Oraux 2022 de Centrale avec Python. Corrigé

On charge d'abord les modules dont on a besoin (en prenant les mêmes alias que dans les documents).

```
from math import *
from scipy import *
import numpy as np
import matplotlib.pyplot as plt

import numpy.linalg as alg
import scipy.integrate as integr
import scipy.optimize as resol
import numpy.random as rd
from numpy.polynomial import *
```

1) a)

```
def essai(n,p) :
     s = 0
     for k in range(n) :
         s = s+rd.binomial(1,p(k))
        ### variante :    if rd.random() >= p(k) :  s = s+1
     if s % 2 == 0 :  return 1
     else :  return 0
def pi(n,p) :
     s = 0 ; N = 1000
     for i in range(N) : s = s + essai(n,p)
     return s/N
```

Exemple :

```
def p(x) :  return 1/(1+x)**2
print(pi(50,p)
```

b)

```
N = [ n for n in range(20)]
Y = [ pi(n,p) for n in N]
plt.plot(N,Y) ; plt.show()
```

On obtient une courbe convergeant (par oscillations chaotiques) vers une constante 0.39....

c) Par les probas totales, $\pi_n = P(X_n \text{ pair}) = p_n P(X_{n-1} \text{ impair}) + (1-p_n)P(X_{n-1} \text{ pair})$.

Donc $\pi_n = p_n(1-\pi_{n-1}) + (1-p_n)\pi_{n-1} = (1-2p_n)\pi_{n-1} + p_n$.

Donc $\pi_n - \frac{1}{2} = (1-2p_n)\left(\pi_{n-1} - \frac{1}{2}\right)$, et ainsi $\pi_n - \frac{1}{2} = \frac{1}{2}\prod_{k=1}^{n}(1-2p_k)$, car $P(X_0 \text{ pair}) = 1$.

c)

```
x = 1/2 ; Z = [1/2]
for k in range(1,101) :   x = x*(1-2*p(k)) ; Z.append(1/2-x)
plt.plot(N,Z) ; plt.show()
```

On retrouve une suite convergant vers 0.39...

d) Il suffit d'étudier $\ln(\pi_n)$, valable car tous les termes du produit sont strictement positifs.

2) a)

```
def matriceA(n) :
    A = np.zeros((n+1,n+1))    # on part de la matrice nulle, qu'on modifie ensuite
    for i in range(n) :   A[i,i+1] = i+1 ; A[i+1,i] = n-i
    return A
```

b)

```
A = matriceA(3) ; V,P = alg.eig(A) ; print(V)
```

On obtient -3, -1, 3 et 1 comme valeurs propres réelles distinctes de A_3, donc A diagonalisable.

```
print(dot(alg.inv(P),np.dot(matriceA(3),P)))
# renvoie la matrice diagonale : les matrices hors-diagonale sont de l'ordre ≤ 10^-16 ....
```

Remarque culturelle :
Si $ab > 0$, la matrice $\begin{pmatrix} 0 & a \\ b & 0 \end{pmatrix}$ est semablable à $\begin{pmatrix} 0 & a\alpha \\ b/\alpha & 0 \end{pmatrix}$, et on prend $\alpha = \sqrt{\frac{b}{a}}$.
En prenant de façon générale $D = \text{Diag}(\alpha_1, ..., \alpha_n)$, on obtient $D^{-1}AD$ symétrique, donc diagonalisable.

c) La loi de N_o est constante de valeur j, où j est le nombre de boules dans la première urne.

d) On pose $T_k = 1$ si le k-ième déplacement se fait de la première urne vers la seconde, et 0 sinon.

On utilise la formule des probas totales selon la valeur de Z_k.

e) On a $Z_{k+1} = \frac{1}{n}AZ_k$. On prend ici $Z_0 = E_1$.

```
def loi(k,n) :
    Z = np.zeros(n) ; Z[0] = 1
    Z = np.array([1/(n+1)]*(n+1))
    return dot(alg.matrix_power(1/n*A,k),Z)
```

Par exemple, `loi(50,3)` renvoie la loi binomiale `[ 0.125 0.375 0.375 0.125]`.

Il y a convergence vers la loi binomiale $\mathcal{B}(n, \frac{1}{2})$, qui est la loi invariante dans cette chaine de Markov.

f) $\varphi_n(P) = P' + X(nP - XP')$.

3) a) et c) Th de la bijection avec $f_n(x) = x + x^2 + ... + x^n$ et pour $n \geq 2$, $f_n(0) < 1 < f_n(1)$.

b) $n! \sim \sqrt{2\pi n} n^n e^{-n}$.

c) $f_{n+1}(u_n) > f_n(u_n) = 0 = f_{n+1}(u_{n+1})$, donc $u_n > u_{n+1}$, donc $(u_n)_{n\geq 1}$ décroissante.

d) e)

```
def u(n) :
    def f(x) :
        s = -1
        for k in range(1,n+1) :  s = s + x**k
        return s
    return resol.fsolve(f,1)[0]
print(u(50))   ### proche de 0.5
print(2**20*(u(20)-0.5))   ### proche de 0.25
```

Attention : **ne pas choisir n trop grand, car sinon erreurs d'arrondis**

f) $f_n(x) = x\dfrac{1-x^n}{1-x} - 1$.

Comme $f_n(y_n) = 0$, alors $u^{n+1} - 2u_n + 1 = 0$, c'est-à-dire $u_n^{n+1} = 2(u_n - L)$.

Donc $(u_n - L) = O(u_n^{n+1}) = O(k^n)$, où $k = \frac{3}{4}$, donc a fortiori $u_n - L = \mathfrak{o}(1/n)$,

Donc $u_n^{n+1} = (L + O(k^n))^{n+1} = (L + \mathfrak{o}\,(1/n))^{n+1} \sim L^{n+1}$.

On en déduit $(u_n - L) = \frac{1}{2}u_n^{n+1} \sim \frac{1}{2}L^{n+1}$, d'où $\lim_{n\to+\infty} v_n = \frac{1}{4}$.

4) a)

```
def F(x) :
    def g(t) :  return exp(-t*x)/(1+t)
    return(integr.quad(g,0,np.inf)[0])
X = np.arange(0.1,50,0.1) ; Y = [F(x) for x in X]
plt.plot(X,Y) ; plt.show()
```

Domaine de définition : $]0, +\infty[$, $\lim_{x\to+\infty} F(x) = 0$ car $F(x) \leq \int_0^{+\infty} \exp(-tx)\ dt = \dfrac{1}{x}$.

b) On a $F'(x) = -\int_0^{+\infty} \dfrac{t}{1+t}\ \exp(-tx)\ dt = -\int_0^{+\infty} \exp(-tx)\ dt + \int_0^{+\infty} \dfrac{\exp(-tx)}{1+t}\ dt = -\dfrac{1}{x} + F(x)$.

5) a) Pour calculer le nombre d'éléments distincts, le plus simple est d'utiliser un dicitonnaire.

```
def different(L) :
    dico = {}
    for x in L : dico[x] = 1
    return len(dico)
```

Exemple : `print(different([1,3,5,1,3,3]))` renvoie 3

Autre méthode : On peut aussi dans le cas d'entiers, classer les éléments, puis les compter :

```
def different(L) :
    L = sorted(L) ; n = len(L) ; s = 1
    for k in range(1,n) :
        if L[k]>L[k-1] :  s = s+1
    return s
```

b)

```
def simulX(n) :
    L = []
    for k in range(n) :  L.append(rd.randint(1,n+1))
    return n-different(L)
```

c) d) e) Y_i suit la loi de Bernoulli $\mathcal{B}((1-\frac{1}{n})^n)$, donc $E(X_n) = \sum_{i=1}^{n} E(Y_i) = n(1-\frac{1}{n})^n \sim ne^{-1}$.

f)

```
def espeVarX(n) :
    s = 0 ; t = 0 ; N = 1000
    for k in range(N) :
        d = simulX(n) ; s = s + d ; t = t + d**2
    return(s/N,t/N-(s/N)**2)
```

g) $\mathrm{Cov}(Y_i, Y_j) = (1-\frac{2}{n})^n - (1-\frac{1}{n})^{2n}$, et on conclut avec $V(X_n) = nV(Y_1) + n(n-1)\,\mathrm{Cov}(Y_1, Y_2)$.

6) a) grad $f(x,y) = (3x^2 - 3, 6xy)$ et les points critiques sont $(1,0)$ et (-1, 0).

b)

```
def f(x,y) :  return x**3-3*x*(1+y**2)
f = np.vectorize(f)
X = np.arange(-2, 2, 0.01)
Y = np.arange(-2, 2, 0.01)
X, Y = np.meshgrid(X, Y) ; Z = f(X, Y)
C = np.arange(-3,3.5,0.5)
plt.axis('equal')
plt.contour(X, Y, Z, C)
plt.show()
```

c)

```
def h(t) :  return f(1+t,t)
T = np.arange(-1,1,0.1)
HT = [h(t) for t in T]
```

```
plt.plot(T,S) ; plt.show()
```

Variante pour les trois dernières lignes :

```
h = np.vectorize(h) ; plt.plot(T,h(T)) ; plt.show()
```

On voit que h n'admet pas d'extremum en $t = 0$, donc $(1, 0)$,n'est pas un estremum local de f.

d) La Hessienne $H(x, y) = \begin{pmatrix} 6x & -6y \\ -6y & 6x \end{pmatrix}$, donc $H(1, 0) = \begin{pmatrix} 6 & 0 \\ 0 & -6 \end{pmatrix}$.

La Hessienne admet à la fois une valeur propre positive et une valeur propre négative.

Donc f n'admet pas en $(1, 0)$ un extremum local (il s'agit en fait d'un point col).

e) On a $f(-x, y) = -f(x, y)$, donc f admet n'admet pas en $(1, 0)$ un extremum local.

f) B compact et f n'a pas de point critique à l'intérieur de B.

g)

```
plt.clf()  # permet de réinitialiser la fenêtre graphique "clear figure"
plt.contour(X, Y, Z, [f(-2+0.4*k,0) for k in range(11)])
T = np.arange(0,2*np.pi,0.1) ; plt.plot(np.cos(T),np.sin(T))
plt.show()
```

h) f n'atteint pas ses extrema (car les seuls points critiques sont des points cols). En fait, on a $\sup f = \lim_{x\to+\infty} f(x, 0) = +\infty$ et $\inf f = \lim_{x\to-\infty} f(x, 0) = -\infty$.

7) a)

```
def mat(a,L) :
    n = len(a) ; A = np.zeros((n,n))
    for i in range(n) :  A[i,i] = a[i]
    for i in range(n-1) :  A[i,i+1] = 1 ; A[i,n-1] = L[i]
    return A
print(mat([1,2,3],[1,2])) # on teste sur un exemple
```

d) Base de degrés échelonnés.

e) On construit les polynômes P_k. On prend $Q = \prod_{i=0}^{n-1}(X - \lambda_i)$ et on détermine μ_k par récurrence forte descendante en considérant le coefficient dominant de $Q + \sum_{i=k+1}^{n-1} \mu_i P_i$. On a notamment $\mu_n = 1$.

On a $\sum_{i=0}^{n-1} \lambda_i = \sum_{i=0}^{n-1} a_i$, donc $\prod_{i=0}^{n-1}(X - \lambda_i)$ et $\prod_{i=0}^{n-1}(X - a_i)$ ont les mêmes termes en X^{n-1} et X^n.

Donc $\mu_{n-1} = 0$, ce qui justifie qu'on se limite à $\sum_{i=k+1}^{n-2} \mu_i P_i$.

```
def decompose(a,Y) :
    n = len(a)
    P = [0]*(n+1) ; P[0] = Polynomial([1])
    for k in range(n) :  P[k+1] = P[k]*Polynomial([-a[k],1])
```

```
    Q = Polynomial([1])
    for i in range(n) :  Q = Q*Polynomial([-Y[i],1])
        # mieux vaut vérifier que c'est OK : print(Q) ; print(P[n])
    L = [0]*n ; Q = Q - P[n]
    for k in range(n-2,-1,-1) :
        if Q.degree() < k :  L[k] = 0
        else :  L[k] = - Q.coef[k]
        # il faut faire attention si  L[k] est nul
    Q = Q + L[k]*P[k]
    return L
```

f)

```
a = [2,2,2] ; Y = [1,2,3]
L = decompose(a,Y) ; A = mat(a,L)  # L vaut [0,1]
print(np.poly(A))    # attention, renvoie Les coefficients de χ_A dans l'ordre décroissant.
```

On obtient bien la liste $[1, -6, 11, -6]$ qui correspond bien au polynôme $(X-1)(X-2)(X-3)$.

g) Par récurrence en développant $\det(XI_n - A)$ selon la première colonne, ou bien sans récurrence en développant selon la dernière ligne.