Devoir d'informatique 4 – Corrigé

D.Malka – MPSI 2019-2020 – Lycée Jeanne d'Albret

31.03.2020

1 Mise en orbite de la sonde

1. **Modélisation du mouvement de la sonde lors de la phase d'assistance gravitationnelle**

1.1 Expressions des fonctions S_x et S_y. On projette l'équation :

$$\frac{d^2\vec{r}(t)}{dt^2} = -Gm_s\frac{\vec{r}(t)}{\|\vec{r}(t)\|^3} - -Gm_v\frac{\vec{r}(t)-\vec{r}_v(t)}{\|\vec{r}(t)-\vec{r}_v(t)\|^3}$$

sur les vecteur de base $(\vec{e}_x, \vec{e}_y)$:

$$\frac{d^2x(t)}{dt^2} = -Gm_s\frac{x}{\|\vec{r}(t)\|^3} - -Gm_v\frac{x-x_v}{\|\vec{r}(t)-\vec{r}_v(t)\|^3} \quad (1)$$
$$\frac{d^2y(t)}{dt^2} = -Gm_s\frac{y}{\|\vec{r}(t)\|^3} - -Gm_v\frac{y-y_v}{\|\vec{r}(t)-\vec{r}_v(t)\|^3} \quad (2)$$

avec $\|\vec{r}(t)\|^3 = (x^2+y^2)^{3/2}$, $\|\vec{r}(t)-\vec{r}_v(t)\|^3 = ((x-x_v)^2+(y-y_v)^2)^{3/2}$, $x_v(t) = r_v\cos(\Omega t+\phi)$ et $y_v(t) = r_v\sin(\Omega t+\phi)$, il vient :

$$\frac{d^2x(t)}{dt^2} = -Gm_s\frac{x}{(x^2+y^2)^{3/2}} - -Gm_v\frac{x-r_v\cos(\Omega t+\phi)}{((x-r_v\cos(\Omega t+\phi))^2+(y-r_v\sin(\Omega t+\phi))^2)^{3/2}} \quad (1)$$
$$\frac{d^2y(t)}{dt^2} = -Gm_s\frac{y}{(x^2+y^2)^{3/2}} - -Gm_v\frac{y-r_v\sin(\Omega t+\phi)}{((x-r_v\cos(\Omega t+\phi))^2+(y-r_v\sin(\Omega t+\phi))^2)^{3/2}} \quad (2)$$

Soit :

$$\frac{d^2x(t)}{dt^2} = S_x(x(t),y(t),t) \quad (1)$$
$$\frac{d^2y(t)}{dt^2} = S_y(x(t),y(t),t) \quad (2)$$

avec :

$$S_x : (x,y,t) \to -Gm_s\frac{x}{(x^2+y^2)^{3/2}} - -Gm_v\frac{x-r_v\cos(\Omega t+\phi)}{((x-r_v\cos(\Omega t+\phi))^2+(y-r_v\sin(\Omega t+\phi))^2)^{3/2}}$$

$$S_y : (x,y,t) \to -Gm_s\frac{y}{(x^2+y^2)^{3/2}} - -Gm_v\frac{y-r_v\sin(\Omega t+\phi)}{((x-r_v\cos(\Omega t+\phi))^2+(y-r_v\sin(\Omega t+\phi))^2)^{3/2}}$$

1.2 Fonction `eval_sm(x,y,t)` qui prend en argument x, y et l'instant t et qui renvoie les images s_x et s_y par S_x et S_y.

```
1 def eval_sm(x,y,t):
2     xv=rv*cos(w*t+phi)
3     yv=rv*sin(w*t+phi)
4     r=x**2+y**2
5     r_rel=(x-xv)**2+(y-yv)**2
6     sx=-Gms*x/r**3/2-Gmv*(x-xv)/r_rel**3/2
7     sy=-Gms*y/r**3/2-Gmv*(y-yv)/r_rel**3/2
8     return sx,sy
```

2. Résolution numérique

2.1 Schéma numérique d'Euler explicite sur les vitesses u_i et v_i à partir des équations du mouvement :

$$\boxed{u_{i+1} = u_i + s_x \Delta t} \qquad \boxed{v_{i+1} = v_i + s_y \Delta t}$$

2.2 Schéma numérique d'Euler explicite sur les positions x_i et y_i à partir des définitions de $v_x = \dfrac{dx}{dt}$ et $v_y = \dfrac{dy}{dt}$

$$\boxed{x_{i+1} = x_i + u_i \Delta t} \qquad \boxed{y_{i+1} = y_i + v_i \Delta t}$$

2.3 Programme de calcul de la trajectoire de la sonde.

```
1  def orbite(init,delta_t,n):
2      """
3      Calcule, par la methode d'Euler explicite la trajectoire de la sonde soumis
4      a l'influence gravitationnelle du Soleil et de Venus. Renvoie la postion
5      (x,y) et la vitesse (v_x,v_y) au cours du mouvement.
6
7      init : list or array, contient les vitesses et positions initiales
8      [x(0),y(0),v_x(0),v_y(0)].
9      delta_t : float, pas d'integration.
10     n : int, nombre de points calcules. Duree de la simulation : T=n*delta_t
11     x,y,u,v : array. Contiennent respectivement x(t),y(t),v_x(t),v_y(t).
12     """
13     t=arange(0,n*delta_t,delta_t)
14     x=zeros(n);x[0]=init[0]
15     y=zeros(n);y[0]=init[1]
16     u=zeros(n);u[0]=init[2]
17     v=zeros(n);v[0]=init[3]
18
19     for i in range(n-1):
20         sx,sy=eval_sm(x[i],y[i],t[i])
21         x[i+1]=x[i]+u[i]*delta_t
22         u[i+1]=u[i]+sx*delta_t
23         y[i+1]=y[i]+v[i]*delta_t
24         v[i+1]=v[i]+sy*delta_t
25
26     return x,y,u,v
```

2.4 Quantité de mémoire nécessaire pour réaliser cette simulation numérique pour une durée de simulation de 30 jours et pour un pas de temps de 1 seconde sachant que les variables `x`, `y`, `u`, `v` contiennent des flottants codés sur 8 octets.

L'essentiel de la mémoire est occupée par les tableaux `x`, `y`, `u`, `v`. La longueur de chaque tableau vaut $n = \lfloor \dfrac{T}{\Delta_t} \rfloor$, la taille de chaque flottant contenu dans les tableaux est $s = 8$ octets. A l'issu de la simulation, l'espace mémoire occupée vaut :

$$\boxed{S = 4s\lfloor \frac{T}{\Delta_t} \rfloor}$$

A.N. : $s = 8$ octets, $T = 30$ jours et $\Delta_t = 1$ s donne : $S \approx 80$ Mo.

Sur les ordinateurs modernes, allouer 80 Mo de mémoire vive au programme ne pose aucun problème.

2 Compression de données sans pertes

1. **Structure des données brutes**

1.1 Sur n bits, le plus grand entier naturel représentable est $2^n - 1$, soit, pour $n = 12$, $2^{12} - 1 = 4095$. Si plus de photons arrive sur le détecteur durant le temps d'acquisition, il va y avoir dépassement de capacité. La valeur enregistrée sera tronquée. Par exemple, s'il arrive 4096 photons alors la valeur à enregistrer devrait être 1 000 000 000 000 mais le bit de poids fort sera perdu par décalage vers la droite et donc la valeur enregistrée sera 000000000000 soit 0 !

1.2 Taille T, en octets, d'un cube de données :

$$\boxed{T = 64 \times 64 \times 352 \times 12 = 2{,}16\,\text{Mo}}$$

2. **Principe de la compression sans pertes et compression optimale**

Des contraintes techniques imposent une compression sans pertes des données du cube avant transmission. Une fois compressé, la taille du cube ne doit pas dépasser $T_c = 1\,\text{Mo}$.

2.1 Taux de compression minimale $\tau = \dfrac{T - T_c}{T}$ à appliquer aux données. $T = 2{,}16\,\text{Mo}$ et $T_c = 1\,\text{Mo}$ donne :

$$\boxed{\tau = 53{,}7\,\%}$$

2.2 Il faut 24 bits pour coder la séquence 4-5-7-0-7-8-4-1-7-4 à comparer au 30 bits nécessaires pour coder cette même séquence par la représentation naturelle sur 3 bits.

2.3 Reste à choisir l'algorithme de codage des données. On peut montrer que la longueur optimale L_c^* de la séquence S de données une fois codée est minorée via l'entropie de Shannon H :

$$L_c^* \geq nH(S) \quad \text{avec} \quad H(S) = -\sum_i p_i \log_2 p_i$$

où $p_i = \dfrac{n_i}{n}$ où n_i est le nombre d'occurrences d'une valeur v_i de la séquence S et n le nombre de termes de la séquence S.

2.3.1 Entropie associée à la séquence 4-5-7-0-7-8-4-1-7-4. $n = 10$.

v_i	n_i	$p_i = n_i/n$
4	3	0,3
5	1	0,1
7	3	0,3
0	1	0,1
1	1	0,1
8	1	0,1

$$H(S) = -\sum_i p_i \log_2 p_i = -2 \times 0.3 \times \log_2(0.3) - 4 \times 0.1 \times \log_2(0.1) = 2{,}37$$

D'où $L_c^* \geq nH(S)$ soit $L_c^* \geq 23{,}7$ bits. Comme L_c^* doit être entier, $L_c^* \geq 24$ bits. Le codage utilisé dans l'exemple est donc optimal.

2.3.2 Fonction `occurrence` prenant en argument une séquence de données (type `list`) et renvoyant le nombre d'occurrences de chaque valeur v_i de la séquence d'entrée. La bonne structure de données pour écrire cette fonction serait un dictionnaire (réduirait la complexité en temps) mais elle n'est pas au programme.

```
1 #L ne contient que des entiers, ce qu'onn exploite en creeant un dictionnaire.
2 def occurrence(L):
3     """
4     Compute frequencies F[i] of each value L[i] of L.
5     L: list of unsigned int, data to compute occurencies
6     F: array, F[k] is the number of occurencies of the value k
7     """
8     m=max(L)
9     F=zeros(m+1)
```

```
    for k in L:
        F[k]+=1
    return F
```

2.3.3 Fonction `entropie` prenant en argument une séquence de données (type `list`) et renvoyant l'entropie de Shannon associée à l'entrée.

```
def entropie(S):
    """
    Compute Shannon entropy H of the data S
    S: list
    H: float
    """
    n=len(S)
    #Calcul des frequences
    p=occurrence(S)/n;print(p)
    q=len(p)
    H=0
    for i in range(q):
        if p[i]!=0:
            H=H-p[i]*np.log2(p[i])
    return H
```

2.3.4 Complexités en temps des fonctions `occurrence` et `entropie` : $O(n)$.

3. Compression par le codage de Rice

3.1 Mot binaire obtenu après codage selon l'algorithme de Rice de la séquence des valeurs 2 à 8 des δ_k du tableau fig.**??**.

k	δ	Code de Rice
2	4	10 0
3	10	110 10
4	18	11 110 10
5	18	11 110 10
6	6	10 10
7	1	10 11
8	3	10 11

Code Rice de :
— de 18 : $q = 18/2^2 = 4$ qu'on code de façon unaire 11 110 ; $r = 18\%2^2 = 2$ qu'on code en binaire 10. D'où le code 11 110 10.
— de 10 : $q = 10/2^2 = 2$ qu'on code de façon unaire 110 ; $r = 18\%2^2 = 2$ qu'on code en binaire 10. D'où le code 110 10.

3.2 Fonction `Rice` prenant en argument un entier non signé de type `int` et renvoyant une liste contenant le code de Rice de cet entier. Le principe est de calculer la représentation unaire du quotient q de la division euclidienne de l'entrée x par 2^p (fonction `un`), la représentation binaire du reste r de la division euclidienne de l'entrée x par 2^p (fonction `bin`), puis de concaténer les deux listes obtenues. Attention, l'algorithme de la fonction `bin` place le bits de poids fort à gauche donc il faut inverser la liste d'où l'implémentation de `reverse`

```
def reverse(L):
    n=len(L)
    for i in range(n//2):
        L[i],L[n-i-1]=L[n-i-1],L[i]
    return None


def un(n):
    """
    Calcule et renvoie la representation unaire de n sous forme de liste
    n : unsigned int
    n_un : list
    """
    n_un=[]
```

```
    for i in range(n):
        n_un.append(1)
    n_un.append(0)
    return n_un

def bin(n,p):
    """
    Calcule et renvoie la representation binaire de n sous forme de liste
    n : unsigned int
    p : int, number of bits used for the binary representation
    n_bin : list
    """
    n_bin=[]
    while n!=0:
        n_bin.append(n%2)
        n=n//2
    print(n_bin)
    #Completion avec des 0 pour une representation sur p bits
    while len(n_bin)<p:
        n_bin.append(0)
    reverse(n_bin)
    return n_bin

def rice(x,p):
    """
    Calcule et renvoie le code de Rice de parametre pde l'entier x sous forme
    d'une liste de bits
    x: int
    p: int
    xc : list of 0 and 1, code de Rice de x
    """
    q=un(x//2**p)
    r=bin(x%2**p,p)
    xc=q+r
    return xc
```

3.3 En représentation binaire, la division par 2 d'un nombre revient à un décalage de un bit vers la droite (avec pertes du bit de poids faible). Par exemple, 6 s'écrit en binaire 110, divisé par 2, devient 3 soit 011. De façon générale, la division par 2^p s'implémente comme un décalage de p bits vers la droite. Le reste r est simplement le nombre formé par les bits $p-1$ à 0 du nombre à diviser : il suffit de les recopier. *C'est ce qui explique la très grande efficacité du codage de Rice*