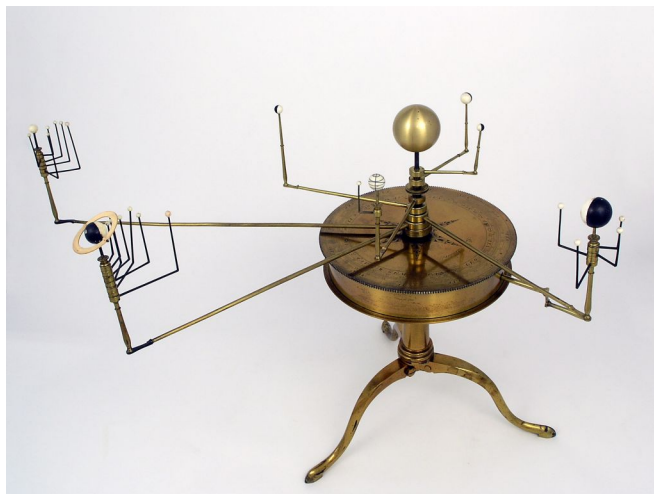


Projet : Gravitation et orbites



Dans ce projet, vous allez étudier la physique des orbites des planètes, dont des exoplanètes. À l'échelle du système solaire, le comportement des corps célestes se décrit, à une très bonne approximation, par les lois de Newton (chapitre 2, section 3 des notes de cours) et par la force gravitationnelle (chapitre 2, section 8.1 des NoC). Avec cette liste courte de principes et d'équations, on peut, en principe, décrire avec précision l'évolution temporelle du système – c'est-à-dire, les orbites des planètes. Cependant, des résolutions analytiques des systèmes à plusieurs corps s'avèrent difficiles, voire impossibles. On a recours, donc, à une résolution numérique.

Les compétences à acquérir sont les suivantes :

- La physique des forces, la gravitation, les lois de Newton.
- La manipulation des vecteurs.
- Les changements d'unités.
- L'analyse des données, l'ajustement d'un modèle linéaire aux données.
- La résolution d'un problème par la méthode numérique d'itération.
- Une compréhension des limites de la méthode, et comment évaluer son domaine de validité.

1. Introduction : Les exoplanètes, et la troisième loi de Kepler

L'astronome Kepler a identifié trois lois décrivant les orbites des planètes autour d'un astre, dont la troisième est la suivante :

$$\left(\frac{2\pi}{P}\right)^2 a^3 = k$$

où P est la période de révolution (techniquement, la période sidérale) de la planète, a est le demi-grand axe de la trajectoire elliptique de la planète, et k est une constante.

Newton a montré que la constante k est donnée par la formule suivante :

$$k = \mathcal{G}(M + m) \approx \mathcal{G}M$$

où m est la masse de la planète, M est la masse de l'astre, et \mathcal{G} est la constante gravitationnelle : $\mathcal{G} = 6,674 \times 10^{-11} \text{ m}^3 \text{ kg}^{-1} \text{ s}^{-2}$. L'approximation $\mathcal{G}(M + m) \approx \mathcal{G}M$ est valable dans la limite où la masse de l'astre est grande devant celle de la planète. Dans cette limite, la valeur de k est identique pour toutes les planètes d'un système solaire.

Kepler étudiait notre système solaire, mais ses lois doivent s'appliquer plus largement. La NASA a rassemblé [une base de données d'observations d'exoplanètes](#), c'est-à-dire des planètes se situant dans des systèmes solaires autres que le nôtre. Dans le fichier `nasa-exoplanetes.csv`, vous trouverez une fraction de ces données, dans un format adapté à la lecture par un code python. Les données se regroupent sous la forme de quatre colonnes et plusieurs rangées (lignes). Ces quatres colonnes représentent :

1. la période de l'orbite de la planète, P , en jours ;
2. le demi-[grand axe](#) de la trajectoire, a , en au ([unités astronomique](#)) ;
3. la masse de la planète, m , en unités de [masses terrestres](#) ($1 M_T$ = la masse de la Terre) ;
4. la masse de l'astre, M , en unités de [masses solaires](#) ($1 M_S$ = la masse du Soleil).

Dans cette première partie, nous allons lire et étudier ces données afin de tester la théorie de Kepler.

Étape 1 : lecture du fichier

[Q1] En utilisant `numpy`, lire le contenu du fichier CSV et stocker le dans un tableau (array) bidimensionnel que vous appellerez `fiche`. (Par exemple, la méthode `numpy.loadtxt()` permettrait d'obtenir un tel tableau.)

```
[1]: # [Q1 : votre code ici]
```

Étape 2 : calcul

Ensuite, nous allons traiter les données, une ligne à la fois. Pour ce faire, il vous faudra une boucle `for`. **Pour chaque planète** (chaque ligne du tableau) :

- [Q2] Extraire les paramètres P , a , m , et M .
- [Q3] Calculer la valeur de $f = \frac{a^3}{P^2 M}$ et imprimer sa valeur sous une forme semblable à la suivante : La valeur de f de la [...]ième planète est [...] au³ jour⁻² MS⁻¹ (où vous remplacerez les [...] par des chiffres). Stocker cette valeur dans un array.

```
[2]: # [votre code ici pour Q2]

# [votre code ici pour Q3, utilisant une boucle 'for']
```

[Q4] Que constatez-vous ?

(vos observations ici)

[Q5] Calculer la moyenne et l'écart-type de l'ensemble de vos valeurs de f (en utilisant `numpy.mean()` et `numpy.std()` ou autrement).

```
[3]: # [votre code ici pour Q5]
```

Étape 3 : graphique

Il convient de visualiser des données sous forme d'un graphique.

[Q6] Pour chaque planète, calculer les deux quantités suivantes. Stockez-les dans des arrays (un array pour les x , un autre pour les y).

- les abscisses $x = P^2 M$
- les ordonnées $y = a^3$

et puis tracer un nuage de points (en utilisant `matplotlib.pyplot.scatter()` ou autrement). N'oubliez pas d'ajouter un titre et nommer vos axes (avec les bonnes unités).

[4]: # [votre code ici pour Q6]

Étape 4 : ajustement et interpretation

[Q7] Effectuer un ajustement linéaire aux données (avec `numpy.polyfit()` ou autrement). Dessiner un graphique incluant la fonction ajustée et les données (sous forme d'un nuage de points).

[5]: # [votre code ici pour Q7]

[Q8] Commenter l'ajustement. (Les données, sont-elles cohérentes avec la fonction ?)

[Q9] Selon la loi de Kepler, la droite passe par l'origine. Est-ce bien le cas ? (L'ordonnée à l'origine, est-elle cohérente avec zéro ?) Que vaut le coefficient directeur de la droite ? Est-il cohérent avec la moyenne calculée précédemment ?

(vos observations ici pour Q8 et Q9)

Pour aller plus loin : [Q10] Le coefficient directeur de la droite, est-il cohérent avec la théorie ? * Attention, cette question Q10 nécessite un peu de travail. Les équations de l'introduction ci-dessus permettent de déterminer la valeur théorique, en fonction de constantes connues (G et π). Cependant, G s'exprime en unités SI ($\text{m}^3 \text{kg}^{-1} \text{s}^{-2}$), mais le coefficient directeur s'exprime en d'autres unités (au, jours, masses solaires). Il faudra passer d'un système d'unités à l'autre. Pour ce faire, posez-vous les questions suivantes : Combien de mètres vaut 1au (une unité astronomique) ? Combien de s vaut un jour ? Combien de kg vaut la masse solaire M_S ?

[6]: # [votre code ici pour Q10]

(vos observations ici pour Q10)

2. Modélisation simple d'un système simple

2.1 : Introduction

Rappelons la loi de Newton (le principe fondamental de la dynamique):

$$m_{\text{tot}} \frac{d\vec{v}_C}{dt} = \vec{F}_{\text{tot}}$$

où m_{tot} est la masse totale d'un système,

\vec{v}_C est la vitesse (vectorielle) du centre de masse C du système, et

\vec{F}_{tot} est la résultante des forces extérieures qui s'appliquent sur le système.

Rappelons également comment calculer la force gravitationnelle entre deux corps :

$$\vec{F}_{A \rightarrow B} = -G \frac{m_A m_B}{||\vec{AB}||^2} \vec{u}_{AB}$$

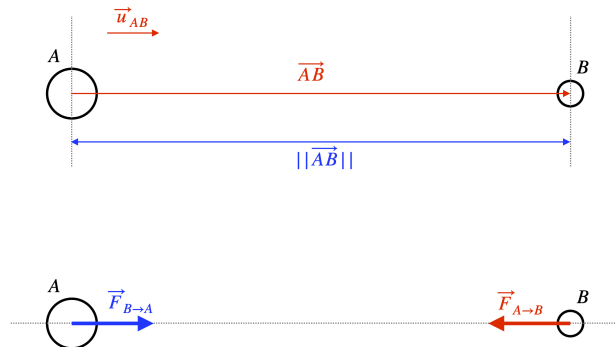
où $\vec{F}_{A \rightarrow B}$ est la force exercée par un corps (ou un système d'objets) A de masse m_A sur un corps B de masse m_B ,

\vec{AB} est la distance vectorielle entre les centres de masse des deux corps,

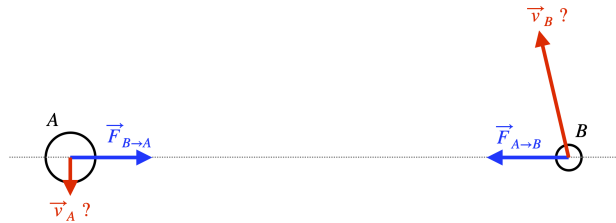
G est la constante gravitationnelle, et \vec{u}_{AB} est le vecteur unitaire de la distance entre les deux corps :

$$\vec{u}_{AB} = \frac{\vec{AB}}{||\vec{AB}||}$$

Nous allons mettre en oeuvre une résolution numérique de ces équations. Nous étudierons d'abord des systèmes suffisamment simples qu'ils permettent aussi des résolutions *analytiques*. L'intérêt est de comparer les deux approches afin de valider l'approche numérique et de comprendre ses limites.



Vous trouverez ci-dessus une illustration des vecteurs pour les deux corps A et B. Noter que les deux forces $\vec{F}_{A \rightarrow B}$ et $\vec{F}_{B \rightarrow A}$ sont de normes identiques mais de sens opposés ; dans le cas de la gravitation, elles dirigent A vers B et, inversement, B vers A. Noter en plus que nous n'avons à aucun point indiqué les vitesses des deux corps, car les forces gravitationnelles ne dépendent pas d'eux (s'expriment uniquement en fonction des masses et de la distance vectorielle \vec{AB}). Par exemple, voici **une configuration possible** des vecteurs vitesses \vec{v}_A et \vec{v}_B :



2.2 : Calculer la distance et la force entre deux corps

Un vecteur en trois dimensions, par exemple la vitesse du corps A, peut se représenter sous la forme de trois nombres. Dans ce projet, nous utiliserons les coordonnées cartésiennes $\vec{x} = (x, y, z)$, et nous représenterons des vecteurs en python par des objets `numpy.array`.

Nous aurons besoin, à plusieurs reprises, de calculer la distance vectorielle (vecteur déplacement) entre deux points, et d'en déduire sa norme et sa direction. Par exemple :

```
[7]: import numpy
import math

# Nous définissons deux vecteurs, qui représentent des positions de deux objets :
exempleVecA = numpy.array([0.0, 0.0, 1.0]) # en mètres
exempleVecB = numpy.array([0.0, 2.0, 1.0]) # en mètres

# Nous calculons le vecteur déplacement :
exempleVecDiff = exempleVecB - exempleVecA # en mètres
# Le produit scalaire de cette différence avec elle-même (la magnitude au carré) :
exempleVecDiffMagCarre = exempleVecDiff.dot(exempleVecDiff) # en m^2
# La magnitude :
exempleVecDiffMag = math.sqrt(exempleVecDiffMagCarre) # en mètres
```

```
# Le vecteur unitaire associé :
exempleVecDiffUnitaire = exempleVecDiff / exempleVecDiffMag # sans unités
```

[Q11] Muni de ces outils, écrire une fonction python qui calcule la force vectorielle entre deux corps A et B, dont les positions sont x_A et x_B (deux objets de type `numpy.array`), et dont les masses sont m_A et m_B . La force étant un vecteur, la fonction doit renvoyer un `numpy.array`. Vous pouvez utiliser la trame suivante :

```
def forceGravitationnelle(xA, xB, mA, mB):
    # votre code ici
    F = [...] # vous calculez la force
    return F
```

[8] : `# [votre code ici pour Q11]`

Il est une bonne pratique de tester et de vérifier le code que l'on vient d'écrire. Dans l'exemple ci-dessus, nous avons défini deux objets séparés d'une distance de deux mètres dans la direction $+y$. Soient leurs masses 1kg. On peut calculer la norme de la force (on obtient $1,6685 \times 10^{-11} \text{ kg m s}^{-2}$). Et l'orientation ? $\vec{F}_{A \rightarrow B}$ est la force exercée par A sur B, c'est-à-dire la force subie par B ; la force gravitationnelle étant attractive, elle va diriger B vers A, donc dans la direction $-y$, c'est-à-dire $(0, -1, 0)$. Est-ce bien le cas ? Le code ci-dessous effectuera ce test (après que vous avez défini la fonction `forceGravitationnelle` dans la Q11 ; si vous avez choisi un autre nom, il faudrait faire une modification correspondante) :

```
exempleForce = forceGravitationnelle(exempleVecA, exempleVecB, 1.0, 1.0)
exempleForceMagnitude = math.sqrt(exempleForce.dot(exempleForce))
print("La force est", exempleForce, "kg m s^{-2}.")
print("Sa magnitude vaut ", exempleForceMagnitude, "kg m s^{-2}")
print("Son vecteur unitaire est", exempleForce/exempleForceMagnitude)
```

[Q12] Ajouter ces lignes de code et effectuer ce test. Si tout est correct, le résultat de ce test simple devrait se ressembler à la suivante :

```
La force est [-0.0000e+00 -1.6685e-11 -0.0000e+00] kg m s^{-2}.
Sa magnitude vaut  1.6685e-11 kg m s^{-2}
Son vecteur unitaire est [-0. -1. -0.]
```

[9] : `# [votre code ici pour Q12]`

2.3 : Premier jet : un champ gravitationnel statique

Ayant écrit et testé ces outils, nous pouvons ensuite simuler un système plus intéressant ! On considère un système constitué du Soleil (corps A) et de la Terre (corps B). Nous souhaitons modéliser le trajet de la Terre autour du Soleil. Dans ce premier exercice, le Soleil est supposé fixe et immobile à l'origine.

La masse du Soleil vaut $1,9885 \times 10^{30} \text{ kg}$. La masse de la Terre vaut $5,972 \times 10^{24} \text{ kg}$.

Exercice : calculer la force subie par la Terre

À $t=0$ (c'est-à-dire dans les conditions initiales), la Terre se trouve à une distance $d_0 = 149\,596\,006 \text{ km}$ du Soleil. On place la Terre à $x_B = (d_0, 0, 0)$. Le soleil est forcément à $x_A = (0, 0, 0)$.

[Q13] Utiliser votre fonction pour évaluer la force gravitationnelle (vectorielle) subie par la Terre à cet instant, en unités SI.

[10] : `# [votre code ici pour Q13]`

[Q14] Ensuite, faites le calcul vous-même (sur papier, ou avec une calculatrice) et vérifiez que vos deux valeurs sont cohérentes.

(vos observations ici pour Q14)

2.4 : L'évolution du système

Mais la Terre ne va pas rester à cette position initiale : elle se déplace ! Nous calculerons son trajet en suivant une approche numérique itérative, en faisant une série de calculs par pas de temps dt .

Supposons que le pas dt soit suffisamment court pour que la réponse du système soit linéaire, c'est-à-dire

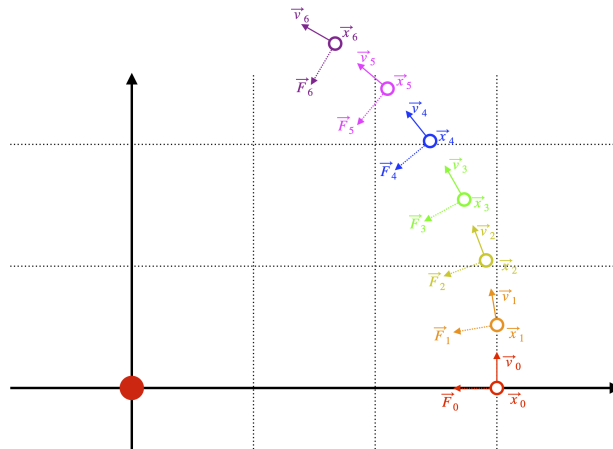
$$\vec{x}_B(t + dt) = \vec{x}_B(t) + d\vec{x} = \vec{x}_B(t) + \vec{v}_B(t)dt$$

$$\vec{v}_B(t + dt) = \vec{v}_B(t) + d\vec{v} = \vec{v}_B(t) + \vec{a}_B(t)dt$$

où $\vec{x}_B(t)$ est la position de la Terre au temps t , $\vec{v}_B(t)$ est sa vitesse, et $\vec{a}_B(t)$ est son accélération. On procède comme suite :

- On définit les conditions initiales : les valeurs des paramètres $\vec{x}_B(t)$ et $\vec{v}_B(t)$ à $t = 0$.
- On calcule $\vec{F}_{A \rightarrow B}(t)$ et ainsi $\vec{a}_B(t)$ à $t = 0$.
- Sachant la vitesse et l'accélération à $t = 0$, on calcule les paramètres à dt : $\vec{x}_B(dt)$ et $\vec{v}_B(dt)$. On en déduit la force et ainsi l'accélération à $t = dt$.
- Avec ces nouvelles valeurs à $t = dt$, on calcule les paramètres à $t = 2dt$.
- Avec ces nouvelles valeurs à $t = 2dt$, on calcule les paramètres à $t = 3dt$.
- (etc)

Voici une illustration de l'approche (avec un pas plutôt grossier). À chaque point $i = 0, 1, \dots, 6$, on détermine la position (\vec{x}_i), la vitesse (\vec{v}_i), et la force subie (\vec{F}_i). Notez que, par exemple, on calcule $\vec{x}_1 = \vec{x}_0 + \vec{v}_0 dt$ et non avec une formule trigonométrique.



Ceci est une approche itérative. Parce que nous allons refaire les calculs de nombreuses fois, nous aimerions écrire un bloc de code réutilisable dans des boucles.

Exercice : avancer un pas

[Q15] Écrire une fonction qui prend en entrée les valeurs actuelles de x_A , x_B , et v_B et un pas dt , qui fait avancer la position et la vitesse de la Terre d'un pas, et qui renvoie les nouvelles valeurs. (Dans ce modèle simplifié, le Soleil reste fixe à l'origine. Parce qu'il ne bouge pas, nous ne précisons pas sa vitesse nulle v_A .) Vous pouvez utiliser la trame suivante :

```
def champStatiqueAvanceUnPas(xA, xB, vB, dt):  
    # votre code ici
```

```
nouvelle_xB = [...]
nouvelle_vB = [...]
return nouvelle_xB, nouvelle_vB
```

Point technique : Python traite des nombres simples (comme dt) un peu différemment des objets composés (comme xB). Selon la structure de votre code, vous pourriez avoir besoin de créer une nouvelle copie d'un vecteur avant de le manipuler, par exemple :

```
dxB = dt * numpy.array(vB)
```

[11]: # [votre code ici pour Q15]

[Q16] Ensuite, tester la fonction. Par exemple, avec une vitesse initiale de **29,78 km/s** vers $+y$ et un pas de 1s, on attend que la Terre se déplace d'environ 30km vers $+y$ lors du pas, et que sa vitesse évolue vers le Soleil (vers $-x$). Ajouter un test et vérifier qu'il donne un résultat raisonnable.

[12]: # [votre code ici pour Q16]

2.5 : Point sur le choix du pas

Le pas (dt) n'est pas un paramètre physique : il ne correspond pas à une propriété observable du système solaire. En principe, on devrait trouver des résultats semblables pour tout choix *valable* de dt (c'est-à-dire, pour tout choix qui valide l'hypothèse que l'évolution du système est linéaire pendant l'intervalle dt). Cependant, il est un paramètre de notre modèle, et un mauvais choix du pas peut fausser le résultat. En général, on veut que le pas soit le plus court possible. Mais la précision a son prix : plus dt est petit, plus le temps de calcul augmente. Un pas très, très petit peut aussi engendrer des problèmes de précision numérique. Dans ce projet, vous devrez choisir un pas adapté au problème, et étudier l'effet de la variation de sa valeur.

Exercice : L'orbite de la terre On va maintenant suivre la Terre autour du Soleil. On prend des valeurs initiales suivantes :

$$\vec{x}(t=0) = (d_0, 0, 0)$$

$$\vec{v}(t=0) = (0, v_0, 0)$$

où

$$d_0 = 149\,596\,006 \text{ km}$$

$$v_0 = 29,78 \text{ km s}^{-1}$$

[Q17] Choisir un pas dt , et déterminer le nombre N de pas correspondant à un an. Ensuite, utilisez votre fonction (définie dans la question 15) N fois et déterminer la position de la Terre après un an.

[13]: # [votre code ici pour Q17, à réutiliser ensuite en étudiant les Q19-21]

(vos remarques sur le choix de dt ici pour Q17)

[Q18] Visualiser l'évolution temporelle de la position de la Terre. * Remarques : Il est convenable de pouvoir visualiser ce que fait le calcul. À cette fin, vous pouvez utiliser la fonction `scatter()` de `matplotlib.pyplot`. Astuce : au lieu de dessiner un point par pas, vous pourriez dessiner un point par 1000 pas (ou 100, ou 10^4 , ou etc, en fonction de sa durée) ; l'opération modulo (%), le reste de la division d'entiers) pourrait vous aider. Vous pouvez ainsi utiliser un pas court sans stocker un nombre excessif de points.

[14]: # [votre code ici pour Q18]

Questions à considérer, muni de votre outil de simulation numérique :

- [Q19] Est-ce que la Terre est revenue à son point de départ après un an ? (À quelle précision ?)
- [Q20] Que se passe-t-il si le pas dt varie ? (Vous pouvez tenter, par exemple, {10000s, 1000s, 100s, 10s, 1s} ou {1jour, 1heure, 1min, 1s}. Attention, le calcul pourrait devenir long dans le cas d'un pas court.)
- [Q21] Que se passe-t-il après plusieurs cycles, par exemple au fil de 10 ans, ou de 100 ans ?

(vos observations ici pour les Q19-21)

3. Un système à deux corps [s'il vous reste le temps nécessaire]

Dans la réalité, le Soleil n'est pas fixe et immobile (troisième loi de Newton). Parce que la force subie par la Terre dépend de la position du Soleil, ce déplacement changera l'évolution subséquente du système. Son comportement devient plus complexe : le mouvement de la Terre dépend du mouvement du Soleil, qui dépend du mouvement de la Terre.

Nous adapterons notre simulation numérique pour qu'elle tienne compte du mouvement du Soleil. Pour ce faire, nous retenons la même approche itérative ; la différence, c'est qu'il faut maintenant suivre davantage de paramètres.

[Q22] D'abord, écrire une nouvelle fonction (prennant en entrée les valeurs actuelles de x_A , x_B , v_A , v_B et un pas dt) qui fait avancer les positions et les vitesses d'un pas, et renvoie les nouvelles valeurs. Vous pouvez utiliser la trame suivante :

```
def systemeDeuxCorpsAvanceUnPas(xA, xB, vA, vB, dt):  
    # votre code ici  
    nouvelle_xA = [...]  
    nouvelle_vA = [...]  
    nouvelle_xB = [...]  
    nouvelle_vB = [...]  
    return nouvelle_xA, nouvelle_vA, nouvelle_xB, nouvelle_vB
```

[15]: # [votre code ici pour Q22]

[Q23] Choisir les valeurs initiales des paramètres pour que le centre de masse (barycentre) du système soit (i) à l'origine, et (ii) au repos, c'est-à-dire de vitesse nulle. (Ces choix n'ont pas de signification physique, dans un sens fondamental, mais ils rendront plus simple le modèle du système.) Comme auparavant, on veut que la distance Terre-Soleil à $t = 0$ soit d_0 et la vitesse relative à $t = 0$ soit v_0 .

(vos remarques ici sur le choix des valeurs initiales pour Q23)

[16]: # [votre code ici pour Q23, à réutiliser ensuite pour Q25-26]

[Q24] Ensuite, utiliser cette fonction pour calculer l'évolution du système.

Questions à considérer :

- [Q25] Est-ce que le système se comporte différemment du modèle simplifié ci-dessus (d'un Soleil fixe) ? Pourquoi ?
- [Q26] Est-ce que la position et la vitesse du centre de masse sont censées varier au fil du temps ? Est-ce le cas en pratique, selon votre modèle ?

(vos remarques ici pour Q25)

(vos remarques ici pour Q26)

[17]: # [votre code ici pour Q25-26, le cas échéant]

Pour aller plus loin

Ci-dessous, quelques idées qui pourrait vous inspirer. La liste n'est pas exhaustive !

Idées pour la partie 1 :

Appliquer la troisième loi de Kepler à d'autres planètes ; incertitudes

Les données du fichier `nasa-exoplanetes.csv` concernent les planètes suivantes : 11 Com b, 11 UMi b, 14 And b, 14 Her b, 16 Cyg B b, 18 Del b, 24 Boo b, 24 Sex b, 24 Sex c, 2MASS J19383260+4603591 b, 30 Ari B b, 4 UMa b, 42 Dra b, 47 UMa b, 47 UMa c, 51 Peg b, 55 Cnc b, 55 Cnc c.

Vous pouvez tester sa validité avec d'autres planètes de la base de données de la NASA. Attention : elle comprend énormément d'informations sur des milliers de planètes (et parfois plusieurs mesures par planète). Il n'est pas conseillé de traiter un grand nombre de planètes (pour qu'il reste un projet de physique et non d'informatique). Par contre, il pourrait être intéressant de tester la loi de Kepler dans des conditions extrêmes (p.ex. des planètes très massives, ou très loin de leurs astres, ou dans des systèmes à plusieurs astres).

La plupart des mesures pertinentes ont des incertitudes associées. Pouvez-vous les inclure dans votre graphique, et/ou dans votre ajustement ?

Idées pour les parties 2+3 :

La méthode de Verlet

L'algorithme itératif proposé dans la partie 2.4 correspond, en effet, à la méthode d'Euler. Il souffre des imprécisions si le pas dt n'est pas très court. Un autre algorithme, plus précis, est [la méthode de Verlet](#). Au deuxième ordre, on peut l'exprimer sous la forme suivante :

$$\begin{aligned}\vec{v}_B(t + dt) &= \vec{v}_B(t) + \vec{a}_B(t)dt \\ \vec{x}_B(t + dt) &= \vec{x}_B(t) + \left[\frac{\vec{v}_B(t) + \vec{v}_B(t + dt)}{2} \right] dt\end{aligned}$$

Vous pouvez mettre en comparaison la précision des deux méthodes (d'Euler et de Verlet) avec des pas grossiers, raisonnables, et fins. Les résultats, sont-ils cohérents ? Sous quelles conditions les méthodes sont-elles fiables ?

Un modèle plus réaliste du système Terre-Soleil ; vérifications numériques de la troisième loi de Kepler (via la simulation des parties 2-3)

Les valeurs initiales du rayon et de la vitesse de la Terre (proposées dans les sections 2.3 et 2.4) ont été calculées tel qu'on obtienne une orbite quasi-circulaire ; elles correspondent bien aux valeurs expérimentales du demi-grand axe a et de la vitesse *moyenne* de [l'orbite](#). Cependant, dans la réalité l'orbite n'est pas strictement circulaire ; vous pouvez simuler l'orbite avec de valeurs plus précises de l'orbite de la Terre, ou d'une autre planète ayant une excentricité plus importante. Que vaut la période selon votre simulation ? Est-ce que cette valeur valide la troisième loi de Kepler ?

Un système à n corps

Dans la partie 3, nous avons étudié un système à deux corps (le Soleil et la Terre). Pouvez-vous généraliser l'algorithme à plusieurs corps ? (Par exemple : le Soleil plus l'ensemble des planètes du [système solaire](#)).

On garde l'approche itérative de base, mais à chaque itération il faudra calculer toutes les forces par paire (par exemple, la force Soleil-Terre, la force Soleil-Mars, la force Terre-Mars, ...) puis pour chaque corps calculer la somme vectorielle des forces (la résultante), puis en déduire l'accélération, puis mettre à jour la position et la vitesse. L'orbite de la Terre, est-elle sensible à la présence des autres planètes ?

D'autres systèmes

À l'aide de [la base de données NASA](#), vous pouvez simuler l'évolution d'un système astre-exoplanète. Se comporte-il d'une manière similaire à notre système solaire ? Y-a-t'il des différences ?

(Image du planétaire : Birmingham Museums Trust, via [Wikimedia Commons](#))