Devoir Maison 06

Autour de la dynamique gravitationnelle

Modéliser les interactions physiques entre un grand nombre de constituants mène à l'écriture de systèmes différentiels pour lesquels, en dehors de quelques situations particulières, il n'existe aucune solution analytique. Les problèmes de dynamique gravitationnelle et de dynamique moléculaire en sont deux exemples. Afin d'analyser le comportement temporel de tels systèmes, l'informatique peut apporter une aide substantielle en permettant leur simulation numérique.

I) Quelques fonctions utilitaires

Q1. Donner la valeur des expressions Python suivantes :
- `[1,2,3] + [4,5,6]`
- `2 * [1,2,3]`

Q2. Écrire une fonction Python `smul` à deux paramètres, un nombre et une liste de nombres, qui multiplie chaque élément de la liste par le nombre et renvoie une nouvelle liste : `smul(2,[1,2,3])` → `[2,4,6]`.

Q3. Arithmétique de listes

Q3.1) Écrire une fonction Python `vsom` qui prend en paramètres deux listes de nombres de même longueur et qui renvoie une nouvelle liste constituée de la somme terme à terme de ces deux listes : `vsom([1,2,3],[4,5,6])`→`[5,7,9]`.

Q3.2) Écrire une fonction Python `vdif` qui prend en paramètres deux listes de nombres de même longueur et qui renvoie une nouvelle liste constituée de la différence terme à terme de ces deux listes (la première moins la deuxième) :
`vdif([1,2,3],[4,5,6])`→`[-3,-3,-3]`.

II) Étude de schémas numériques

Soient y une fonction de classe C^2 sur $\mathbb{R}$ et t_{min} et t_{max} deux réels tels que $t_{min} < t_{max}$.
On note I l'intervalle $[t_{min}, t_{max}]$.
On s'intéresse à une équation différentielle du second ordre de la forme :

$$\forall t \in I, \quad y''(t) = f\left(y(t)\right) \tag{E1}$$

où f est une fonction donnée, continue sur $\mathbb{R}$.
De nombreux systèmes physiques peuvent être décrits par une équation de ce type. On suppose connues les valeurs $y_0 = y(t_{min})$ et $z_0 = y'(t_{min})$. On suppose également que le système physique étudié est conservatif. Ce qui entraine l'existence d'une quantité indépendante du temps (énergie, quantité de mouvement. . .), notée E, qui vérifie l'équation (E2) où $g' = -f$.

$$\forall t \in I, \quad \frac{1}{2}y'^2(t) + g\left(y(t)\right) = E \tag{E2}$$

Q4. *Mise en forme du problème*

Pour résoudre numériquement l'équation différentielle (E1) on introduit la fonction $z : I \to \mathbb{R}$ définie par : $\forall t \in I,\ z(t) = y'(t)$.

Q4.1) Montrer que l'équation (E1) peut se mettre sous la forme d'un système différentiel du premier ordre en $y(t)$ et $z(t)$, noté $(\mathcal{S})$.

Q4.2) Soit n un entier strictement supérieur à 1 et $J_n = \{0, \ldots, n-1\}$.

On pose $h = \dfrac{t_{max} - t_{min}}{n-1}$ et $\forall i \in J_n,\ t_i = t_{min} + i.h$.

Montrer que, pour tout entier $i \in \{0, \ldots, n-1\}$,

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} z(t)dt \text{ et } z(t_{i+1}) = z(t_i) + \int_{t_i}^{t_{i+1}} f\left(y(t)\right) dt \qquad \text{(E3)}$$

La suite du problème exploite les notations introduites dans cette partie et présente deux méthodes numériques dans lesquelles les intégrales précédentes sont remplacées par une valeur approchée.

Q5. *Schéma d'Euler explicite*

Dans le schéma d'Euler explicite, chaque terme sous le signe intégrale est remplacé par sa valeur prise en la borne inférieure. c'est-à-dire que pour tout $t \in [t_i; t_{i+1}]$, $z(t) = z(t_i)$ et $f\left(y(t)\right) = f\left(y(t_i)\right)$.

Q5.1) Dans ce schéma, montrer que les équations (E3) permettent de définir deux suites $(y_i)_{i \in J_n}$ et $(z_i)_{i \in J_n}$ où y_i et z_i sont des valeurs approchées de $y(t_i)$ et $z(t_i)$. Donner les relations de récurrence permettant de déterminer les valeurs de y_i et z_i connaissant y_0 et z_0.

Q5.2) Écrire une fonction `euler` qui reçoit en argument les paramètres qui vous semblent pertinents et qui renvoie deux listes de nombres correspondant aux valeurs associées aux suites $(y_i)_{i \in J_n}$ et $(z_i)_{i \in J_n}$.

Vous justifierez le choix des paramètres transmis à la fonction.

Q5.3) Pour illustrer cette méthode, on considère l'équation différentielle

$$\forall t \in I,\ y''(t) = -\omega^2 y(t)$$

dans laquelle ω est un nombre réel strictement positif.

Q5.3.a) Montrer que $E = \frac{1}{2}y'^2(t) + \dfrac{\omega^2}{2}y^2(t)$ est indépendante de t et vérifie une équation de la forme (E2).

Q5.3.b) On note E_i la valeur approchée de E à l'instant t_i, $i \in J_n$, calculée en utilisant les valeurs approchées de $y(t_i)$ et $z(t_i)$ obtenues à la question Q5.1. Montrer que $E_{i+1} - E_i = h^2\omega^2 E_i$.

Q5.3.c) Qu'aurait donné un schéma numérique qui satisfait à la conservation de E ?

Q5.3.d) Pour $\omega = 1$, en portant les valeurs de y_i et z_i sur l'axe des abscisses et l'axe des ordonnées respectivement, quelle serait l'allure d'un graphe qui respecte la conservation de E ?

Q5.3.e) La mise en œuvre de la méthode d'Euler explicite génère le résultat graphique donné Figure 1 à gauche. Dans un système d'unités adapté, les calculs ont été menés en prenant

$$y_0 = 3, z_0 = 0, t_{min} = 0, t_max = 3, \omega = 1 \text{ et } n = 100$$

En quoi ce graphe confirme-t-il que le schéma d'Euler explicite ne conserve pas E ? Pouvez-vous justifier son allure ?

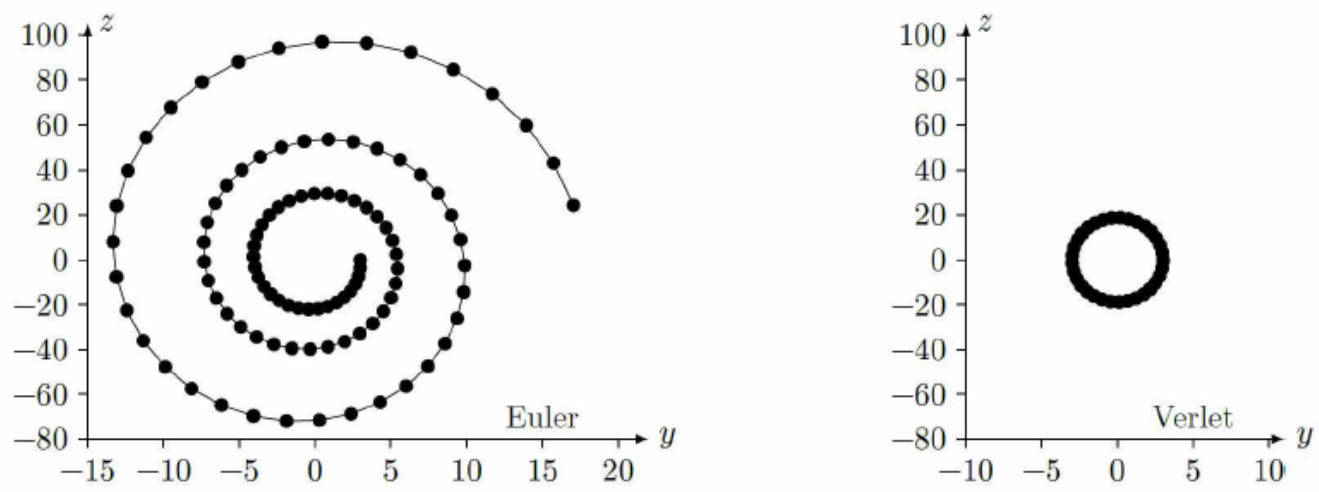

Figure 1

Q6. *Schéma de Verlet*

Le physicien français Loup Verlet a proposé en 1967 un schéma numérique d'intégration d'une équation de la forme (E1) dans lequel, en notant $f_i = f(y_i)$ et $f_{i+1} = f(y_{i+1})$, les relations de récurrence s'écrivent

$$y_{i+1} = y_i + hz_i + \frac{h^2}{2} f_i \text{ et } z_{i+1} = z_i + \frac{h}{2}(f_i + f_{i+1})$$

Q6.1) Écrire une fonction `verlet` qui reçoit en argument les paramètres qui vous semblent pertinents et qui renvoie deux listes de nombres correspondant aux valeurs associées aux suites $(y_i)_{i\in J_n}$ et $(z_i)_{i\in J_n}$.

Q6.2) On reprend l'exemple de l'oscillateur harmonique (question Q5.3) et on compare les résultats obtenus à l'aide des schémas d'Euler et de Verlet.

Q6.2.a) Montrer que dans le schéma de Verlet, on a $E_{i+1} - E_i = O(h^3)$.

Q6.2.b) La mise en œuvre du schéma de Verlet avec les mêmes paramètres que ceux utilisés au Q5.3.e donne le résultat de la Figure 1 à droite. Interpréter l'allure de ce graphe.

Q6.2.c) Que peut-on conclure sur le schéma de Verlet ?

III) Exploitation d'une base de données

À partir de mesures régulièrement effectuées par différents observatoires, une base de données des caractéristiques et des états des corps célestes de notre Système solaire est maintenue à jour.

L'objectif de cette partie est d'extraire de cette base de données les informations nécessaires à la mise en œuvre des fonctions développées dans les parties précédentes, puis de les utiliser pour prévoir les positions futures des différentes planètes.

Les données à extraire sont les masses des corps étudiés et leurs états (position et vitesse) à l'instant t_{min} du début de la simulation.

Une version simplifiée, réduite à deux tables, de la base de données du Système solaire est :

- La table `CORPS` répertorie les corps étudiés, elle contient les colonnes :
  - `id_corps` (clé primaire) entier identifiant chaque corps ;
  - `nom`, chaine de caractères, désigne le nom usuel du corps ;
  - `masse` de type flottant, contient la masse du corps.
- La table `ETAT` rassemble l'historique des états successifs (positions et vitesses) des corps étudiés. Elle est constituée de huit colonnes :
  - `id_corps` de type entier, identifie le corps concerné ;
  - `datem` est la date de la mesure, sous forme d'un entier donnant le nombre de secondes écoulées depuis un instant d'origine ;
  - trois colonnes de type flottant pour les composantes de la position x, y, z ;
  - trois colonnes de type flottant pour les composantes de la vitesse v_x, v_y, v_z.

Les masses sont exprimées en kilogrammes, les distances en unités astronomiques (1 au = $1,5 \cdot 10^{11}$ m) et les vitesses en kilomètres par seconde. Le référentiel utilisé pour exprimer les composantes des positions et des vitesses est galiléen, orthonormé et son centre est situé à proximité du Soleil.

Q7. Écrire une requête SQL qui renvoie la liste des masses de tous les corps étudiés.

Q8. Les états des différents corps ne sont pas forcément tous déterminés exactement au même instant. Nous allons assimiler l'état initial (à la date t_{min}) de chaque corps à son dernier état connu antérieur à t_{min}.

Dans toute la suite, on supposera que la valeur de t_{min}, sous le format utilisé dans la table `ETAT`, est accessible à toute requête SQL via l'expression `tmin()`.

Par exemple, pour afficher le nombre corps mesuré à t_{min}, la requête est

```
SELECT count(*) FROM ETAT WHERE datem=tmin();
```

Q10.1) On souhaite vérifier que tous les corps étudiés disposent d'un état connu antérieur à `tmin()`.

Écrire une requête SQL qui renvoie le nombre de corps qui ont au moins un état connu antérieur à `tmin()`

Q10.2) Écrire une requête SQL qui renvoie, pour chaque corps, son identifiant et la date de son dernier état antérieur à `tmin()`.

Q10.3) Le résultat de la requête précédente est stocké dans une nouvelle table `date_mesure` à deux colonnes :

- `id_corps` de type entier, contient l'identifiant du corps considéré ;
- `date_der` de type entier, correspond à la date du dernier état connu du corps considéré, antérieur à `tmin()`.

Pour simplifier la simulation, on décide de négliger l'influence des corps ayant une masse strictement inférieure à une valeur fixée `masse_min()` et de ne s'intéresser qu'aux corps situés dans un cube, centré sur l'origine du référentiel de référence et d'arête `arete()` donnée. Les faces de ce cube sont parallèles aux plans formés par les axes du référentiel de référence.

Écrire une requête SQL qui renvoie la masse et l'état initial (sous la forme masse, x, y, z, v_x, v_y, v_z) de chaque corps retenu pour participer à la simulation. Classez les corps dans l'ordre croissant par rapport à leur distance à l'origine du référentiel.

Q1. Sur les listes Python, l'opérateur + effectue la concatenation (en creant une troisieme liste, d'où son inefficacité par rapport à append).

[1,2,3] + [4,5,6] renvoie [1,2,3,4,5,6].

De meme la multiplication d'une liste par un entier effectué la concatenation du nombre demande d'exemplaires de ladite liste.

2*[1,2,3] renvoie [1,2,3,1,2,3].

Ces opérations ne correspondent donc pas a ce que l'on souhaiterait pour operer sur les vecteurs ! Pour cela il est conseille d'utiliser les tableaux numpy... mais ce sujet incite à tout faire avec des listes... Il faut respecter l'esprit de l'enonce !

Q2. On peut contruire la nouvelle liste à l'aide d'une boucle et de append, qui donne le code :

```
def smul(k,L):
    res=[]
    for x in L:
        res.append(k*x)
    return res
```

Attention ! On pourrait aussi modifier en mémoire les éléments de L par des instructions du genre L[i]=k*L[i], mais l'énonce demande explicitement de renvoyer une nouvelle liste, ce qui suppose (en creux) qu'il ne faut pas modifier L...

Q3. 1. Même principe. . .

```
def vsom(L1,L2):
    res=[]
    for i in range(len(L1):
        res.append(L1[i]+L2[i])
    return res
```

N.B. : inutile ici de gérer les erreurs (avec par exemple tester len(L1)==len(L2)), l'énonce a pris la peine de le preciser !

2. Idem. . . On pourrait aussi composer les deux fonctions precedentes, mais ce serait peu efficace !

```
def vdif(L1,L2):
    res=[]
    for i in range(len(L1):
        res.append(L1[i]-L2[i])
    return res
```

Q4. 1. Classiquement, l'équation scalaire d'ordre 2 est equivalente au systeme d'ordre 1

$$\forall t \in I \begin{cases} y'(t) &= z(t) \\ z'(t) &= f\left(y(t)\right). \end{cases}$$

2. On imagine que l'énonce suppose (implicitement !) que y est solution de (E1). En particulier y et z sont C^1 sur I et, pour $i \in \{0, \ldots, n-2\}$, On primitive sur l'intervalle

$[t_i, t_{i+1}]$:

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} y'(t)dt \text{ et } z(t_{i+1}) = z(t_i) + \int_{t_i}^{t_{i+1}} z'(t)dt.$$

Autrement dit, y et z vérifiant le systeme de la question précédente :

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} z(t)dt \text{ et } z(t_{i+1}) = z(t_i) + \int_{t_i}^{t_{i+1}} f(y(t))dt.$$

Q5. 1. Selon le principe du schema d'Euler explicite, les integrales ci-dessus sont remplacées respectivement par hz_i et $hf(y_i)$ (puisque $t_{i+1} - t_i = h$ par construction). J'obtiens ainsi la definition par récurrence des suites $(y_i)_{i\in Jn}$ et $(z_i)_{i\in Jn}$:

$$\forall i \in [[0, n-2]] y_{i+1} = y_i + hz_i \text{ et } z_{i+1} = z_i + hf(y_i).$$

2. Il est alors aisé de coder cela. Avec numpy, il faut initialiser les tableaux à la bonne taille, ici on peut construire les listes progressivement à coups de append ! Je choisis comme paramètres la fonction f, les valeurs initiales $y0$ et $z0$, les bornes $tmin$ et $tmax$ et enfin l'entier n. Ce sont les donnees utilisees dans l'enonce. On pourrait utiliser une fonction f definie comme variable globale, mais ce serait plus rigide.

```
def euler(f,y0,z0,tm,tM,n):
    h=(tM-tm)/(n-1)
    y=[y0]
    z=[z0]
    for i in range(n-1):
        y.append(y[-1]+h*z[-1])
        z.append(z[-1]+h*f(y[-1]))
    return y,z
```

Noter que y[-1] et z[-1] sont bien connus lors du passage dans la boucle pour la valeur i (une recurrence immediate prouverait l'invariant de boucle : au debut de l'execution du corps de la boucle pour la valeur i, les listes y et z sont de longueur i+1).

3. a. En multipliant l'équation considérée ici par $y'(t)$, j'obtiens

$$\forall t \in I,\ y'(t)y''(t) - \omega^2 y(t)y'(t) = 0,$$

c'est-a-dire que la fonction $\frac{1}{2}y'^2 - \frac{1}{2}\omega^2 y$ est constante sur l'intervalle I. Donc, en posant $E = \frac{1}{2}z_0^2 + \frac{1}{2}\omega^2 y_0^2$:

$$\forall t \in I,\ \frac{1}{2}y'^2(t) + \frac{\omega^2}{2}y^2(t) = E.$$

b. Avec les notations précédentes, pour $i \in J_n$, $2Ei = z_i^2 + \omega^2 y_i^2$ d'où

$$\begin{aligned} 2E_{i+1} &= \left(z_i + \omega^2 h y_i\right)^2 + \omega^2 (y_i + hz_i)^2 \\ &= z_i^2 + \omega^2 y_i^2 + h^2\omega^2 \left(\omega^2 y_i^2 + z_i^2\right) \\ &= 2E_i \left(1 + h^2\omega^2\right) \end{aligned}$$

soit finalement $E_{i+1} - E_i = h^2\omega^2 E_i$.

c. Un schema respectant la conservation de E verifierait $E_{i+1} - E_i = 0$!!

d. Avec un tel schema, les couples (y_i, z_i) seraient sur le cercle d'équation cartésienne $y^2 + z^2 = 2E$.

e. La forme en spirale montre bien que les points ne sont pas sur un cercle (qui contient au plus deux points d'abscisse donnée, par exemple !). La spirale divergente (le point initial (3, 0) est à l'interieur) correspond bien au resultat du b, ou l'on a vu que l'énergie augmente suivant ce schéma.

Q6. 1. Reprenons le principe (et les parametres !) de Q5.2 :

```
def verlet(f,y0,z0,tm,tM,n):
    h=(tM-tm)/(n-1)
    y=[y0]
    z=[z0]
    for i in range(n-1):
        fi=f(y[i])
        y.append(y[i]+h*(z[i]+h/2*fi))
        z.append(z[i]+h/2*(fi+f(y[i+1])))
    return y,z
```

Je prends soin de calculer une seule fois $f(y_i)$. Une fois la liste y allongee, je dispose de y_{i+1} qui sert pour calculer z_{i+1} selon le schema de Verlet.

2. 1. J'ai ici selon le schema de Verlet, en regroupant tous les termes dominés par h^3,

$$y_{i+1}^2 = y_i^2 + 2hy_iz_i + h^2z_i^2h^2\omega^2y_i^2 + O\left(h^3\right)$$

et

$$\begin{aligned} z_{i+1}^2 &= z_i^2h\omega^2z_i(y_i + y_{i+1}) + h^2\omega^2\frac{1}{4}(y_i + y_{i+1})^2 \\ &= z_i^2 2h\omega^2y_iz_i - h^2\omega^2z_i^2 + h^2\omega^2y_i^2 + O\left(h^3\right) \end{aligned}$$

car

$$(y_i + y_{i+1})^2 = 4y_i^2 + O(h)$$

d'où

$$\begin{aligned} 2E_{i+1} &= z_{i+1}^2 + \omega^2y_{i+1}^2 \\ &= 2E_i + O\left(h^3\right) \end{aligned}$$

soit au final $E_{i+1} - E_i = O\left(h^3\right)$.

2. Cette fois-ci les points (y_i, z_i) semblent bien rester sur un cercle.

3. En conclusion le schéma de Verlet fournit une approximation plus réaliste, sûrement grace à la meilleure conservation de l'énergie.

Q7. L'enonce parle de liste mais il attend vraisemblablement la simple requete :
SELECT masse FROM corps

Q8. 1. Une question où il vaut mieux connaitre le mot clé DISTINCT...

```
SELECT COUNT(*) FROM (SELECT DISTINCT id_corps FROM etat WHERE datem<tmin())
```

Autre version, plus compacte, si l'on sait que DISTINCT peut s'appliquer dans le COUNT :

```
SELECT COUNT(DISTINCT id_corps) FROM etat WHERE datem<tmin()
```

2. Il s'agit ici de déterminer le maximum des datem inferieurs a tmin, cela pour chaque corps, d'ou le GROUP BY.

   ```
   SELECT id_corps,MAX(datem) FROM etat GROUP BY id_corps WHERE datem < tmin()
   ```

   Autre version avec HAVING, ou il faut filtrer les paquets crees par GROUP BY en fonction de la valeur du MAX :

   ```
   SELECT id_corps,MAX(datem) AS m FROM etat GROUP BY id_corps HAVING m < tmin()
   ```

3. Question plus delicate, puisqu'il faut effectuer deux jointures afin de récuperer toutes les données requises pour les corps sélectionnés...

   Inutile de tester la date puisque l'énonce stipule que la table `date_mesure` ne contient qu'un enregistrement pour chaque corps selectionnable.

   ```
   SELECT masse,x,y,z,vx,vy,vz FROM corps AS c JOIN etat AS e
   ON c.id_corps=e.id_corps JOIN date_mesure AS d
   ON e.id_corps=d.id_corps WHERE masse>masse_min() AND
   ABS(x)<arete()/2 AND ABS(y)<arete()/2 AND ABS(z)<arete()/2
   ORDER BY x*x+y*y+z*z
   ```

Q9. Pas de difficulte majeure, j'appelle la fonction `etat_suiv` pour determiner les positions successives. Attention tout de meme aux changements d'unité et à effectuer une copie de pos avant de la modifier !

```
def simulation_verlet(deltat,n):
   pos = [smul(1.5e11,p) for p in p0] #conversion des ua en m
   vit = [smul(1e3,v) for v in v0] #conversion des km/s en m/s
   L = [pos[:]] #initialisation de la liste a renvoyer
   for i in range(n):#nb d'iterations
      pos,vit = etat_suiv(m,pos,vit,deltat) #etat suivant
      L.append([smul(1/1.5e11,p) for p in pos]) #reconversion et ajout
   return L
```