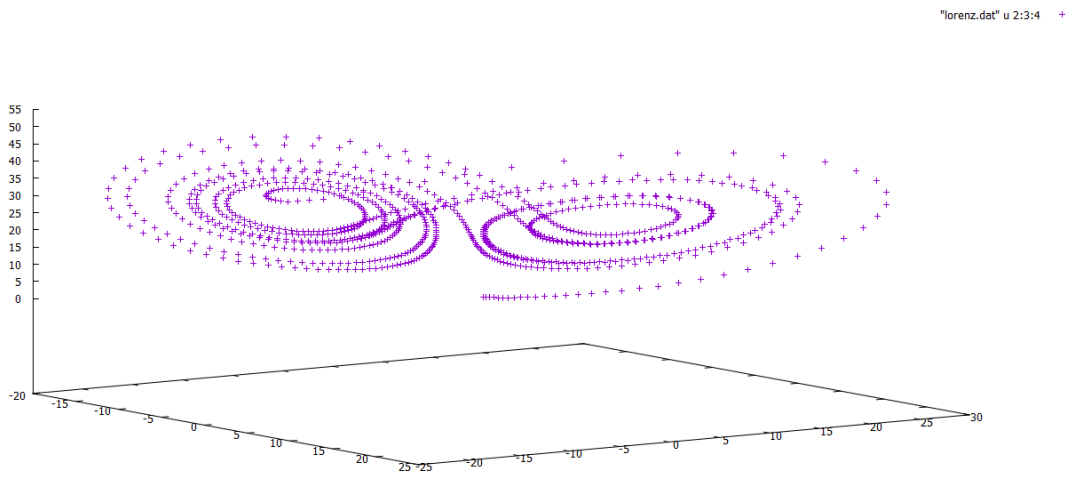
Projet Informatique
Modélisation de la trajectoire d'un point



« 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 ».

Notre objectif final est donc de pouvoir dessiner la trajectoire d'un point suivant le système des équations de Lorenz mais aussi suivant d'autres systèmes d'équations. Pour parvenir à ce résultat final nous avons cherché à bien organiser notre travail, afin de, lors de l'écriture du code, partir avec un plan détaillé déjà étudié et faciliter l'écriture du programme. C'est ce que nous allons vous détailler dans la suite.

Table des matières

I. Librairie Structures/Types

Cette librairie contiendra des déclarations de structures et de types créés pour faciliter le code. Elles permettront de rassembler les différentes variables par catégories, afin de ne pas les mélanger lors de l'écriture.

La structure principalement utilisée dans le programme est un triplet de float que l'on nommera "triplet":

```
Struct triplet{
    float x;
    float y;
    float z;
};
```

Ensuite on définit un type nommé "tri" à partir de cette structure:

```
typedef struct triplet tri;
```

II. Librairie Systèmes Dynamiques

Cette librairie contiendra les fonctions utilisées pour calculer la vitesse et la position d'un point suivant les équations de Lorenz, ou de tout autre système que nous déciderons d'implémenter (comme par exemple l'attracteur de Rössler). Cette librairie "Systèmes Dynamiques" permettra donc de regrouper toutes les fonctions calculant des coordonnées de vitesses ou de points pour les systèmes dynamiques utilisés.

1) Fonction vitesse_lorenz

La fonction vitesse_lorenz prendra en entrée les coordonnées (x,y,z) et les paramètres pour le calcul de la trajectoire.

La fonction permettra de calculer les coordonnées vitesses à partir des équations de Lorentz et des coordonnées.

Déclaration : tri vitesse_lorenz(tri coordposition, tri paratrajectoire)

Elle retournera les trois coordonnées calculées sous forme d'un triplet de float.

On aura :

coordvitesse.x=paratrajectoire.σ(coordposition.y-coordposition.x)

coordvitesse.y=coordposition.x(paratrajectoire.ρ-coordposition.z)-coordposition.y

coordvitesse.z=coordposition.x*coordposition.y-paratrajectoire.β*coordposition.z

2) Fonction position_lorenz

La fonction permettra de calculer les positions à partir des vitesses et des coordonnées.

La fonction position_lorenz prendra en entrée les coordonnées (x,y,z), l'incrément dt et les vitesses.

Déclaration : tri position_lorenz(tri coordposition, float dt, tri coordvitesse)

Elle retournera les trois coordonnées calculées sous forme d'un triplet de float. On aura :

coordposition.x=coordposition.x+coordvitesse.x*dt

coordposition.y= coordposition.y+coordvitesse.y*dt

coordposition.z= coordposition.z+coordvitesse.z*dt

3) Fonction vitesse_Rössler

La fonction vitesse_Rössler prendra en entrée les coordonnées (x,y,z) et les paramètres pour le calcul de la trajectoire.

La fonction permettra de calculer les coordonnées vitesses à partir des équations de Lorentz et des coordonnées.

Déclaration : tri vitesse_Rössler(tri coordposition, tri paratrajectoire)

Elle retournera les trois coordonnées calculée sous forme d'un triplet de float. On aura :

coordvitesse.x=-coordposition.y-coordposition.z)

coordvitesse.y=coordposition.x+(paratrajectoire.σ*coordposition.y)

coordvitesse.z=paratrajectoire.ρ+coordposition.z*(coordposition.x-paratrajectoire.β)

4) Fonction position_Rössler

La fonction permettra de calculer les positions à partir des vitesses et des coordonnées.

La fonction position_Rössler prendra en entrée les coordonnées (x,y,z), l'incrément dt et les vitesses.

Déclaration : tri position_Rössler(tri coordposition, float dt, tri coordvitesse)

Elle retournera les trois coordonnées calculées sous forme d'un triplet de float. On aura :

coordposition.x=coordposition.x+coordvitesse.x*dt

coordposition.y= coordposition.y+coordvitesse.y*dt

coordposition.z= coordposition.z+coordvitesse.z*dt

5) Fonction position_Hénon

La fonction permettra de calculer les positions d'un point dans un plan suivant les coordonnées précédentes uniquement.

La fonction position_Hénon prendra en entrée les coordonnées (x,y,z) (z étant fixé à 0 pour toute la simulation, cela évite de déclarer une nouvelle structure à 2 floats juste pour ce système), ainsi que les coordonnés pour le calcul (a,b,c) (la troisième étant encore une fois fixée à 0 car le système dépend seulement de 2 paramètres).

Déclaration : tri position_Hénon(tri coordposition, tri paratraj);

Elle retournera les trois coordonnées calculées sous forme d'un triplet de float. (z étant toujours égal à 0)

On aura :

coordposition.x = coordposition.y + 1 - a*(coordposition.x)^2

coordposition.y = b*coordposition.x

coordposition.z = 0

III. Librairie Interaction fichier

Cette librairie contiendra les fonctions permettant de lire et écrire dans un fichier (et afficher). C'est une librairie contenant des fonctions axées sur les interactions avec un fichier.

1) Fonction ecr_fichier

Déclaration: int ecr_fichier(FILE* fichier, tri coord, float temps);

Cette fonction prend en entrée un pointeur de fichier ainsi que les coordonnés d'un point (sous la forme d'un triplet de float) avec le temps correspondant, et les écrit dans un fichier ouvert précédemment dans le main. Le fichier sera ouvert en mode ajout et fermé une seul fois dans le main, pour éviter un ralentissement des calculs dû au fait de l'ouvrir et le fermer à chaque fois dans la fonction ecr_fichier.

2) Fonction lire_fichier

Déclaration: float lire_fichier(FILE* fichier, int position);

Cette fonction prend en entrée un pointeur de fichier et la position d'un nombre contenu dans le fichier (il y a 4 float par ligne: le temps et les coordonnées x,y et z) le but de la fonction est de se déplacer jusqu'au float demandé et de le renvoyer.

IV. Librairie Choix

Dans cette librairie seront regroupées toutes les fonctions qui demandent à l'utilisateur d'entrer les informations (positions initiales, incréments, paramètres,...) que ce dernier souhaite utiliser. Toutes ces fonctions sont liées car elles utilisent toutes une interaction avec l'utilisateur. Pour toutes, Il faudra être assez clair avec l'utilisateur pour qu'il rentre les bonnes données et dans les bons types, c'est pour cela qu'on utilisera la librairie nommée « entrées » contenant des fonctions vérifiant que l'on entre bien les types nécessaires au fonctionnement

1) Fonction demander_position

La fonction permettra de demander à l'utilisateur la position initiale.

La fonction « demander_position » ne prendra rien en entrée.

Déclaration : tri demander_position();

Elle retournera donc les trois coordonnées entrée par l'utilisateur sous forme d'un triplet de float.

2) Fonction demander_increment

La fonction permettra de demander à l'utilisateur la valeur de l'incrément "dt".

La fonction « demander_increment » ne prendra rien en entrée.

Déclaration : float demander_increment();

Elle retournera donc la valeur de "dt" choisie par l'utilisateur.

3) Fonction demander_parametres

La fonction permettra de demander à l'utilisateur les valeurs de paramètres qu'il souhaite utiliser dans son système de Lorenz.

La fonction « demander_parametres » ne prendra rien en entrée.

Déclaration : tri demander_parametres();

Elle retournera un triplet de float contenant les valeurs des paramètres que l'utilisateur souhaite utiliser.

4) Fonction demander_tmax

La fonction permettra de demander à l'utilisateur le temps où l'on étudie la trajectoire.

La fonction « demander_tmax » ne prendra rien en entrée.

Déclaration : float demander_tmax();

Elle retournera la valeur Tmax choisie par l'utilisateur.

V. Librairie Entrees

Librairie contenant des fonctions qui permettront de vérifier le bon type des éléments rentrés par l'utilisateur du programme, ainsi que les fonctions dont elles ont besoin pour fonctionner.

1) Fonction lire_fin_ligne

Fonction comptant le nombre de caractères non blancs dans le buffer jusqu'à la fin de la ligne.

Déclaration: int lire_fin_ligne ();

2) Fonction lire_format

Fonction lisant une variable avec le type donné en argument.

Déclaration: int lire_format (char* str, void * adresse);

Elle prend comme arguments une chaîne de caractères du type "%d" par exemple, pour ensuite faire un scanf sur le type correspondant, et une adresse ou sera enregistré ce qui a été lu.

Elle appelle lire_format pour vérifier qu'il n'y ai bien plus aucun caractère dans la ligne après le scanf.

3) Fonction lire_entier

Fonction permettant de lire un entier uniquement, la lecture recommence en boucle tant que les caractères entrés ne représentent pas un entier.

Déclaration: int lire_entier(int * entier);

Elle prend comme argument l'adresse ou sera stocké l'entier lu.

Elle appelle lire_format avec les arguments ("%d", entier).

4) Fonction lire_decimal

Fonction permettant de lire un float uniquement, la lecture recommence en boucle tant que les caractères entrés ne représentent pas un float (un entier sera lu aussi, car c'est un float particulier).

Déclaration: float lire_decimal(float *);

Elle prend comme argument l'adresse ou sera stocké le float lu.

Elle appelle lire_format avec les arguments ("%f", float).

VI. Librairie Simulation

Librairie contenant toutes les fonctions de simulation des systèmes dynamiques jusqu'au temps final (appelant les fonctions de mise à jour de la vitesse et de la position des systèmes dynamiques). Elle contiendra entre autres simul_lorenz.

1) Fonction simul_lorenz

void simul_lorenz(FILE* fichier ,float tpssimulation, float dt, tri coordpos, tri parametres);

Cette fonction appelle récursivement jusqu'à temps = Tmax la fonction de mise à jour de la vitesse vitesse_lorenz avec les arguments (coordpos, parametres) puis celle de la position position_lorenz avec les arguments (coordpos, dt, coordvit (renvoyé par vitesse_lorenz)) et écrit à chaque itération les nouvelles coordonnées dans un fichier en appelant ecr_fichier avec les arguments (fichier, coord, temps) temps étant actualisé à chaque boucle en lui ajoutant dt.

2) Fonction simul_Rössler

void simul_Rössler(FILE* fichier, float tpssimulation, float dt, tri coord, tri parametres);

Cette fonction appelle récursivement jusqu'à temps = Tmax la fonction de mise à jour de la vitesse vitesse_Rössler avec les arguments (coordpos, parametres) puis celle de la position position_Rössler avec les arguments (coordpos, dt, coordvit (renvoyé par vitesse_Rössler)) et écrit à chaque itération les nouvelles coordonnées dans un fichier en appelant ecr_fichier avec les arguments (fichier, coord, temps) temps étant actualisé à chaque boucle en lui ajoutant dt.

3) Fonction simul_Hénon

void simul_Hénon(FILE* fichier, float tpssimulation, float dt, tri coord, tri parametres)

Cette fonction appelle récursivement jusqu'à temps = Tmax la fonction de mise à jour de la position position_Hénon avec les arguments (coordpos, parametres) et écrit à chaque itération les nouvelles coordonnées dans un fichier en appelant ecr_fichier avec les arguments (fichier, coord, temps) temps étant actualisé à chaque boucle en lui ajoutant dt.

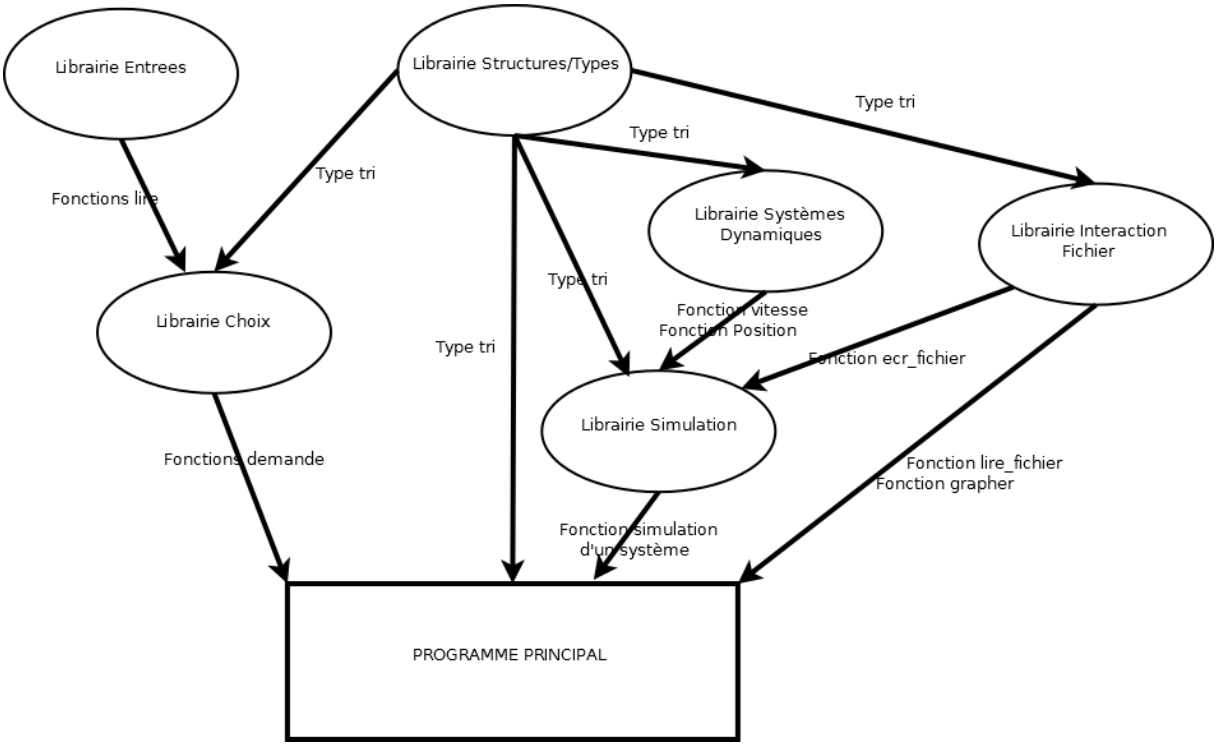
VII. Dépendances entre librairies

Toutes les librairies sont dépendantes entre elles. En effet, elles permettront que le programme aboutisse.

La première librairie sur les structures et les types est essentielle car elle est utilisée dans chaque librairie, ainsi que dans le programme principal, elle permet de définir le type des variables à donner mais aussi à renvoyer. Dans la suite les librairies s'enchaînent.

La librairie “Simulation” dépend de “Systèmes Dynamiques” et de “Interaction fichier”, elle a besoin de ces deux librairies pour un fonctionnement correct.

La librairie “Choix” n’a besoin que du type tri, et est appelée directement par le programme principal au début pour tout initialiser.



VIII. Le main

Dans le programme principal on retrouvera, dans l'ordre :

- Les fonctions de la librairie "Choix", afin de tout d'abord demander à l'utilisateur quel système dynamique utiliser et ensuite initialiser la position initiale et les différents paramètres aux choix faits par l'utilisateur.
- La fonction simulation correspondant au système dynamique choisit, qui se chargera d'appeler récursivement jusqu'à Tmax:
    - La fonction vitesse qui calcule la vitesse en chaque point
    - La fonction position qui calcule la position de chaque point
    - La fonction ecr_fichier qui va écrire dans un fichier les coordonnées à l'instant t.
- La fonction qui lit les données dans le fichier
- Une fonction qui permettra d'afficher la courbe (à déclarer et définir plus tard)

IX. Tests pour notre programme de base avec les équations de Lorenz

Pour vérifier si notre programme renvoit des bonnes valeurs on peut comparer les valeurs rendues par le problème avec d'autres déjà connues pour des valeurs particulières (valeurs obtenues par calcul direct):

Test n°1 : dt=0.01, x=1, y=2, z=3, σ=10, ρ=28, β=8/3

t	x	y	z	vx	vy	vz
0	1	2	3	10	23	-6
0,01	1,1	2,23	2,94	11,3	25,34	-5,39
0,02	1,21	2,48	2,89	12,7	27,98	-4,68
0,03	1,34	2,76	2,84	14,23	30,95	-3,87
0,04	1,48	3,07	2,8	15,9	34,28	-2,91
0,05	1,64	3,42	2,77	17,74	37,99	-1,78
0,06	1,82	3,8	2,75	19,77	42,12	-0,44
0,07	2,02	4,22	2,75	22	46,7	1,17
0,08	2,24	4,68	2,76	24,47	51,76	3,11
0,09	2,48	5,2	2,79	27,2	57,34	5,46
0,1	2,75	5,77	2,85	30,22	63,48	8,31
0,11	3,06	6,41	2,93	33,54	70,19	11,77
0,12	3,39	7,11	3,05	37,21	77,5	15,99
0,13	3,76	7,89	3,21	41,24	85,4	21,12
0,14	4,18	8,74	3,42	45,65	93,89	27,38
0,15	4,63	9,68	3,69	50,48	102,91	34,99
0,16	5,14	10,71	4,04	55,72	112,35	44,22
0,17	5,69	11,83	4,48	61,38	122,06	55,41
0,18	6,31	13,05	5,04	67,45	131,78	68,89
0,19	6,98	14,37	5,73	73,88	141,14	85,06
0,2	7,72	15,78	6,58	80,61	149,61	104,31

Test n°2 : dt=0.01, x=0, y=0, z=0, σ=10, ρ=28, β=8/3

Tout doit être égal à 0.

Test n°3 : dt=0.01 , x=1, y=2, z=1, σ=0, ρ=0, β=0

t	x	y	z	vx	vy	vz
0	1	2	1	0	-3	2
0,01	1	1,97	1,02	9,7	25,01	-0,75
0,02	1,1	2,22	1,01	11,23	27,39	-0,26
0,03	1,21	2,49	1,01	12,85	30,15	0,32
0,04	1,34	2,8	1,01	14,58	33,31	1,04
0,05	1,48	3,13	1,02	16,45	36,89	1,91
0,06	1,65	3,5	1,04	18,49	40,93	2,98
0,07	1,83	3,91	1,07	20,74	45,45	4,3
0,08	2,04	4,36	1,12	23,21	50,49	5,92
0,09	2,27	4,87	1,17	25,94	56,09	7,93
0,1	2,53	5,43	1,25	28,95	62,29	10,4
0,11	2,82	6,05	1,36	32,29	69,12	13,45
0,12	3,14	6,74	1,49	35,97	76,6	17,22
0,13	3,5	7,51	1,66	40,03	84,77	21,87
0,14	3,9	8,35	1,88	44,51	93,61	27,6
0,15	4,35	9,29	2,16	49,42	103,1	34,65
0,16	4,84	10,32	2,51	54,78	113,16	43,31
0,17	5,39	11,45	2,94	60,62	123,66	53,91
0,18	6	12,69	3,48	66,93	134,38	66,83
0,19	6,67	14,03	4,15	73,67	144,99	82,5
0,2	7,4	15,48	4,97	80,8	155,01	101,38

Nous allons donc, tout au long du projet, s'aider de ce plan de travail. Cela nous permettra d'être plus efficace et de pouvoir bien se répartir le travail entre nous.

Il est possible qu'au fur et à mesure du travail de codage, nous découvrons d'autres façons de procéder que celles décrites dans notre plan, plus simples à mettre en œuvre ou permettant un code plus compact et optimisé et que nous soyons amenés à modifier en partie le plan nous guidant, cela fait aussi partie de notre travail. Mais nous garderons toujours l'objectif en tête.