Université Lyon 1 Claude Bernard
Polytech Lyon
Année 2024/2025

Projet informatique 3

Auteur:
Baptiste Ridolfi

1 Introduction

Nous allons voir tout au long de ce rapport le problème à "N corps". Ce problème représente N corps dans l'espace, tous soumis à l'attraction gravitationnelle les uns des autres. Pour ce faire, on utilisera dans un premier un dossier de fichier écrit en orientée-objet en C++ qui ressortira un fichier ".txt". Ce fichier sera par la suite implémenté dans un code Python qui nous permettra d'afficher une animation disponible sur un lien YouTube.

2 Programmation en C++

Dans cette première partie, nous allons créer 3 classes au total.

La première classe "Coordonnées" nous permettra de décrire les coordonnées d'un point dans l'espace, les planètes du système solaire, dans un repère absolu.

La deuxième, qui héritera de "coordonnées" se prénomme "Vecteur", elle permettra par la suite de définir la vitesse de nos planètes. Ce vecteur a une direction, un sens et une norme afin de définir comment nos points vont se mouvoir.

Finalement, la classe "Point Matériel" nous permettra de définir chacune de nos planètes par une masse, une position et une vitesse.

2.1 Classe coordonnées

Dans cette classe nous allons donc définir plusieurs méthodes en prenant en argument les coordonnées (X,Y) d'un point. Les deux principales méthodes sont donc "distance" qui prend un argument un autre point et qui calcule la distance entre deux points, ainsi que "translation" qui permet de translater un point d'une valeur dx, et dy.

Elles sont définies comme suit.

$$Translation : A' = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} dx \\ dy \end{bmatrix}$$

$$Distance : d = \sqrt{(x_b - x_a)^2 + (y_b - y_a)}$$

Cette classe nous sert de base pour la prochaine classe "Vecteur" qui sera sa classe mère.

2.2 Classe Vecteur

Cette classe vecteur, qui hérite de la classe "coordonnées" va faire apparaître plusieurs nouvelles méthodes propre au vecteur. Elle prend en coordonnées la composante V_x **et** V_y d'un vecteur, qui dans notre cas, sera la vitesse de nos planètes. Tout d'abord, on calcule la norme 2 entre 2 vecteurs que l'on verra plus tard, puis on calcule l'angle entre les composantes X et Y d'un vecteur. On crée ensuite une méthode rotation, qui nous permet de faire tourner notre vecteur d'un certain angle.

$$\text{Norme 2 : } \sqrt{V_x^2 + V_y^2}$$

$$\text{Angle : } \arctan(\frac{V_y}{V_x})$$

$$\text{Rotation : } \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} * \begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{matrix} V_x cos\theta - V_y \sin\theta \\ V_x \sin\theta + V_y cos\theta \end{matrix}$$

Ensuite on surcharge les opérateurs "+" qui permet d'additionner deux objets Vecteurs entre eux, puis "*" qui permettra de faire une homothétie, elle prend en argument un double et un vecteur. Ces deux surcharges renverront chacune un vecteur. Finalement, on surcharge une nouvelle fois l'opérateur "+" pour qu'il fasse une translation d'un point au bout d'un vecteur.

Ils sont définis comme suit.

```
vect operator+(vect A, vect B) // Addition des coordonnees du vecteur
{
    double v_x,v_y;
    v_x=B.getvX()+A.getvX();
    v_y=B.getvY()+A.getvY();
    vect resultant(v_x,v_y);
    return resultant;
}

vect operator*(vect A, double a) // Agrandissement/Reduction du vecteur
{
    double v_x,v_y;
    v_x=a*A.getvX();
    v_y=a*A.getvY();
    vect result(v_x,v_y);
    return result;
}

coordonnee operator+( vect A, coordonnee P) // On prend le point P et on le met au bout du vecteur
     (translation)
{
    double mx, my;
    mx = P.getX() + A.getvX(); // Utiliser getX() et getY()
    my = P.getY() + A.getvY();
    coordonnee resultat(mx, my);
    return resultat;
}

}
```

Listing 1: Surcharge d'opérateur

Ces deux classes étant faites, on peut désormais définir proprement nos points matériels dans la prochaine classe.

2.3 Classe Point Matériel

Dans cette troisième et dernière classe, nous allons utiliser les deux précédentes classes afin de définir la position et la vitesse de notre point qui seront pris en arguments par nos objets point mat. A ces deux arguments, on rajoute finalement un double qui servira pour la masse de nos objets.

Cette classe voit plusieurs méthodes apparaître. Deux méthodes qui permettront de calculer la nouvelle position ainsi que la nouvelle vitesse à chaque indentation de temps dans le main (que nous verrons plus tard). On utilisera la méthode d'Euler pour calculer la position et la vitesse. En complément de ces deux nouvelles méthodes, on crée 2 méthodes "setvitesse" et "setcoord" qui permettront de modifier directement les arguments de la classe pointmat.

Finalement, une dernière méthode permettra de calculer la force entre 2 points selon la loi universelle de la gravitation.

Méthode :

Méthode d'Euler :

$$\text{Position : } \vec{x}(t+\delta t) = \vec{x}(t) + \delta t \cdot \vec{v}(t) \tag{1}$$

$$\text{Vitesse : } \vec{v}(t+\delta t) = \vec{v}(t) + \delta t \cdot \vec{a}(t) \tag{2}$$

Loi universelle de la gravitation :

$$\vec{F}_{a\to b} = -Gm_1m_2 * \frac{\overrightarrow{x_1x_2}}{||\overrightarrow{x_1x_2}||^3}$$

Avec $G = 6.67 * 10^{-11} m^3 \cdot kg^{-1} \cdot s^{-2}$

2.4 Main

Nos 3 classes sont maintenant terminées, on peut finalement écrire notre fichier texte, contenant l'ensemble des coordonnées des planètes du système solaire.

Dans un premier temps, on crée un tableau dynamique "lescorps" qui contiendra l'ensemble des planètes définies par la classe Point mat. On récupère les informations nécessaires sur internet, et on obtient les coordonnées, vitesse et masse suivantes pour chaque planète :

Planète	**Position initiale (m)**	**Vitesse initiale (m/s)**	**Masse (kg)**
Soleil	(0.0, 0.0)	(0.0, 0.0)	1.989e30
Mercure	(5.79e10, 0.0)	(0.0, 47.87e3)	3.285e23
Vénus	(1.082e11, 0.0)	(0.0, 35.02e3)	4.867e24
Terre	(1.496e11, 0.0)	(0.0, 29.78e3)	5.972e24
Mars	(2.279e11, 0.0)	(0.0, 24.077e3)	6.39e23
Jupiter	(7.785e11, 0.0)	(0.0, 13.07e3)	1.898e27
Saturne	(1.433e12, 0.0)	(0.0, 9.69e3)	5.683e26
Uranus	(2.877e12, 0.0)	(0.0, 6.81e3)	8.681e25
Neptune	(4.503e12, 0.0)	(0.0, 5.43e3)	1.024e26

Ensuite, on rentre toutes ces coordonnées dans le tableau puis on commence note boucle temporelle mentionnée plus haut. Je choisis de calculer les coordonnées sur 165 ans qui est le temps de révolution de Neptune. Cela permettra d'avoir une simulation complète de notre système solaire. Pour ce faire, on créer 3 boucles : une première qui est la boucle de temps. Puis deux autres qui vont permettre de parcourir le tableau en prenant une planète "j" et de calculer les forces engendrées par les autres planètes "i". Toujours dans la boucle j, on calcule ensuite l'accélération avec le PFD :

$$\sum_{j=0}^{n} \frac{\vec{F}_j}{m_j} = a_j$$

De cette accélération, on en déduit la vitesse puis les coordonnées de la planète à l'instant $t + dt$. Voir : (1) et (2). Puis finalement, on écrit dans le fichier "pointmat.txt", l'ensemble des coordonnées obtenues par ces boucles.Vous retrouverez le pseudo-code de notre main juste en dessous.

```
Boucle t de 0 a 3600 x 24 x 365 x 165 s:
    Boucle j de 0 a ncorps - 1 : avec ncorps qui est le nombre de planete soit 9
        Forcetotale <- (0.0, 0.0) // On initialise la force a 0 pour chaque nouveau calcul
        Boucle i de 0 a ncorps - 1 :
            Si i est different de j :
                Forcetotale <- Forcetotale + force(lescorps[i], lescorps[j])
        Fin Boucle i

        acceleration j <- Forcetotale / masse corps j)
        lescorps[j].vitesse <- acceleration * dt + lescorps[j].vitesse
        lescorps[j].position <- lescorps[j].vitesse * dt + lescorps[j].position

        Ecrire position x planete j et position y planete j dans le fichier
        Si j < ncorps - 1 :
            Ajouter une tabulation dans le fichier
    Fin Boucle j

    Ajouter un saut de ligne dans le fichier
Fin Boucle t

Fermer le fichier
```

On exécute et on obtient un fichier texte que l'on choisit d'exploiter avec Python afin de tracer les animations.

3 Résultats après programmation Python

Nous sommes désormais dans Python, on programme un code permettant d'obtenir et de tracer les graphes et animations des planètes orbitant autour du Soleil, on obtient alors les résultats suivants :

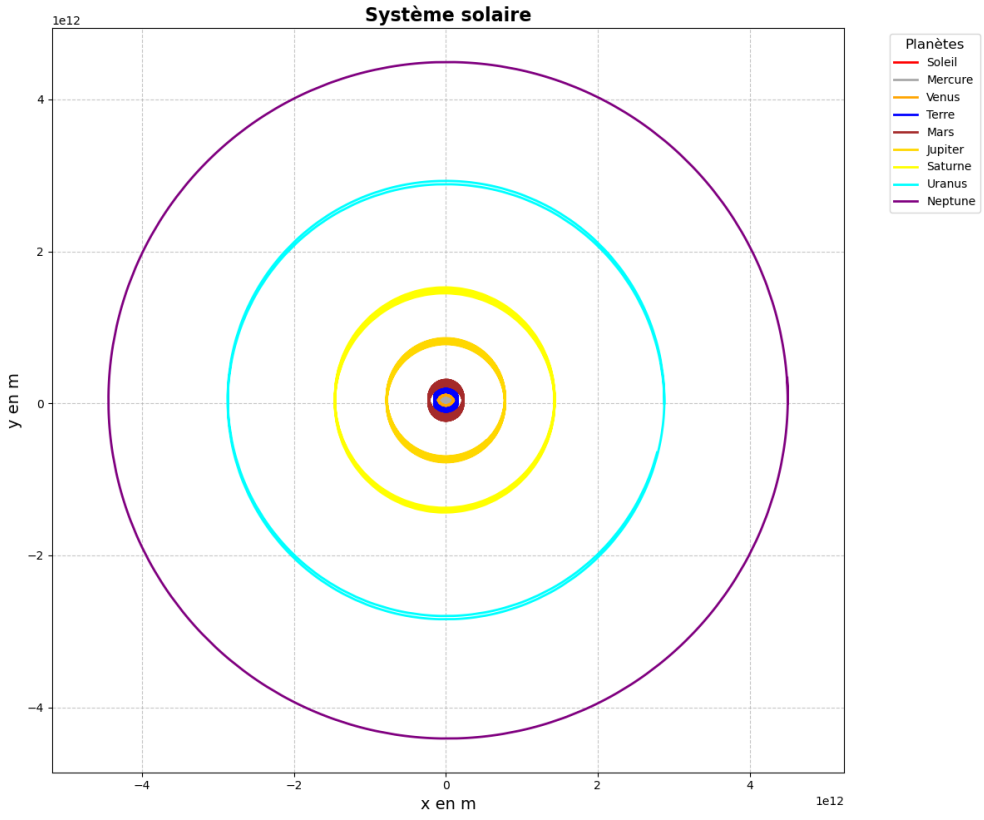

Figure 1: Système solaire

Cette image nous montre bel et bien que notre programme C++, fonctionne puisque toutes les planètes orbitent

autour du Soleil. Cependant, étant donné que les planètes dérivent vers les y positifs, on ne voit pas le Soleil ni les courbes exactes que prennent les planètes proches du Soleil.

C'est pourquoi on trace deux nouvelles animations :

- La première avec Mercure, Vénus, la Terre et Mars orbitant autour du Soleil : Disponible sur ce lien YouTube : Simulation Soleil - Mercure - Vénus - Terre - Mars
- La deuxième avec l'ensemble du système solaire qui gravitent autour du Soleil : Disponible sur ce lien YouTube : Simulation du système solaire - 165 ans

Afin de limiter une surcharge de l'animation et de la mémoire qui pourrait ralentir les animations, j'ai limité le nombre de points afficher pour la trajectoire de la planète.