Epreuve orale Math-info de Centrale. Oraux choisis. Corrigé

```
from math import * ; from scipy import *
import numpy as np   # on peut ajouter si on préfère : from numpy import *
import matplotlib.pyplot as plt
```

Exercice A

1) On considère le système d'ordre 1 associé : $\begin{cases} y_0' = y_1 \\ y_1' = -y_1/x - y_0 \end{cases}$, et on représente les (x, y_0)

```
import scipy.integrate as integr

def f(Y,x) :  return np.array([Y[1],-Y[0]-Y[1]/x])
X = np.arange(0.001,20,0.01) ; Y0 = np.array([1,0]) # prendre 0.001 car sinon problème en 0

V = np.array(integr.odeint(f, Y0 , T))
Y = V[:,0]
plt.plot(X,Y) ; plt.show()
```

2) On approche $B(x)$ par $B_N(x) = \sum_{n=0}^{N-1} (-1)^n \dfrac{1}{4^n (n!)^2} x^{2n}$, avec $|B(x) - B_N(x)| \leq \dfrac{x^{2N}}{4^n (N!)^2}$ par le CSSA (le terme général est décroissant pour $n \geq N$ lorsque $x^2 \leq 4N^2$, ce qui est le cas dans la valeur qu'on va trouver ensuite pour $x = 20$ et $N = 37$).

On prend donc N tel que $\dfrac{20^{2N}}{4^n (N!)^2} \leq 10^{-12}$, c'est-à-dire $\dfrac{10^N}{N!} \leq 10^{-6}$.

```
N = 1 ; x = 10
while x > 10**(-6) :  N = N+1 ; x = N*10/n
print(N)   # renvoie 37

def B(x) :
     s = 1 ; a = 1 ; m = 1
     for n in range(1,N) :
     a = (- a) * x**2 / 4 / n **2 ; s = s + a
              # on calcule les coefficients a_n en utilisant la récurrence a_0 = 1 et a_n = -a_{n-1} x^2/(4n^2)
     return(s)
print(B(2))   #   test

X = [ 0.1*k for k in range(201) ]
B = np.array([B(x) for x in X])
plt.plot(X,B) ; plt.show()
```

Exercice B

On considère $f_n(x) = \sum_{k=1}^{n} \dfrac{x^k}{k} - 1$.

1) On a $f_n(0) = -1$, $f_n(1) > 0$ et $f_n' > 0$ sur $[0, 1]$. Donc x_n existe et est unique.

2)
```
def f(n,x) :
     s = 0 ; y = 0
     for k in range(1,n+1) :  s = s+x**/k
     return s-1
```

Remarque : La fonction `f(n,x)` renvoie $\sum_{k=1}^{n} \dfrac{x^k}{k} - 1$.

```
import scipy.optimize as resol
import scipy.optimize as resol
def xx(n) :
     def g(x) :  return f(n,x)  # il faut définir f_n comme une fonction à une variable x
```

```
    y = resol.fsolve(g,0)     # 0 est la valeur initiale (méthode de Newton)
    return y[0]    # attention : fsolve renvoie une liste
print(xx(50))    # on vérifie déjà pour une valeur (qui ici est proche de la limite L ≃ 0.64)
N = [n for n in range(1,51)]
X = [xx(n) for n in N ]
plt.plot(N,X)
```

3) On a $f_{n+1}(x_{n+1}) = 0 = f_n(x_n) < f_{n+1}(x_n)$, et comme f_{n+1} croît strictement, on a $x_{n+1} < x_n$.

Donc $(x_n)_{n\in\mathbb{N}^*}$ est décroissante et minorée, donc elle converge.

4) Il suffit de prendre $\alpha = x_2$, car $(x_n)_{n\geq 2}$ est décroissante.

5) cf cours DSE : on a $\lim_{n\to+\infty} f_n(x) = -\ln(1-x) - 1$.

6) En fait, on va utiliser implicitement la cv uniforme de $(f_n)_{n1}$ vers f sur $[0,\alpha]$:

On a $\forall x \in [0,\alpha]$, $|f(x) - f_n(x)| = \sum_{k=n+1}^{+\infty} \frac{x^k}{k} \leq \sum_{k=n+1}^{+\infty} \frac{\alpha^k}{k} = f(\alpha) - f_n(\alpha)$.

En particulier, $|f(x_n)| = |f(x_n) - f_n(x_n)| \leq f(\alpha) - f_n(\alpha)$ qui tend lorsque n tend vers $+\infty$.

D'où par passage à la limpite et continuité de f en L, on a : $|f(L)| = 0$, c'est-à-dire $f(L) = 0$.

Donc $\ln(1-L) = -1$, c'est-à-dire $L = 1 - 1/e$.

Avec PYTHON, on vérifie la valeur trouvée précédemment : `print(1-exp(-1))` affiche 0.63...

Exercice C

1) Le principe est le suivant : On définit une matrice M, puis on regarde si la matrice de passage renvoyée par la fonction `eig(M)` est inversible (dans ce cas, M est inversible).

Remarque : Attention, il convient d'écrire $|\det(M)| < 10^{-6}$ plutôt que $\det(M) = 0$.

On utilise une fonction `essai()` qui fait *un* test (et renvoie 1 si M diagonalisable), et ensuite on moyenne sur 1000 essais. On obtient donc le programme suivant :

```
import numpy.linalg as alg ; import numpy.random as rd
def essai() :
    X,Y = rd.geometric(0.5,2) ; M = np.array([[0,X-Y],[X+Y,0]])
    V,P = alg.eig(M)
    if abs(alg.det(P)) < 10**(-6) :
        return 0
    else :  return 1
print(essai()) # on fait un test pour s'assurer que le programme tourne
s = 0
for i in range(1000) :  s = s + essai()
print(s/1000) # renvoie 0.69
```

2) Le polynôme caractéristique de M est $\lambda^2 - \Delta$, où $\Delta = X^2 - Y^2$.

Il admet deux racines distinctes ssi $\Delta \neq 0$. Dans ce cas M est diagonalisable dans $\mathcal{M}_2(\mathbb{C})$.

Il admet $\lambda = 0$ comme racine double si $\Delta = 0$. Si M était diagonalisable, elle serait la matrice nulle, ce qui est impossible ici car X et Y ne peuvent être nuls (attention, en effet, elles sont à valeurs dans $\mathbb{N}^*$).

Donc M est diagonalisable ssi $\Delta \neq 0$, donc $1 - r = \sum_{n=1}^{+\infty} P(X = n, Y = n) = \sum_{n=1}^{+\infty} (q^{n-1}p)^2 = \frac{p^2}{1-q^2} = \frac{p}{1+q}$.

On en conclut que $r = 1 - \frac{p}{1+q} = \frac{2q}{1+q}$. Lorsque $p = q = \frac{1}{2}$, on obtient $r = \frac{2}{3} \approx 0.67$

3) M est diagonalisable dans $\mathcal{M}_2(\mathbb{R})$ ssi $\Delta > 0$, c'est-à-dire $X > Y$.

On a $P(X > Y) = \sum_{n=1}^{+\infty} P(Y = n)P(X > n) = \sum_{n=1}^{+\infty} (q^{n-1}p)(q^n) = \frac{pq}{1-q^2} = \frac{q}{1+q} = \frac{1}{2}r$.

Remarque : Le résultat $\frac{1}{2}r$ est logique, car $X^2 - Y^2$ et $Y^2 - X^2$ ayant même loi, on a $P(\Delta > 0) = P(\Delta < 0)$.

Exercice D

```
import scipy.integrate as integr

def f(x) :
     def g(t) :
         return exp(-t*x)/(1+t)
     return g

def F(x) :
     return integr.quad(f(x),0,inf)[0]

X = [0.01+0.1*k for k in range(101)]
Y = [F(x) for x in X]
plt.plot(X,Y)
```

On conjecture que f est strictement décroissante de $]0,+\infty[$, que $\lim_{x\to 0} F(x) = +\infty$ et $\lim_{x\to+\infty} F(x) = 0$.

On a (cf cours de dérivation des intégrales) $F'(x) = -\int_0^{+\infty} \frac{t}{1+t}\exp(-tx)\ dt < 0$.

On a $\frac{t}{1+t} = 1 - \frac{1}{1+t}$, donc $F'(x) = F(x) - \frac{1}{x}$. On en déduit F de classe C^∞.

Remarque : *Preuve mathématique* : $0 \le F(x) \le \int_0^{+\infty} \exp(-tx)\ dt = \frac{1}{x}$, donc $\lim_{x\to+\infty} F(x) = 0$.

Par une IPP, on a $F(x) = \int_0^{+\infty} \frac{\exp(-u)}{x+u}\ du = -\ln(x) + \int_0^{+\infty} \ln(x+u)\exp(-u)\ du$.

Par cv dominée, $\lim_{x\to 0} \int_0^{+\infty} \ln(x+u)\exp(-u)\ du = \int_0^{+\infty} \ln(u)\exp(-u)\ du$ donc $F(x) = -\ln(x) + O(1)$.

Exercice E

1) $E(X) = \frac{1}{r+1}\sum_{k=0}^{r} k = \frac{r}{2}$ et $G_X(z) = \frac{1}{r+1}(1 + z + ... + z^r)$ et $G_{S_{r,n}}(z) = (G_X(z))^n$.

2) `import numpy.random as rd ; from numpy.polynomial import *`

a)
```
def S(r,n) :
     L = [1/(r+1) for i in range(r+1)]
     return (Polynomial(L)**n).coef

def loi(r,n) :
     L = S(r,n)
     T = [k for k in range(n*r+1)] ; plt.plot(T,L)
loi(2,5) ; plt.show()     # test
```

b)
```
def uniforme(r) :  return rd.randint(r+1)

def simul(r,n) :
     s = 0
     for i in range(n) :  s = s + uniforme(r)
     return(s)

def loi2(r,n) :
     T = [k for k in range(n*r+1)]
     V = [0]*(n*r+1)
     N = r*n*100
     for k in range(N) :
         V[simul(r,n)] += 1/N
     plt.plot(T,V)
loi2(2,5) ; plt.show()     # test
```

3) a)

```
def somme(k,L) :
```

```
    s = 0
    for i in range(min(k,len(L))) : s += L[i]
    return s
```

b)

```
def u(lam,r,n) :      # attention : lambda est un mot clé du langage PYTHON
    L = S(r,n)
    return somme(int(lam*r*n),L)
for lam in [0,1/4,1/3,1/2,2/3,3/4,1] ] :
    print( u(lam,3,20) )
```

On conjecture que $\lim_{n\to+\infty} u_{r,n}(\lambda) = 0$, $\frac{1}{2}$ ou 1 selon que $\lambda < \frac{1}{2}$, $\lambda = \frac{1}{2}$ ou $\lambda > \frac{1}{2}$.

c) On a par symétrie de la loi : $u_{p,n} = \frac{1}{2} + O\left(\frac{1}{\sqrt{p}}\right)$ si $\lambda = \frac{1}{2}$ (le terme correctif provient du terme central).

Montrons à l'aide de Bienaymé-Tchebychev que $\lim_{n\to+\infty} u_{p,n} = 0$ si $\lambda < \frac{1}{2}$ et $\lim_{p\to+\infty} u_{p,n} = 1$ si $\lambda > \frac{1}{2}$

On a $E(S_p) = \frac{1}{2}pn$. On a $V(S_p) = n\sigma^2$, où σ^2 ne dépend pas de n. Et $P(\left|S_p - \frac{1}{2}rn\right| \geq \gamma) \leq \dfrac{n\sigma^2}{\gamma^2}$.

On prend $\gamma = \varepsilon n$, avec $\varepsilon > 0$. Donc $\lim_{p\to+\infty} P((\frac{1}{2} - \varepsilon)n \leq S_{p,n} \leq (\frac{1}{2} + \varepsilon)n) = 1$, d'où on peut conclure.

Exercice F

```
import numpy.linalg as alg
from numpy.polynomial import *
```

Pour $n \in \mathbb{N}$, on pose $C_n = \binom{2n}{n}$.et $H_n = (C_{i+j-2})_{1\leq i\leq n+1, 1\leq j\leq n+1} \in \mathcal{M}_{n+1}(\mathbb{R})$.

1) H_n est réelle et symétrique.

2) Pour récupérer les coefficients binomiaux, on peut utiliser les coefficients de $(1+x)^n$.
(*Remarque* : On peut aussi calculer les $\binom{n}{k}$ par : $\binom{n}{k} = \prod_{j=0}^{k} \frac{n-j}{j+1}$. Eviter à tout prix les factorielles).

```
def binome(n) :
    p = Polynomial([1,1]) ; p = p**(2*n)
    return p.coef[n]
```

`print(binome(5))` renvoie le flottant 252.0 (qu'on pourrait convertir en entier via `int`)

```
def H(n) :
    M = np.zeros((n+1,n+1))
    for i in range(n+1) :
        for j in range(n+1) :
            M[i,j] = binome(i+j)
    return M
for n in range(1,4) :
    M = H(n)
    print("pour n=", n)
    print("det = ", n,alg.det(M), " ; valeurs propres = ", alg.eigvals(M))
```

On obtient :
pour $n = 1$, det = 2.0 ; valeurs propres = [0.298437886.70156212] ... etc ...

3) $P_iP_j = (1+X)^{2(i+j)}$, donc le coefficient de X^{i+j} vaut $\binom{2(i+j)}{i+j} = \sum_{a+b=i+j} \binom{2i}{a}\binom{2j}{b}$..

Remarque : $\sum_{a+b=i+j} \binom{2i}{a}\binom{2j}{b} = \sum_{a+b=i+j} \binom{2i}{a}\binom{2j}{2j-b} = \sum_a \binom{2i}{a}\binom{2j}{j-i+a}$.

On considère la matrice Q_n de la famille $(X^nP_0, X^{n-1}P_1, ..., P_n)$ dans la base canonique de $\mathbb{R}_{2n}[X]$.
Le coefficient (i,j) de $Q_n^TQ_n$ vaut bien $\sum_x \binom{2i}{x-n+i}\binom{2j}{x-n+j} = \sum_x \binom{2i}{a}\binom{2j}{j-i+a} = \binom{2(i+j)}{i+j} = C_{i+j}$.
La matrice Q_n est inversible car la matrice $(P_n, ..., X^{n-1}P_1, X^nP_0)$ est triangulaire inférieure inversible.