

**Devoir Surveillé n°4 d'Informatique**Samedi 30 Mai 2026 (*Durée : 2 heures*)

Modéliser les interactions gravitationnelles entre un grand nombre de corps mène à l'écriture de systèmes différentiels pour lesquels, en dehors de situations particulières, il n'existe aucune solution analytique. L'informatique peut alors apporter une aide substantielle en permettant de simuler numériquement le comportement temporel de tels systèmes.

L'objet de ce sujet est l'étude de solutions algorithmiques permettant de simuler une dynamique gravitationnelle afin, par exemple, de prédire une éclipse ou le passage d'une comète.

Les programmes doivent être écrits en Python et les requêtes sur les base de données en SQL. Vous pouvez définir et utiliser toute fonction auxiliaire dont vous estimerez avoir besoin. Vous pouvez utiliser librement les fonctions standards de Python et celles de la bibliothèque *math*, mais pas celles de la bibliothèque *numpy* ; ainsi, on utilisera des listes de trois nombres pour représenter les vecteurs de l'espace et pas des tableaux de type *array*.

Lorsque le sujet demande l'écriture d'une fonction, la réponse doit commencer par l'entête de la fonction. D'autre part, si le sujet précise que la fonction prend un paramètre d'un certain type ou qui répond à une certaine condition, la fonction n'a pas à vérifier la conformité de l'argument reçu.

Enfin, n'oubliez pas de faire appel aux fonctions Python écrites dans les parties précédentes !

**Rappel :** quand une fonction doit renvoyer des listes, il peut être utile de commencer par créer des listes de longueur  $n$  via une instruction de la forme  $L=[0]*n$ .

## I – Quelques fonctions utiles

1) Quels sont les résultats des instructions Python suivantes ?

$$(a) \quad [1, 2, 3] + [4, 5, 6]$$

$$(b) \quad 2 * [1, 2, 3]$$

2) Écrire une fonction `smul` qui prend en paramètres un nombre  $x$  et une liste de nombres  $L$ , et qui renvoie la liste obtenue en multipliant chaque élément de la liste  $L$  par le nombre  $x$ .

Ainsi, `smul(2, [1, 2, 3])` doit renvoyer `[2, 4, 6]`.

3) Écrire une fonction `vsom` qui prend en paramètres deux listes de nombres  $L1$  et  $L2$  de même longueur, et qui renvoie la liste obtenue en sommant terme à terme ces deux listes.

Ainsi, `vsom([1, 2, 3], [4, 7, 11])` doit renvoyer `[5, 9, 14]`.

4) Écrire une fonction `vdiff` qui prend en paramètres deux listes de nombres  $L1$  et  $L2$  de même longueur, et qui renvoie la liste obtenue en retranchant terme à terme la liste  $L2$  à la liste  $L1$ .

Ainsi, `vdiff([1, 2, 3], [4, 5, 6])` doit renvoyer `[-3, -3, -3]`.

## II – Étude de schémas numériques

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

$$\forall t \in I \quad y''(t) = f(y(t)) \quad (1)$$

où  $f$  est une fonction continue sur  $\mathbb{R}$ . On suppose que les valeurs  $y_0 = y(t_{\min})$  et  $z_0 = y'(t_{\min})$  sont connues. On suppose également que le système physique étudié est conservatif : ceci entraîne l'existence d'une quantité indépendante du temps notée  $E$  et qui vérifie l'équation

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

où  $g$  est une fonction dérivable telle que  $g' = -f$ .

### 1) Mise en forme du problème

Pour résoudre numériquement l'équation différentielle (1), on définit la fonction  $z = y'$  sur  $I$ .

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

*Indication : il faut commencer par exprimer  $y'$  et  $z'$  en fonction de  $y$  et  $z$ .*

(b) Soit  $n \in \mathbb{N}^*$ . On pose  $h = (t_{\max} - t_{\min}) / n$  et  $t_i = t_{\min} + ih$  pour tout  $i \in \llbracket 0; n \rrbracket$ .

Montrer que pour tout  $i \in \llbracket 0; n - 1 \rrbracket$

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} z(t) dt \quad \text{et} \quad z(t_{i+1}) = z(t_i) + \int_{t_i}^{t_{i+1}} f(y(t)) dt \quad (3)$$

La suite du problème exploite les notations introduites dans cette partie et présente deux méthodes numériques dans lesquelles les intégrales précédentes sont remplacées par des valeurs approchées.

## 2) Schéma d'Euler

Ce schéma est obtenu en remplaçant chaque terme sous le signe intégrale par sa valeur prise en la borne inférieure (c'est en fait la méthode des rectangles à gauche) : ainsi, on remplacera  $z(t)$  par  $z(t_i)$  dans la première intégrale.

- (a) Montrer que les équations (3) permettent de définir deux suites  $(y_i)$  et  $(z_i)$ , où  $y_i$  et  $z_i$  sont des valeurs approchées de  $y(t_i)$  et  $z(t_i)$  pour tout  $i \in \llbracket 0; n \rrbracket$ .

*Indication : ces suites vérifient  $y_{i+1} = y_i + h z_i$  et  $z_{i+1} = z_i + h f(y_i)$  pour tout  $i \in \llbracket 0; n-1 \rrbracket$ .*

- (b) Écrire une fonction `euler` qui reçoit en arguments les paramètres qui vous semblent pertinents et qui renvoie les deux listes  $[y_0, y_1, \dots, y_n]$  et  $[z_0, z_1, \dots, z_n]$ .

Vous justifierez le choix des paramètres transmis à la fonction.

## 3) Un exemple

Pour illustrer cette méthode, on fixe un réel  $\omega$  et l'on considère l'équation différentielle

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

- (a) Montrer que l'on peut définir une quantité  $E$ , indépendante du temps, qui vérifie une équation de la même forme que (2).

*Indication : on peut procéder par calcul direct, ou bien identifier les fonctions  $f$  et  $g$  intervenant dans les relations (1) et (2).*

- (b) Pour tout  $i \in \llbracket 0; n \rrbracket$ , on note  $E_i$  la valeur approchée de  $E$  à l'instant  $t_i$  calculée à l'aide des valeurs approchées  $y_i$  et  $z_i$  obtenues à la question 2.(a).

Montrer, à l'aide des relations de récurrence établies en 2.(a), que  $E_{i+1} - E_i = h^2 \omega^2 E_i$ .

- (c) Qu'aurait donné un schéma numérique respectant la conservation de  $E$  ?

- (d) Quelle forme va prendre ici le portrait de phase ( $z$  en fonction de  $y$ ) ?

## 4) Schéma de Verlet

Pour remédier aux imprécisions du schéma d'Euler, le français Verlet a proposé un schéma numérique d'une équation de la forme (1) dans lequel, en notant  $f_i = f(y_i)$  et  $f_{i+1} = f(y_{i+1})$ , les relations de récurrence satisfaites par les suites  $(y_i)$  et  $(z_i)$  s'écrivent

$$\forall i \in \llbracket 0; n-1 \rrbracket \quad y_{i+1} = y_i + h z_i + \frac{h^2}{2} f_i \quad \text{et} \quad z_{i+1} = z_i + \frac{h}{2} (f_i + f_{i+1}) \quad (4)$$

Écrire alors une fonction `verlet` qui reçoit en arguments les paramètres qui vous semblent pertinents et qui renvoie les deux listes  $[y_0, y_1, \dots, y_n]$  et  $[z_0, z_1, \dots, z_n]$ .

*Indication : c'est une adaptation de la fonction `euler`.*

En reprenant l'exemple de la question 3), on montre que l'on a  $E_{i+1} - E_i = O(h^3)$  avec le schéma de Verlet, qui permet donc une intégration bien plus précise que le schéma d'Euler. Il est plus approprié à l'étude d'un tel système conservatif.

### III – Problème à N corps

Soit  $N \geq 2$  un entier. On s'intéresse à présent à la dynamique gravitationnelle d'un système de  $N$  corps, que l'on assimile à des points matériels  $P_j$  de masses  $m_j$  pour  $j \in \llbracket 0; N-1 \rrbracket$ . Le mouvement de ces points est étudié dans un référentiel galiléen muni d'une base orthonormée. L'action exercée par un corps  $k$  sur un autre corps  $j$  est décrite par la force gravitationnelle

$$\vec{F}_{k/j} = G \frac{m_j m_k}{r_{jk}^3} \vec{P_j P_k} \quad (5)$$

où  $r_{jk} = \|\vec{P_j P_k}\|$  est la distance séparant les deux corps et  $G = 6,67 \times 10^{-11} \text{ N} \cdot \text{m}^2 \cdot \text{kg}^{-2}$  la constante de gravitation universelle.

Soient  $i \in \llbracket 0; n \rrbracket$  et  $j \in \llbracket 0; N-1 \rrbracket$ . À l'instant  $t_i$ , le corps  $j$  est repéré par ses coordonnées cartésiennes  $(x_{ij}, y_{ij}, z_{ij})$  et les composantes de son vecteur vitesse  $(vx_{ij}, vy_{ij}, vz_{ij})$  dans le référentiel choisi. Trois listes vont être utilisées pour représenter ce système en Python.

- `masse` donne les masses de chaque corps : `masse[j] = m_j`
- `position` contient les positions successives de chaque corps : `position[i][j] = [x_{ij}, y_{ij}, z_{ij}]`
- `vitesse` mémorise les vitesses successives de chaque corps : `vitesse[i][j] = [vx_{ij}, vy_{ij}, vz_{ij}]`

Notons que `masse` est de longueur  $N$  tandis que `position` et `vitesse` sont de longueur  $n+1$ .

L'objet de la suite du problème est de construire ces listes en mettant en œuvre l'algorithme de Verlet. On va utiliser les fonctions définies dans la partie I pour manipuler les vecteurs de l'espace.

Par exemple, si `a` et `b` sont les listes des coordonnées de deux points A et B, le vecteur  $\vec{AB}$  se calcule avec l'instruction `vdifff(b,a)` et le vecteur  $3\vec{AB}$  avec `smul(3,vdifff(b,a))`.

#### 1) Position du problème

- (a) Exprimer la force  $\vec{F}_j$  exercée sur le corps  $P_j$  par les  $N-1$  autres corps.
- (b) Écrire une fonction `force2(m1,p1,m2,p2)` qui prend en paramètres :
  - les masses `m1` et `m2` (en kilogrammes) de deux corps ;
  - leurs positions `p1` et `p2` sous forme de listes de trois coordonnées cartésiennes (en mètres)
 et qui renvoie la force exercée par le corps 2 sur le corps 1 sous forme d'une liste de trois composantes (en newtons).

*Indication : on commencera par écrire une fonction `norme(v)` qui renvoie la norme euclidienne d'un vecteur `v` de l'espace, et l'on utilisera la relation (5).*

- (c) Écrire de même une fonction `forceN(j,m,pos)` qui prend en paramètres l'indice `j` d'un corps, la liste `m` des masses des  $N$  corps du système étudié ainsi que la liste `pos` de leurs positions, et qui renvoie la force  $\vec{F}_j$  exercée sur le corps  $j$  par tous les autres corps.

*Indication : utiliser le résultat de la question 1.(a) et la fonction `force2`.*

#### 2) Approche numérique

- (a) Expliciter la structure et la signification de `position[i]` et `vitesse[i]`.

*Dans le schéma de Verlet exposé à la question II.4, les quantités  $y_i$ ,  $z_i$  et  $f_i$  sont des grandeurs vectorielles correspondant à la position, la vitesse et l'accélération du corps considéré à l'instant  $t_i$ . Elles permettent de calculer sa position et sa vitesse à l'instant suivant  $t_{i+1}$ .*

*Dans cette partie, les calculs sont plus compliqués (on ne peut pas réutiliser la fonction `verlet` de la partie II) car il faut calculer les positions et les vitesses de tous les corps à chaque instant. On va donc commencer par créer deux fonctions auxiliaires qui calculent la liste des positions et la liste des vitesses de tous les corps à l'instant  $t_{i+1}$  à partir de celles à l'instant  $t_i$ .*

- (b) Écrire une fonction `pos_suiv(m, pos, vit, h)` qui prend en paramètres la liste `m` des masses (en kg) des  $N$  corps du système étudié, la liste `pos` de leurs positions (en m) à l'instant  $t_i$ , la liste `vit` de leurs vitesses (en  $\text{m} \cdot \text{s}^{-1}$ ) au même instant et le pas d'intégration `h` (en s), et qui renvoie la liste des positions de ces  $N$  corps à l'instant  $t_{i+1}$  calculées en utilisant le schéma de Verlet.

*Indication : l'accélération subie par le corps  $j$  vaut  $\vec{F}_j/m_j$ .*

- (c) Écrire une fonction `vit_suiv(m, pos, pos_nouv, vit, h)` qui prend en paramètres la liste `m` des masses, la liste `pos` des positions à l'instant  $t_i$ , la liste `pos_nouv` des positions à l'instant  $t_{i+1}$ , la liste `vit` des vitesses à l'instant  $t_i$  et le pas d'intégration `h`, et qui renvoie la liste des vitesses des  $N$  corps à l'instant  $t_{i+1}$  calculées en utilisant le schéma de Verlet.

**Remarque :** dans le schéma de Verlet, on a en effet besoin de connaître les anciennes positions et les nouvelles positions pour mettre à jour les vitesses des corps.

- (d) Estimer la complexité temporelle des fonctions `pos_suiv` et `vit_suiv` sous la forme  $\mathcal{O}(N^\alpha)$ .

- (e) On dispose maintenant sous Python des variables globales `masse`, `p0` et `v0` correspondant respectivement à la liste des masses, la liste des positions et la liste des vitesses de nos  $N$  corps à l'instant initial  $t_0 = t_{\min}$ .

*Autrement dit, on a `p0=position[0]` et `v0=vitesse[0]`.*

Écrire une fonction `simulation_verlet(h, n)` prend en paramètres le pas d'intégration en secondes ( $h > 0$ ) et le nombre d'itérations ( $n > 0$ ), et qui renvoie la liste `position` des positions successives des  $N$  corps aux instants  $t_0, t_1, \dots, t_n$ .

## Partie IV – Exploitation d'une base de données

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

L'objectif de cette partie est d'extraire de cette base les informations nécessaires à la mise en œuvre des fonctions développées dans les parties précédentes, afin de prévoir les positions futures des différents corps. 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 (on parlera d'*état initial*).

Nous allons utiliser une version simplifiée, réduite à deux tables, de la BDD du système solaire.

- La table `corps` répertorie les corps étudiés et contient les attributs suivants :
  - `id_corps` (clef primaire), un entier identifiant chaque corps ;
  - `nom`, une chaîne de caractères donnant le nom usuel du corps ;
  - `masse`, un flottant égal à la masse du corps.
- La table `etat` rassemble l'historique des états successifs (positions et vitesses) des corps étudiés. Elle contient les huit attributs suivants :
  - `id_corps`, un entier identifiant le corps concerné ;
  - `date_m` la date de la mesure, sous forme d'un entier donnant le nombre de secondes écoulées depuis un instant d'origine ;
  - `x`, `y`, `z` trois flottants donnant les composantes de la position ;
  - `vx`, `vy`, `vz` trois flottants donnant les composantes de la vitesse.

Les masses sont exprimées en kg, les distances en unités astronomiques ( $1 \text{ u.a.} = 1,5 \times 10^{11} \text{ m}$ ) et les vitesses en  $\text{km} \cdot \text{s}^{-1}$ . Le référentiel utilisé pour exprimer les composantes des positions et des vitesses est galiléen, orthonormé et son origine est située à proximité du Soleil.

- 1) (a) Écrire une requête SQL qui renvoie la liste des masses de tous les corps étudiés.  
 (b) Reprendre la requête précédente, en affichant les noms des corps et en triant les résultats par ordre décroissant de masse.
- 2) Les états des différents corps ne sont pas forcément tous connus au même instant. Nous allons assimiler l'état initial (à la date  $t_{\min}$ ) de chaque corps à son dernier état connu antérieur à  $t_{\min}$  (c'est-à-dire à la plus grande date inférieure 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()`.

- (a) On souhaite d'abord vérifier que tous les corps étudiés disposent d'un état connu antérieur à `tmin()`. Le nombre de corps présents dans la base est obtenu grâce à la requête `SELECT count(*) FROM corps`.

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

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

*Remarque : il faut utiliser une fonction d'agrégation et regrouper les données par corps.*

- 3) La requête précédente produit une nouvelle table `date_mesure` qui contient deux attributs :
  - `id_corps`, un entier identifiant le corps considéré ;
  - `date_der`, un entier correspondant à la date du dernier état connu 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 une sphère centrée sur l'origine du référentiel et de rayon donné `rayon()`.

Écrire une requête SQL qui renvoie la masse et l'état initial de chaque corps retenu pour participer à la simulation. On classera les corps dans l'ordre croissant de leurs distances par rapport à l'origine du référentiel.

*Remarque : les données qui nous intéressent se trouvant dans les trois tables, il faut utiliser deux jointures. En particulier, il faut utiliser la date du dernier état connu de chaque corps pour trouver son état initial.*

- 4) On dispose maintenant sous Python des variables globales `t0`, `p0`, `v0` et `masse` initialisées à partir du résultat de la requête précédente :
  - `t0` est un entier qui donne la date des conditions initiales ; il correspond à  $t_{\min}$  et `tmin()`
  - `p0` est une liste de longueur  $N$  ; chacun de ses éléments est une liste de la forme  $[x, y, z]$  représentant la position initiale d'un corps, en unités astronomiques
  - `v0` est une liste de structure identique, mais elle indique les vitesses initiales des corps considérés, en  $\text{km} \cdot \text{s}^{-1}$
  - `masse` a été définie dans la partie III

Écrire une fonction `simulation_verlet(deltat, n)` prend en paramètres un incrément de temps (c'est le pas) en secondes (`deltat > 0`) et un nombre d'itérations (`n > 0`), et qui renvoie la liste des positions des corps considérés pour chaque instant `t0`, `t0+deltat`, ..., `t0+n*deltat` (c'est en fait la variable `position` définie dans la partie III).

Les calculs seront menés en utilisant le schéma d'intégration de Verlet et le résultat sera fourni en unités astronomiques. On pourra bien entendu utiliser les fonctions Python définies dans les parties I et III. Attention toutefois aux unités utilisées dans les différentes parties.