Et pourtant, elle tourne !

Louis R. Couture

Abstract

Since the beginning of time, planets have been moving around the sun. Even in ancient civilizations, humans were able to find and detect planets and other astronomical objects. With the arrival of computational technologies in our societies, it is now possible to represent the orbits and positions of astronomical objects on a computer screen. However, the equations describing the path of planets can be tedious and involves complex integrals. The goal of this project was to determinate if it was possible to approximate the position and orbits of different planets through theorems from differential calculus with a 10% gap from theoretical values. It was found that it was indeed possible, but the gap would varies depending on factors such as the vicinity of celestial objects.

Mots-clés
Astrophysique, mathématiques, dynamique, informatique, planètes.

Introduction

L'univers est vaste et a toujours fasciné l'homme. C'est pourquoi ce dernier a longtemps cherché à la comprendre. Dans l'atteinte de ce but, il a usé de son savoir et de son génie pour créer des outils afin de faciliter la compréhension de l'univers. Les récentes avancées en informatique ont permis notamment de modéliser le comportement des planètes, soit comment elles se déplacent les unes par rapport aux autres et ce, en faisant appel aux notions de la physique mécanique (pour l'attraction gravitationnelle entre les corps), aux techniques de calcul différentiel (pour approximer la position des corps célestes, puisque les équations décrivant leurs trajectoires peuvent être très complexes), ainsi que les concepts de la programmation informatique.

Modéliser la trajectoire des planètes et des objets célestes est utile lorsqu'il s'agit d'avoir la position d'un corps à un instant donné. Concrètement, cela est utilisé lorsqu'on envoie des sondes vers Mars par exemple, pour trouver sa position lorsque cette sonde sera arrivée sur la planète en question. Également, plusieurs astéroïdes orbitant autour de notre étoile et pourraient causer des dommages non négligeables sur la planète Terre, d'où l'importance de connaitre la trajectoire. Il est possible de penser à l'écrasement de la météorite de Tunguska en Sibérie (Large Synaptic

Survey Telescope, 2018), en 1908, qui aurait pu causer des millions de morts si elle s'était écrasée dans une zone plus populeuse. Un autre astéroïde, nommé Bennu, d'une taille de 500 m, pourrait frapper de plein fouet la Terre le 21 septembre 2135, et c'est pourquoi la NASA tente d'éviter un contact catastrophique, et ce, en envoyant une sonde explorer ce corps mystérieux. (Couronne, 2018)

Ainsi, dans ce projet, la trajectoire des corps du système solaire sera modélisée et programmée en utilisant l'environnement de développement intégré de Maple et celui de Python avec l'interface de matplotlib. L'hypothèse suivante sera vérifiée : il est possible de déterminer la position des planètes et d'autres corps célestes avec une précision de 10% avec la méthode de Range-Kutta.

Orbite

Il faut comprendre que l'orbite, ou la trajectoire d'un objet céleste résulte de l'ensemble des forces gravitationnelles qui s'appliquent sur l'objet. La force gravitationnelle ($\overrightarrow{F_g}$) entre deux objets, la position ($\vec{r}$), la masse des deux objets (m_1, m_2) sont donnés par:

$\overrightarrow{F_g} = \frac{-Gm_1m_2}{\vec{r}^2}$ (Séguin, 2010)

Où $G = 6{,}67 \cdot 10^{-11}\ \mathrm{Nm^2/kg^2}$

Par exemple, pour déterminer la position de la Lune, il faut considérer minimalement la force gravitationnelle entre la Lune et la Terre, et celle de la Lune et du Soleil. Pour calculer ces 2 forces, il faut connaitre la position en 3D entre la Terre et de la Lune (le Soleil est à l'origine, car on considère le modèle héliocentrique), et celle entre la Lune et le Soleil. Aussi, puisque la force gravitationnelle est égale à la masse de la Lune multipliée par l'accélération, et que l'accélération est un vecteur, il faut considérer cette dernière, ainsi que la position et la vitesse en x, y et z. De plus, il faut aussi considérer que la Terre bouge à cause de la force gravitationnelle du Soleil simultanément. Cela fait beaucoup de paramètres pour résoudre uniquement la trajectoire de la Lune.

Pour approximer la position de la Lune, il faut trouver l'équation différentielle qui décrit sa position. En considérant les astres qui exercent une attraction non négligeable sur la Lune, et les paramètres de l'équation, soit m_s, la masse du Soleil, m_l, la masse de la Lune, $\overrightarrow{r_{ls}}$, la distance entre la Lune et le Soleil, $\overrightarrow{r_{lt}}$, la distance entre la Lune et la Terre :

Trouver la position de la Lune correspond à résoudre une équation différentielle, soit une équation contenant la variable indépendante (temps), des variables dépendantes (position et vitesse), l'équation différentielle pour déterminer la position de la Lune considérant le Soleil et la Terre est $\frac{-Gm_sm_l}{\overrightarrow{r_{ls}}} + \frac{-Gm_tm_l}{\overrightarrow{r_{lt}}} = m_l\vec{a}$

Résoudre une équation différentielle revient à trouver l'expression qui relie les variables dépendantes et indépendantes, soit d'exprimer la position en fonction du temps.

Méthodes d'approximations

Les mathématiciens ont développé des méthodes qui permettent d'approximer les équations différentielles. Ces méthodes ont une précision variable.

La première, soit la méthode d'Euler, est une approximation du théorème de Taylor qui permet de résoudre une équation différentielle de la forme $y' = f(x,y)$, en ayant un point (x,y) connu (Kreyszig, 1993)

La méthode d'Euler consiste à calculer la pente de la tangente en un point donné, pour approximer la position du prochain point, et de réappliquer cette méthode pour le point suivant et ainsi de suite.

La méthode d'Euler est la suivante :

$$y_{n+1} = y_n + hf(x,y)$$
$$x_{n+1} = x_n + h$$

Où y = f(x) et y' = f(x,y), et h est l'écart entre deux points.

Voici une mise en situation de l'application de la méthode d'Euler : il faut tracer une équation différentielle dont la solution est y'=y-x+1 et qui passe par le point ((x_0 = 0,y_0 = 1). Il est possible de comparer avec la solution exacte qui est $y = x + e^x$.

Pour trouver le point suivant, il faut appliquer la méthode d'Euler.

$$y_1 = y_0 + h\,f(x_0, y_0)$$
$$y_1 = 1 + 1\,(1 - 0 + 1\,) = 3$$
$$x_1 = x_n + h = 0 + 1 = 1$$

Le point suivant serait donc (1,3), alors que la valeur exacte est plutôt (1;3,7). Voici la représentation graphique de cette mise en situation.

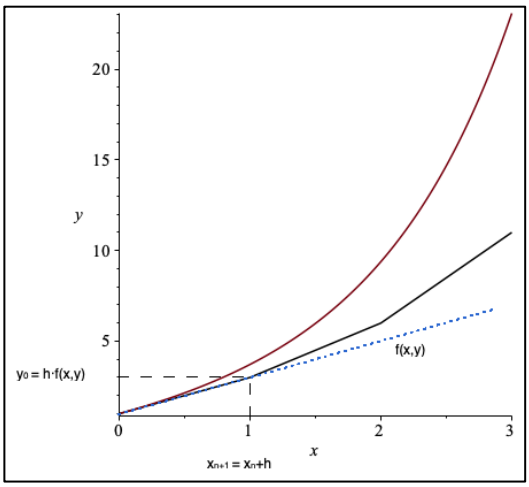

Figure 1. La représentation graphique de la méthode d'Euler

Pour plus de précision, il est possible d'utiliser la méthode de Runge-Kutta II. Au lieu d'approximer la position avec la tangente, cette dernière est calculée deux fois entre deux points.

$$y_{n+1} = y_n + \frac{1}{2}\,h(f(x_n, y_n) + f(x_{n+1}, k)$$
$$x_{n+1} = x_n + h$$
$$k = y_n + h\,f(x_n, x_n)$$

(Kreyszig, 1993)

Il est possible d'être encore plus précis en utilisant la méthode de Runge-Kutta IV, cette méthode est en quelque sorte une amélioration de la méthode de Runge-Kutta II. Au lieu de calculer la pente de la tangente 2 fois d'un point à l'autre, cette pente est calculée 4 fois. La moyenne de ces tangentes se retrouve à pointer où sera le point suivant de l'approximation, qui sera beaucoup plus précise. L'algorithme de Runge Kutta IV est le suivant.

Pour n de 0 à N

$$k_1 = hf(x_n, y_n\,)$$

$$k_2 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right)$$
$$k_3 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2\right)$$
$$k_4 = hf(x_n + h, y_n + k_3)$$
$$x_{n+1} = x_n + h$$
$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$
(Kreyszig, 1993)

Dans ce projet, la précision des différentes méthodes d'approximations sera testée

Matériel et méthodologie

Test des méthodes

Les 3 méthodes ont été testées avec la fonction $y = xe^x$ sur l'intervalle [0,1] sachant que la valeur initiale était de 1, elles ont été comparées

Trajectoire des astres et des planètes

En reprenant l'équation différentielle qui permet de résoudre la position de la Lune et en divisant par la masse de la Lune, il est obtenu

$$\frac{-Gm_s m_l}{\overrightarrow{r_{ls}}} + \frac{-Gm_t m_l}{\overrightarrow{r_{lt}}} = m_l \vec{a}$$
$$-\frac{Gm_s}{\overrightarrow{r_{ls}}} + -\frac{Gm_t}{\overrightarrow{r_{lt}}} = \vec{a} = \frac{d^2\overrightarrow{r_{ls}}}{dt}$$

Cependant, il n'est pas possible de résoudre la méthode de Runge-Kutta IV avec une dérivée seconde. Il faut donc trouver une autre façon puisque $\vec{a} = d\vec{v}/dt$

$$-\frac{Gm_s}{\overrightarrow{r_{ls}}} + -\frac{Gm_t}{\overrightarrow{r_{lt}}} = \vec{a} = \frac{d\vec{v}}{dt}$$

Il est ainsi possible d'obtenir la vitesse de la Lune à chaque point. Comme $v = f(\vec{r}, t) ou \frac{d\vec{r}}{dt}$, il est possible d'approximer la position de la Lune en appliquant la méthode de Runge Kutta IV pour la position en même temps que Runge Kutta IV pour la vitesse.

Il faut aussi décomposer l'équation de l'accélération en x, en y et en z.

Enfin, deux systèmes ont été programmés dans cette expérience. Celui de la Terre, de la Lune et du Soleil et celui de Saturne, de Titan et du Soleil. Le Soleil est toujours à l'origine. Le code est offert en annexe.

Matériel et logiciels utilisés

Ce projet a été fait avec un ordinateur Mac Book Pro datant de 2015. Le logiciel de programmation était initialement Maple, mais il n'était pas possible d'obtenir des graphiques en 3D avec celui-ci, et il a été remplacé par la bibliothèque de programmation Matplotlib, ainsi qu'à l'environnement de programmation du langage informatique Python. Le chiffrier utilisé est Excel.

Unités

Dans ce projet, les unités de position sont en unités astronomiques, (au) les unités de temps sont en jours, et les unités de vitesse se retrouveraient donc à être en au/j. Une unité astronomique correspond à la distance entre la Terre et le Soleil, soit 149 598 600 km (Universalis, 2018).La constante gravitationnelle se retrouve donc à être $1{,}488 \cdot 10^{-34}$ au^3/j^2kg

Résultats

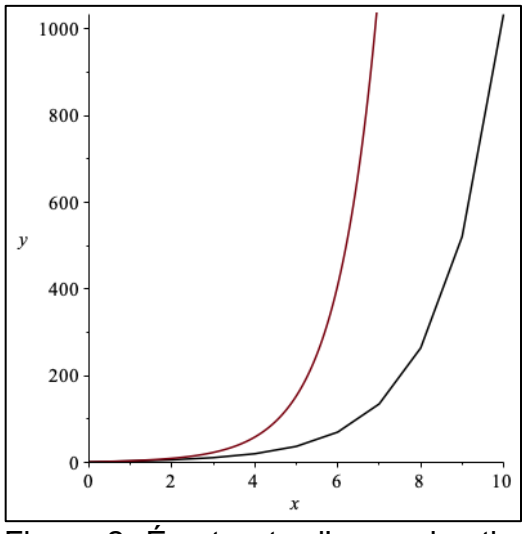

Figure 2. Écart entre l'approximation d'Euler (noir), et la fonction réelle (rouge)

La figure 2 représente la fonction $x+e^x$ et son approximation avec la méthode d'Euler

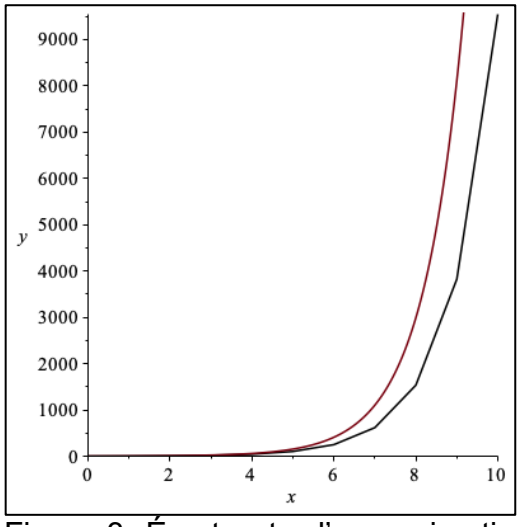

Figure 3. Écart entre l'approximation de Runge-Kutta II (noir) et la fonction réelle (rouge)

La figure 3 représente la fonction $x+e^x$ et son approximation avec la méthode de Runge-Kutta II

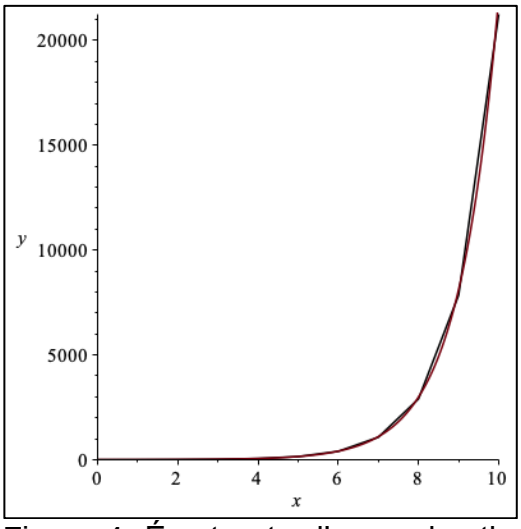

Figure 4. Écart entre l'approximation de Runge-Kutta II (noir) et la fonction réelle (rouge)

La figure 4 représente la fonction $x+e^x$ et son approximation avec la méthode de Runge-Kutta IV

Tableau 1. Écart entre la mesure exacte et l'approximation pour les 3 méthodes.

Méthode	N=10 h=0,1	N=20 h = 0,05
Euler	-0,1245	-0,064
Range Kutta II	-0,0042	0,001
Range Kutta IV	$-2{,}084 \cdot 10^{-6}$	$-1{,}32 \cdot 10^{-7}$

Le tableau 1 représente l'écart entre la valeur exacte et la valeur approximée pour les différentes méthodes et différentes valeurs de pas (h)

Tableau 2. Comparaison entre la position (au) et la vitesse (au/j) de la Terre et de la Lune et les valeurs théoriques

Terre	Vecteur	Norme
Position selon les calculs (au)	0,535i+0,762j+0,3304 k	0,988
Vitesse selon les calculs(au/j)	-0,01474i+0,008496j+0,00368k	0,01741
Position selon la théorie (au)	0,537i+0,761j+0,3299k	0,988
Vitesse selon la théorie (au/j)	-0,01472i+0,008524j+0,00369k	0,01741
Écart position (au)	0,001i+0,001j+0,0005k	0,002
Écart vitesse (au/j)	0,000021i+0,000028j+0,000012k	0,00004
Erreur position (%)		0,219
Erreur vitesse (%)		0,214
Lune		
Position selon les calculs (au)	0,533i+0,763j+0,331k	0,988
Vitesse selon les calculs (au/j)	-0,01499i +0,00797j+0,00349k	0,01733
Position selon la théorie (au)	0,535i+0,762j+0,33061k	0,988
Vitesse selon la théorie (au/j)	-0.01502i+0,00803j+0,00352j	0,01739
Écart position (au)	0,002i+0,001j+0,0004k	0,0021
Écart vitesse (au/j)	0,00003i+0,00005j+0,00003k	0,00007
Erreur position (%)		0,00215
Erreur vitesse (%)		0,00007

Le tableau 2 représente la valeur approximée et réelle de la position de la Terre ainsi que de la Lune au 30 octobre 2019, soit un an terrien après que la simulation a été commencée. Il contient également les vitesses, les écarts entre les données approximées et réelles, le tout sous forme vectorielle et scalaire. Les erreurs en pourcentage ont été calculées. Les données réelles ont été prises sur le site de l'IMCCE (Miriade, 2018).

Tableau 3. Comparaison entre la position (au) et la vitesse (au/j) de Saturne et de Titan et les valeurs théoriques.

Saturne	Vecteur	Norme
Position selon les calculs (au)	1,8i-9,12j-3,85k	10,06
Vitesse selon les calculs (au/j)	0,0052i+0,00099j+0,0002k	0,0053
Position selon la théorie (au)	2,2i-9,02j-3,82k	10,04
Vitesse selon la théorie (au/j)	0,0051i+0,0012j+0,0003k	0,0053
Écart position (au)	0,4i+0,1j+0,03k	0,41
Écart vitesse (au/j)	-0,000051i+0,00021j+0,000089k	0,000234
Erreur position (%)		4,042
Erreur vitesse (%)		4,42
Titan		
Position selon les calculs (au)	1,8i-9,13j-3,84k	10,07
Vitesse selon les calculs(au/j)	0,00752i-0,001j+0,00012k	0,00761
Position selon la théorie (au)	2,2i-9,023j-3,82k	10,043
Vitesse selon la théorie (au/j)	0,00799i+0,003j-0,000083k	0,0085
Écart position (au)	0,4i+0,1j+0,03k	0,4
Écart vitesse (au/j)	0,0005i+0,004j-0,0002k	0,004
Erreur position (%)		4,125
Erreur vitesse (%)		46,612

(Miriade, 2018)

Le tableau 3 représente la valeur approximée et réelle de la position de Saturne ainsi que de Titan au 26 juillet 2048, soit un an saturnien après que la simulation ait été commencée. Il contient également les vitesses, les écarts entre les données approximées et réelles, le tout sous forme vectorielle et scalaire. Les données réelles ont été prises sur le site de l'IMCCE (Miriade, 2018).

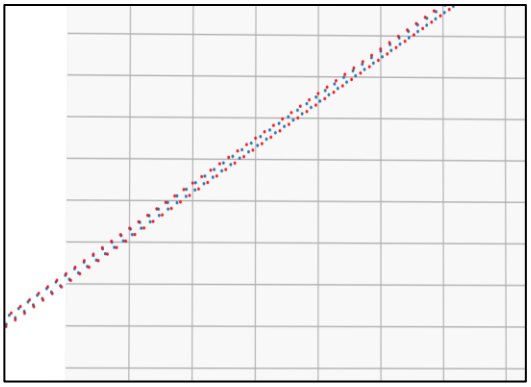

Figure 4. Le mouvement de la Terre (bleu) et de la Lune (rouge) face au Soleil.

La figure 4 montre chaque point approximé avec la méthode de Runge Kutta IV. Les points bleus représentent la position de la Terre et les points rouges représentent la position de la Lune.

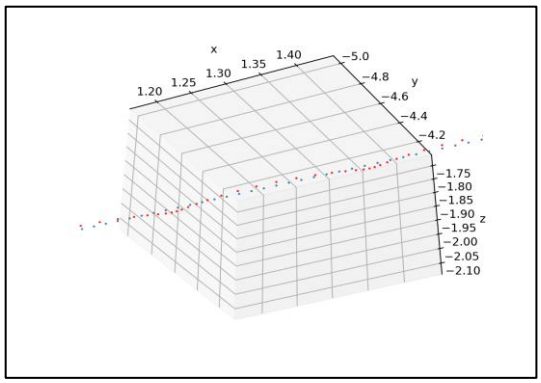

Figure 5. Le mouvement de Saturne (bleu) et de Titan (rouge) face au Soleil

La figure 5 montre chaque point approximé avec la méthode de Range Kutta IV. Les points bleus représentent la position de Saturne et les points rouges la position de Titan.

Discussion

Il fallait avant tout déterminer laquelle de ces méthodes est la plus précise. En regardant les figures 2,3, et 4, il est possible de constater que c'est la méthode de Runge Kutta IV qui est la plus précise. En effet, avec cette méthode, l'approximation de la fonction se superpose presque à la fonction réelle. Il est également possible de constater que si l'écart h est divisé de 2, on multiplie notre précision par 2, avec Euler, par 2^2 avec Runge-Kutta II et par 2^4 avec range Kutta IV. Si la méthode de Runge-Kutta prend 4 fois plus de calcul, elle est 16 fois plus précise, d'où l'avantage. L'hypothèse de départ était de déterminer s'il était possible d'approximer différents corps célestes avec la méthode de Range Kutta IV, et ce, sans dépasser 10% d'écart, soit que le rapport de la variation entre les positions calculées et théoriques sur la valeur théorique ne dépasse pas cette valeur. Cette hypothèse est confirmée. En effet, pour l'attraction le système Soleil-Terre-Lune, les écarts sont en dessous de 0,4%. Comme les données approximées sont très près de ce qu'elles sont en réalité, et ce, malgré le fait que la force gravitationnelle entre la Terre ou la Lune et d'autres planètes du système solaire a été négligée, on peut conclure que l'effet de la force gravitationnelle des planètes relativement éloignées est négligeable. C'est normal puisque le Soleil est fortement plus massif que les autres planètes du système solaire. À titre de comparaison, la planète la plus massive du système solaire, Jupiter, est environ 1047 fois moins massive que le Soleil (NSFA Working Group, 2018). Comme la masse est beaucoup plus petite, sa force gravitationnelle l'est aussi et elle devient moins influente. Saturne peut bien être massive, mais elle ne fait littéralement pas le poids face à notre étoile. Il y a également la grande distance entre la planète Terre et Jupiter ou Saturne, par exemple qui contribue à diminuer leur influence sur la Terre. Il est aussi intéressant de constater qu'en regardant la figure 4, il est possible de voir que la Terre et la Lune semblent aller dans la même direction, en se rapprochant, il est possible de constater que la Lune tourne en ligne presque droite par rapport au Soleil, la plupart des gens se seraient attendus à une figure circulaire comme dans la formule suivante;

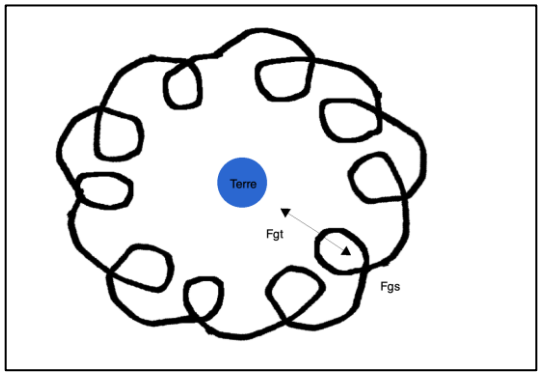

Figure 6. Conception erronée de la trajectoire de la Lune.

Cependant dans ce cas, il aurait fallu que la force gravitationnelle de la Terre (Fg_t) soit supérieure de la force gravitationnelle du Soleil (Fg_s). Ce n'est cependant le cas. Il a d'ailleurs fallu agrandir encore plus pour pouvoir distinguer la Terre de la Lune dans le plan.

Pour Saturne, si les écarts obtenus pour la position sont un peu plus grands, ils restent toutefois sous le maximum de 10%. Il y a, certes un très grand écart pour la vitesse de Titan, mais l'hypothèse concernait seulement la position. Plusieurs explications viennent à l'esprit. Tout d'abord, contrairement à la Lune, Titan n'est pas le seul satellite près de Saturne. Peut-être que Titan est influencée par les autres satellites qui l'avoisinent. Des satellites de masse similaires à Titan orbitent aussi Saturne. Aussi, il y a que Saturne varie sa vitesse beaucoup. Autant Saturne que Titan sont près d'autres grandes planètes massives telles que Jupiter ou Uranus, elles peuvent donc avoir tendance à dévier que des planètes comme la Terre, qui sont plus près du Soleil. Il est possible de constater en regardant la figure 6 que l'orbite de Saturne et de Titan que celle-ci est beaucoup attirée par Saturne que la Lune l'est envers la Terre.

Conclusion

Au terme de ce projet, il est possible d'affirmer que malgré certaines problématiques mineures qu'il est possible d'approximer la position des planètes avec la méthode de Range Kutta IV. Cependant, sa précision va varier d'une planète à l'autre et il faut parfois qu'on diminue l'unité de temps (h), et augmente le nombre d'itérations par ce fait, pour avoir une orbite acceptable. C'est le cas lorsque la vitesse des planètes varie beaucoup. Cependant, augmenter le nombre d'itérations à son prix, celui de la performance du programme. En effet, plus il y a d'itérations, plus il y a de calculs, et plus le programme prend du temps à exécuter. Aussi, plus il y a d'itérations, plus il y a de points et ainsi moins la navigation sur le graphique 3D est fluide. Si quelqu'un souhaitait, par exemple, modéliser la trajectoire des planètes du système solaire à des fins scientifiques, il pourrait en théorie le faire puisque les données seraient fiables, mais il devrait avoir en sa possession un ordinateur capable de très bonnes performances.

<u>Suggestions et perspectives d'avenir</u> :

L'univers est vaste et complexe et même en ayant testé la méthode de Runge Kutta IV pour deux planètes différentes par plusieurs caractéristiques, cela ne peut pas être considéré comme un résultat général pour l'ensemble des objets célestes. Une suggestion en ce sens

serait de refaire la même expérience, mais avec des planètes dans des contextes différents. Il est possible de penser à la planète naine Pluton, par exemple, qui est moins massive que Saturne, mais beaucoup plus éloignée de la Terre, ou avec mercure, qui est près de notre étoile. Il faut également penser qu'avec les avancées technologiques, notamment avec les ordinateurs quantiques, il serait possible d'avoir dans quelques dizaines d'années des ordinateurs beaucoup plus performants. Il serait alors possible de faire un projet similaire, mais avec beaucoup plus de précision.

Remerciements

La réalisation de ce projet n'aurait pas été possible sans mon tuteur, Monsieur Jean-Norbert Fournier, qui m'a apporté du soutien académique notamment pour ce qui a trait aux méthodes d'approximations utilisées, et qui m'a offert de l'assistance technique tout au long du projet. La rédaction de cet article n'aura pas été possible sans l'aide de Madame Chantale Leblond, qui m'a assisté dans la rédaction de l'article. J'aimerais enfin remercier la fondation Python, qui est celle qui a créé le langage à code ouvert utilisé dans la majeure partie de ce projet et utilisé pour dessiner les graphiques en 3D.

Bibliographie

Couronne, I. (2018,3 Décembre). *Une sonde de la NASA à la conquête de l'astéroïde Bennu*. Récupéré sur LeDevoir.com: https://www.ledevoir.com/societe/science/542775/une-sonde-de-la-nasa-a-la-conquete-de-l-asteroide-bennu

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science \& Engineering, 9*(3), 90-95.

Kreyszig, E. (1993). *Advanced Engineering Mathematics Seventh Edition.* Columbus, Ohio, États-Unis d'Amérique: John Wiley & Sons.

Large Synaptic Survey Telescope. (2018, Septembre 3). *Potentially Hazardous Asteroids (PHAs)*. Récupéré sur LSST: https://www.lsst.org/science/solar-system/potentially-hazardous-asteroids

Miriade. (2018,29 Novembre). Récupéré sur IMCEE: http://vo.imcce.fr/webservices/miriade/?forms

NSFA Working Group. (2018). *Numerical Standards for Fundamental Astronomy*. Récupéré sur USNO-Navy.

Séguin, M. (s.d.). *Mécanique.* Montréal: ERPI.

Universalis. (2018, 10 13). *Mesure de la distance Terre-Soleil*. Récupéré sur Universalis: https://www.universalis.fr/encyclopedie/mesure-de-la-distance-Terre-Soleil/

N.B : Ce texte a été rédigé en utilisant l'orthographe rectifiée.

Annexe I : le code python pour la Terre et la Lune

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
axe = fig.add_subplot(1,1,1, projection='3d')



#Orbite de la Terre et de la Lune en 3D

masseSoleil = 1.989*10**30
masseLune = 7.346*10**22
masseTerre = 5.9723*10**24
G = 1.488*10**(-34)

#Les fonctions suivantes déterminent la position en x,y et z de la Terre (t)

def ft(xT,yT,zT,xL,yL,zL):
   return -(G*masseSoleil*xT/(xT**2+yT**2+zT**2)**(3/2))-(G*masseLune*(xT-xL)/((xT-xL)**2+(yT-yL)**2+(zT-
zL)**2)**(3/2))

def fl(xT,yT,zT,xL,yL,zL):
    return -(G*masseSoleil*xL/(xL**2+yL**2+zL**2)**(3/2))-(G*masseTerre*(xL-xT)/((xT-xL)**2+(yT-yL)**2+(zT-
zL)**2)**(3/2))

def gt(xT,yT,zT,xL,yL,zL):
    return -(G*masseSoleil*yT/(xT**2+yT**2+zT**2)**(3/2))-(G*masseLune*(yT-yL)/((xT-xL)**2+(yT-yL)**2+(zT-
zL)**2)**(3/2))

def gl(xT,yT,zT,xL,yL,zL):
   return -(G*masseSoleil*yL/(xL**2+yL**2+zL**2)**(3/2))-(G*masseTerre*(yL-yT)/((xT-xL)**2+(yT-yL)**2+(zT-
zL)**2)**(3/2))

def mt(xT,yT,zT,xL,yL,zL):
   return -(G*masseSoleil*zT/(xT**2+yT**2+zT**2)**(3/2))-(G*masseLune*(zT-zL)/((xT-xL)**2+(yT-yL)**2+(zT-
zL)**2)**(3/2))

def ml(xT,yT,zT,xL,yL,zL):
   return -(G*masseSoleil*zL/(xL**2+yL**2+zL**2)**(3/2))-(G*masseTerre*(zL-zT)/((xT-xL)**2+(yT-yL)**2+(zT-
zL)**2)**(3/2))

#Données en date du 30 octobre 2018
xT = [0] * 366 #fait une liste de longueur 366 avec des zéros partout
yT= [0] * 366
zT = [0] * 366
uxT = [0] * 366
uyT = [0] * 366
uzT = [0] * 366
xL = [0] * 366
yL = [0] * 366
zL = [0] * 366
uxL = [0] * 366
uyL = [0] * 366
uzL = [0] * 366
t = [0] * 366
#xT.insert(0,...)
xT.insert(0,0.5331348236265)
yT.insert(0,0.7633433095910)
```

```
zT.insert(0,0.3309099000574)
uxT.insert(0,-0.0147634662835)
uyT.insert(0,0.0084512752918)
uzT.insert(0,0.0036637693449)
xL.insert(0,0.5356091468900)
yL.insert(0,0.7641156323294)
zL.insert(0,0.3309998421661)
uxL.insert(0,-0.0149654058774)
uyL.insert(0,0.0089588863396)
uzL.insert(0,0.0038754910853)
N = 3650
h = 0.1
t.insert(0,0)
#faire un liste avec un range prédéfini ou mettre des 0 partout dans la définition de la liste
for i in range(0,N):
   #------------Range Kutta I-----------------------------------------------
           t.insert(i+1,t[i]+h)
           kxT1 = h*uxT[i] #dit division par zéro, mais c'est parce qu'on a pas encore fini...
           kxL1 = h*uxL[i]
           kyT1 = h*uyT[i]
           kyL1 = h*uyL[i]
           kzT1 = h*uzT[i]
           kzL1 = h*uzL[i]
           kuxT1 = h*ft(xT[i], yT[i], zT[i], xL[i], yL[i], zL[i])
           kuxL1 = h*fl(xT[i], yT[i], zT[i], xL[i], yL[i], zL[i])
           kuyT1 = h*gt(xT[i], yT[i], zT[i], xL[i], yL[i], zL[i])
           kuyL1 = h*gl(xT[i], yT[i], zT[i], xL[i], yL[i], zL[i])
           kuzT1 = h*mt(xT[i], yT[i], zT[i], xL[i], yL[i], zL[i])
           kuzL1 = h*ml(xT[i], yT[i], zT[i], xL[i], yL[i], zL[i])

#--------------Range Kutta II----------------------------------------
           kxT2 = h*(uxT[i]+(1/2)*kuxT1)
           kxL2 = h*(uxL[i]+(1/2)*kuxL1)
           kyT2 = h*(uyT[i]+(1/2)*kuyT1)
           kyL2 = h*(uyL[i]+(1/2)*kuyL1)
           kzT2 = h*(uzT[i]+(1/2)*kuzT1)
           kzL2 = h*(uzL[i]+(1/2)*kuzL1)
           kuxT2 = h*ft(xT[i]+(1/2)*kxT1, yT[i]+(1/2)*kyT1, zT[i]+(1/2)*kzT1, xL[i]+(1/2)*kxL1,
yL[i]+(1/2)*kyL1, zL[i]+(1/2)*kzL1)
           kuxL2 = h*fl(xT[i]+(1/2)*kxT1, yT[i]+(1/2)*kyT1, zT[i]+(1/2)*kzT1, xL[i]+(1/2)*kxL1,
yL[i]+(1/2)*kyL1, zL[i]+(1/2)*kzL1)
           kuyT2 = h*gt(xT[i]+(1/2)*kxT1, yT[i]+(1/2)*kyT1, zT[i]+(1/2)*kzT1, xL[i]+(1/2)*kxL1,
yL[i]+(1/2)*kyL1, zL[i]+(1/2)*kzL1)
           kuyL2 = h*gl(xT[i]+(1/2)*kxT1, yT[i]+(1/2)*kyT1, zT[i]+(1/2)*kzT1, xL[i]+(1/2)*kxL1,
yL[i]+(1/2)*kyL1, zL[i]+(1/2)*kzL1)
           kuzT2 = h*mt(xT[i]+(1/2)*kxT1, yT[i]+(1/2)*kyT1, zT[i]+(1/2)*kzT1, xL[i]+(1/2)*kxL1,
yL[i]+(1/2)*kyL1, zL[i]+(1/2)*kzL1)
           kuzL2 = h*ml(xT[i]+(1/2)*kxT1, yT[i]+(1/2)*kyT1, zT[i]+(1/2)*kzT1, xL[i]+(1/2)*kxL1,
yL[i]+(1/2)*kyL1, zL[i]+(1/2)*kzL1)

#--------------Range Kutta III-------------------------------------------------------------------------------
------------------------
           kxT3 = h*(uxT[i]+(1/2)*kuxT2)
           kxL3 = h*(uxL[i]+(1/2)*kuxL2)
           kyT3 = h*(uyT[i]+(1/2)*kuyT2)
           kyL3 = h*(uyL[i]+(1/2)*kuyL2)
           kzT3 = h*(uzT[i]+(1/2)*kuzT2)
           kzL3 = h*(uzL[i]+(1/2)*kuzL2)
           kuxT3 = h*ft(xT[i]+(1/2)*kxT2, yT[i]+(1/2)*kyT2, zT[i]+(1/2)*kzT2, xL[i]+(1/2)*kxL2,
yL[i]+(1/2)*kyL2, zL[i]+(1/2)*kzL2)
           kuxL3 = h*fl(xT[i]+(1/2)*kxT2, yT[i]+(1/2)*kyT2, zT[i]+(1/2)*kzT2, xL[i]+(1/2)*kxL2,
yL[i]+(1/2)*kyL2, zL[i]+(1/2)*kzL2)
```

```
            kuyT3 = h*gt(xT[i]+(1/2)*kxT2, yT[i]+(1/2)*kyT2, zT[i]+(1/2)*kzT2, xL[i]+(1/2)*kxL2,
yL[i]+(1/2)*kyL2, zL[i]+(1/2)*kzL2)
            kuyL3 = h*gl(xT[i]+(1/2)*kxT2, yT[i]+(1/2)*kyT2, zT[i]+(1/2)*kzT2, xL[i]+(1/2)*kxL2,
yL[i]+(1/2)*kyL2, zL[i]+(1/2)*kzL2)
            kuzT3 = h*mt(xT[i]+(1/2)*kxT2, yT[i]+(1/2)*kyT2, zT[i]+(1/2)*kzT2, xL[i]+(1/2)*kxL2,
yL[i]+(1/2)*kyL2, zL[i]+(1/2)*kzL2)
            kuzL3 = h*ml(xT[i]+(1/2)*kxT2, yT[i]+(1/2)*kyT2, zT[i]+(1/2)*kzT2, xL[i]+(1/2)*kxL2,
yL[i]+(1/2)*kyL2, zL[i]+(1/2)*kzL2)
#----------------------Range Kutta IV-------------------------------------------------------------------------
------------------------
            kxT4 = h*(uxT[i]+kuxT3)
            kxL4 = h*(uxL[i]+kuxL3)
            kyT4 = h*(uyT[i]+kuyT3)
            kyL4 = h*(uyL[i]+kuyL3)
            kzT4 = h*(uzT[i]+kuzT3)
            kzL4 = h*(uzL[i]+kuzL3)
            kuxT4 = h*ft(xT[i]+kxT3, yT[i]+kyT3, zT[i]+kzT3, xL[i]+kxL3, yL[i]+kyL3, zL[i]+kzL3)
            kuxL4 = h*fl(xT[i]+kxT3, yT[i]+kyT3, zT[i]+kzT3, xL[i]+kxL3, yL[i]+kyL3, zL[i]+kzL3)
            kuyT4 = h*gt(xT[i]+kxT3, yT[i]+kyT3, zT[i]+kzT3, xL[i]+kxL3, yL[i]+kyL3, zL[i]+kzL3)
            kuyL4 = h*gl(xT[i]+kxT3, yT[i]+kyT3, zT[i]+kzT3, xL[i]+kxL3, yL[i]+kyL3, zL[i]+kzL3)
            kuzT4 = h*mt(xT[i]+kxT3, yT[i]+kyT3, zT[i]+kzT3, xL[i]+kxL3, yL[i]+kyL3, zL[i]+kzL3)
            kuzL4 = h*ml(xT[i]+kxT3, yT[i]+kyT3, zT[i]+kzT3, xL[i]+kxL3, yL[i]+kyL3, zL[i]+kzL3)
#----------------------Mise à jour des valeurs-----------------------------------------------------------------
----------------
            xT.insert(i+1,xT[i]+(1/6)*(kxT1+2*kxT2+2*kxT3+kxT4))
            xL.insert(i+1,xL[i]+(1/6)*(kxL1+2*kxL2+2*kxL3+kxL4))
            yT.insert(i+1,yT[i]+(1/6)*(kyT1+2*kyT2+2*kyT3+kyT4))
            yL.insert(i+1,yL[i]+(1/6)*(kyL1+2*kyL2+2*kyL3+kyL4))
            zT.insert(i+1,zT[i]+(1/6)*(kzT1+2*kzT2+2*kzT3+kzT4))
            zL.insert(i+1,zL[i]+(1/6)*(kzL1+2*kzL2+2*kzL3+kzL4))

            uxT.insert(i+1,uxT[i]+(1/6)*(kuxT1+2*kuxT2+2*kuxT3+kuxT4))
            uxL.insert(i+1,uxL[i]+(1/6)*(kuxL1+2*kuxL2+2*kuxL3+kuxL4))
            uyT.insert(i+1,uyT[i]+(1/6)*(kuyT1+2*kuyT2+2*kuyT3+kuyT4))
            uyL.insert(i+1,uyL[i]+(1/6)*(kuyL1+2*kuyL2+2*kuyL3+kuyL4)) #l'erreur ici
            uzT.insert(i+1,uzT[i]+(1/6)*(kuzT1+2*kuzT2+2*kuzT3+kuzT4))
            uzL.insert(i+1,uzL[i]+(1/6)*(kuzL1+2*kuzL2+2*kuzL3+kuzL4))


            #voir à 250 pour tout les x




print("xT:",xT[365],"yT:",yT[365],"zT:",zT[365],"uxT:",uxT[365],"uyT:",uyT[365],"uzT:",uzT[365])

print("xL:",xL[365],"yL:",yL[365],"zL:",zL[365],"uxL:",uxL[365],"uyL:",uyL[365],"uzL:",uzL[365])



axe.scatter(xT,yT,zT,s = 1,c = None)

axe.scatter(xL,yL,zL,s=1,c="r")




axe.set_xlabel('x')
```

```
axe.set_ylabel('y')
axe.set_zlabel('z')

plt.show()
```

Annexe II : le code pour Saturne et Titan

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
axe = fig.add_subplot(1,1,1, projection='3d')



#Orbite de la Saturne et de la Titan en 3D

masseSoleil = 1.989*10**30
masseTitan = 1.3455*10**23
masseSaturne = 5.6834*10**26
G = 1.488*10**(-34)

#Les fonctions suivantes déterminent la position en x,y et z de la Saturne (t)

def ft(xS,yS,zS,xT,yT,zT):
   return -(G*masseSoleil*xS/(xS**2+yS**2+zS**2)**(3/2))-(G*masseTitan*(xS-xT)/((xS-xT)**2+(yS-
yT)**2+(zS-zT)**2)**(3/2))

def fl(xS,yS,zS,xT,yT,zT):
    return -(G*masseSoleil*xT/(xT**2+yT**2+zT**2)**(3/2))-(G*masseSaturne*(xT-xS)/((xS-
xT)**2+(yS-yT)**2+(zS-zT)**2)**(3/2))

def gt(xS,yS,zS,xT,yT,zT):
    return -(G*masseSoleil*yS/(xS**2+yS**2+zS**2)**(3/2))-(G*masseTitan*(yS-yT)/((xS-xT)**2+(yS-
yT)**2+(zS-zT)**2)**(3/2))

def gl(xS,yS,zS,xT,yT,zT):
   return -(G*masseSoleil*yT/(xT**2+yT**2+zT**2)**(3/2))-(G*masseSaturne*(yT-yS)/((xS-
xT)**2+(yS-yT)**2+(zS-zT)**2)**(3/2))

def mt(xS,yS,zS,xT,yT,zT):
   return -(G*masseSoleil*zS/(xS**2+yS**2+zS**2)**(3/2))-(G*masseTitan*(zS-zT)/((xS-xT)**2+(yS-
yT)**2+(zS-zT)**2)**(3/2))

def ml(xS,yS,zS,xT,yT,zT):
```

```
   return -(G*masseSoleil*zT/(xT**2+yT**2+zT**2)**(3/2))-(G*masseSaturne*(zT-zS)/((xS-
xT)**2+(yS-yT)**2+(zS-zT)**2)**(3/2))

#Données en date du 29 nov 2018
N = 160
xS = [0] * N #fait une liste de longueur 366 avec des zéros partout
yS= [0] * N
zS = [0] * N
uxS = [0] * N
uyS = [0] * N
uzS = [0] * N
xT = [0] * N
yT = [0] * N
zT = [0] * N
uxT = [0] * N
uyT = [0] * N
uzT = [0] * N
t = [0] * N
#xS.insert(0,...)
xS.insert(0,1.7885918672527)
yS.insert(0,-9.1239278546549)
zS.insert(0,-3.8457231721355)
uxS.insert(0,0.0051903405303)
uyS.insert(0,0.0009809505764)
uzS.insert(0,0.0001817597102)
xT.insert(0,1.7802702126894)
yT.insert(0,-9.1246124827660)
zT.insert(0,-3.8449405622395)
uxT.insert(0,0.0054961854928)
uyT.insert(0,-0.0021321671600)
uzT.insert(0,0.0003630774268)
h = 1
t.insert(0,0)
#faire un liste avec un range prédéfini ou mettre des 0 partout dans la définition de la liste
for i in range(0,N):
   #------------Range Kutta I-----------------------------------------------
           t.insert(i+1,t[i]+h)
           kxS1 = h*uxS[i] #dit division par zéro, mais c'est parce qu'on a pas encore fini...
           kxT1 = h*uxT[i]
           kyS1 = h*uyS[i]
           kyT1 = h*uyT[i]
           kzS1 = h*uzS[i]
           kzT1 = h*uzT[i]
           kuxS1 = h*ft(xS[i], yS[i], zS[i], xT[i], yT[i], zT[i])
           kuxT1 = h*fl(xS[i], yS[i], zS[i], xT[i], yT[i], zT[i])
           kuyS1 = h*gt(xS[i], yS[i], zS[i], xT[i], yT[i], zT[i])
           kuyT1 = h*gl(xS[i], yS[i], zS[i], xT[i], yT[i], zT[i])
           kuzS1 = h*mt(xS[i], yS[i], zS[i], xT[i], yT[i], zT[i])
           kuzT1 = h*ml(xS[i], yS[i], zS[i], xT[i], yT[i], zT[i])

#--------------Range Kutta II----------------------------------------
           kxS2 = h*(uxS[i]+(1/2)*kuxS1)
           kxT2 = h*(uxT[i]+(1/2)*kuxT1)
           kyS2 = h*(uyS[i]+(1/2)*kuyS1)
           kyT2 = h*(uyT[i]+(1/2)*kuyT1)
```

```
            kzS2 = h*(uzS[i]+(1/2)*kuzS1)
            kzT2 = h*(uzT[i]+(1/2)*kuzT1)
            kuxS2 = h*ft(xS[i]+(1/2)*kxS1, yS[i]+(1/2)*kyS1, zS[i]+(1/2)*kzS1, xT[i]+(1/2)*kxT1,
yT[i]+(1/2)*kyT1, zT[i]+(1/2)*kzT1)
            kuxT2 = h*fl(xS[i]+(1/2)*kxS1, yS[i]+(1/2)*kyS1, zS[i]+(1/2)*kzS1, xT[i]+(1/2)*kxT1,
yT[i]+(1/2)*kyT1, zT[i]+(1/2)*kzT1)
            kuyS2 = h*gt(xS[i]+(1/2)*kxS1, yS[i]+(1/2)*kyS1, zS[i]+(1/2)*kzS1, xT[i]+(1/2)*kxT1,
yT[i]+(1/2)*kyT1, zT[i]+(1/2)*kzT1)
            kuyT2 = h*gl(xS[i]+(1/2)*kxS1, yS[i]+(1/2)*kyS1, zS[i]+(1/2)*kzS1, xT[i]+(1/2)*kxT1,
yT[i]+(1/2)*kyT1, zT[i]+(1/2)*kzT1)
            kuzS2 = h*mt(xS[i]+(1/2)*kxS1, yS[i]+(1/2)*kyS1, zS[i]+(1/2)*kzS1, xT[i]+(1/2)*kxT1,
yT[i]+(1/2)*kyT1, zT[i]+(1/2)*kzT1)
            kuzT2 = h*ml(xS[i]+(1/2)*kxS1, yS[i]+(1/2)*kyS1, zS[i]+(1/2)*kzS1, xT[i]+(1/2)*kxT1,
yT[i]+(1/2)*kyT1, zT[i]+(1/2)*kzT1)

#-------------Range Kutta III-------------------------------------------------------------------
-------------------------------------
            kxS3 = h*(uxS[i]+(1/2)*kuxS2)
            kxT3 = h*(uxT[i]+(1/2)*kuxT2)
            kyS3 = h*(uyS[i]+(1/2)*kuyS2)
            kyT3 = h*(uyT[i]+(1/2)*kuyT2)
            kzS3 = h*(uzS[i]+(1/2)*kuzS2)
            kzT3 = h*(uzT[i]+(1/2)*kuzT2)
            kuxS3 = h*ft(xS[i]+(1/2)*kxS2, yS[i]+(1/2)*kyS2, zS[i]+(1/2)*kzS2, xT[i]+(1/2)*kxT2,
yT[i]+(1/2)*kyT2, zT[i]+(1/2)*kzT2)
            kuxT3 = h*fl(xS[i]+(1/2)*kxS2, yS[i]+(1/2)*kyS2, zS[i]+(1/2)*kzS2, xT[i]+(1/2)*kxT2,
yT[i]+(1/2)*kyT2, zT[i]+(1/2)*kzT2)
            kuyS3 = h*gt(xS[i]+(1/2)*kxS2, yS[i]+(1/2)*kyS2, zS[i]+(1/2)*kzS2, xT[i]+(1/2)*kxT2,
yT[i]+(1/2)*kyT2, zT[i]+(1/2)*kzT2)
            kuyT3 = h*gl(xS[i]+(1/2)*kxS2, yS[i]+(1/2)*kyS2, zS[i]+(1/2)*kzS2, xT[i]+(1/2)*kxT2,
yT[i]+(1/2)*kyT2, zT[i]+(1/2)*kzT2)
            kuzS3 = h*mt(xS[i]+(1/2)*kxS2, yS[i]+(1/2)*kyS2, zS[i]+(1/2)*kzS2, xT[i]+(1/2)*kxT2,
yT[i]+(1/2)*kyT2, zT[i]+(1/2)*kzT2)
            kuzT3 = h*ml(xS[i]+(1/2)*kxS2, yS[i]+(1/2)*kyS2, zS[i]+(1/2)*kzS2, xT[i]+(1/2)*kxT2,
yT[i]+(1/2)*kyT2, zT[i]+(1/2)*kzT2)
#---------------------Range Kutta IV-------------------------------------------------------------
-------------------------------------
            kxS4 = h*(uxS[i]+kuxS3)
            kxT4 = h*(uxT[i]+kuxT3)
            kyS4 = h*(uyS[i]+kuyS3)
            kyT4 = h*(uyT[i]+kuyT3)
            kzS4 = h*(uzS[i]+kuzS3)
            kzT4 = h*(uzT[i]+kuzT3)
            kuxS4 = h*ft(xS[i]+kxS3, yS[i]+kyS3, zS[i]+kzS3, xT[i]+kxT3, yT[i]+kyT3, zT[i]+kzT3)
            kuxT4 = h*fl(xS[i]+kxS3, yS[i]+kyS3, zS[i]+kzS3, xT[i]+kxT3, yT[i]+kyT3, zT[i]+kzT3)
            kuyS4 = h*gt(xS[i]+kxS3, yS[i]+kyS3, zS[i]+kzS3, xT[i]+kxT3, yT[i]+kyT3, zT[i]+kzT3)
            kuyT4 = h*gl(xS[i]+kxS3, yS[i]+kyS3, zS[i]+kzS3, xT[i]+kxT3, yT[i]+kyT3, zT[i]+kzT3)
            kuzS4 = h*mt(xS[i]+kxS3, yS[i]+kyS3, zS[i]+kzS3, xT[i]+kxT3, yT[i]+kyT3, zT[i]+kzT3)
            kuzT4 = h*ml(xS[i]+kxS3, yS[i]+kyS3, zS[i]+kzS3, xT[i]+kxT3, yT[i]+kyT3, zT[i]+kzT3)
#---------------------Mise à jour des valeurs-----------------------------------------------------
---------------------------
            xS.insert(i+1,xS[i]+(1/6)*(kxS1+2*kxS2+2*kxS3+kxS4))
            xT.insert(i+1,xT[i]+(1/6)*(kxT1+2*kxT2+2*kxT3+kxT4))
            yS.insert(i+1,yS[i]+(1/6)*(kyS1+2*kyS2+2*kyS3+kyS4))
            yT.insert(i+1,yT[i]+(1/6)*(kyT1+2*kyT2+2*kyT3+kyT4))
```

```
            zS.insert(i+1,zS[i]+(1/6)*(kzS1+2*kzS2+2*kzS3+kzS4))
            zT.insert(i+1,zT[i]+(1/6)*(kzT1+2*kzT2+2*kzT3+kzT4))

            uxS.insert(i+1,uxS[i]+(1/6)*(kuxS1+2*kuxS2+2*kuxS3+kuxS4))
            uxT.insert(i+1,uxT[i]+(1/6)*(kuxT1+2*kuxT2+2*kuxT3+kuxT4))
            uyS.insert(i+1,uyS[i]+(1/6)*(kuyS1+2*kuyS2+2*kuyS3+kuyS4))
            uyT.insert(i+1,uyT[i]+(1/6)*(kuyT1+2*kuyT2+2*kuyT3+kuyT4)) #l'erreur ici
            uzS.insert(i+1,uzS[i]+(1/6)*(kuzS1+2*kuzS2+2*kuzS3+kuzS4))
            uzT.insert(i+1,uzT[i]+(1/6)*(kuzT1+2*kuzT2+2*kuzT3+kuzT4))


            #voir à 250 pour tout les x




#print("xS:",xS[1083200],"yS:",yS[1083200],"zS:",zS[1083200],"uxS:",uxS[1083200],"uyS:",uyS[1083
200],"uzS:",uzS[1083200])

#print("xT:",xT[1083200],"yT:",yT[1083200],"zT:",zT[1083200],"uxT:",uxT[1083200],"uyT:",uyT[1083
200],"uzT:",uzT[1083200])



axe.scatter(xS,yS,zS,s = 1,c = None)

axe.scatter(xT,yT,zT,s=1,c="r")




axe.set_xlabel('x')
axe.set_ylabel('y')
axe.set_zlabel('z')

plt.show()
```