

Épreuves orales Math-info de Centrale. Petits exercices d'entraînement.

Modules incontournables à charger dès le début, puis on y ajoute les modules spécifiques

```
from math import *      # ici sans alias sinon on utilise math.log , etc ...
import numpy as np
import matplotlib.pyplot as plt
```

1) Suites numériques

a) On considère la suite $(r_n)_{n \in \mathbb{N}}$ définie par $r_0 = 1$ et $\forall n \in \mathbb{N}^*$, $r_n = 1 + \frac{1}{nr_{n-1}}$.

Représenter $(r_n)_{0 \leq n \leq 100}$. On vérifie ainsi numériquement que $\lim_{n \rightarrow +\infty} r_n = 1$.

Solution :

```
r = 1 ; L = [1] ; N = 100
for n in range(N+1) :
    r = 1 + 1 / (r*n) ; L.append(r) ;
X = [n for n in range(N+1)] ; plt.plot(X,L) ; plt.show()
```

b) On considère la suite $(u_n)_{n \in \mathbb{N}}$ définie par $u_0 = 1$, $u_1 = 1$ et $\forall n \in \mathbb{N}^*$, $u_{n+1} = u_n + \frac{1}{n}u_{n-1}$.

Écrire une fonction $u(n)$ calculant u_n (de façon efficace). Représenter $\left(\frac{u_n}{n}\right)_{1 \leq n \leq 100}$.

Solution :

```
def u(n) :
    x = 1 ; y = 1
    for k in range(1,n+1) : x,y = y,y + 1/k * x # affectations simultanées
    return x

X = [n for n in range(1,101)] ; Y = [u(n)/n for n in X ]
plt.plot(X,Y) ; plt.show()
```

c) Conjecture concernant le rayon de convergence R de $\sum u_n x^n$?

Solution : En posant $r_n = \frac{u_{n+1}}{u_n}$, on a $r_n = 1 + \frac{1}{nr_{n-1}}$.

On visualise par a) que la suite $\lim_{n \rightarrow +\infty} r_n = \leq 1$, donc par d'Alembert, $R = 1$.

Remarque mathématique : On a $r_n \geq 1$, donc $1 \leq r_n \leq 1 + \frac{1}{n}$, donc (cf a)) $\lim_{n \rightarrow +\infty} r_n = 1$.

Remarque culturelle : En fait, on a $u_n \sim \lambda n$. En effet, on note que u_n est croissante et on écrit que pour n assez grand, $r_n \in [1, 1 + \varepsilon]$, ce qui permet de comparer u_n avec v_n défini par $v_{n+1} = v_n (1 + \frac{1}{n})$, et on a $v_n = v_1 \exp(\sum_{k=1}^n \frac{1}{k}) = v_1 \exp(\ln n + \gamma + o(1)) \sim \lambda n$.

2) Suites de zéros

On considère x_n l'unique réel positif vérifiant $\sum_{k=0}^n \frac{x^k}{k!} = 2$.

Écrire une fonction `x(n)` renvoyant x_n . Conjecture sur la valeur de $L = \lim_{n \rightarrow +\infty} x_n$?

Indication : Construire les factorielles en exploitant $\frac{x^{k+1}}{(k+1)!} = \frac{x^k}{k!} \times \frac{x}{k}$.

Utiliser la fonction `fsolve` du module `scipy.optimize` as `resol`

Attention, `resol.fsolve(f,a)` renvoie dans tous les cas une liste de zéros de f , éventuellement vide.

La valeur a correspond à la valeur initiale dans la méthode de Newton.

Il est nécessaire parfois de changer de valeur de a pour que l'algorithme aboutisse à la solution cherchée.

Souvent, la liste contient un unique élément, on récupère sa valeur par `resol.fsolve(f,a)[0]`.

Solution :

```
import scipy.optimize as resol

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

print(x(20)) ; print(log(2)) # renvoie une valeur proche de ln 2.
```

Remarque mathématique : On peut s'attendre à ce que L vérifie $\sum_{k=0}^{+\infty} \frac{L^k}{k!} = 2$, d'où $L = \ln 2$.

Pour le justifier : on a $P_n(L) < 2$ et $P_n(L + \varepsilon) \rightarrow e^{L+\varepsilon}$, donc > 2 pour n assez grand, où $P_n(x) = \sum_{k=0}^n \frac{x^k}{k!}$.

3) Intégrales et intégrales paramétrées. On considère $F(x) = \int_0^{+\infty} \frac{\exp(-tx)}{1+t} dt$.

Écrire une fonction `F(x)` renvoyant $F(x)$. Représenter le graphe de F sur $]0, 50]$.

Indication : Utiliser `quad` du module `scipy.integrate`

Attention, `integr.quad(f,a,b)` renvoie un couple dont le premier élément est une valeur approchée de $\int_a^b f$ et le second est une majoration de l'erreur.

En pratique, on utilise donc `integr.quad(f,a,b)[0]`.

Utiliser aussi `np.inf` pour représenter la borne $+\infty$.

Attention à ne pas considérer la valeur en 0 qui diverge : on prendra $[0.1, 50]$.

Solution :

```
import scipy.integrate as integr

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()
```

4) Matrices diagonalisables

a) Définir une fonction **Matrice** qui étant donnée une liste $a = [a_0, \dots, a_{n-1}]$ renvoie la matrice compagnon

$$A = \begin{pmatrix} 0 & & & -a_0 \\ 1 & 0 & & -a_1 \\ & 1 & 0 & \vdots \\ & & 1 & 0 & -a_{n-2} \\ & & & 1 & -a_{n-1} \end{pmatrix} \in \mathcal{M}_n(\mathbb{R})$$

b) Sur l'exemple $a = [0, 1, 2, \dots, 9]$, vérifier que $\chi_A(X) = X^n + \sum_{k=0}^{n-1} a_k X^k$.

c) On prend $a = [1, 1, 0]$.

Trouver $P \in GL_3(\mathbb{C})$ et $D \in \mathcal{M}_3(\mathbb{C})$ diagonale telles que $P^{-1}AP = D$ est diagonale.

Suggestions : a) Pour définir une matrice, partir de la matrice nulle obtenue avec `zeros((n,n))`, et ensuite modifier les coefficients un par un (à l'aide d'une boucle `for`).

b) On utilise la fonction `np.poly` : attention, elle renvoie le vecteur des coefficients dans l'ordre décroissant (contrairement à ce qui se passe dans la représentation des polynômes du module polynomial).

c) Utiliser la fonction `eig(A)` du module `numpy.linalg` qui renvoie un couple (V, P) , où V est le vecteur dont les coefficients sont les racines du polynôme caractéristique et P une matrice inversible telle que $P^{-1}AP = D$ (lorsque $\det P$ est nul numériquement, A n'est pas diagonalisable).

Solution :

a)

```
def Matrice(a) :
    n = len(a) ; A = np.zeros((n,n))
    for i in range(n-1) : A[i+1,i] = 1
    for i in range(n) : A[i,n-1] = -a[i]
    return A

print(Matrice([2,3,4]))

b) a = list(range(0,10)) ; print(np.poly(Matrice(a)))

c) import numpy.linalg as alg
a = [1,1,0] ; A = Matrice(a)
V,P = alg.eig(A)          # conseils : donner des noms aux objets
D = np.dot(alg.inv(P),np.dot(A,P)) ; print(D)
```

Remarque : Les nombres obtenus sont des complexes (de la forme $0.2 + 0.001j$), proches de 0 pour les coefficients non diagonaux (les coefficients non diagonaux sont ici “numériquement nuls”).

5) Simulations d'une variable aléatoire

On considère X_n somme de n v.a. indépendantes de Bernoulli de paramètres $\frac{1}{k}$, avec $1 \leq k \leq n$.

a) Écrire une fonction réalisant une simulation : `essai(n)` renvoie un entier représentant $X(\omega)$.

b) Évaluer l'espérance et la variance de X_n pour $n \in [10, 50, 100]$. On prendra $N = 1000$ essais.

c) Représenter pour $n = 50$ la loi de X_n sur $\llbracket 0, 30 \rrbracket$, c'est-à-dire le graphe de $(P(X = i))_{0 \leq i \leq 30}$.

d) Retrouver la loi de X_n en utilisant la série génératrice $G_X(z) = \prod_{k=1}^n \left(1 - \frac{1}{k} + \frac{1}{k}z\right)$.

Suggestion : a) Il est conseillé de définir d'abord une fonction aléatoire de Bernoulli `B(p)`.

b) Attention à bien prendre les mêmes essais pour l'évaluation de $E(X)$ et de $E(X^2)$.

c) Construire un tableau V des occurrences de X_n : Pour $0 \leq i \leq 30$, on incrémente $V[i]$ lorsque X_n vaut i .

Puis on représente graphiquement le tableau des $V[i]/N$, pour $0 \leq i \leq 30$.

d) On pourra faire l'exo 6) d'abord.

Solution

```
a) import numpy.random as rd
```

```
def B(p) :
```

```
    return rd.binomial(1,p)
```

```
def simulX(n) :
```

```
    s = 0
```

```
    for k in range(1,n) :    s = s + B(1/k)
```

```
    return s
```

```
print(simulX(20))
```

b)

```
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)
```

c)

```
V = [0]*31 ; N = 1000
```

```
for k in range(N) :
```

```
    i = simulX(n)
```

```
    if i <= 30 :    V[i] = V[i] + 1
```

```
Y = [i for i in range(31)]
```

```

L = [V[i]/N for i in Y]
plt.plot(Y,L) ; plt.show()

d)

from numpy.polynomial import *      # attention, ici, pas d'alias
def loi(n) :
    P = Polynomial([1])
    for k in range(n) : P = P*Polynomial([1-1/k,1/k])
    return P.coef
Y = [i for i in range(31)]
n = 50 ; L = loi(n) ; Z = [L[i] for i in range(31)]
plt.plot(Y,L) ; plt.show()

```

5) bis) Simulations d'une variable aléatoire

On considère $S_n = \sum_{k=1}^n X_k$, où les X_k sont i.i.d. avec $P(X = 1) = p$ et $P(X = -1) = 1 - p$.

a) Écrire une fonction réalisant une simulation : **somme(n,p)** renvoie un entier représentant $X(\omega)$.

b) Écrire une fonction **positif(n,p)** évaluant numériquement la probabilité a_n que $S_n > 0$.

c) Exprimer la loi de S_n mathématiquement, et montrer que si $p > \frac{1}{2}$, $\lim_{n \rightarrow +\infty} a_n = 1$.

Solution : `import numpy.random as rd`

a) On commence par définir une fonction réalisant une v.a. aléatoire de loi X .

```

def x(p) :
    return 2*rd.binomial(1,p)-1

def somme(n,p) :
    s = 0
    for k in range(n) : s = s + x(p)
    return s

```

```

b) def positif(n,p) :
    N = 1000 ; c = 0
    for k in range(N) :
        if somme(n,p)>0 : c = c + 1
    return c/N

```

c) On a $S_n = 2T_n - n$, où T_n suit la loi binomiale $\mathcal{B}(n, p)$.

Donc $P(S_n = 2k - n) = P(T_n = k) = \binom{n}{k} p^k (1-p)^{n-k}$.

Posons $\mu_n = E(S_n) = n(2p - 1)$. Supposons $p > \frac{1}{2}$. Donc $\mu_n > 0$.

On a $P(S_n \leq 0) \leq P(|S_n - \mu_n| > \mu_n)$, donc par Bienaymé-Tchebychev, $P(S_n \leq 0) \leq \frac{V(S_n)}{\mu_n^2}$.

Comme $V(S_n) = nV(X)$, alors $V(S_n) = O(n)$, donc $P(S_n \leq 0) = O_{+\infty}\left(\frac{1}{n}\right)$, d'où $\lim_{n \rightarrow +\infty} a_n = 1$.

6) Polynômes (cf module Polynomial)

Créer une fonction `Pol(L)` qui étant donnée une liste $L = [a_0, \dots, a_{n-1}]$ de réels ou de complexes renvoie le polynôme $P = \prod_{i=0}^{n-1} (X - a_i)$.

Indication : On utilise le module `polynomial`. Commencer par définir une fonction `Monome(a)` qui renvoie $X - a$, puis utiliser la multiplication des polynômes. Noter que le polynôme constant 1 est obtenu avec `Polynomial([1])`.

Solution :

```
from numpy.polynomial import *      # attention, ici, pas d'alias

def Monome(a) : return Polynomial([-a,1])

def Pol(L) :

    P = Polynomial([1])

    for a in L : P = P*Polynomial([-a,1])

    return P.coef

# attention : on renvoie la liste des coefficients (et non le polynôme P, dont le code n'est pas très lisible)

# attention au fait que coef n'est pas une fonction ("not callable").

print(Pol([0,1,2]))    # on vérifie sur un exemple
```

7) a) Tracé des lignes de niveau d'une fonction de deux variables

On considère $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2 \quad (x, y) \mapsto x^2 - y^2$.

Représenter les lignes de niveau $f(x, y) = k$ pour $(x, y) \in [-2, 2]^2$ et $k \in \{-3, -2.5, \dots, 2.5, 3\}$.

Indication : Il faut utiliser le code de l'aide (sans chercher à comprendre le modus operandi ...).

Solution :

```
def f(x,y) : return x**2-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)

K = np.arange(-3,3.5,0.5)

plt.axis('equal')

plt.contour(X, Y, Z, K)

plt.show()
```

b) Tracé d'une courbe paramétrée

Représenter la courbe paramétrée $M(t) = (x(t), y(t))$, où $x(t) = \cos(t)$ et $y(t) = \sin(t)$.

Remarque : Utiliser `axis("equal")` de sorte à obtenir le cercle unité.

Solution : L'aide Centrale utilise les fonctions vectorisées `np.cos` `np.sin`, mais c'est inutile :

```
T = arange(0,2*np.pi+0.1,0.1)
X = [cos(t) for t in T] ; Y = [sin(t) for t in T]
plt.axis('equal') ; plt.plot(X, Y) ; plt.show()
```

Variante pour définir X et Y : on utilise les fonctions vectorisées `np.cos` et `np.sin` (au lieu de `cos` et `sin` du module `math`) : on prend donc `X = np.cos(T)` ; `Y = np.sin(T)`

c) Représenter sur un même graphe les deux courbes précédentes.

Solution :

Il faut reprendre les deux codes précédents, mais ne faire `plt.show()` **qu'une fois à la fin** (autrement dit, il suffit de supprimer le `plt.show()` du a)).

Remarque : `plt.show()` ajoute le nouveau graphe à la fenêtre graphique.

Pour réinitialiser une fenêtre graphique, on utilise `plt.clf()` qui signifie "clear figure".

d) Tracé d'une surface 3D

Tracer la surface $z = x^2 - y^2$ (surface en selle de cheval avec un point col).

Solution : Ne pas oublier de charger la fonction `Axes3D` ; recopier le code donné sur l'aide-mémoire

```
from mpl_toolkits.mplot3d import Axes3D
ax = Axes3D(plt.figure())
def f(x,y) :
    return x**2 - y**2
f=np.vectorize(f)
X = np.arange(-1, 1, 0.02)
Y = np.arange(-1, 1, 0.02)
X, Y = np.meshgrid(X, Y)
Z = f(X, Y)
ax.plot_surface(X, Y, Z)
plt.show()
```

8) Equations différentielles

Représenter sur $[0, 20]$ la fonction $y(t)$ vérifiant $y''(t) + ty(t) = 0$ et $y(0) = 1$ et $y'(0) = 0$.

Indication : On écrit l'équation sous la forme
$$\begin{cases} y'(t) = z(t) \\ z'(t) = -t y(t) \end{cases}$$

Recopier le code correspondant à ce même type d'équation : bien suivre les mêmes indices.

On utilise la fonction `odeint` du module `scipy.integrate`

Solution :

```
import scipy.integrate as integr

def f(Y,t) : return np.array([Y[1],-t*Y[0]])

T = np.arange(0,20,0.01) # discrétisation en temps

Y0 = np.array([1,0]) # valeurs initiales

V = np.array(integr.odeint(f, Y0 , T))

Y = V[:,0] # permet de récupérer la fonction y(t) ; on obtiendrait z(t) avec V[:,1]

plt.plot(X,Y) ; plt.show()
```

9) Évaluer numériquement la somme $S(p) = \sum_{n=p}^{+\infty} \binom{n}{p}^{-1}$ pour $p \in \{2, 3, \dots, 10\}$.

Suggestion : On pourrait commencer par définir une fonction `binome(n,p)`.

On peut aussi récupérer les coefficients binomiaux des coefficients du polynôme $(1 + X)^n$:

```
P = Polynomial([1,1])*n ; L = P.coef #
```

La fonction factorielle n'est pas une fonction standard en PYTHON. De toute façon, il convient d'éviter de calculer des factorielles (entiers trop grands). On pourrait d'ailleurs calculer les $\binom{n}{p}$ par récurrence en utilisant :
$$\binom{n}{p} = \frac{n(n-1)\dots(n-p+1)}{p!} = \prod_{i=0}^{p-1} \frac{n-i}{i+1}$$
 ou bien la formule du triangle de Pascal.

10) Matrices aléatoires et diagonalisation

On considère la matrice $M = \begin{pmatrix} 2 + X & Y - 2 - X \\ 1 & Y - 1 \end{pmatrix}$, où X et Y sont des v.a. de loi géométrique $\mathcal{G}(p)$.

a) Évaluer numériquement la probabilité $\mu(p)$ que M soit diagonalisable (dans $\mathbb{R}$).

Suggestion : Ecrire d'abord une fonction donnant une matrice aléatoire $M(p)$.

On note Δ le discriminant du polynôme caractéristique de M .

On a $\Delta = (\text{tr } M)^2 - 4 \det M$, donc $\Delta \geq 0$ ssi $4 \det M \leq (\text{tr } M)^2$.

On utilise : $V, P = \text{eig}(M)$ et M est diagonalisable (dans $\mathbb{R}$) lorsque $\Delta \geq 0$ et P est inversible.

En fait on a toujours $\Delta \geq 0$. On peut considérer que P est inversible si $|\det P| > 10^{-7}$.

Effectuer alors par exemple 1000 valeurs pour estimer $\mu(p)$.

b) Comparer avec le graphe de μ sur $[0, 1]$ avec celui de la fonction $F : p \mapsto \frac{2 - 2p + p^2}{2 - p}$.

Remarque culturelle mathématique : En fait, ici, les valeurs propres de M sont Y et $X + 1$, et elles sont réelles.

On vérifie alors aisément (...) que, comme souvent, M est diagonalisable ssi son polynôme caractéristique est scindé à racines simples, donc ssi $Y \neq X + 1$.

On obtient donc $\mu = 1 - \lambda$, où $\lambda = \sum_{n=0}^{+\infty} (q^n p)(q^{n+1} p) = \frac{qp^2}{1-q^2} = \frac{qp}{1+q} = \frac{(1-p)p}{2-p}$.