Notebook

April 24, 2015

1 Mines IPT MP 2015

1.1 I Réception des données

Q1

Un rappel sur le codage d'entier en annexe n'aurait pas fait de mal. La plage est $[-2^9, 2^9-1] = [-512, 511]$

Q2

J'interprète pleine échelle on va de -5 à 5 volts valeurs extrêmes comprises. On dispose de $2^{10} = 1024$ points ou 1023 écarts donc la résolution sera de $\frac{10}{1023}\ldots$

Pas sûr de moi sur ce coup-là...

J'aurai plutôt dit $511 = 5V$ mais bon...

Q3

On aurait pu traduire, checksum = somme de contrôle

```
In [1]: def lectmesures():
            ''' retourne None si entête pas correcte ?
            '''
            typemes = com.read(1)
            if typemes not in ['U', 'I', 'P']:
                return None

            RETOUR = [typemes]
            nbentree = int(com.read(3))

            Ldonnees = []
            for i in range(nbentree):
                Ldonnees.append(int(com.read(4)))

            RETOUR.append(Ldonnees)
            RETOUR.append(int(com.read(4)))

            return RETOUR
```

Q4

Le concepteur du sujet n'a toujours pas lu la pep8... [] hérité du java...

```
In [2]: def check(mesure, Checksum):
            Lmes = mesure[1]
            S = sum([abs(a) for a in Lmes])
            return (S % 10000) == mesure[2]
```

Q5

```
In [3]: def affichage(mesure):
            Lmes = mesure[1]
            Lmesexacte = [4*a for a in Lmes]  # si je comprends bien l'énoncé... pas sûr
            n = len(Lmes)
            Labsc = [2*a for a in range(n)]

            plt.plot(Labsc, Lmesexacte)
            plt.show()
```

1.2 II Analyse des mesures

Q6

```
In [4]: def trapeze(Lmes, tps=2):
            ''' améliorable
            '''
            n = len(Lmes)
            S = 0.0
            for i in range(1, n):
                S += Lmes[i-1] + L[i]

            return S / tps*(n - 1)
```

Q7

```
In [5]: import math

        def ecarttype(Lmes, tps=2):
            ''' idem '''
            moy = trapeze(Lmes, tps)
            Lmes2 = [(a - moy)**2 for a in Lmes]
            retour = trapeze(Lmes2, tps)

            return math.sqrt(retour)
```

1.3 Base de données

Q8

Constantes Imin et Imax ???
SELECT nSerie FROM testfin WHERE Imoy > Imin AND Imoy < Imax

Q9

SELECT nSerie, Iec, fichierMes FROM testfin WHERE Iec < (SELECT AVG(Iec) FROM testfin)

Q10

SELECT nSerie, fichierMes FROM testfin
JOIN
(SELECT nSerie AS n2 FROM testfin EXCEPT SELECT nSerie FROM production)
ON nSerie = n2;

1.4 Compression

Q11

test() renvoie un booléen, les deux autres fonctions sont utilisées pour renvoyer respectivement un chaine de caractère (str) et un entier (int) mais bon c'est tordu comme formulation...

Q12

```
In [6]: import huffmannorigine as h
```

```
In [7]: node1 = h.make_huffman_tree(('F', 1), ('E', 1))
        print(node1)

[('F', 1), ('E', 1), ['F', 'E'], 2]

In [8]: node2 = h.make_huffman_tree(('D', 1), ('B', 1))
        print(node2)

[('D', 1), ('B', 1), ['D', 'B'], 2]
```

Q13

```
In [9]: node3 = h.make_huffman_tree(node1, node2)
        print(node3)

[[('F', 1), ('E', 1), ['F', 'E'], 2], [('D', 1), ('B', 1), ['D', 'B'], 2], ['F', 'E', 'D', 'B'], 4]
```

Q14

```
In [10]: f = h.freq_table('AABBCB')

In [11]: list(f.items())

Out[11]: [('A', 2), ('B', 3), ('C', 1)]
```

L'énoncé aurait du obligatoirement le préciser.

Q15

C'est une fonction récursive, je suppose que c'est la réponse attendue...

Q16

On repère un "espèce d'invariant de récursivité" pour parler dans le même style que l'auteur. On début la liste est vide donc l'énoncé est vrai puis onn récurre, je passe.

Q17

```
In [12]: ## 5 Et si j'indentais correctement ?
         # plus sérieusement
         ## 5 création de l'arbre avec lst
         ## 6  création d'un sous arbres avec les 2 plus petits poids
         ## 7 on supprime les deux éléments en question
         ## 8 et on rajoute le sous-arbre nouvellement crée à la bonne place
```

Q18

Dans le meilleur des cas, avec un appel à pos=0, on fait deux comparaisons en insérant un élément de poids strictement plus petit que tous le autres. Danns le pire des cas, il a le plus grand poids strict et on effectue $2n + 1$ comparaisons.

Q19

La formulation est complètement tordu puisque lst est une variable interne à la fonction build_huffman_tree. Comptons vaguement qq chose c'est-à-dire les comparaisons après lst.sort en fonction de n = len(lst).

On fait n//2 itérations avec un appel à insert_item de complexité entre 2 et 2p+1 où p est le n local puis on rajoute une création d'arbre.

Dans le meilleur des cas, la création de sous-arbre donne un poids qui est inférieur à l'élément suivant, ce qui fait de l'ordre de 2*n//2 = n comparasions (à un chouïa près) (poids: 1, 1, 2, 3, 5... fibonacci quoi)

Dans le pire des cas on crée à chaque fois un sous-arbre de poids maximal (1, 1, 1, 1, 1, 1,...) ce qui fait un nombre de comparaisons de $\sum_{p=0}^{n//2}(2p+1)$ soit un ordre n^2 comparaisons.

Critique On ne compte pas les opérations de gestion de mémoire (garbage collector) des instructions del et de création de listes en pagaille...

Q20

```
In [13]: htree = h.build_huffman_tree('ZBBCB')
         print(htree)
```

```
[('B', 3), ('Z', 1), ('C', 1)]
[[('Z', 1), ('C', 1), ['Z', 'C'], 2], ('B', 3), ['Z', 'C', 'B'], 5]
```

Q21
Je me dispense de l'arbre, B = 1, C = 00 et Z = 01 par exemple
Q22
En version numpy

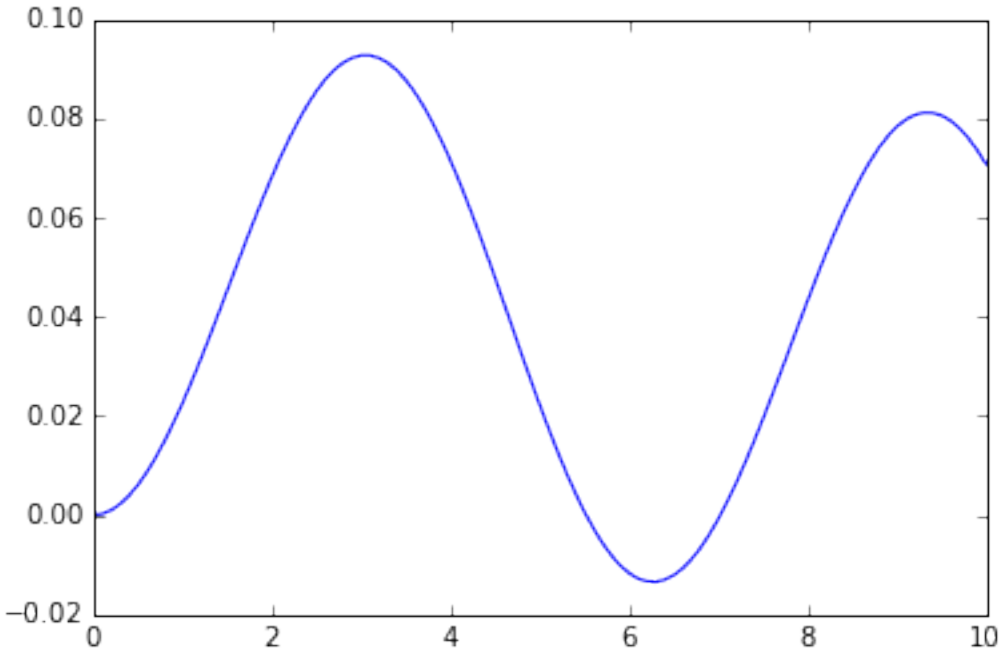
```
In [14]: % matplotlib inline
         import numpy as np
         import matplotlib.pyplot as plt
         from scipy.integrate import odeint


         k = 0.5 # par exemple
         T = np.linspace(0, 10, 101)
         e = np.sin

         def f(y, t):
             return - k * (y - e(t)) / 10.0

         s = odeint(f, 0, T)

         plt.plot(T, s)
         plt.show()
```



Version scilab, quelque chose du genre
k=0.5;

e=sin;
function [ydot]=f(t, y)

```
ydot = -k*(y - e(t))/10.
```

endfunction
y0=0;
t0=0;
T=0:0.1:10;
y = ode(y0,t0,T,f);
plot(T,y);

Q23

```
In [15]: def test(k, smes, seuil=1e-5):

             T = np.linspace(0, 10, 101)
             e = np.sin

             def f(y, t):
                 return - k * (y - e(t)) / 10

             s = odeint(f, 0, T)

             # on suppose smes correct par rapport à T
             n = len(T)
             L = [(smes[i] - s[i])**2 for i in range(n)]
             test = sum(L)

             return test < seuil**2
```