ENSAE exercice de préparation pour le TD noté, mardi 27 novembre 2012

Ces exercices abordent des sujets en rapport avec le TD noté.

1

On construit une séquence selon le procédé suivant :

1. On tire un nombre entier entre 0 et 2.
2. On l'ajoute à la séquence.
3. Si le nombre tiré est 0, on s'arrête.
4. Si c'est 1, on tire à nouveau une fois et on répète le même processus depuis l'étape 2.
5. Si c'est 2, on tire deux nombres qu'on ajoute à la séquence et on tire encore autant de fois que la somme des deux nombres tirés. Pour chacun d'entre eux, on répète le processus depuis l'étape 3.

Rappel : voici deux lignes de code permettant de générer un nombre aléatoire entier entre 0 et 5 inclus

```
import random
i = random.randint(0,5)
```

1) Construire une fonction qui construit une séquence telle que celle définie plus haut.

2) Construire une fonction qui calcule la moyenne des longueurs des séquences obtenues (sur 1000 séquences par exemple) ?

Correction

L'énoncé peut suggérer qu'il faut agir différemment selon que le nombre entier aléatoire est 0, 1 ou 2. Cependant, il est utile de remarquer que le nombre de tirages restant à faire dépend de la longueur de la séquence de nombres et de la somme des nombres qu'elle contient. Si on note `s` la séquence de nombres aléatoires dans un état intermédiaire, le nombre de tirages aléatoires restant à effectuer est égal à $\texttt{sum(s)} - \texttt{len(s)} + 1$. On vérifie que cela fonctionne lorsque `s` contient juste un nombre.

La seconde question ne pose pas de problème puisqu'il s'agit de faire une moyenne des longueurs d'un grand nombre de séquences, 100 dans le programme qui suit.

```
#coding:latin-1
import random

# question 1
def sequence () :
    res = [ ]
    nb = 1
    while nb > 0 :
        i = random.randint(0,2)
        nb += i - 1
        res.append (i)
```

```
    return res

# question 2
def moyenne (nb_tirage) :
    somme = 0.0
    for i in range(nb_tirage) :
        s = sequence()
        somme += len(s)
    return somme / nb_tirage

s = sequence ()
print len(s),s
m = moyenne (100)
print m
```

Après quelques exécutions, on remarque que cette moyenne n'est pas jamais la même ou tout simplement que le programme ne se termine pas. Il y a deux explications possibles :

1. Soit 100 tirages ne suffisent pas pour faire converger la moyenne des longueurs,
2. Soit la moyenne n'existe pas.

Le programme informatique ne permet pas de répondre de manière certaine à cette question, il permet juste d'avoir une intuition qui serait dans ce cas que la moyenne n'existe pas. La preuve doit être mathématique.

Aparté

On note $S_n = (N_1, N_2, ..., N_n)$ une séquence de nombres aléatoires tirés dans l'ensemble $\{0, 1, 2\}$ avec les probabilités (a, b, c). Cette séquence correspond à celle de l'énoncé dans le cas où $a = b = c = \frac{1}{3}$. L'intuition est que si $a > c$, la moyenne des longueurs des séquences est finie et si $a \leqslant c$ alors elle est infinie.

La démonstration qui suit repose sur l'indépendance des tirages aléatoires. On note la variable aléatoire $\#S$ qui désigne la longueur d'une séquence générée selon le processus décrit au paragraphe précédent. On décompose cette variable comme ceci :

$$\#S = k + \#S_k \tag{1}$$

Où $\#S_k$ est la longueur de la séquence après le $k^{\text{ième}}$ élément exclu. En tenant compte de l'indépendance des tirages et en supposant que l'espérance de $\#S$ existe, on peut dire que :

$$\mathbb{E}\left(\#S_k | \sum_{j=0}^{k} N_j \leqslant k+1\right) = 0 \tag{2}$$

$$\mathbb{E}\left(\#S_k | \sum_{j=0}^{k} N_j = k+2\right) = \mathbb{E}\left(\#S\right) \tag{3}$$

$$\mathbb{E}\left(\#S_k | \sum_{j=0}^{k} N_j = k+3\right) = 2\mathbb{E}\left(\#S\right) \tag{4}$$

La première assertion est évidente. Si à partir d'un certain rang, la somme des nombres tirés est inférieure à $k + 1$, la séquence s'arrête. La deuxième assertion l'est également, à partir du rang k, il

reste un nombre à tirer, l'espérance de la longueur de la séquence qui commence à partir de cette position est identique à celle au début de celle-ci.

Pour la dernière assertion, imaginons que nous devions tirer 2 nombres aléatoires. On peut le faire aux positions $k+1$ et $k+2$ ou on peut tirer le premier nombre, continuer la séquence, attendre qu'il n'y ait plus de nombres à tirer puis tirer le second. Dans ce cas, on comprend que l'espérance de la longueur est bien 2 fois celle d'une séquence. On en déduit que :

$$\mathbb{E}\left(\#S_k\right) = \sum_{m=0}^{\infty} \mathbb{E}\left(\#S_k | \sum_{j=0}^{k} N_j = k+m+1\right) \mathbb{P}\left(\sum_{j=0}^{k} N_j = k+m+1\right) \tag{5}$$

$$= \sum_{m=0}^{\infty} m\mathbb{E}\left(\#S\right) \mathbb{P}\left(\sum_{j=0}^{k} N_j = k+m+1\right) \tag{6}$$

Pour utiliser ce raisonnement, on isole le premier nombre s_0 de la séquence aléatoire S en $S = (N_0, S\ N_0)$. $\#S_1$ est la séquence S privée de N_0. On applique le résultat précédent :

$$\mathbb{E}\left(\#S\right) = \begin{cases} 1 & \text{si } N_0 = 0 \\ 1 + \mathbb{E}\left(\#S\right) & \text{si } N_0 = 1 \\ 1 + 2\mathbb{E}\left(\#S\right) & \text{si } N_0 = 2 \end{cases} \tag{7}$$

On en déduit que :

$$\begin{aligned} \mathbb{E}\left(\#S\right) &= a + b\left[1 + \mathbb{E}\left(\#S\right)\right] + c\left[1 + 2\mathbb{E}\left(\#S\right)\right] \\ &= a + b + c + \mathbb{E}\left(\#S\right)(b + 2c) \\ &= 1 + \mathbb{E}\left(\#S\right)(1 - a + c) \end{aligned} \tag{8}$$

On en déduit que :

$$\mathbb{E}\left(\#S\right) = \frac{1}{a-c} \tag{9}$$

Sachant que cette quantité est forcément positive, elle n'est définie que si $a > c$. Le programme suivant permet de vérifier qu'en simulant des séquences pour plusieurs valeurs a, b, c, on retrouve bien une espérance proche de celle donnée par cette formule.

```
#coding:latin-1
import random

def randomint (a,b,c) :
    x = random.random()
    if x <= a : return 0
    elif x <= a+b : return 1
    else : return 2
```

```
def sequence (a,b,c) :
    res = [  ]
    nb = 1
    while nb > 0 :
        i = randomint(a,b,c)
        nb += i - 1
        res.append (i)
    return res

def moyenne (nb_tirage,a,b,c) :
    somme = 0.0
    for i in range(nb_tirage) :
        s = sequence(a,b,c)
        somme += len(s)
    return somme / nb_tirage

a,c = 0.3, 0.2
b = 1-a-c

moy = 1.0 / (a-c)
print "calcul",moy

m1 = moyenne (100000, a,b,c)
print "simulée", m1
```

Second aparté

$S_n = (N_1, N_2, ..., N_n)$ est toujours une séquence de nombres aléatoires tirés dans l'ensemble $\{0, 1, 2\}$ avec des probabilités équiprobables (a, b, c), le nombre de tirages T_n restant à effectuer après n tirages est :

$$T_n = \sum_{i=1}^{n} N_i - n + 1 \tag{10}$$

Si définit la séquence $S'_n = (N'_1, ..., N'_n) = (N_1 - 1, ..., N_n - 1)$ où N'_i est une variable aléatoire à valeur dans l'ensemble $\{-1, 0, +1\}$:

$$T_n = \sum_{i=1}^{n} N'_i + 1 = T'_n + 1 \tag{11}$$

La séquence S'_n est de longueur finie s'il existe n tel que $T'_n = 0$. T'_n est en quelque sorte une marche aléatoire dont on peut définir l'espérance et la variance :

$$\mathbb{E}\left(T'_n\right) = \sum_{i=1}^{n} \mathbb{E}\left(N'_i\right) = n\mathbb{E}\left(N'_1\right) = 0 \tag{12}$$

$$\mathbb{V}\left(T'_n\right) = \sum_{i=1}^{n} \mathbb{V}\left(N'_i\right) = n\mathbb{V}\left(N'_1\right) = n\left[\mathbb{E}\left((N'_1)^2\right) - \left(\mathbb{E}\left(N'_1\right)\right)^2\right] = n(a + c - (c - a)^2) \tag{13}$$

Dans la suite, on pose $e = -1$. On définit le nombre U tel que : $U = \inf\{u | T'_u = e\}$. U est la longueur de la séquence S. C'est aussi une variable aléatoire qui vérifie :

$$\begin{cases} T'_U = e \\ \forall u < U,\ T'_u \neq e \end{cases} \tag{14}$$

U est un temps d'arrêt pour le processus aléatoire $(S'n)_n$. On s'intéresse maintenant à la séquence S'_n. Etant donné que chaque élément peut prendre trois valeurs, il existe 3^n séquences différentes. On va chercher à calculer la probabilité de chaque séquence S'_n vérifiant :

$$\forall u < n,\ T'_u = \sum_{i=1}^{u} N'_i \neq e \tag{15}$$

Donc notre cas, on suppose $e = -1$ ce qui implique pour la marche aléatoire de rester positive. Le raisonnement serait le même pour $e > 0$. La probabilité $\mathbb{P}(U = u)$ revient à énumérer toutes les marches aléatoires (ou le nombre de chemins) qui terminent à e tout en restant supérieures à e entre 1 et u exclu. On définit p_{ui} le nombre de chemins terminant par $T'_u = e$ et ne passant jamais par e ($\forall k < u,\ T'_k \neq e$). On définit :

$$p_{ue} = f_e(u) = \mathbb{P}\left(T'_u = e,\ T'_k \neq e\ \forall k < u\right) \tag{16}$$

Si $e < 0$, on peut construire p_{ui} par récurrence :

$$p_{1k} = \begin{cases} 0 & \text{si } k \neq 1 \\ 1 & \text{si } k = 0 \end{cases} \tag{17}$$

$$p_{uk} = \begin{cases} c\,p_{u-1,k-1} & +b\,p_{u-1,k} & +a\,p_{u-1,k+1} & \text{si } k > e+1 \\ & b\,p_{u-1,k} & +a\,p_{u-1,k+1} & \text{si } k = e+1 \\ & & a\,p_{u-1,k+1} & \text{si } k = e \\ 0 & & & \text{sinon} \end{cases} \tag{18}$$

A partir de cette définition, on peut désormais écrire que si l'espérance de U existe, alors :

$$\mathbb{E}(U) = \sum_{u=1}^{\infty} u\mathbb{P}(U = u) = \lim_{n\to\infty} \sum_{u=1}^{n} u\mathbb{P}(U = u) \tag{19}$$

$$= \lim_{n\to\infty} \sum_{u=1}^{n} u\mathbb{P}(T_u = e, T_k \neq e\,\forall k < u) \tag{20}$$

$$= \lim_{n\to\infty} \sum_{u=1}^{n} u p_{ue} = \lim_{n\to\infty} r_n \tag{21}$$

On repasse à l'informatique pour avoir l'intuition mathématique de la limite de $(r_u)_u$ lorsque u tend vers l'infini.

```
#coding:latin-1
import random, math

# question 1
def calcul_suite_a (n, e, a,b,c) :
    p = {}
    p [0,0]  = 1
    for u in range (1,n+1) :
        for k in range (e,u+2) :
            if   k == e   : p [u,k] = a * p.get ( (u-1,k+1), 0 )
```

```
            elif k == e+1 : p [u,k] = a * p.get ( (u-1,k+1), 0 ) + \
                                      b * p.get ( (u-1,k  ), 0 )
            elif k >  e+1 : p [u,k] = a * p.get ( (u-1,k+1), 0 ) + \
                                      b * p.get ( (u-1,k  ), 0 ) + \
                                      c * p.get ( (u-1,k-1), 0 )
    return p

def affiche_proba (ps, e) :
    n     = max ( [ k[1] for k,z in ps.iteritems () ] )
    moy = 0.0
    logru = []
    logu  = []
    for u in range (1, n+1) :
        p = ps.get((u,e),0)*1.0
        moy += p * u
        mes = "u % 3d P(U=u) %1.6g r_u %1.6g" % (u, p, moy)
        if u < 3 or u %50 == 0 : print mes
        logru.append(math.log(moy))
        logu.append(math.log(u))

    import pylab
    pylab.plot ( logu, logru, "o")
    pylab.show()

a,c = 1.0/3, 1.0/3
b = 1-a-c
e = -1

su = calcul_suite_a(600,e,a,b,c)
affiche_proba(su, e)
```

Pour $a > c$, on vérifie que la suite $(r_u)_u$ converge vers l'expression (9). Pour $a = b = c = \frac{1}{3}$, cela donne :

```
u    1 P(U=u) 0.333333    r_u  0.333333
u    2 P(U=u) 0.111111    r_u  0.555556
u   50 P(U=u) 0.0013566   r_u  5.57241
u  100 P(U=u) 0.00048407  r_u  8.38753
u  150 P(U=u) 0.000264311 r_u 10.5627
u  200 P(U=u) 0.000171942 r_u 12.4016
u  250 P(U=u) 0.000123146 r_u 14.0242
u  300 P(U=u) 9.37388e-05 r_u 15.4926
u  350 P(U=u) 7.44204e-05 r_u 16.8438
u  400 P(U=u) 6.09325e-05 r_u 18.1021
u  450 P(U=u) 5.10779e-05 r_u 19.2843
u  500 P(U=u) 4.36202e-05 r_u 20.4028
u  550 P(U=u) 3.78157e-05 r_u 21.4669
u  600 P(U=u) 3.31933e-05 r_u 22.4839
```

Il en ressort que la suite $P(U = u)$ semble tendre vers 0 à l'infini. Ceci signifierait que la probabilité de construire une séquence S_n infinie est nulle. Mais la suite $(r_u)_u$ semble tendre vers l'infini ce qui signifirait que la moyenne des longueurs des séquences initiales S_n n'existe pas. On vérifie en traçant le graphe $(\log u, \log r_u)_u$ (voir figure 1). Il suggère que $r_n \sim cn^\alpha$ avec $0 < \alpha < 1$.

Il reste à démontrer formellement que la suite r_n tend vers l'infini. Pour cela, on définit :

$$f_e(n) = \mathbb{P}(U = n) = \mathbb{P}\left(T'_n = e, T'_i \neq e \text{ si } i < n\right) \tag{22}$$

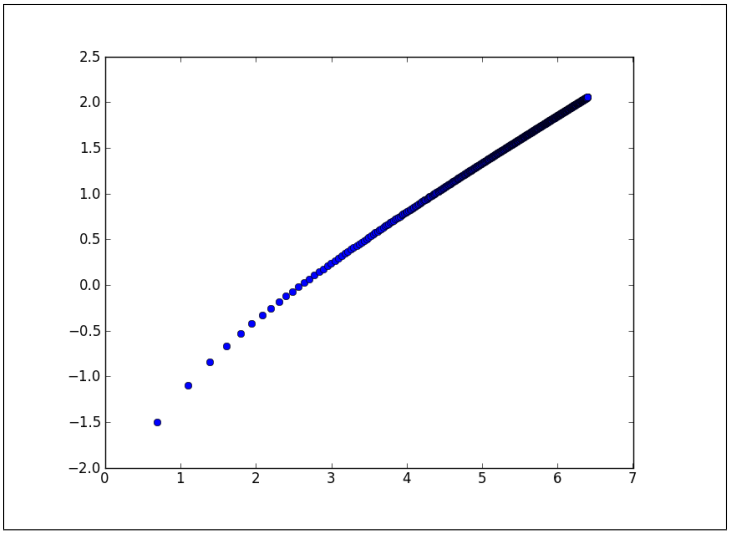

FIGURE 1 : *Graphe* $(\log u, \log r_u)_u$ *défini par (21).*

$f_e(n)$ est différent de la suite p_{n0}. La marche aléatoire pour atteindre -1 au temps n doit nécessaire commencer par 0 ou 1. Si on note, k le premier temps auquel elle passe par 0, on peut décompser $f_{-1}(n)$:

$$f_{-1}(1) = c \tag{23}$$

$$f_{-1}(n) = bf_{-1}(n-1) + c\left[\sum_{k=2}^{n-1} \mathbb{P}\left(\sum_{i=2}^{k} S'_i = -1\right) \mathbb{P}\left(\sum_{i=k+1}^{n} S'_i = -1\right)\right] \tag{24}$$

On en déduit que :

$$f_{-1}(1) = a \tag{25}$$

$$f_{-1}(n) = bf_{-1}(n-1) + c\left[\sum_{k=2}^{n-1} f_{-1}(k-1) f_{-1}(n-k)\right] \tag{26}$$

On pose :

$$F_e(s) = \sum_{n=1}^{\infty} f_e(n) s^n \tag{27}$$

$F_e(1) = \sum_{n=1}^{\infty} f_e(n)$ correspond à la probabilité que la marché aléatoire atteint e en un temps fini. La moyenne des longueurs des séquences $S'n$ est défini comme le temps moyen d'arrivée en e : $\sum_{n=1}^{\infty} nf_e(n)$. On utilise le théorème de la convergence monotone [1] pour montrer que :

1. http://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_de_convergence_monotone

$$\sum_{n=1}^{\infty} n f_e(n) s^n = F'_e(s)$$

$$\implies \lim_{s\to 1} \sum_{n=1}^{\infty} n f_e(n) s^n = \sum_{n=1}^{\infty} n f_e(n) = F'_e(1) \tag{28}$$

On cherche une relation fonctionnelle du type $x[F_{-1}(s)]^2 + yF_{-1}(s) + z = 0$ en utilisant (26).

$$\begin{aligned}
[F_{-1}(s)]^2 &= \left[\sum_{n=1}^{\infty} f_{-1}(n) s^n\right]^2 \\
&= \sum_{n=3}^{\infty} s^{n-1} \left[\sum_{k=2}^{n-1} f_{-1}(k-1) f_{-1}(n-k)\right] \\
&= \sum_{n=3}^{\infty} s^{n-1} \frac{1}{c} \left[f_{-1}(n) - b f_{-1}(n-1)\right] \\
&= \frac{1}{cs} \sum_{n=3}^{\infty} s^n f_{-1}(n) - \frac{b}{cs} \sum_{n=3}^{\infty} s^n f_{-1}(n-1) \\
&= \frac{1}{cs} \left[F_{-1}(s) - f_{-1}(1)s\right] - \frac{b}{c} F_{-1}(s) \\
&= F_{-1}(s) \left[\frac{1}{cs} - \frac{b}{c}\right] - \frac{f_{-1}(1)}{c} \\
\implies \quad & cs[F_{-1}(s)]^2 - F_{-1}(s)(1-bs) - s f_{-1}(1) = 0
\end{aligned} \tag{29}$$

En résolvant le polynôme $csx^2 - (1-bs)x - sf_{-1}(1) = 0$, il est possible de déterminer l'expression de $F_{-1}(s)$. On rappelle que $f_{-1}(1) = a$. Une seule des solutions du polynôme du second degré est positive [2].

$$F_{-1}(s) = \frac{(1-bs) + \sqrt{(1-bs)^2 + 4acs^2}}{2cs} \tag{30}$$

Il ne reste plus qu'à dériver et à trouver la limite lorsque $s \to 1$.

A suivre.

2

On considère une matrice 10×10 remplie de 0 et de 1 aléatoirement avec la probabilité d'avoir 1 égale à 0,2.

1) Construire une telle matrice.

2) Compter le nombre de points m_{ij} de la matrice vérifiant les conditions suivantes :

2. Les solutions d'un polynôme du second degré de type $ax^2 + bx + c = 0$ sont de type : $x = \frac{-b+/-\sqrt{b^2-4ac}}{2a}$.

1. $m_{ij} = 0$
2. $m_{i-1,j} = 1$ ou $m_{i+1,j} = 1$ ou $m_{i,j-1} = 1$ ou $m_{i,j+1} = 1$

Correction

Pour obtenir 1 avec une probabilité de 0,2, il suffit de tirer un nombre aléatoire N entre 1 et 5 inclus et de ne considérer que le cas où $N = 1$.

```
import random
N = 10
M = [ [  1 if random.randint(1,5) == 1 else 0 for i in range (N) ] for j in range(N) ]
for l in M : print l

nb = 0
for i in range(N) :
    for j in range (N) :
        if   i > 0   and M[i-1][j] == 1 : nb += 1
        elif i < N-1 and M[i+1][j] == 1 : nb += 1
        elif j > 0   and M[i][j-1] == 1 : nb += 1
        elif j < N-1 and M[i][j+1] == 1 : nb += 1
print nb
```

Cela donne pour un exemple :

```
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 1, 1]
[0, 0, 0, 0, 1, 0, 1, 1, 0, 1]
[0, 0, 1, 0, 1, 0, 0, 1, 1, 1]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
46
```

Ce n'est pas évident de vérifier qu'on ne s'est pas trompé. Un moyen simple consiste à prendre une valeur de N plus petite.

```
[0, 1, 0]
[0, 0, 1]
[0, 0, 0]
4
```

3

On considère une matrice 10×10 remplie de nombres entiers aléatoires tirés entre 0 et 100. On apelle M cette matrice.

1) Créer une autre matrice N qui vérifie : $N_{ij} = \frac{M_{ij}}{\sum_{i=1}^{10} M_{ij}}$. Le module `numpy` simplifie l'écriture du programme.

Correction

```
import random, numpy
N  = 10
M  = [ [  1.0 if random.randint(1,5) == 1 else 0.0 for i in range (N) ] for j in range(N) ]
M  = numpy.matrix(M)
MM = numpy.matrix(M)
for i in range (N) :
    s = numpy.sum(M[i,:])  # ou M[i,:].sum()
    if s > 0 : MM [i,:] = M [i,:] / s
print MM
```

Le dernier exercice cache deux pièges. Le premier est le problème des divisions entières. Si on remplace `1.0` et `0.0` par 1 et 0 sur la troisième ligne, tous les nombres manipulés deviennent entiers. La matrice `MM` est alors peuplée de 0 et de 1 uniquement. Le second piège intervient quand on pense avoir résolu le premier en forçant une division réelle en multipliant d'un côté par `1.0` :

```
import random, numpy
N  = 5
M  = [ [  1 if random.randint(1,5) == 1 else 0 for i in range (N) ] for j in range(N) ]
M  = numpy.matrix (M)
MM = numpy.matrix(M)
for i in range (N) :
    s = numpy.sum(M[i,:])
    if s > 0 : MM [i,:] = M [i,:]*1.0 / s   # multiplication par 1.0
print MM
```

Comme initialement, on a créé la matrice `M` avec des entiers, le module `numpy` refuse l'ajout ultérieur de nombres réels. On peut regretter que le module `numpy` soit aussi strict ou ne jette une erreur indiquant au programmeur qu'il s'est trompé. Le fait même de vérifier les types des objets contenus dans la matrice et ceux qu'on lui injecte aurait le désavantage de ralentir les calculs.