Devoir d'informatique n° 4 (2 heures)

Ce devoir est constitué d'une partie de l'épreuve d'IPT de Centrale 2015 et d'un petit exercice. L'ordre des exercices ne correspond à aucun critère de difficulté ou de longueur : vous pouvez les traiter dans l'ordre que vous voulez. Veillez à soigner la copie tant pour l'écriture, la propreté que pour la rédaction, la rigueur et l'argumentation. De plus, on prêtera une attention particuliére au respect des alignements et des indentations des séquence d'instructions Python. La calculatrice est interdite.

Vous numéroterez vos copies et ferez apparaître clairement sur la première page le nombre de copies.

Consignes particulières : Lorsque l'on demande d'écrire une fonction classique d'opérations sur les listes ou tout autre objet, et qu'une telle fonction ou méthode existe, vous n'êtes évidemment pas autorisé à l'utiliser

PROBLEME : Centrale 2015

CONCOURS CENTRALE-SUPÉLEC ~~3 heures~~

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

Les programmes doivent être écrits en langage python ~~et les requêtes de base de données en langage SQL~~. Les candidats sont libres de définir et de programmer toute fonction auxiliaire dont ils estiment avoir besoin pour répondre aux questions posées. Ils veilleront dans ce cas à définir précisément, le rôle de chaque fonction introduite, ses paramètres et son résultat. Ils peuvent également utiliser librement les fonctions de la bibliothèque standard Python, en particulier celles du module **math** dont on supposera toutes les fonctions importées.

Lorsque le sujet demande l'écriture d'une fonction python, la réponse doit commencer par l'entête de la fonction (instruction) **def***). 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. La lisibilité des codes produits, tant en python ~~qu'en SQL~~, est un élément important d'appréciation.*

I. Première partie

I.A. Donner la valeur des expressions python suivantes :

I.A.1) `[1, 2, 3] + [4, 5, 6]`

I.A.2) `2*[1, 2, 3]`

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

I.C. Arithmétique des listes

I.C.1) Écrire une fonction python **vsom** qui prend en paramètre 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]`.

I.C.2) Écrire une fonction python **vdif** qui prend en paramètre 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 $\mathscr{C}^2$ 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 :

$$\forall t \in I \qquad y''(t) = f(y(t)) \qquad \text{(II.1)}$$

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 (II.2) où g vérifie $g' = -f$.

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

II.A. ***Mise en forme du problème***

Pour résoudre numériquement l'équation différentielle (II.1), on introduit la fonction $z : I \to \mathbb{R}$ définie par $\forall t \in I, z(t) = y'(t)$.

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

II.A.2) Soit n un entier strictement supérieur à 1 et $J_n = [\![0, n-1]\!]$.

On pose $h = \dfrac{t_{\max} - t_{\min}}{n-1}$ et $\forall i \in J_n, \quad t_i = t_{\min} + ih$. Montrer que, pour tout entier $i \in [\![0, n-2]\!]$,

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} z(t)\mathrm{d}t \quad \text{et} \quad z(t_{i+1}) = z(t_i) + \int_{t_i}^{t_{i+1}} f(y(t))\mathrm{d}t \qquad \text{(II.3)}$$

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

II.B. ***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.

II.B.1) Dans ce schéma, montrer que les équations (II.3) 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.

II.B.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.

II.B.3) Pour illustrer cette méthode, on considère l'équation différentielle $y''(t) = -\omega^2 y(t)$ dans laquelle ω est un nombre réel.

II.B.3)a) Montrer qu'on peut définir une quantité E , indépendante du temps, vérifiant une équation de la forme (II.2).*Indication : prendre pour g la primitive de $-f$ s'annulant en 0*

II.B.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 II.B.1. Montrer que $E_{i+1} - E_i = -h^2\omega^2 E_i$.

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

II.B.3)d) 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 du graphe qui respecte la conservation de E ?

II.B.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 $y_0 = 3, z_0 = 0, t_{\min} = 0, t_{\max} = 3, \omega = 2\pi$ et $n = 100$.

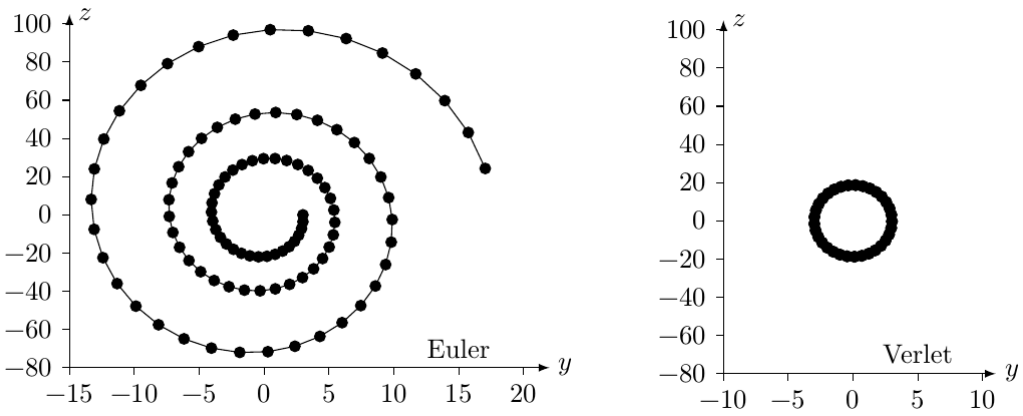


Figure 1

En quoi ce graphe confirme-t-il que le schéma numérique ne conserve pas E ? Pouvez-vous justifier son allure ?

II.C. Schéma de Verlet

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

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

II.C.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 $(z_i)_{i\in J_n}$.

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

II.C.2)a) ~~Montrer que~~ *On supposera pour la suite que* dans le schéma de Verlet, on a $E_{i+1} - E_i = O\left(h^3\right)$.

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

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

III. Problème à N corps

On s'intéresse à présent à la dynamique d'un système de N corps massifs en interaction gravitationnelle. Dans la suite, les corps considérés sont assimilés à des points matériels P_j de masses m_j où $j \in [\![0, N-1]\!]$, $\quad N \geqslant 2$ étant un entier positif donné. Le mouvement de ces points est étudié dans un référentiel galiléen muni d'une base orthonormée. L'interaction entre deux corps j et k est modélisée par la force gravitationnelle. L'action exercée par le corps k sur le corps j est décrite par la force F $\vec{F}_{k/j} = G\dfrac{m_j m_k}{r_{jk}^3}\overrightarrow{P_jP_k}$ où r_{jk} est la distance séparant les corps j et k ($r_{jk} = \left\|\overrightarrow{P_jP_k}\right\|$) et $G = 6,67 \times 10^{-11}\text{N}\cdot\text{m}^2\cdot\text{kg}^{-2}$ la constante de gravitation universelle.

À tout instant t_i avec $i \in [\![0, n]\!]$, chaque corps de masse m_i est repéré par ses coordonnées cartésiennes (x_{ij}, y_{ij}, z_{ij}) et les composantes de son vecteur vitesse $(v_{xij}, v_{xij}, v_{xij})$ dans le référentiel de référence.

Trois listes sont utilisées pour représenter ce système en python :

- **masse** conserve 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]` $= (v_{xij}, v_{xij}, v_{xij})$.

L'objet de la suite du problème est de construire ces listes en mettant en œuvre l'algorithme de Verlet.

III.A. ***Position du problème***

III.A.1) Exprimer la force $\overrightarrow{F_j}$ exercée sur le corps P_j par l'ensemble des autres corps P_j, avec $k \neq j$.

III.A.2) Écrire une fonction python `distance(p1, p2)` qui prend en paramètre les positions (`p1` et `p2`, sous forme de listes de trois coordonnées cartésiennes en mètres) de deux corps 1 et 2 et qui renvoie la distance entre ces deux corps.

III.A.3) Écrire une fonction python `force2(m1, p1, m2, p2)` qui prend en paramètre les masses (`m1` et `m2` en kilogrammes) et les positions (`p1` et `p2`, sous forme de listes de trois coordonnées cartésiennes en mètres) de deux corps 1 et 2 et qui renvoie la valeur de la force exercée par le corps 2 sur le corps 1, sous la forme d'une liste à trois éléments représentant les composantes de la force dans la base de référence, en newtons.

III.A.4) Écrire une fonction `forceN(j, m, pos)` qui prend en paramètre l'indice `j` d'un corps, la liste des masses des N corps du système étudié ainsi que la liste de leurs positions et qui renvoie $\vec{F}_j$, la force exercée par tous les autres corps sur le corps `j`, sous la forme d'une liste de ses trois composantes cartésiennes.

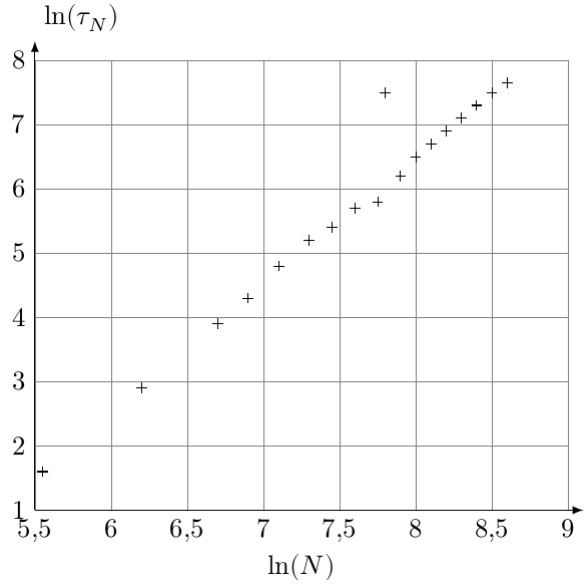
III.B. ***Approche numérique***

III.B.1) Expliciter la structure et la signification de `position[i]` et `vitesse[i]`.

III.B.2) Écrire une fonction `pos_suiv(m, pos, vit, h)` qui prend en paramètres la liste des masses des N corps du système étudié (en kilogrammes), la liste de leurs positions (en mètres) à l'instant t_i, la liste de leurs vitesses (en mètres par seconde) au même instant et le pas d'intégration h (en secondes) et qui renvoie la liste des positions des N corps à l'instant t_{i+1} calculées en utilisant le schéma de Verlet.

III.B.3) Écrire une fonction `etat_suiv(m, pos, vit, h)` qui prend les mêmes paramètres que la fonction `pos_suiv` et qui renvoie la liste des positions (en mètres) et la liste des vitesses (en m/s) des N corps à l'instant t_{i+1} calculées en utilisant le schéma de Verlet.

III.B.4) En notant τ_N la durée des calculs pour un nombre N de corps, la mise en œuvre de la fonction `etat_suiv` a donné le résultat graphique de la figure 2 où on a porté $\ln(N)$ en abscisse et $\ln(\tau_N)$ en ordonnée.



III.B.4)a) Quelle relation simple peut-on établir entre $\ln(\tau_N)$ et $\ln(N)$ à partir de la figure 2 ?

III.B.4)b) Quelle hypothèse peut-on émettre quant à la complexité de l'algorithme étudié ?

III.B.5) Complexité

III.B.5)a) Estimer la complexité temporelle de la fonction `etat_suiv` sous la forme $O(N^\alpha)$.

III.B.5)b) Comparer avec le résultat obtenu à la question III.B.4.

Exercice : Recherche d'une solution approchée à une équation

On considère la fonction définie sur $\mathbb{R}$ par : $f(x) = x^3 + 3x^2 - 6x + 2$

1. Montrer que l'équation $f(x) = 0$ possède une et une seule solution sur $]0, 1[$.
2. Pour déterminer une valeur approchée de cette solution, on va utiliser la méthode Newton en partant de $u = 0$.
   (a) Tracer sur un même graphe, la courbe de f sur $[0, 1]$ ainsi que le graphe permettant de déterminer le premier terme obtenu par la méthode de Newton.
   (b) Ecrire une séquence d'instructions permettant de calculer les 10 premiers termes obtenus par la méthode de Newton.
   (c) Déterminer sous forme de fractions (puis en donnant des valeurs décimales approchées) les deux premiers termes de la suite obtenue par la méthode de Newton appliquée à la fonction f en démarrant à 0.

CORRECTION

PROBLEME : Centrale 2015

I. Première partie

A. Donner la valeur des expressions python suivantes :

1) `[1, 2, 3] + [4, 5, 6]` donne **[1, 2, 3, 4, 5, 6]**

2) `2*[1, 2, 3]` donne **[1, 2, 3, 1, 2, 3]**

B.

```
>>> def smul(a, L):
>>>         return( [a * x for x in L])
```

C. Arithmétique des listes

1)

```
>>> def vsom(L1, L2):
>>>   res = [0] * len(L1)
>>>   for k in range(len(L1):
>>>     res[k] = L1[k] + L2[k]
>>>   return(res)
```

2)

```
>>> def vdif(L1, L2):
>>>   res = [0] * len(L1)
>>>   for k in range(len(L1):
>>>     res[k] = L1[k] - L2[k]
>>>   return(res)
```

II. Étude de schémas numériques

A. ***Mise en forme du problème***

1) $$y''(t) = f(y(t)) \Longleftrightarrow \begin{cases} y'(t) = z(t) \\ z'(t) = f(y(t)) \end{cases} \quad (S)$$

2) Soit $n > 1$ un entier et $J_n = [\![0, n-1]\!]$, $h = \dfrac{t_{\max} - t_{\min}}{n-1}$ et $\forall i \in J_n, \quad t_i = t_{\min} + ih$. y est une primitive de z et z est une primitive de $f \circ y$ donc : $\forall i \in [\![0, n-2]\!]$,

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} z(t)\mathrm{d}t \quad \textbf{et} \quad z(t_{i+1}) = z(t_i) + \int_{t_i}^{t_{i+1}} f(y(t))\mathrm{d}t$$

B. ***Schéma d'Euler explicite***

Dans ce schéma, chaque terme sous l'intégrale est remplacé par sa valeur prise en la borne inférieure.

1) Les équations (II.3) permettent de définir les 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 $z(t_i)$ par les relations de récurrence : $\forall i \in [\![0, n-2]\!], \begin{cases} y_{i+1} = y_i + hz_i \\ z_{i+1} = z_i + hf(y_i) \end{cases}$.

Ainsi connaissant y_0 et z_0, on peut calculer tous les termes des suites $(y_i)_{i\in J_n}$ et $(z_i)_{i\in J_n}$.

2)

```
>>> def euler(f, n, h, y0, z0):
>>>         y, z, Ly, Lz = y0 , z0, [y0], [z0]
>>>         for i in range(n-1):
>>>                 fy = f(y)
>>>                 y = y + h * z
>>>                 z = z + h * fy
>>>                 Ly.append(y) ; Lz.append(z)
>>>         return(Ly, Lz)
```

3) On considère l'équation différentielle $y''(t) = -\omega^2 y(t)$ où $\omega \in \mathbb{R}$.

a) Ici la fonction f est la fonction $f : y \to -\omega^2 y$. On peut choisir g, primitive de f, sous la forme : $g : y \to \dfrac{\omega^2}{2}y^2$. Ici, la résolution littérale de l'équation différentielle, donne : $y(t) = A\cos(\omega t + \phi)$. Aussi :
$E = \dfrac{1}{2}y'^2(t) + g(y(t)) = \dfrac{A^2\omega^2}{2}\cos^2(\omega t + \phi) + \dfrac{A^2\omega^2}{2}\sin^2(\omega t + \phi) = \dfrac{A^2\omega^2}{2}$ qui est bien une constante.

b) La valeur approchée de E proposée est : $E_i = \dfrac{1}{2}z_i^2 + g(y_i) = \dfrac{1}{2}z_i^2 + \dfrac{\omega^2}{2}y_i^2$. On a :
$2(E_{i+1} - E_i) = z_{i+1}^2 - z_i^2 + \omega^2 y_{i+1}^2 - \omega^2 y_i^2 = (z_i - \omega^2 h y_i)^2 + \omega^2(y_i + h z_i)^2 - z_i^2 - \omega^2 y_i^2$
Donc $2(E_{i+1} - E_i) = -2\omega^2 z_i h y_i + h^2\omega^4 y_i^2 + 2\omega^2 y_i z_i h + h^2\omega^2 z_i^2 = 2h^2\omega^2(z_i^2 + \omega^2 y_i^2)$
Donc : $\boxed{E_{i+1} - E_i = h^2\omega^2 E_i}$.

c) Un schéma numérique qui satisfait à la conservation de E aurait donné : $\boxed{E_{i+1} - E_i = 0}$.

d) Si on avait eu la conservation de E, on aurait : $z_i^2 + \omega^2 y_i^2 = 2E = Cte$. Le graphe correspondant est **une ellipse** (ou un cercle en modifiant l'échelle des abscisses ou des ordonnées).

e) Dans le graphe de gauche de la figure 1, on a une spirale qui part vers l'extérieur. Pour un y donné, les valeurs de $|z|$ correspondant à un point de cette spirale croissent ce qui induit un accroissement de la quantité $\dfrac{1}{2}z^2 + \dfrac{\omega^2}{2}y^2$ donc un accroissement de l'énergie E

C. Schéma de Verlet

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

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

1)

```
>>> def verlet(f, n, h, y0, z0):
>>>         y, z, Ly, Lz, FI = y0 , z0, [y0], [z0], f(y)
>>>         for i in range(n-1):
>>>                 y = y + h * z + FI*h*h/2
>>>                 GI = f(y)
>>>                 z = z + (FI + GI)*h/2
>>>                 Ly.append(y) ; Lz.append(z)
>>>                 FI = GI
>>>         return(Ly, Lz)
```

2) On reprend l'exemple de la question II.B.3).

a) On a $E_i = \dfrac{1}{2}z_i^2 + g(y_i) = \dfrac{1}{2}z_i^2 + \dfrac{\omega^2}{2}y_i^2$ donc $2(E_{i+1} - E_i) = (z_{i+1}^2 - z_i^2) + \omega^2(y_{i+1}^2 - y_i^2)$. Or :
$f_i = -\omega^2 y_i$ et $f_{i+1} = -\omega^2 y_{i+1}$
$z_{i+1}^2 - z_i^2 = (z_{i+1} - z_i)(z_{i+1} + z_i) = -h\omega^2(2z_i - h\omega^2 y_i)\left(y_i + z_i\dfrac{h}{2}\right) + O(h^3)$
$y_{i+1}^2 - y_i^2 = (y_{i+1} - y_i)(y_{i+1} + y_i) = h\omega^2(2y_i + h z_i)\left(z_i - y_i\dfrac{h\omega^2}{2}\right) + O(h^3)$

Donc on a $E_{i+1} - E_i = O(h^3)$.

b) On observe une trajectoire fermée (à la précision de la représentation graphique) : le schéma semble conserver l'énergie

c) Le schéma de Verlet permet une intégration plus précise que le schéma d'Euler pour les problèmes où la dérivée seconde y'' de la position ne dépend que de la position y et non de la vitesse z. On constate d'ailleurs que le schéma d'Euler correspond à une *linéarisation ou DL à l'ordre 1* du schéma de Verlet.

III. Problème à N corps

On s'intéresse à présent à la dynamique d'un système de N corps massifs en interaction gravitationnelle. Dans la suite, les corps considérés sont assimilés à des points matériels P_j de masses m_j où $j \in [\![0, N-1]\!]$, $N \geqslant 2$ étant un entier positif donné. Le mouvement de ces points est étudié dans un référentiel galiléen muni d'une base orthonormée. L'interaction entre deux corps j et k est modélisée par la force gravitationnelle. L'action exercée par le corps k sur le corps j est décrite par la force F $\vec{F}_{k/j} = G\frac{m_j m_k}{r_{jk}^3}\overrightarrow{P_jP_k}$ où r_{jk} est la distance séparant les corps j et k ($r_{jk} = \left\|\overrightarrow{P_jP_k}\right\|$) et $G = 6,67 \times 10^{-11}\,\mathrm{N}\cdot\mathrm{m}^2\cdot\mathrm{kg}^{-2}$ la constante de gravitation universelle.

À tout instant t_i avec $i \in [\![0, n]\!]$, chaque corps de masse m_i est repéré par ses coordonnées cartésiennes (x_{ij}, y_{ij}, z_{ij}) et les composantes de son vecteur vitesse $(v_{xij}, v_{xij}, v_{xij})$ dans le référentiel de référence.

Trois listes sont utilisées pour représenter ce système en python :

- **masse** conserve 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]` $= (v_{xij}, v_{xij}, v_{xij})$.

L'objet de la suite du problème est de construire ces listes en mettant en œuvre l'algorithme de Verlet.

III.A. *Position du problème*

III.A.1) La force totale s'exerçant sur le corps j est la somme de toutes les forces exercées sur j par tous les autres corps : $\boxed{\overrightarrow{F_j} = \sum_{k\neq j}\overrightarrow{F_{k/j}} = \sum_{k\neq j} G\frac{m_j m_k}{r_{jk}^3}\overrightarrow{P_jP_k}}$

III.A.2)

```
>>> def distance(p1, p2):
>>>         p1p2 = vdif(p1,p2)
>>>         dis = 0
>>>         for i in range(len(p1p2)):
>>>                 dis += p1p2**2
>>>         return(sqrt(dis))
```

III.A.3)

```
>>> def force2(m1, p1, m2, p2):
>>>         r12 = distance(p1,p2)
>>>         pos = vdif(p2, p1)
>>>         G = 6.67e(-11)
>>>         coeff = G * m1 * m2/(r12**3)
>>>         return(smul(coeff,pos))
```

III.A.4)

```
>>> def forceN(j,m,pos):
>>>         force = [0]*len(pos[j])
>>>         for k in range(len(m)):
>>>                 if k != j:
>>>                         F_k_sur_j = force2(m[j], pos[j], m[k], pos[k])
>>>                         force = vsom(force, F_k_sur_j)
>>>         return(force)
```

III.B. *Approche numérique*

III.B.1) `position[i]` et `vitesse[i]` sont des listes de N listes de 3 flottants.
`position[i]` contient les listes des coordonnées des N corps à l'instant t_i
`vitesse[i]` contient les listes des composantes des vitesses des N corps à l'instant t_i

III.B.2)

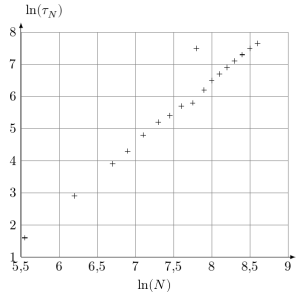
```
>>> def pos_suiv(m, pos, vit, h):
>>>         pos2 = []
>>>         for j in range(len(m)):
>>>                 force = forceN(j, m, pos)
>>>                 newpj = [0]*len(pos[j])
```

```
>>>                 for k in range(len(pos[j])):
>>>                         newpj[k] = pos[j][k] + h*vit[j][k]
    +(h*h/2)*force[k]/m[j]
>>>                 pos2.append(newpj)
>>>         return(pos2)
```

III.B.3)

```
>>> def etat_suiv(m, pos, vit, h):
>>>         newpos = pos_suiv(m, pos, vit, h)
>>>         newvit = []
>>>         for j in range(len(m)):
>>>                 fi = smul(1/m[j], forceN(j,m,pos))
>>>                 fip1 = smul(1/m[j], forceN(j,m,newpos))
>>>                 nextvitj = [0]*len(vit[j])
>>>                 for k in range(len(vit[j])):
>>>                         nextvitj[k] = vit[j][k] + (h/2)*(fi[k] + fip1[k])
>>>                 newvit.append(nextvitj)
>>>         return(newpos, newvit)
```

III.B.4) Si τ_N durée des calculs pour N corps, la mise en œuvre de la fonction `etat_suiv` a donné le résultat graphique de la figure 2 où on a porté $\ln(N)$ en abscisse et $\ln(\tau_N)$ en ordonnée.



a) Le graphe de la figure 2 semble une relation du type $\ln(\tau_N) = \alpha \ln(N) + \beta$. Ainsi on a une relation du type : $\boxed{\tau_N = \tau_1 N^\alpha}$.

b) Par ailleurs il semble que le coefficient α soit "proche" de 2... On peut donc émettre l'hypothèse que la complexité de l'algorithme soit quadratique en nombre de corps

III.B.5) Complexité

a) ☞ Pour `pos_suiv`. On effectue N = len(m) appels à `forceN` (et d'autres calculs et fonctions à temps constant). Or `forceN` est de commplexité $O(N)$ car il a $N-1$ appels à `force2`

☞ Pour `etat_suiv`. On a un appel à `pos_suiv` et 2N = len(m) appels à `forceN` (et d'autres calculs et fonctions à temps constant).

Donc $\boxed{\textbf{la complexité temporelle de la fonction } \texttt{etat\_suiv} \textbf{ est } O\left(N^2\right)}$.

b) Cela correspond....

Exercice : Recherche d'une solution approchée à une équation

On considère la fonction définie sur $\mathbb{R}$ par : $f(x) = x^3 + 3x^2 - 6x + 2$

1. $f'(x) = 3\left(x^2 + 2x - 2\right) = 3\left((x+1)^2 - 3\right)$ qui s'annule et change en un seul point a de $]0,1[$: $a = \sqrt{3} - 1$. Donc f est décroissante sur $[0,a]$ et croissante sur $[a,1]$. Or $f(0) = 2 > 0$, $f(1) = 0$ et $f(a) < 0$ Donc f possède une et une seule solution sur $]0,1[$: il est entre 0 et a.
2. Pour déterminer une valeur approchée de cette solution, on va utiliser la méthode Newton en partant de $u = 0$.

   (a)

   (b)

```
>>> f = lambda x: x**3 + 3*x**2 - 6*x + 2
>>> fp = lambda x: 3*(x**2 + 2*x-2)
>>> u, k = 0, 0
>>> while k<6:
>>>         u, k = u-f(u)/fp(u) , k+1
>>>         print(u)
```

(c) On trouve $u_1 = \frac{1}{3} = 0.33333...$ et $u_2 = \frac{43}{99} = 0.43434343...$