# Enigma, modélisateur de la trajectoire d'un point

*Par Hugo AVRIL et Mihaja RAZAFIMAHEFA*

# Introduction

Enigma est un projet réalisé durant notre année universitaire L2 CUPGE en Informatique.
Ecrite en C et compilée avec Makefile, Enigma permer de représenter graphiquement la trajectoire d'un point dont sa position est gouvernée par un système dynamique. Cette représentation est rendu possible à l'aide du programme `gnuplot` .

# Utilisation

A lancement du programme, l'utilisateur a tout d'abord le choix entre une configuration (1) automatique ou (2) manuelle de la mise à jour de la vitesse.

Ensuite il est présenté avec plusieurs choix de systèmes dynamiques pour calculer sa trajectoire :

1. **Lorenz**
2. **Hugo**
3. **Mihaja**

> **Remarque**
> Les équations 2 et 3 ne sont données qu'à titre d'exemple pour illustrer la modularité du programme.

L'utilisateur est ensuite tenu de remplir toutes les paramètres d'une trajectoire :

- la position initiale (par ses composantes $x$, $y$ et $z$)
- l'incrément $dt$
- le temps d'arrêt $T_{max}$
- les constantes des systèmes dynamiques

Si l'utilisateur a choisi une configuration manuelle de la vitesse, celui-ci doit indiquer l'incrément $dx$, $dy$ et $dz$ à chaque incrément $dt$.

Le programme écrit alors chaque nouvelle position dans un fichier `sortie.dat` pour chaque incrément $dt$. Lorsque $T_{max}$ sera atteint, le programme ouvrira automatique `gnuplot` avec la trajectoire calculée. L'utilisateur vera alors la courbe du point modélisé.

> **Remarque**
> L'utilisateur est tenu d'avoir `gnuplot` installé sur son système d'exploitation pour un bon fonctionnement du programme

# Structure

Le programme est compilé à partir d'un `Makefile` principal : celui se trouvant à la racine du projet. Il va tout d'abord compiler les librairies organisées en **modules**, dont les commandes se trouve dans des sous- `Makefile` , dans le dossier de chaque module. Puis il va compiler le programme principale `main` .

## main.c

`main` est le programme qui orchestre la création et le calcul de la trajectoire, puis effectue les commandes systèmes pour afficher la courbe sur `gnuplot` .
Pour cela, il se sert d'un pipe pour pouvoir y imprimer les commandes que `gnuplot` exécutera. Ainsi, l'utilisateur n'a besoin que de compiler le programme avec `make` pour pouvoir afficher sa courbe.

# Les modules élémentaires

A l'exception de `main` , le code source est organisé en modules, comprenant plusieurs fonctions et rassemblées par leur similarité ou par leur contribution au programme.
Les fichiers `.h` sont aussi organisés en fonctions des modules et permettent de les définir, en regroupant les structures communes avec leurs opérations possibles.

## calcule_trajectoire

Ce module constitue le coeur du programme, celui qui initialise la trajectoire, demande sa configuration en appelant les fonctions d'entrées, et écrit la trajectoire calculée dans le fichier de sortie.

Une trajectoire est symbolysée par ses paramètres (position, constantes, ...) et par un système dynamique associé :

```
struct trajectoire {
    struct parametres *parametres;
    struct systeme_dynamique *equation_mouvement;
};
```

Les fonctions du modules sont :

- `nouvelle_trajectoire` : demande le choix des paramètres du programme à l'utilisateur, crée une nouvelle trajectoire et y attribue une équation du mouvement en fonction de l'entrée utilisateur.
- `calcul_trajectoire` : ouvre le fichier de sortie, appelle la fonction qui symbolise l'équation du système dynamique et écrit ce qu'il a calculé pour chaque $t \in [0; T_{max}]$

# entrees

Ce module est responsable de toutes les entrées utilisateurs du programme. Il regroupe notamment la lecture des constantes d'un système, des paramètres initiales ainsi que de l'équation de la vitesse si l'entrée est manuelle.

Les fonctions du modules sont :

- `demande_hugo` , `demande_lorenz` , `demande_mihaja` : crée une structure représentant un ensemble de constante pour un système donné et demande à l'utilisateur de spécifier leur valeur.

Un ensemble de constantes est défini par un tableau les stockant, le nombre de constantes qui est aussi la longueur du tableau, et enfin un tableau de caractères qui contient les premieres lettres de chaque constante :

```
struct constantes {
    int nbre_constantes;
    // * l'ordre des lettres est important : il permet d'identifier sa place dans float *t
    char *lettres_constantes;
    float *t;
};
```

- `demande_parametres` : demande les paramètres initiales du système (position initiale, $dt$, $T_{max}$) et calcule en même temps le nombre de points que contiendra la courbe

Toutes ces valeurs sont stockées dans une seule structure les regroupant :

```
struct parametres {
    struct position *position_initiale;
    float dt;
    float t_max;
    int nbre_points;
};
```

Une position est notamment représentée par ses composantes `x`, `y` et `z` au sein d'une structure symbolique :

```
struct position {
    float x;
    float y;
    float z;
};
```

- `demande_vitesse` : demande l'équation $\frac{du}{dt}$ ($u \in \{x, y, z\}$) représentant la mise à jour de la vitesse. Il implémente notamment la notation polonaise pour pouvoir facilement transcrire l'entrée utilisateur en équation. Cette fonction à elle seule est dépendante de deux modules outils : **liste** et **lecture**

## equations

Ce module regroupe les équations de la trajectoire d'un système ainsi qu'une équation "custom" pour calculer la vitesse à partir de l'entrée manuelle de l'utilisateur.

En effet, un système dynamique est symbolisé par une unique structure :

```
struct systeme_dynamique {
    struct constantes *constantes;
    deplacement equation;
};
```

Les constantes ayant déjà été abordé, le point intéressant ici est `deplacement equation`.

`deplacement` est un type spécial :

```
typedef struct position* (*deplacement)(struct position *p, struct constantes *cte, float dt);
```

C'est le type modèle qui effectue le calcul du changement de position. Il prend en entrée la position actuelle $(x, y, z)$, les constantes du système et l'incrément $dt$. Puis renvoie l'adresse de la nouvelle position $(x + \frac{dx}{dt} \cdot dt, y + \frac{dy}{dt} \cdot dt, z + \frac{dz}{dt} \cdot dt)$.

> **Remarque**
> Toutes les fonctions `_transform` sont du type `deplacement`. C'est la norme adoptée par le projet.

Chaque système dynamique se voit être attribué de la bonne fonction `_transform` au moment de son initialisation, sauf dans le cas d'une mise à jour de la vitesse manuelle de l'utilisateur, auquel cas la fonction `custom_transform` lui sera attribuée.

# Les modules complémentaires ou outils

## lecture

Ce module contient deux fonctions utiles à `demande_vitesse` :

- `evaluer_operation` : permet de faire l'operation arithmétique de deux nombres à partir d'un chaîne de caractères représentant une opération ($+$, $-$, $*$, $/$)
- `is_entree_valide` : vérifie à partir des caractères autorisées si l'entrée utilisateur est bien valide pour les opérations qui suivront. Dans le cadre de `demande_vitesse`, les caractères autorisées sont toujours les premieres lettres de chaque constante (défini dans `struct constantes`), les lettres composant une position ($x$, $y$ et $z$) et enfin les lettres d'opérations arithmétiques ($+$, $-$, $*$, $/$).

## liste

Ce module contient une implémentation des listes liées en C afin d'obtenir, à partir de la notation polonaise, une pile de données de l'entrée utilisateur.

Une liste est symbolisée par :

```
struct liste_valeurs
{
    int nbre_valeurs;
    struct valeur *premier;
};
```

Avec `struct valeur` qui un élément de la liste, symbolisé par :

```
struct valeur {
    float val;
    struct valeur *suivant;
};
```

On remarque ici donc qu'une liste liée n'est qu'une suite d'éléments (de type `struct valeur`) contenant sa valeur et l'élément suivant.

Pour pouvoir manipuler une liste liée, le module contient trois fonctions :

- `nouvelle_liste` : permet d'instancier une nouvelle liste ne contenant aucun élément (`nbre_valeurs = 0`)
- `ajouter_liste` : permet d'ajouter un nouvel élément **au début de la liste** (pour respecter la notation polonaise). Dans le cas où la liste est encore vide, il assigne la nouvelle valeur à `premier->val`. Sinon il crée un nouvelle élément `valeur` où il assignera son élément suivant au premier élément de la liste. Le premier élément de la liste aura alors l'adresse de `valeur`. Ensuite il incrémente `nbre_valeurs`.
- `supprimer_liste` : permet de supprimer **le premier élément de la liste** (toujours pour respecter la notation polonaise et la notion de pile). Pour cela, il stocke d'abord l'adresse du premier élément puis pointe le premier élément de la liste à l'élément suivant. Le premier élément n'étant plus lié à la liste, il le libère de la mémoire avec `free`. Enfin, il décrémente `nbre_valeurs`.

Pour plus d'informations sur l'implémentation de ces fonctions, la théorie des listes et des piles, plus de détails ici.

## position

Ce module permet de manipuler facilement une `struct position` avec plusieurs fonctions :

- `fprintf_position` : imprime la position à un instant $t$ donné dans un fichier
- `get_position` : permet d'obtenir une composante à partir d'une position donnée
- `nouvelle_position` : permet de créer une nouvelle position à partir des composantes données