teaching sciences for innovation

Proposition de corrigé

Concours : Concours Centrale-Supélec

Année : 2015

Filière : MP - PC - PSI - TSI

Épreuve : Informatique

Ceci est une proposition de corrigé des concours de CPGE, réalisée bénévolement par des enseignants de Sciences Industrielles de l'Ingénieur et d'Informatique, membres de l'UPSTI (Union des Professeurs de Sciences et Techniques Industrielles), et publiée sur le site de l'association :

https://www.upsti.fr/espace-etudiants/annales-de-concours

A l'attention des étudiants

Ce document vous apportera des éléments de corrections pour le sujet traité, mais n'est ni un corrigé officiel du concours, ni un corrigé détaillé ou exhaustif de l'épreuve en question.

L'UPSTI ne répondra pas directement aux questions que peuvent soulever ces corrigés : nous vous invitons à vous rapprocher de vos enseignants si vous souhaitez des compléments d'information, et à vous adresser à eux pour nous faire remonter vos éventuelles remarques.

Licence et Copyright

Informez-vous !

Retrouvez plus d'information sur les Sciences de l'Ingénieur, l'orientation, les Grandes Ecoles ainsi que sur les Olympiades de Sciences de l'Ingénieur et sur les Sciences de l'Ingénieur au Féminin sur notre site : www.upsti.fr

L'équipe UPSTI

Élément de corrigé - Informatique - Concours Centrale Supélec 2015

I Quelques fonctions utilitaires

I.A Expressions Python

I.A.1) Concaténation

$[1, 2, 3, 4, 5, 6]$

I.A.2) Concaténation multiple

$[1, 2, 3, 1, 2, 3]$

I.B Fonction smul

```
def smul(n, liste ):
        """
        Multiplie chaque élément de la liste par un nombre et renvoie une nouvelle liste
        """
        liste2 =[]
        for i in liste :
                liste2 .append(i*n)
        return liste2
```

I.C Arithmétique de liste

I.C.1) Fonction vsom

```
def vsom(liste1 , liste2 ):
        """
        Prend en paramètre deux listes de nombres de même longueur et renvoie une nouvelle
        liste constituée de la somme terme àterme de ces deux listes .
        """
        liste3 =[]
        for i in range(len( liste1 )):
                liste3 .append( liste1 [i]+ liste2 [i])
        return liste3
```

I.C.2) Fonction vdif

```
def vdif( liste1 , liste2 ):
        """
        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 .
        """
        liste3 =[]
        for i in range(len( liste1 )):
                liste3 .append( liste1 [i]- liste2 [i])
        return liste3
```

II Étude de schémas numériques

II.A Mise en forme du problème

II.A.1) Détermination de (S)

On a par définition $y'(t) = z(t)$ or y est de classe C^2 donc on peut obtenir la deuxième expression par dérivation.

$$\boxed{(S) \Leftrightarrow \begin{cases} y'(t) = z(t) \\ z'(t) = f\left(y(t)\right) \end{cases}}$$

II.A.2)

D'après la question précédente, on a :

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

De la même manière :

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

II.B Schéma d'Euler explicite

II.B.1)

D'après l'approximation d'Euler, on peut écrire :

$$\begin{cases} \left(t_{k+1} - t_k\right).z(t_k) \approx \int_{t_k}^{t_{k+1}} z(t)dt \\ \left(t_{k+1} - t_k\right).f(y(t_k)) \approx \int_{t_k}^{t_{k+1}} .f(y(t))dt \end{cases}$$

Or

$$\forall k \in [0, n-2]\,,\ h = t_{k+1} - t_k$$

,

ce qui donne

$$\begin{cases} h.z(t_k) \approx \int_{t_k}^{t_{k+1}} z(t)dt \\ h.f(y(t_k)) \approx \int_{t_k}^{t_{k+1}} f(y(t))dt \end{cases}$$

En utilisant II.3 on obtient :

$$\begin{cases} y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} z(t)dt \approx y(t_i) + h.z(t_i) \\ z(t_{i+1}) = z(t_i) + \int_{t_i}^{t_{i+1}} f(y(t))dt \approx z(t_i) + h.f(y(t_i)) \end{cases}$$

On pose respectivement y_i et z_i les valeurs approchées de $y(t_i)$ et $z(t_i)$ on obtient les relations de récurrence demandées :

$$\boxed{\begin{cases} y(t_{i+1}) = y(t_i) + h.z(t_i) \\ z(t_{i+1}) = z(t_i) + h.f(y(i_k)) \end{cases}}$$

II.B.2)

Pour calculer tous les y_i et z_i, il est nécessaire de connaître : f, y_0, z_0, i et h d'après les formules de récurrence précédentes. Pour le pas et le temps d'arrêt, deux choix sont possibles : donner t_{max}, t_{min} et le nombre de points n ou donner h et n.

```
def euler(f,y0,z0,n,h):
    """
    Euler explicite
    """
    valeursy=[y0]
    valeursz=[z0]
    for k in range(n-1):
        y=valeursy[-1]+h*valeursz[-1]
        z=valeursz[-1]+h*f(valeursy[-1])
        valeursy.append(y)
        valeursz.append(z)
    return [valeursy, valeursz]
```

II.B.3)

a) $y''(t) = -\omega^2.y(t)$

En multipliant chaque membre de l'égalité par $y'(t)$, on a :

$y'(t).y''(t) = -\omega^2.y(t).y'(t)$

$\frac{1}{2}\left(2y'(t).y''(t)\right) = (-\omega^2)\frac{1}{2}.\left(2y(t).y'(t)\right)$

donc en intégrant :

$$\exists E \; tq \quad \frac{1}{2}y'(t)^2 + \frac{\omega^2}{2}y(t)^2 = E$$

Ainsi en posant $g : x \mapsto \frac{\omega^2}{2}x^2$ avec $g' = -f$, on obtient :

$$\boxed{\frac{1}{2}y'(t)^2 + g(y(t)) = E}$$

b)

$$E(t_i) = \frac{1}{2}y'(t_i)^2 + g(y(t_i)) = \frac{1}{2}z(t_i)^2 + g(y(t_i))$$

$$E_i = \frac{1}{2}z_i^2 + g(y_i) = \frac{1}{2}z_i^2 + \frac{\omega^2}{2}(y_i)^2$$

$$E_{i+1} = \frac{1}{2}z_{i+1}^2 + \frac{\omega^2}{2}(y_{i+1})^2$$

Soit

$$E_{i+1} - E_i = \frac{1}{2}z_{i+1}^2 + \frac{\omega^2}{2}(y_{i+1})^2 - \frac{1}{2}z_i^2 - \frac{\omega^2}{2}(y_i)^2$$

En utilisant les relations de récurrence :

$$E_{i+1} - E_i = \frac{1}{2}(z_i - h\omega^2 y_i)^2 + \frac{\omega^2}{2}(y_i + hz_i)^2 - \frac{1}{2}z_i^2 - \frac{\omega^2}{2}(y_i)^2$$

Après simplification :

$$E_{i+1} - E_i = \frac{1}{2}h^2\omega^2(z_i^2 + \omega^2 y_i^2)$$

$$\boxed{E_{i+1} - E_i = h^2\omega^2 E_i}$$

c) Pour conserver l'énergie il faut :

$$E_{i+1} - E_i = 0$$

Tous les E_i sont égaux à E or $E_i = \frac{1}{2}(z_i^2 + \omega^2 y_i^2)$ donc $z_i^2 + \omega^2 y_i^2 = constante$.
On obtient l'équation d'une ellipse.
Exemples, circuit RLC avec R nulle (non dissipatif), barre en liaison pivot sans frottement.
d) Allure d'une ellipse centrée en $y_i = z_i = 0$.

e) L'allure n'est pas une ellipse. Il y a divergence des y_i et des z_i. Le système ainsi modélisé n'est pas conservatif en énergie. En calculant le rapport $\frac{E_{i+1}}{E_i}$, on trouve :

$$\frac{E_{i+1}}{E_i} = 1 + h^2\omega^2$$

qui est strictement supérieur à 1. Donc, la raison de la suite est supérieur à 1, donc divergente. Cela signifie que l'énergie n'est pas finie, et donc diverge. Par conséquent, les y_i et les z_i divergent aussi. D'où l'allure de la figure de gauche.

II.C Schéma de Verlet

II.C.1) Fonction verlet

Comme pour la méthode d'Euler, pour calculer tous les y_i et z_i, il est nécessaire de connaître : f, y_0, z_0, i et h.

```
def verlet(f,y0,z0,n,h):
    valeursy =[ y0 ]
    valeursz =[ z0 ]
    h2sur2=h**2/2. # Hors de la boucle pour la complexité
    for k in range ( n-1 ):
        fi =f(valeursy[-1])
        y = y + h * valeursz[ -1]+h2sur2*fi
        z = z + h/2. * (fi +f(y))
```

```
        valeursy . append ( y )
        valeursz . append ( z )
    return [ valeursy , valeursz ]
```

II.C.2) Comparaison Euler/Verlet

a) D'après IIB3b :

$$E_{i+1} - E_i = \frac{1}{2}z_{i+1}^2 + \frac{\omega^2}{2}(y_{i+1})^2 - \frac{1}{2}z_i^2 - \frac{\omega^2}{2}(y_i)^2$$

En utilisant le schéma de Verlet :

$$E_{i+1} - E_i = \frac{1}{2}(z_i + \frac{h}{2}(-\omega^2 y_i - \omega^2(y_i + hz_i - \frac{h^2}{2}\omega^2 y_i))^2 + \frac{\omega^2}{2}(y_i + hz_i - \frac{h^2}{2}\omega^2 y_i)^2 - \frac{1}{2}z_i^2 - \frac{\omega^2}{2}(y_i)^2$$

Après simplification :

$$\begin{aligned} E_{i+1} - E_i = h^3 &\left(\frac{1}{4}\omega^4 z_i y_i - \frac{1}{2}\omega^4 z_i y_i + \frac{1}{2}\omega^4 z_i y_i\right) \\ &+h^4\left(\frac{1}{8}\omega^4 z_i^2 - \frac{1}{4}\omega^6 y_i^2 + \frac{1}{8}\omega^6 y_i^2\right) \\ &+h^5\left(-1/8\omega^6 z_i y_i\right) \\ &+h^6\left(\frac{1}{32}\omega^8 y_i^2\right) \end{aligned}$$

Soit :

$$\boxed{E_{i+1} - E_i = O(h^3)}$$

b) L'allure des positions est une ellipse. On à donc l'impression que l'énergie est conservée. Cependant le calcul précédent ne donne pas comme résultat la conservation, mais plutôt une faible variation en fonction du pas. Cette variation étant faible elle n'est pas visible sur la figure tracée.

c) Le schéma de Verlet semble plus stable. La formule de récurrence sur z_{i+1} fait apparaitre une prise d'information au pas $i+1$ pour f. On se rapproche donc des avantages d'un schéma implicite.

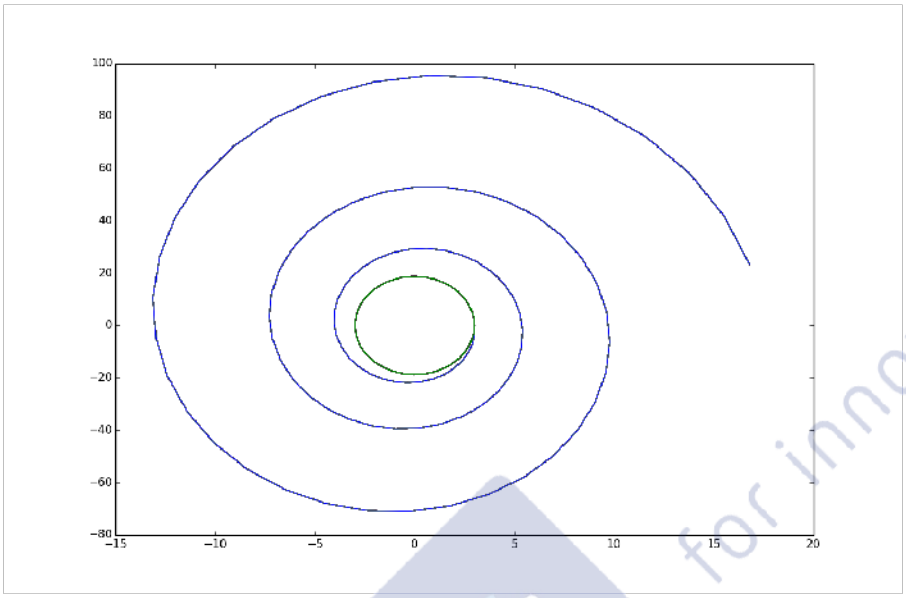
Complément sur la méthode de tracé et résultats :

```
y0=3
z0=0
tmin=0
i=100
h=3./i

def f(x):
    omega=2*pi
    return - omega**2*x


Y1,Z1=euler(f,y0,z0,tmin,i,h)
Y2,Z2=verlet(f,y0,z0,tmin,i,h)

import matplotlib.pyplot as plt
plt.figure(1)
plt.plot(Y1,Z1)
plt.plot(Y2,Z2)
plt.show()
```



Euler en bleu, Verlet en vert

III Problème à N corps

III.A Position du problème

III.A.1) Expression de Fj

Immédiat :

$$\boxed{\overrightarrow{F_j} = \sum_{k \neq j}^{n} G \frac{m_j m_k}{r_{jk}^3} \overrightarrow{PjPk}}$$

III.A.2) Fonction force2

```
def force2(m1,p1,m2,p2):
        """
        Renvoie la valeur de la force exercée par le corps 2 sur le corps 1
        """
        G = 6.67E-11        # Constante de gravitation universelle
        p1p2 = vdif(p2,p1)  # Vecteur P1P2
        facteur = G*m1*m2/(norme(p1p2))**3 # Calcul de G*mj*mk/r^3
        return smul(facteur,p1p2)          # Renvoi de l'expression finale


def norme(vecteur):
        """
        Calcul de la norme d'un vecteur de dimension n
        """
        somme = 0;
        for i in range(len(vecteur)):
                somme += (vecteur[i])**2
        return math.sqrt(somme)
```

III.A.3) Fonction forceN

```
def forceN(j, m, pos):
        """
        Renvoie la force exercée par tous les autres corps sur le corps j
        """
        force = [0,0,0]      # Déclaration du vecteur force
        for k in range(len(pos)):
                if k != j:
                        force = vsom(force,force2(m[j], pos[j], m[k],pos[k]))
        return force
```

III.B Approche numérique

III.B.1) Étude de position[i] et vitesse[i]

- `position[i]` : liste de N listes donnant les coordonnées cartésiennes (liste de trois valeurs) de chacun des corps à l'instant t_i ;
- `vitesse[i]` : liste de N listes donnant les coordonnées cartésiennes du vecteur vitesse de chacun des points P_j à l'instant t_i.

III.B.2) Fonction pos_suiv

On pose : $\overrightarrow{q_j} = \overrightarrow{OP_j}$

En appliquant le principe fondamental de la dynamique à un corps $j \in N$, on trouve :

$$m_j \ddot{\overrightarrow{q_j}} = -G \sum_{k \neq j}^{n} \frac{m_j m_k \left(\overrightarrow{q_j} - \overrightarrow{q_k}\right)}{|\overrightarrow{q_j} - \overrightarrow{q_k}|^3}$$

```
def pos_suiv(m,pos,vit,h):
    n=len(m)
    h2sur2=h**2/2. # Sorti de la boucle pour la complexité
    acc=[]
    pos2=[]
    for j in range(n):
        acc.append(smul(1/m[j],forceN(j,m,pos)))
        pos2.append( vsom(pos[j],vsom(smul(h,vit[j]), smul(h2sur2,acc[j]))))
    return[pos2,acc]
    # Ce n'est pas demandé dans l'énoncé,
    #mais on sort les accélérations à ce pas pour les avoir
    #dans la suite afin d'éviter quelques calculs inutiles.
```

III.B.3) Fonction etat_suiv

```
def etat_suiv(m,pos,vit,h):
        posip1 , acci = pos_suiv(m, pos, vit, h)
        accip1=[]
        vit2=[]
        for j in range(len(m)):
                accip1.append(smul(1/m[j],forceN(j, m, posip1)))
                vit2.append(vsom(vit[j], smul(h/2., (vsom(acci[j], accip1[j])))))

        return posip1, vit2
```

III.B.4)

a) A partir d'une analyse du comportement présenté, on peut proposer une loi affine :

$$ln(\tau_N) = aln(N) + b$$

.

b) Le coefficient directeur de la droite est environ égale à 2. En passant l'équation précédente à l'exponentielle on a $\tau_N \approx AN^a$ ce qui donne $\tau_N = O(N^2)$: Complexité quadratique.

III.B.5) Complexité temporelle

etat_suivant fait appel à *pos_suiv*.

Dans une boucle sur les n éléments *pos_suiv* fait appel à $forceN$ qui fait x*n calculs pour déterminer les efforts sur chacun des corps.

Ce n'est pas nécessaire mais on peut évaluer x : $x \approx 3(3coordonnes) + 3(vdif) + 3(norme) + 4(2 * 1/1 * *) + 3(smul)$.

Donc *pos_suiv* a une complexité en $O(N^2)$.

etat_suivant a une boucle sur n élément. Pour chaque élément on multiplie une liste de dimension 3 par un scalaire. Cette liste est crée par la fonction $forceN$ qui fait $x * n$ itérations.

Soit une complexité en $O(N^2)$.

La boucle suivante parcours aussi chacun des termes pour faire la mise à jour des vitesses. On peut évaluer son cout constant pour chaque corps : On a une somme de trois composantes, puis une multiplication des composantes par un scalaire et à nouveau une somme soit 9 calculs.

Soit une complexité en $O(N)$.

Soit une complexité totale en $O(N^2)$.

b) Les résultats correspondent.

IV Exploitation d'une base de données

IV.A Liste des masses

```
SELECT masse FROM corps
```

IV.B Requêtes avancées

IV.B.1) Comptage des corps

```
SELECT count( DISTINCT id_corp) FROM etat WHERE datem < tmin()
```

IV.B.2) Récupération du dernier état des corps

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

IV.B.3) Simplification du problème

```
SELECT a.masse, b.x, b.y, b.z, b.vx, b.vy, b.vz FROM date_mesure AS c
JOIN etat AS b ON c.id_corps=b.id_corps AND c.date_der=b.datem
JOIN corps AS a ON c.id_corps=a.id_corps
WHERE ABS(b.x) < arete()/2 AND ABS(b.y) < arete()/2
        AND ABS(b.z) < arete()/2 AND a.masse > masse_min()
ORDER BY SQRT(POWER(b.x,2)+POWER(b.y,2)+POWER(b.z,2)) ASC
```

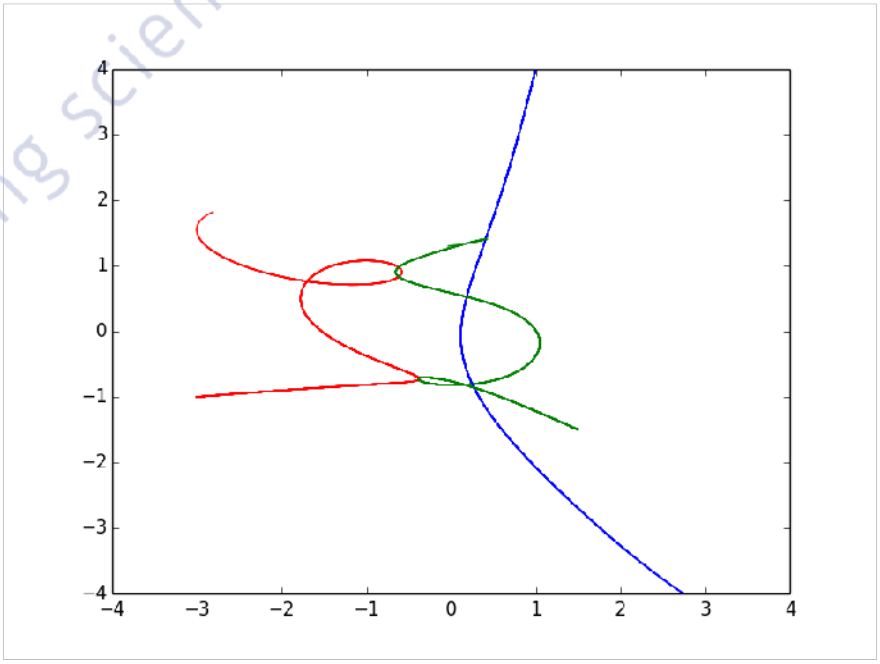
IV.C Fonction simulation_verlet

```
def simulation_verlet(h,n,t0, p0, v0 ,masse):
        # Ajout d'arguments ou alors il faut supposer que t0, p0, v0
        # et masse sont des variables globales
        #Données
        #Valeur d'une unité astronomique en m
        ua = 149597871000
        lua=1./ua
        kV=1000.
        N=len(p0)
        P0_USI=[]
        V0_USI=[]

        #Pour chaque corps i appartenant a N
        for j in range(N):
                P0_USI.append(smul(ua,p0[j]))   # conversion du vecteur position en m
                V0_USI.append(smul(kV,v0[j]))   # conversion du vecteur vitesse en m/s

        # Initialisation
        pos=P0_USI
        position = [pos]
        vit=V0_USI

        #Résolution
        #Pour chaque position k du corps i, on calcule
        #la position et la vitesse a l'instant t+deltat
        for i in range(n):
                pos, vit = etat_suiv(masse,pos,vit,h)
                #Conversion
                posC=[]
                for j in range(N):
                        posC.append(smul(lua,pos[j]))
                position.append(posC)
        return position
```



Cas test d'un système à trois corps avec une simplification des unités (cf code en annexe)