

I – Quelques fonctions utiles

1) Attention, il s'agit ici d'opérations de concaténation !

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

(b) $2 * [1, 2, 3]$ donne $[1, 2, 3, 1, 2, 3]$

2) Le plus simple est d'utiliser une compréhension de listes :

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

3) Même chose, en utilisant la fonction `zip()` pour parcourir les deux listes simultanément :

```
def vsom2(L1, L2):
    return [e1+e2 for (e1, e2) in zip(L1, L2)]
```

4) Il suffit évidemment de reprendre la fonction `vsom` en remplaçant l'addition par une soustraction. On peut aussi utiliser la fonction `smul` pour additionner l'opposé du second vecteur :

```
def vdiff(L1, L2):
    return vsom(L1, smul(-1, L2))
```

Remarque : on pouvait aussi utiliser des listes de zéros de longueur convenable

```
def smul(x, L):
    n=len(L)
    M=[0]*n
    for i in range(n):
        M[i]=x*L[i]
    return M
```

```
def vsom(L1, L2):
    n=len(L1)
    L=[0]*n
    for i in range(n):
        L[i]=L1[i]+L2[i]
    return L
```

II – Étude de schémas numériques

1) **Mise en forme du problème**

(a) Comme $z = y'$, alors $z' = y''$ et l'équation différentielle (1) conduit à $z'(t) = f(y(t))$.
On obtient ainsi le système différentiel du premier ordre suivant :

$$\forall t \in I \quad \begin{cases} y'(t) = z(t) \\ z'(t) = f(y(t)) \end{cases} \quad (S)$$

(b) Soit $i \in \llbracket 0; n-1 \rrbracket$. D'après la question précédente, on a

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

d'où

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

De même,

$$\int_{t_i}^{t_{i+1}} f(y(t)) dt = \int_{t_i}^{t_{i+1}} z'(t) dt = z(t_{i+1}) - z(t_i)$$

si bien que

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

2) Schéma d'Euler

(a) Soit $i \in \llbracket 0; n-1 \rrbracket$. On effectuant l'approximation suggérée par le sujet, on obtient

$$\int_{t_i}^{t_{i+1}} z(t) dt \simeq \int_{t_i}^{t_{i+1}} z(t_i) dt = (t_{i+1} - t_i) z(t_i) = h z(t_i)$$

d'où $y(t_{i+1}) \simeq y(t_i) + h z(t_i)$

et $\int_{t_i}^{t_{i+1}} f(y(t)) dt \simeq \int_{t_i}^{t_{i+1}} f(y(t_i)) dt = (t_{i+1} - t_i) f(y(t_i)) = h f(y(t_i))$

d'où $z(t_{i+1}) \simeq z(t_i) + h f(y(t_i))$

On définit alors les deux suites $(y_i)_{i \in \llbracket 0; n \rrbracket}$ et $(z_i)_{i \in \llbracket 0; n \rrbracket}$ de valeurs approchées en posant

$$\boxed{\forall i \in \llbracket 0; n-1 \rrbracket \quad \begin{cases} y_{i+1} = y_i + h z_i \\ z_{i+1} = z_i + h f(y_i) \end{cases}}$$

(b) La connaissance des valeurs initiales y_0 et z_0 permet ainsi de déterminer de proche en proche les valeurs successives de y_i et z_i . Les calculs ci-dessus faisant également intervenir la fonction f et le pas $h = (t_{\max} - t_{\min})/n$, la fonction euler va donc recevoir en argument les paramètres $y_0, z_0, f, t_{\min}, t_{\max}$ et n .

```
def euler(y0,z0,f,tmin,tmax,n):
    h=(tmax-tmin)/n
    Y=[y0]*(n+1)
    Z=[z0]*(n+1)
    for i in range(n):
        Y[i+1]=Y[i]+h*Z[i]
        Z[i+1]=Z[i]+h*f(Y[i])
    return Y,Z
```

Dans cette fonction, on crée deux listes Y et Z de longueur $(n+1)$, ce qui permet de recopier la relation de récurrence sans risque d'erreur.

3) Un exemple

(a) C'est un classique toujours de bon goût. L'équation différentielle conduit à

$$\forall t \in I \quad y''(t) + \omega^2 y(t) = 0$$

d'où $\forall t \in I \quad y''(t)y'(t) + \omega^2 y'(t)y(t) = 0$

En intégrant cette relation sur I, on en déduit qu'il existe une constante positive E telle que

$$\boxed{\forall t \in I \quad \frac{y'(t)^2}{2} + \frac{\omega^2 y(t)^2}{2} = C^{\text{te}} = E}$$

Remarque : guidés par l'énoncé et la relation (2), on pouvait noter que $f(y) = -\omega^2 y$ implique que $g(y) = \frac{\omega^2 y^2}{2}$. Ceci nous pousse à considérer la fonction E définie par

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

On vérifie que la dérivée de E est nulle, d'où la constance de cette fonction.

(b) Soit $i \in \llbracket 0; n-1 \rrbracket$. On a alors

$$\begin{aligned}
 2E_{i+1} &= z_{i+1}^2 + \omega^2 y_{i+1}^2 \\
 &= (z_i + h f(y_i))^2 + \omega^2 (y_i + h z_i)^2 \\
 &= (z_i - h \omega^2 y_i)^2 + \omega^2 (y_i + h z_i)^2 \\
 &= z_i^2 - 2h \omega^2 y_i z_i + h^2 \omega^4 y_i^2 + \omega^2 y_i^2 + 2h \omega^2 y_i z_i + h^2 \omega^2 z_i^2 \\
 &= z_i^2 + h^2 \omega^4 y_i^2 + \omega^2 y_i^2 + h^2 \omega^2 z_i^2
 \end{aligned}$$

Or, $2E_i = z_i^2 + \omega^2 y_i^2$

d'où
$$\begin{aligned}
 2(E_{i+1} - E_i) &= h^2 \omega^4 y_i^2 + h^2 \omega^2 z_i^2 \\
 &= h^2 \omega^2 (z_i^2 + \omega^2 y_i^2) \\
 &= 2h^2 \omega^2 E_i
 \end{aligned}$$

si bien que

$$\forall i \in \llbracket 0; n-1 \rrbracket \quad E_{i+1} - E_i = h^2 \omega^2 E_i$$

(c) Un tel schéma aurait dû donner $E_{i+1} - E_i = 0$ puisque E_i reste constante.

(d) La quantité E_i étant forcément strictement positive, la relation précédente montre que $E_{i+1} - E_i > 0$ et donc que E_i augmente. Ainsi, le point de coordonnées (y_i, z_i) va s'éloigner de l'origine. Par conséquent, le portrait de phase va nous donner, au lieu d'une belle ellipse qui se referme, une affreuse spirale qui part vers l'extérieur. Horreur !

4) Schéma de Verlet

On retrouve les mêmes arguments et les mêmes idées que pour la fonction euler car le schéma de Verlet est similaire à celui d'Euler.

```

def verlet(y0,z0,f,tmin,tmax,n):
    h=(tmax-tmin)/n
    Y=[y0]*(n+1)
    Z=[z0]*(n+1)
    for i in range(n):
        Y[i+1]=Y[i]+h*Z[i]+0.5*h*h*f(Y[i])
        Z[i+1]=Z[i]+0.5*h*(f(Y[i])+f(Y[i+1]))
    return Y,Z

```

III – Problème à N corps

1) Position du problème

- (a) La force totale s'exerçant sur le corps P_j est la somme de toutes les forces exercées sur celui-ci par les corps P_k avec $k \neq j$, si bien que

$$\vec{F}_j = \sum_{\substack{k=0 \\ k \neq j}}^{N-1} \vec{F}_{k/j} = \sum_{\substack{k=0 \\ k \neq j}}^{N-1} G \frac{m_j m_k}{r_{jk}^3} \vec{P}_j \vec{P}_k$$

- (b) Commençons par écrire une fonction qui renvoie la norme euclidienne d'un vecteur de $\mathbb{R}^3$.

```
def norme(v):
    (a,b,c)=v
    return (a**2+b**2+c**2)**0.5
```

Avec les fonctions de la partie I, on dispose maintenant de tous les outils permettant de manipuler les vecteurs de l'espace sous forme de listes de trois nombres.

On n'oublie pas de créer une variable globale pour G.

```
G = 6.67e-11
def force2(m1,p1,m2,p2):
    P1P2=vdiff(p2,p1)
    nbre=G*m1*m2/norme(P1P2)**3
    return smul(nbre,P1P2)
```

- (c) Pour avoir la force totale exercée sur le corps j , on additionne les forces exercées sur lui par tous les autres corps (cf. question 1.(a)) à l'aide d'une boucle qui parcourt ceux-ci.

```
def forceN(j,m,pos):
    N=len(m)
    F=[0,0,0]
    for k in range(N):
        if k!=j:
            F_kj=force2(m[j],pos[j],m[k],pos[k])
            F=vsom(F,F_kj)
    return F
```

2) Approche numérique

- (a) Les objets `position[i]` et `vitesse[i]` sont respectivement la liste des positions et la liste des vitesses de toutes les particules à l'instant t_i . Ce sont donc des listes de N listes de trois éléments (les coordonnées des vecteurs).
- (b) L'accélération du corps j est donnée par $\vec{F}_j/m_j$. Elle correspond à la quantité $f(y)$ dans le schéma de Verlet. On calcule grâce à celui-ci les nouvelles positions des différents corps, qui sont placées dans la liste `pos_nouv`.

```
def pos_suiv(m,pos,vit,h):
    N=len(m)
    pos_nouv=[0]*N
    for j in range(N):
        mj,yj,zj=m[j],pos[j],vit[j]
        fj=smul(1/mj,forceN(j,m,pos))
        dyj=vsom(smul(h,zj),smul(0.5*h*h,fj))
        pos_nouv[j]=vsom(yj,dyj)
    return pos_nouv
```

- (c) La mise à jour des vitesses fait appel aux accélérations obtenues dans les nouvelles positions, d'où la nécessité de l'argument `pos_nouv`. Comme dans la question précédente, on s'inspire des notations de la question II.4 pour plus de clarté.

```
def vit_suiv(m,pos,pos_nouv,vit,h):
    N=len(m)
    vit_nouv=[0]*N
    for j in range(N):
        mj,zj=m[j],vit[j]
        fj=smul(1/mj,forceN(j,m,pos))
        fj_nouv=smul(1/mj,forceN(j,m,pos_nouv))
        dzj=smul(0.5*h,vsom(fj,fj_nouv))
        vit_nouv[j]=vsom(zj,dzj)
    return vit_nouv
```

- (d) Tous les appels aux fonctions `vsom`, `vdiff`, `smul` et `norme` se font en temps constant car elles s'appliquent à des liste de longueur 3. Par conséquent :

- La fonction `forceN` a une complexité en $\mathcal{O}(N)$ du fait de la boucle `for`.
- La fonction `pos_suiv`, qui l'appelle N fois, a une complexité en $\mathcal{O}(N^2)$.
- La fonction `vit_suiv` a de même une complexité en $\mathcal{O}(N^2)$.

- (e) On écrit une boucle qui effectue les n itérations voulues et renvoie la liste `position` des positions des corps aux $n + 1$ instants $(t_i)_{i \in \llbracket 0;n \rrbracket}$.

```
def simulation_verlet(h,n):
    position=[p0]*(n+1)
    vitesse=[v0]*(n+1)
    for i in range(n):
        position[i+1]=pos_suiv(masse,position[i],vitesse[i],h)
        vitesse[i+1]=vit_suiv(masse,position[i],position[i+1],vitesse[i],h)
    return position
```

Partie IV – Exploitation d’une base de données

1) (a) `SELECT masse FROM corps`

(b) `SELECT nom,masse FROM corps ORDER BY masse DESC`

2) (a) Attention, il faut compter le nombre de corps **distincts** satisfaisant la condition requise.

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

(b) Comme le suggère l’énoncé, on va effectuer une agrégation afin de calculer le plus grand (fonction d’agrégation `MAX`) des temps inférieurs à `tmin()` (sélection en amont) pour chaque corps (on regroupe les entrées selon les valeurs de `id_corps`):

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

3) Il faut passer par deux jointures puisque l’attribut `masse` se trouve seulement dans la table `corps`, que les positions et les vitesses se trouvent dans la table `etat`, et qu’il faut utiliser la table `date_mesure` pour trouver la date de l’état initial dans la table `etat`

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

4) On va d’abord créer une fonction auxiliaire `conversion` qui permet de multiplier tous les vecteurs d’une liste de vecteurs `liste` par une constante `k`. Elle va nous servir à exprimer les positions en m et les vitesses en $\text{m} \cdot \text{s}^{-1}$ afin de pouvoir utiliser la fonction `etat_suiv` définie dans la partie III.

```
def conversion(k,liste):
    return [smul(k,element) for element in liste]
```

Ensuite, on commence par exprimer les positions en m et les vitesses en $\text{m} \cdot \text{s}^{-1}$, puis on écrit une boucle globale qui effectue les n itérations voulues et renvoie la liste `position` des positions des corps aux $n + 1$ instants.

Il ne faut pas oublier que la fonction `etat_suiv` renvoie des positions en m, que l’on doit exprimer en unités astronomiques avant de les ajouter à la liste `position`.

```
def simulation_verlet(deltat,n):
    pos=conversion(1.5e11,p0)
    vit=conversion(1e3,v0)
    position=[p0]
    for j in range(n):
        pos,vit=etat_suiv(m,pos,vit,deltat)
        pos_ua=conversion(1/1.5e11,pos)
        position.append(pos_ua)
    return position
```