



Compte-rendu TP Gravitation

Algorithmique Mathématique 5 et Python :
Analyse Numérique des Systèmes Dynamiques

Lore Mounho et Louise Vannetzel
Étudiantes en Licence 3 - Mathématiques

Enseignant référent - Mr Poncet

Année Universitaire 2023-2024

Sommaire

- I. Introduction
- II. Partie 1 : Formalisme hamiltonien d'une équation non-linéaire d'ordre 2
- III. Partie 2 : Schémas numériques
- IV. Partie 3 : Application, système gravitant en 2D
- V. Partie IV : Système gravitant 3D
- VI. Conclusion

Introduction

L'objectif de ce TP est de réaliser des expérimentations numériques à partir de systèmes dynamiques de la forme : $y'(t) = f(y,t)$. On souhaite simuler la trajectoire d'un satellite galiléen autour de Jupiter en utilisant différents schémas numériques et en comparant leur comportement. Pour cela, on utilise des données fournies par l'IMCCE sur la position et la vitesse à un temps donné des satellites Io, Europe, Ganymède et Callisto. On étudie ce problème de gravitation en dimension 2 et en dimension 3 et on effectue des calculs numériques de la période orbitale des satellites et des erreurs relatives à ceux-ci .

PARTIE I :

3) on sait que l'équa-diff (*) s'écrit : $\begin{cases} q'(t) = p(t) \\ p'(t) = \frac{-q(t)}{\|q(t)\|_2^3} \end{cases}$

on veut montrer qu'elle s'écrit : $\begin{cases} q'(t) = p(t) \\ p'(t) = \nabla V(q(t)) \end{cases}$

$\nabla V(q) = \nabla \frac{1}{\|q\|_2}$ avec $\frac{1}{\|q\|_2} = \frac{1}{\sqrt{q_1^2 + q_2^2 + q_3^2}} = (q_1^2 + q_2^2 + q_3^2)^{-\frac{1}{2}}$ = $g \circ f(q(t))$ avec :

$f: q \mapsto \|q\|^2$ et $g: x \mapsto x^{-\frac{1}{2}}$; $f': q \mapsto 2\|q\|$ et $g': x \mapsto -\frac{1}{2}x^{-\frac{3}{2}}$

$$\begin{aligned} \nabla V(q) &= g'(f(q)) \times f'(q) = g'(\|q\|^2) \times 2\|q\|_2 \\ &= -\frac{1}{2}\|q\|^{-\frac{3}{2}} \times 2\|q\| = -\|q(t)\|^{-3} \times \|q(t)\| \\ &= \begin{vmatrix} -q_1 \\ \hline -q_2 \\ \hline 0 \\ \hline 0 \end{vmatrix} = \frac{-q(t)}{\|q(t)\|_2^3} \end{aligned}$$

Donc $p'(t) = \nabla V(q(t))$, $\forall t \geq 0$.

2) En déduire que $H(q, p) = \frac{1}{2} \|p\|_2^2 - V(q)$ est le hamiltonien du système ci-dessus.
on doit montrer que :

$$x'(t) = J \nabla H(x(t)) = \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix} \nabla H(x(t)) \Leftrightarrow \nabla H(x(t)) = J^{-1} x'(t)$$

$$\nabla H(x(t)) = \nabla \left(\frac{1}{2} \|p\|_2^2 - V(q) \right) = \begin{vmatrix} 0 \\ \nabla_q \frac{1}{2} \|p\|_2^2 - \nabla_q V(q) \\ \nabla_p \frac{1}{2} \|p\|_2^2 - \nabla_p V(q) \\ 0 \end{vmatrix} = \begin{vmatrix} -\nabla_q V(q) \\ p \\ \frac{1}{2} \times 2p \end{vmatrix}$$

$$J^{-1} x'(t) = \begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix} \begin{pmatrix} p \\ \nabla V(q) \end{pmatrix} = \begin{vmatrix} -\nabla V(q) \\ p \end{vmatrix}$$

D'où : $x'(t) = J \nabla H(x(t))$ soit $H(q, p)$ est le hamiltonien du système.

3) Justifier que $H(q(t), p(t))$ est constant. Pour cela on montre que $\frac{dH(x(t))}{dt} = 0$.

$$\begin{aligned} \frac{dH(x(t))}{dt} &= \langle \nabla H(x(t)), x'(t) \rangle = \langle \nabla H(x(t)), J \nabla H(x(t)) \rangle = \alpha \\ &= \langle J \nabla H(x(t)), \nabla H(x(t)) \rangle \text{ par symétrie} \\ &= \langle \nabla H(x(t)), J^T \nabla H(x(t)) \rangle \\ &= \langle \nabla H(x(t)), -J \nabla H(x(t)) \rangle \text{ car } J^T = -J \\ &= -\langle \nabla H(x(t)), J \nabla H(x(t)) \rangle \text{ car le produit scalaire est bilinéaire} \\ &= -\alpha \end{aligned}$$

on a donc $\alpha = -\alpha \Leftrightarrow \alpha = 0 \Leftrightarrow \frac{dH(x(t))}{dt} = 0$

Donc $H(x(t))$ est constant et vaut toujours $H(x(0))$.

PARTIE II :

(F)

4) Montrez que la solution analytique $q(t)$ dans \mathbb{R}^2 de l'équation :

$$\text{(**)} \quad \begin{cases} q''(t) = \frac{-q(t)}{\|q(t)\|^3} & \forall t \geq 0 \\ q(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ et } q'(0) = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{cases} \Leftrightarrow \begin{cases} q'(t) = p(t) \\ p'(t) = \frac{-q(t)}{\|q(t)\|^3} \end{cases} \quad \forall t \geq 0$$

est $q(t) = (\cos(t), \sin(t))$.

Si $q(t) = (\cos(t), \sin(t))$ est la solution alors elle vérifie (**):

$$\begin{aligned} q'(t) &= (-\sin(t), \cos(t)) \\ q''(t) &= (-\cos(t), -\sin(t)) \end{aligned}$$

$$\text{Or } \frac{-q(t)}{\|q(t)\|^3} = \frac{(-\cos(t), -\sin(t))}{\|q(t)\|^3} \quad \text{car } \|q(t)\|_2^2 = (\cos t)^2 + (\sin t)^2 = 1$$

$$\text{donc } \|q(t)\|_2^3 = \|q(t)\|_2 = \sqrt{(\cos t)^2 + (\sin t)^2} = 1$$

$$= (-\cos(t), -\sin(t))$$

$$\text{D'où } q''(t) = \frac{-q(t)}{\|q(t)\|^3}. \quad \text{De plus } q(0) = (\cos(0), \sin(0)) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ et } q'(0) = (-\sin(0), \cos(0)) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$\Rightarrow q_{\text{ex}}(t) = (\cos(t), \sin(t))$$

II.2. On cherche à approcher la solution de l'équation (**) en utilisant les schémas (EE), (PM) et (RK4). Pour cela, on prend pour chaque schéma numérique deux valeurs de N différentes pour montrer que l'on obtient l'ordre théorique de la méthode.

Pour le schéma PM, on choisit les valeurs de N suivantes : N=12000 puis N=24000.

Pour EE, on prend N=48000 puis N=96000. Enfin pour RK4, on choisit N=1500 puis N=3000.

On utilise comme valeur d'erreur, la valeur finale : $q_{\text{ex}}(4\pi) = (1, 0)^T$. L'exécution du programme **P2qs2.py**, nous permet d'obtenir les erreurs situées dans le tableau ci-dessous.

Méthode	St	Err	St*	Err*	Err/Err *	St/St *	Pnum
PM	$\frac{4\pi}{12\ 000}$	$-8.0556 e^{-6}$	$\frac{4\pi}{24\ 000}$	$-2.0118 e^{-6}$	4.0042	2	≈ 2
EE	$\frac{4\pi}{48\ 000}$	-0.0618	$\frac{4\pi}{96\ 000}$	-0.0309	≈ 1.996	2	≈ 1
RK4	$\frac{4\pi}{1\ 500}$	$1.554 e^{-9}$	$\frac{4\pi}{3\ 000}$	$9.2929 e^{-11}$	≈ 16.724	2	≈ 4

Détail des calculs pour $Pnum = \log(\frac{Err}{Err^*}) / \log(\frac{St}{St^*})$

- PM : $\log(4.0042) / \log(2) \approx 2$
- EE : $\log(1.996) / \log(2) \approx 1$
- RK4 : $\log(16.724) / \log(2) \approx 4$

Le 'Pnum' obtenu après calculs est bien égal à l'ordre théorique correspondant à chacun de ces schémas.

II.3. On choisit un $N=15\ 000$ tel que RK4 montre une erreur de l'ordre de 10^{-14} . En effet, pour ce N on a : $Err \approx 1.9056e^{-14}$. On souhaite calculer, pour chacun de ces schémas, l'évolution de $H(qn, pn)$. On sait d'après la question I.3 que $H(qex, pex)$ est constant, c'est-à-dire ne dépend pas du temps.

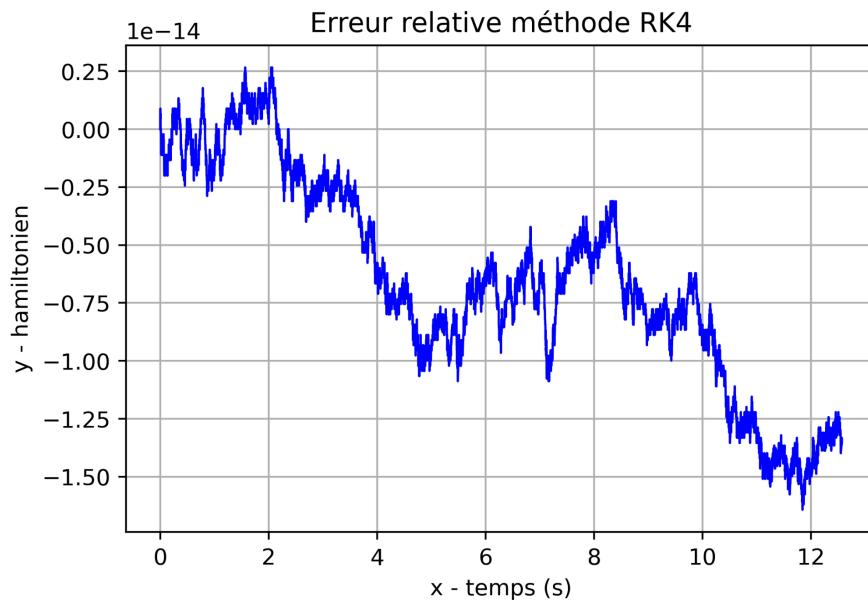
En effet, en se servant de la formule de H énoncée à la question 2 de la partie I et des conditions initiales on a :

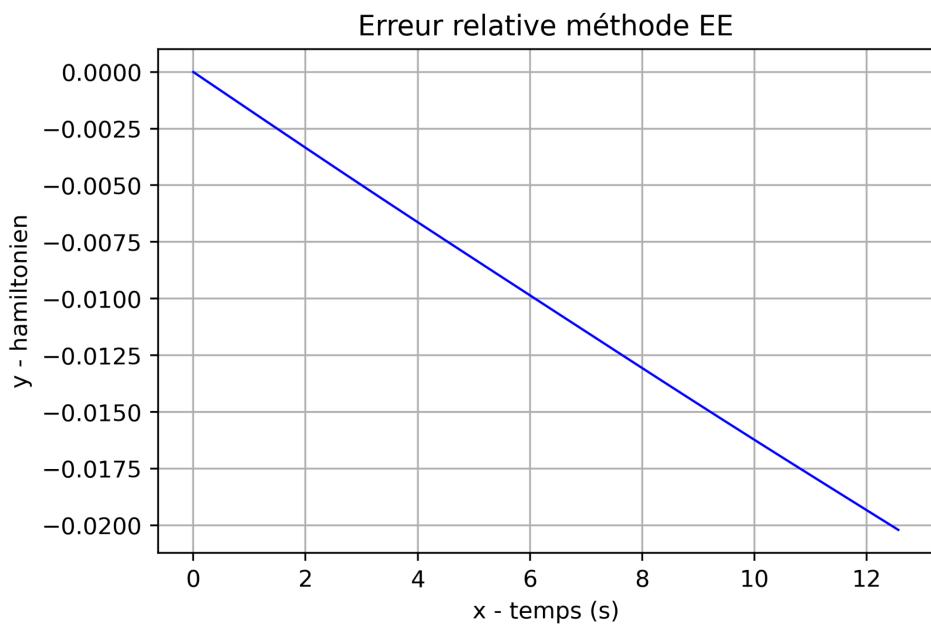
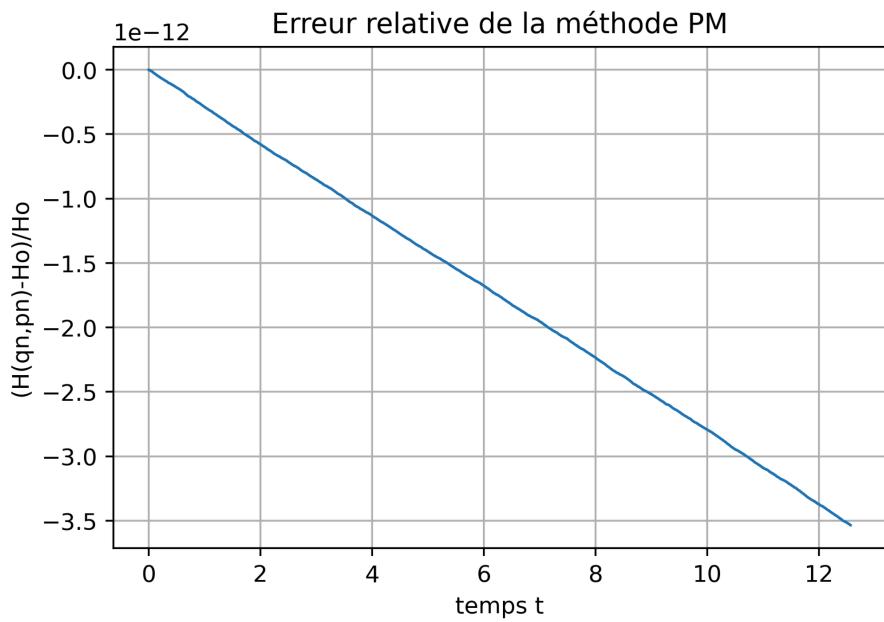
$$H(x_0) = H\left(\left(\frac{1}{2}\right), \left(\frac{0}{1}\right)\right) = \frac{1}{2} (1^2 + 0^2) - \frac{1}{\sqrt{0^2 + 1^2}} = -\frac{1}{2}$$

Ensuite, nous avons codé la ligne qui suit pour tracer l'erreur relative de chaque méthode.

```
# plotting the points
plt.plot(SolT, (SolH-H0)/H0, 'b-')
```

Le fichier **P2qs3.py** nous a permis d'obtenir les graphes suivants :





On remarque grâce aux graphes que l'erreur relative de la méthode RK4 est bien moins importante (de l'ordre de 1e-14) que celle pour les méthodes EE (de l'ordre de 0,1) et PM (de l'ordre de 1e-12). La méthode à privilégier dans ce TP est donc RK4, car elle est beaucoup plus précise que EE et PM.

Partie III : Application, système gravitant en 2D

Dans cette partie, on néglige l'excentricité c'est-à-dire on suppose que les orbites sont circulaires ; et on considère que $GM = 1$.

Nom	Diamètre	Masse	Distance moyenne de Jupiter	Période orbitale	Excentricité	Vitesse
	km	kg	km	jours		km/s
Io	3630	$8.94 \cdot 10^{22}$	421600	1.769138	0.0041	17.34
Europe	3138	$4.8 \cdot 10^{22}$	670900	3.551181	0.009	13.74
Ganymède	5268	$1.48 \cdot 10^{23}$	$1.07 \cdot 10^6$	7.154553	0.002	10.9
Calisto	4806	$1.07 \cdot 10^{23}$	$1.883 \cdot 10^6$	16.68902	0.007	8.21

III.1. Montrons à partir des valeurs du tableau ci-dessus que la vitesse orbitale est compatible avec la période orbitale et la distance moyenne de Jupiter si l'on néglige l'excentricité de l'orbite.

Pour cela on utilise la formule suivante : $T_{\text{orb}} = 2\pi d/V$ avec :

d = distance moyenne de Jupiter convertie en m

V =vitesse du satellite en m/s

T_{orb} =période orbitale en s du satellite

Après calculs, on obtient les valeurs suivantes que l'on convertie en jours :

Io : $T_{\text{orb}} = 152\ 767,6428\text{s} = 1,7681\text{j}$

Europe : $T_{\text{orb}} = 306\ 796,8721\text{s} \approx 3,5508\text{j}$

Ganymède : $T_{\text{orb}} = 616\ 789,7503\text{s} \approx 7,1387\text{j}$

Callisto : $T_{\text{orb}} = 1\ 441\ 076,484\text{s} \approx 16,6791\text{j}$

Ces périodes orbitales sont très proches de celles indiquées dans le tableau. Par conséquent, en négligeant l'excentricité de l'orbite, il y a toujours compatibilité entre les mesures.

III.2. On désire calculer numériquement la période orbitale de Io en utilisant la vitesse orbitale et la distance moyenne à Jupiter du tableau ci-dessus. Le premier point pour y arriver est de déterminer quelles conditions initiales utiliser pour l'équation de gravitation.

On initialise donc les vecteurs position et vitesse de la façon suivante :

$\mathbf{q}_0 = (\mathbf{r}, 0)$ et $\mathbf{p}_0 = (0, \mathbf{V})$ avec \mathbf{r} la distance moyenne entre le satellite et Jupiter.

On a donc : $\mathbf{X}_0 = (421600e^3, 0, 0, 17.34e^3)$

III.3. On intègre alors l'équation différentielle (*) (*cf partie I*) avec le schéma RK4, et un pas de temps de 8 secondes. On souhaite montrer que l'on fait une erreur de moins de 0.1% sur la période orbitale de Io. Pour cela, on choisit un temps final de 2j ce qui correspond à 172 800s ; car $172\ 800 = 2*(24*60*60)$.

Puis, on crée un code dans lequel on initialise q0 et p0 grâce aux données trouvées à la question 2.

```
# Boucle sur le temps
for n in range(N+1):
    # Pour le plot
    SolT[n]=t
    SolY[:,n]=Y
    if (SolY[1,n]>0)& (SolY[1,n-1]<0) :
        print(t,(t-Tio)/Tio)
```

Les lignes de code suivantes nous permettent d'afficher le temps où la deuxième composante de q change de signe ; on obtient $t = 152\ 976\text{s}$. Cela correspond au temps mis par le satellite pour effectuer un tour autour de Jupiter. L'erreur liée à la période orbitale est calculée grâce à la formule $\frac{(t-T_{\text{Io}})}{T_{\text{Io}}}$ avec $T_{\text{Io}} = 152853.5232\text{s}$ ce qui correspond à 1.769138j .

Après exécution du code **P3qs3.py**, on obtient : $Err \approx 0.0008013 \leq 0.01$ soit 0.1% . Cela montre que l'on retrouve à moins de 0.1% près, la période de Io.

III.4. On fait la même chose pour Callisto, en prenant cette fois-ci un pas de temps de 60s . On choisit un temps final de 18j soit $1\ 555\ 000\text{s}$; car $1\ 555\ 000 = 18*(24*60*60)$. Ensuite, on initialise q0 et p0 avec la distance moyenne entre Jupiter et Callisto en mètres soit $1.883\text{e}9\text{ m}$ et avec la vitesse 8210 m/s . On a donc :

$$X_0 = (1.883e^9, 0, 0, 8.21).$$

Ici, la deuxième composante de q change de signe à $t = 1\ 446\ 600\text{s}$. De plus, $T_{\text{Orbi}} = 1\ 441\ 931.328\text{s}$.

D'où après exécution du code **P3qs4.py**, on obtient : $Err \approx 0.003238 \leq 0.0035$ soit 0.35% . Cela montre que l'on retrouve à moins de 0.35% près, la période de Callisto.

III.5. Nous avons plusieurs hypothèses concernant la principale source d'erreur pour la période orbitale de Callisto.

- Tout d'abord, nous pensons que négliger l'excentricité des orbites peut être une source d'erreur, car les données que l'on utilise sont moins précises et le problème de gravitation que l'on étudie est plus éloigné de la réalité.
- Enfin, une autre source d'erreur peut être la distance entre les satellites et Jupiter. En effet, d'après le tableau ci-dessous on remarque que plus la distance est grande, plus l'erreur l'est aussi.

	Erreur	Distance à Jupiter (m)
Io	$8.013e^{-4}$	$4.216e^8$
Callisto	$3.2380e^{-3}$	$1.883e^9$

Partie IV : Système gravitant en 3D

Dans cette partie, on considère le problème de gravitation en 3D, afin de calculer les positions et trajectoires de chaque satellite. Nous devons donc réécrire l'équation en dimension 3 de la fonction f correspondante.

3) On a en dimension 3 :

$$f: \mathbb{R}^+ \times \mathbb{R}^6 \rightarrow \mathbb{R}^6$$

$$(t, x) \mapsto f(t, x) = \dot{x}(t) = \begin{cases} p_1(t) \\ p_2(t) \\ p_3(t) \\ -GMq_1/\|q_1(t)\|^3 \\ -GMq_2/\|q_2(t)\|^3 \\ -GMq_3/\|q_3(t)\|^3 \end{cases}$$

```
# Dimension de l'espace
NDIM = 6
# Méthode numérique choisie
methode = 'MethRK4'
GM = 6.672e-11 * 1.8987e27

# Fonction définissant le système dynamique
def f(t, Y):
    q1 = Y[0]; q2 = Y[1]; q3 = Y[2]; p1 = Y[3]; p2 = Y[4]; p3 = Y[5];
    r = sqrt(q1**2 + q2**2 + q3**2)
    return array([p1, p2, p3, -GM*q1/r**3, -GM*q2/r**3, -GM*q3/r**3], float)
```

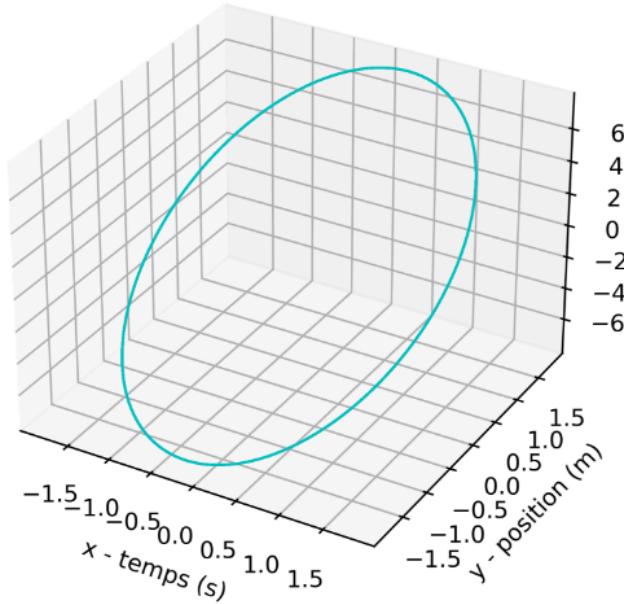
On utilise RK4 pour résoudre cette équation en dimension 3 avec un pas de temps d'une minute.

IV.2. On se sert des tableaux de données fournies par l'IMCCE, afin de tester la performance de nos schémas numériques sur ce type de problème. On initialise le système de la question IV.1 avec les données des premières lignes de chaque tableau, que l'on convertit en mètres.

On obtient alors les trajectoires de chaque satellite, autour de la planète Jupiter. Après exécution des codes : **P4qs2-Amalthee.py**, **P4qs2-Callisto.py**, **P4qs2-Europe.py**, **P4qs2-Io.py** et **P4qs2-Ganymede.py**.

Par exemple pour Amalthee, en choisissant un pas de temps d'une minute et une période de 18j on obtient le graphe suivant :

Trajectoire de Amalthee



IV.3. On souhaite calculer numériquement l'erreur relative de la période orbitale de Io avec ces nouvelles valeurs initiales. En utilisant un pas de temps d'une minute et en intégrant sur 30j, on obtient la période orbitale de Io (1.769138j) avec une erreur relative de moins de 0.15%.

En effet, après exécution du code **P4qs3.py**, on obtient les données ci-dessous (*cf P4qs3-données*). On détermine t de la même manière qu'à la question III.3 et on a pour $t = 294540 - 141480 = 153060s$; $Err = 0.00135 \leq 0.0015$ soit 0.15%.

```
print(294540-141480,(294540-141480-Tio)/Tio) #calcul erreur
```

```

1 141480 -0.07440798852322428
2 294540 0.9269428262678083
3 447600 1.928293641058841
4 600720 2.9300369885095328
5 753780 3.931387803300566
6 906840 4.932738618091599
7 1059900 5.934089432882631
8 1213020 6.935832780333323
9 1366080 7.937183595124355
10 1519140 8.938534409915388
11 1672200 9.93988522470642
12 1825320 10.941628572157112
13 1978380 11.942979386948146
14 2131440 12.944330201739177
15 2284500 13.945681016530209
16 2437560 14.947031831321242
17 2590680 15.948775178771934
18
19 153060 0.0013508147910325956

```

Cela montre que l'on retrouve à moins de 0.15% près la période de Io.

IV.4. De même, en utilisant un pas de temps d'une minute et en intégrant sur 30j, on obtient la période orbitale de Europe (3.551181j) avec une erreur relative de moins de 0.35%. En effet, après exécution du code **P4qs4.py**, on obtient les données ci-dessous (*cf P4qs4-données*) et on a pour $t = 463380 - 156360 = 307020\text{s}$; $Err = 0.000645 \leq 0.0008$ soit 0.08%.

```
1 156360 -0.4903886278333258
2 463380 0.5102565722345451
3 770400 1.510901772302416
4 1077420 2.511546972370287
5 1384440 3.512192172438158
6 1691460 4.512837372506029
7 1998480 5.5134825725739
8 2305500 6.514127772641771
9
10 307020 0.0006452000678709411
```

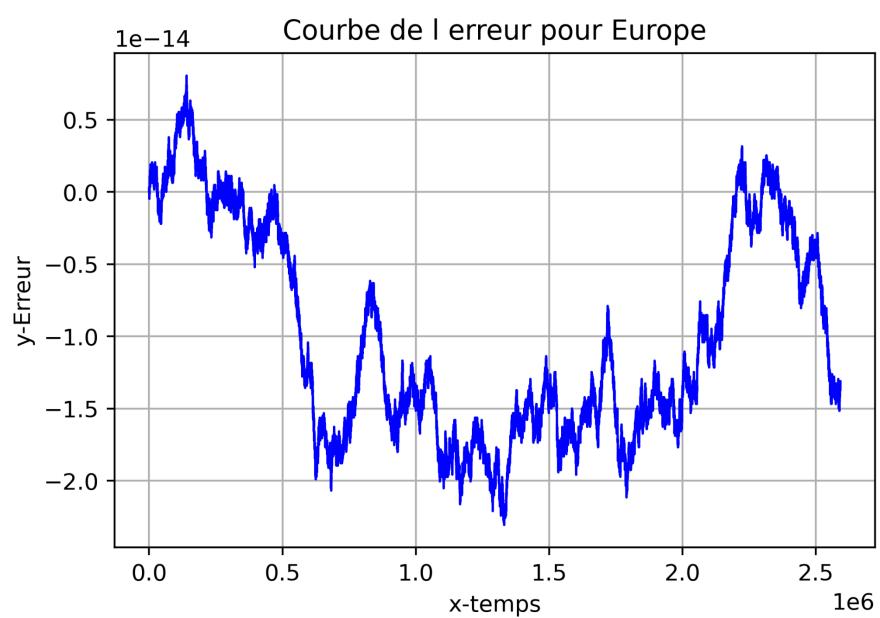
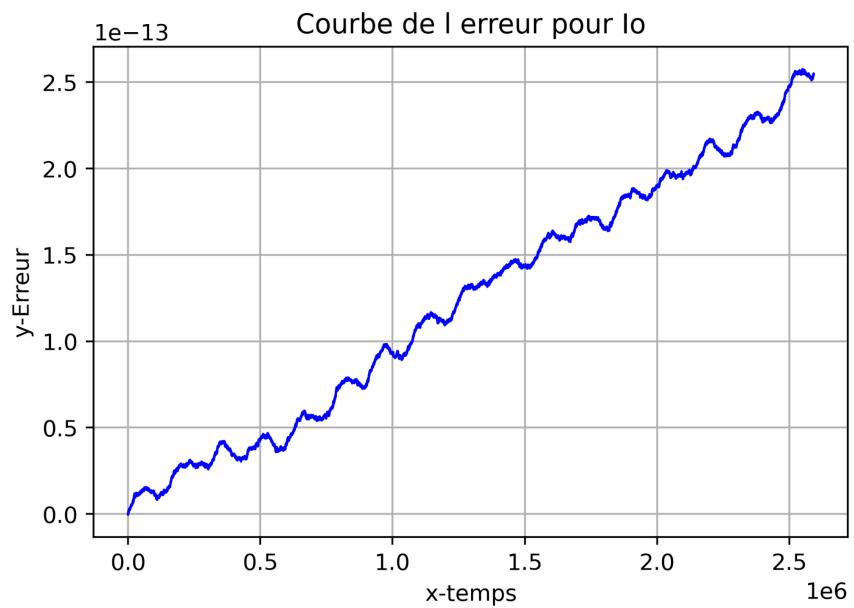
Cela montre que l'on retrouve à moins de 0.08% près, la période du satellite Europe.

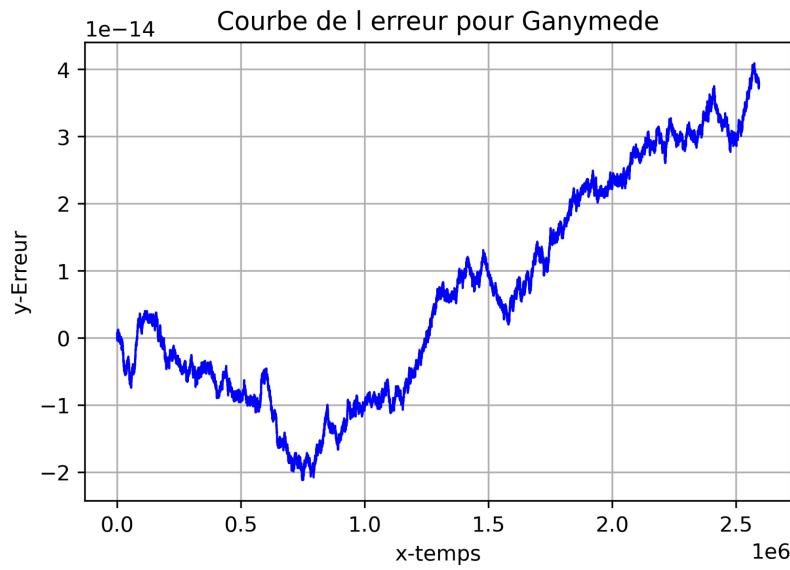
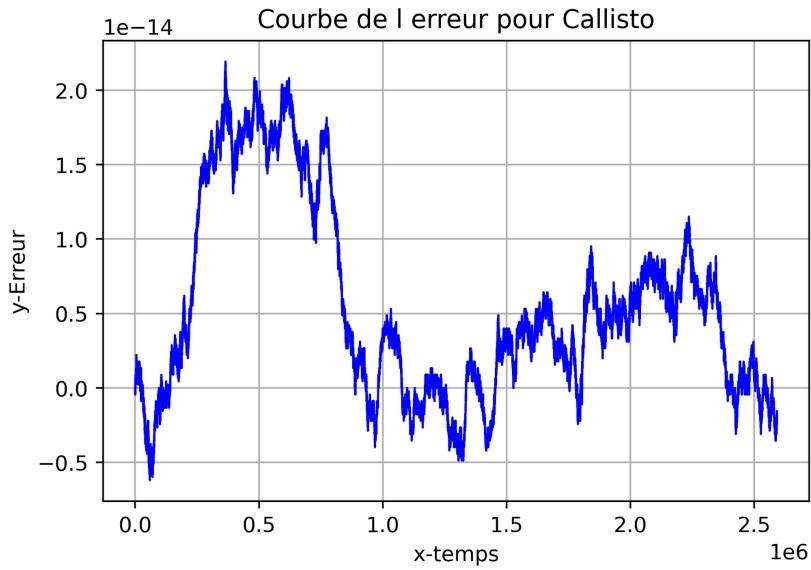
IV.5. En gardant le même pas de temps, utilisons la formule : $\frac{H(X(t)) - H(X(0))}{H(X(0))}$ pour observer l'erreur sur le hamiltonien pour les quatres satellites.

Dans un premier temps, on code le hamiltonien de la manière suivante :

```
def H(X):
    q1 = X[0]
    q2= X[1]
    q3= X[2]
    p1=X[3]
    p2=X[4]
    p3= X[5]
    return 0.5*(p1**2 + p2**2+ p3**2)-1/sqrt(q1**2+q2**2+q3**2)
```

On exécute les codes **P4qs5-Io.py**, **P4qs5-Europe.py**, **P4qs5-Ganymede.py**, **P4qs5-Callisto.py** et on obtient les graphes suivants :



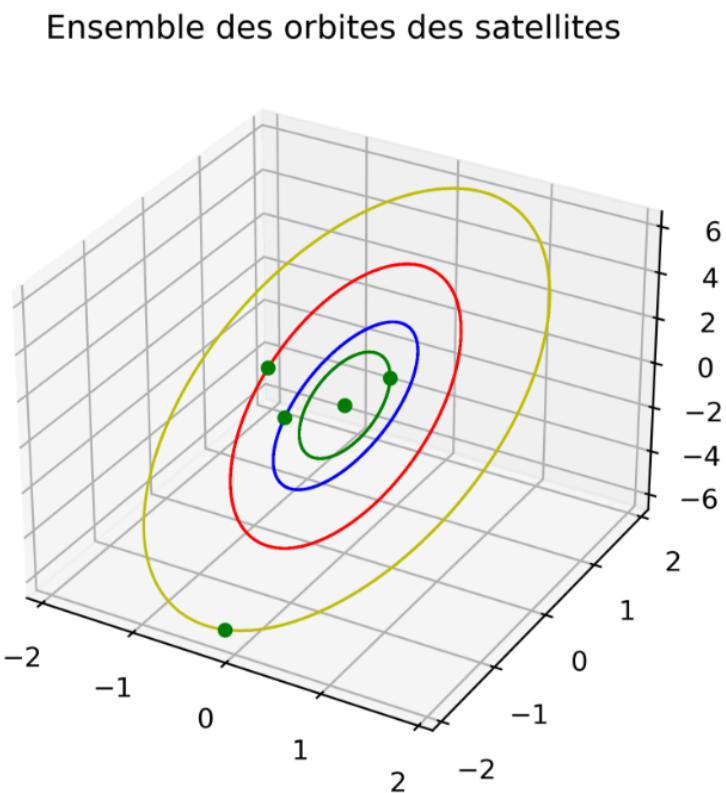


On remarque que pour les satellites Io et Ganymède, l'erreur sur le hamiltonien augmente. Pour Io, elle augmente proportionnellement au temps. La courbe de l'erreur pour Europe nous indique une erreur maximale (environ $0.7e-14$) à environ $0.2s$, tandis que l'erreur maximale pour Callisto est atteinte à environ $0.5s$ (environ $2e-14$).

IV.6. Pour finir, on crée un module appelé **cc2partie4.py** où est définie une fonction Courbe contenant la fonction f , ainsi que la méthode utilisée (RK4). On crée aussi **P4qs6.py** où se trouvent les instructions pour le tracé du graphe, ainsi que les conditions initiales. Procéder de cette manière nous permet d'afficher toutes les orbites sur le même graphe.

On trace donc les orbites des satellites Europe (en bleu), Ganymède (en rouge), Callisto (en jaune) et Io (en vert), sur une période de 30 jours et un pas de temps d'une minute, en utilisant leur configuration initiale au 15 avril 2024 à minuit.

On obtient le graphe suivant :



Conclusion

Dans la partie II, nous avons étudié l'évolution de l'hamiltonien pour chacun des schémas (PM), (EE) et (RK4). Le graphe représentant l'erreur relative de l'hamiltonien nous a montré que le schéma ayant l'erreur la plus faible était (RK4). C'est pourquoi dans les parties 3 et 4, c'est le schéma qui est utilisé.

La partie III nous a permis de calculer numériquement les périodes orbitales de Io et de Callisto en étant en dimension 2. Cela nous a permis de mettre en évidence les principales sources d'erreurs possibles pour les périodes orbitales.

Enfin dans la partie IV, nous avons travaillé en dimension 3 pour s'intéresser aux positions et trajectoires des satellites. Nous avons trouvé d'autres erreurs qui ont confirmé les hypothèses faites dans la partie III.

Pour conclure, ce TP nous a permis de nous familiariser à la syntaxe de Python. Nous avons eu quelques difficultés à afficher les graphes correctement, notamment pour la question 6 de la partie 4, car le logiciel Spyder que nous avions sur nos ordinateurs ne fonctionnait pas correctement.