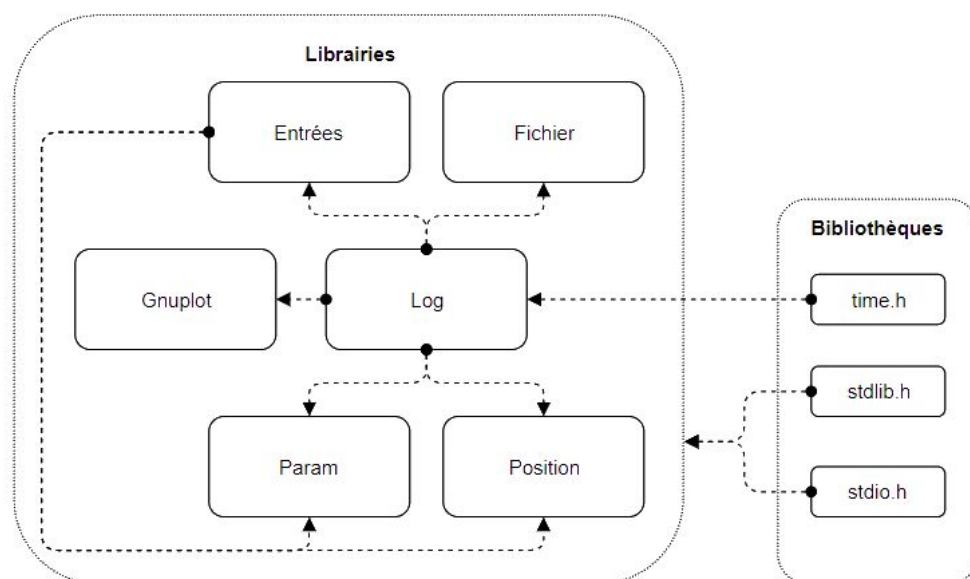


figure 7 : Dépendances des librairies

## V. Organisation du programme principal

### 1. Initialisation

Cette phase d'initialisation se décompose en fait en deux étapes, tout d'abord la création du fichier .log ainsi que l'initialisation du fichier position.dat (on le vide s'il existe, sinon, on le crée).

Ce n'est qu'ensuite que l'on commence à écrire dans le terminal afin d'interagir avec l'utilisateur.

Le plus important et pas des moindre est la sélection du système dynamique (position initiale et paramètres dépendant du système) puis on va en augmentant la complexité, c'est-à-dire le choix de la position initiale, le temps d'arrêt, l'incrément de temps et enfin le choix des paramètres.

### 2. Calcul

Après tests de notre programme avec plusieurs cobayes, nous nous sommes aperçus de la certaine impatience quant à l'affichage de la courbe, c'est pour cela que nous affichons le nombre de calcul de position à prévoir (entre 10 000 et 10 000 000 positions).

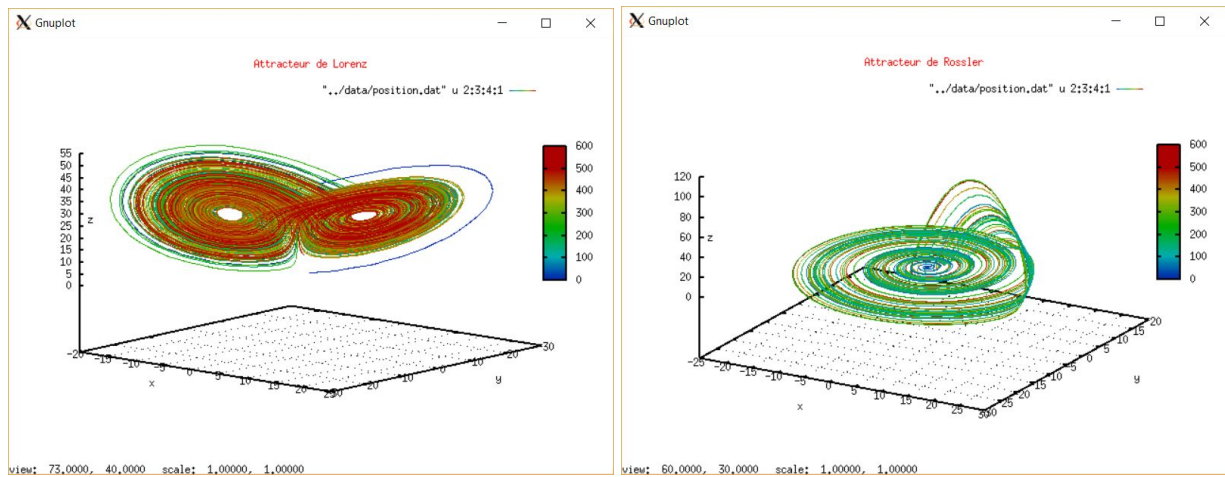
Ensuite, une ouverture du fichier position.dat est faite (raison cf. Tests) ainsi qu'une boucle while qui calcule un point à un temps  $t$ , le stocke puis calcule le point à  $t+dt$  jusqu'au temps  $T_{max}$ .

Bien entendu, une fermeture du fichier position.dat est faite en fin de cette phase.

### 3. Affichage

Cette dernière phase consiste seulement à dessiner la courbe.

Pour cela, un appel à la fonction `trace_courbe` (cf. Gnuplot) et donc fait agir le fichier shell associé au système dynamique ainsi que le fichier de paramètre Gnuplot associé. Une exécution sous Linux, MacOS ou bash (Windows) conduit à une ouverture de la courbe dans une autre fenêtre.



figures 8 et 9 : Attracteurs de Lorenz et Rössler ( $T_{max} = 500s$ ,  $dt = 0.005s$ )

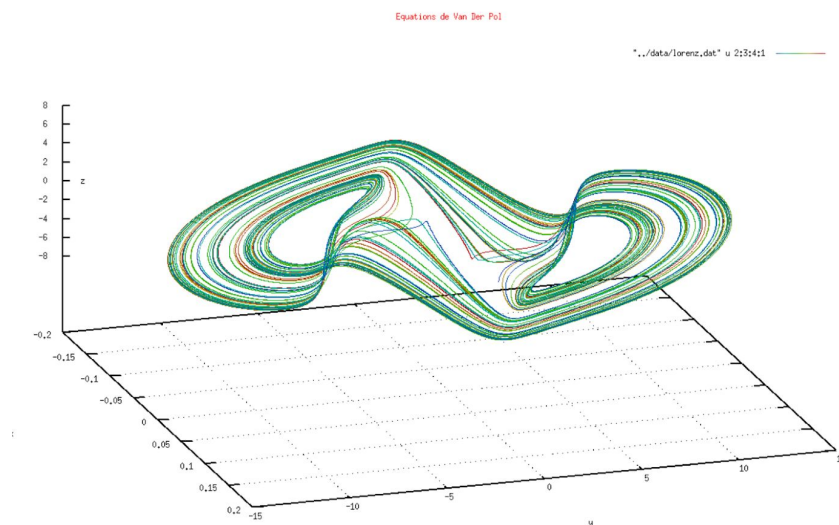


figure 10 : Attracteur de Van Der Pol ( $T_{max} = 5000s$ ,  $dt = 0.005s$ )

## VI. Tests

Tout d'abord, nous avons été soucieux du bon fonctionnement de nos fonctions, c'est ainsi que, avec une rigueur assez extrême, chacune des fonctions est sensée avoir été testée, d'abord seule, puis dans le cadre de son utilisation.

Aussi, rapidement, nous nous sommes munis de fichiers de positions de référence pour chaque système dynamiques afin de comparer les fonctionnements de notre programme avec ce qui est attendu.

Par ailleurs, le fichier `.log` a été d'une aide immense permettant de suivre en direct l'exécution du programme.

Pour citer un exemple, il nous a permis de rendre compte d'une lenteur conséquente d'exécution lorsque nous ouvrons et fermions le fichier `position.dat` à chaque écriture et donc de corriger le tir.

L'outil `gdb`, lui aussi, a été d'une grande aide devant nos rencontres avec les segmentations fault.

Une attention particulière a été portée par des sessions de test où le but principal était de tordre le programme dans tous les sens, dans des conditions non optimales pour arriver à trouver un ou plusieurs bugs.

Ces tests ont été effectués sur différents systèmes d'exploitations (Bash et Debian) et par différents cobayes nous permettant de rendre compte, par exemple, du crash obligatoire devant un trop grand nombre de calcul avec des microprocesseurs moins récents.

Du même acabit, un bug encore inexpliqué aujourd'hui a amené une modification du `makefile` : un programme `<fichier>.exe` compilé directement sous Debian ne peut pas se lancer, il faut le compiler en `<fichier>.out` puis le renommer en `<fichier>.exe`.

De même, il faut systématiquement ajouter les droits de lecture et d'exécution aux fichiers shells qui parfois sautent.

## VII. Conclusion

Ce projet nous a permis d'apprendre les bases du travail collectif (répartitions des tâches et entraide) sur un projet informatique. C'était un bon moyen de confronter nos acquis avec un projet ayant une finalité, ce qui est plus motivant que de simples exercices et les difficultés rencontrées nous ont permis de prendre de l'expérience et des habitudes de travail et de réflexions pour éviter ces problèmes dans un futur plus ou moins proche.

De plus, utiliser Git permet une compréhension plus poussée de son fonctionnement et un apprentissage par l'échec de certaines manipulations (on peut citer en exemple des `git push -f` ou des `git pull` mal gérés).

Avec une vision plus lointaine, ce projet a été une parfaite préparation pour notre participation à la Nuit de l'Info en nous obligeant à collaborer ensemble sans nous marcher dessus.