Correction – Séance Python

Préparation à l'oral Agro/Véto

BCPST 2_A — 2025/2026

– –

Imports utilisés dans toute la séance :

```
import random as rd
import numpy as np
from math import *
```

– –

Partie A — Exercices avec préparation

Thème 1 — Simulation d'une loi et loi des grands nombres

Ex 1 : 1) Simulation d'une loi exponentielle.
On utilise la méthode d'inversion : si $U \hookrightarrow \mathcal{U}([0,1])$, alors

$$X = -\frac{1}{\lambda}\ln(1-U) \hookrightarrow \mathcal{E}(\lambda).$$

Voir la démonstration dans le cours ou dans la feuille info 21.

```
def expo(lam):
    U = rd.random()
    return -1/lam * log(1 - U)
```

2) Simulation de $\min(X,Y)$ avec $X \hookrightarrow \mathcal{E}(\lambda)$ et $Y \hookrightarrow \mathcal{E}(\mu)$ indépendantes.

```
def minXY(lam, mu):
    X = expo(lam)
    Y = expo(mu)
    return min(X, Y)
```

3) Estimation de $\mathbb{E}[\min(X,Y)]$ par la loi des grands nombres.

```
def estime_E(lam, mu, N):
    total = 0
    for i in range(N):
        total += minXY(lam, mu)
    return total / N                # moyenne empirique -> esperance
```

4) Vérification numérique pour $\lambda = 1$, $\mu = 2$, $N = 100\,000$. On admet $\min(X,Y) \hookrightarrow \mathcal{E}(\lambda+\mu)$, d'espérance $\dfrac{1}{\lambda+\mu}$.

```
print(estime_E(1, 2, 100_000))     # doit etre proche de 1/3
```

$$\boxed{\mathbb{E}[\min(X,Y)] = \frac{1}{\lambda+\mu} = \frac{1}{3} \approx 0{,}333}$$

– –

Thème 2 — Chaînes de Markov discrètes

La chaîne $(Y_n)_{n\in\mathbb{N}}$ est à valeurs dans $\{0,1,2,3\}$, avec $Y_0 = 1$. Sachant $Y_n = k$, on a $Y_{n+1} \hookrightarrow \mathcal{B}\left(3, \frac{k}{3}\right)$.

Ex 2 : 1) Fonction de transition.

```
def transition(k):
    somme = 0
    for i in range(3):          # 3 tirages de Bernoulli de param. k/3
        if rd.random() < k/3:
            somme += 1
    return somme
```

2) Simulation d'une trajectoire $Y_0, Y_1, \ldots, Y_n$.

```
def chaine(n):
    L = [1]                             # Y0 = 1
    for i in range(n):
        L.append(transition(L[-1]))
    return L
```

3) Estimation de $\mathbb{E}[Y_n]$ par simulation de N chaînes.

```
def estime_E_Yn(n, N):
    total = 0
    for i in range(N):
        total += chaine(n)[-1]       # derniere valeur de la trajectoire
    return total / N

N =10000

for n in [1, 5, 10, 20]:
    print("n = ", n, ' E[Yn] -> ', estime_E_Yn(n, N) )
```

4) *(Bonus)* Probabilité d'absorption avant l'étape 10.
Une version naïve qui utilise chaine :

```
def proba_absorption(N):
    count = 0
    for i in range(N):
        traj = chaine(9)                        # avant 10
        if 0 in traj or 3 in traj:              # ou traj[-1] in [0, 3]
            count += 1
    return count / N
```

La version la plus naturelle utilise un `while` avec double condition d'arrêt : absorption *ou* dépassement de 10 étapes.

```
def proba_absorption(N):
    count = 0
    for _ in range(N):
        etat = 1
        n = 0
        while etat != 0 and etat != 3 and n < 10:
            etat = transition(etat) # on avance d'un pas
            n += 1
        if etat == 0 or etat == 3:  # absorption effective
            count += 1
    return count / N
```

Thème 3 — Matrices et numpy

Ex 3 : 1) Construction de $K_n \in \mathcal{M}_{n+1}(\mathbb{R})$.

```
def matrice_K(n):
    M = [ [0 for j in range(n+1)] for i in range(n+1)]
    for i in range(1, n+1):
        M[i-1][i] = i                    # sur-diag. : (K)_{i,i+1} = i
    for j in range(1, n+1):
        M[j][j-1] = -n-1+j               # sous-diag. : (K)_{j+1,j} = -n
            -1+j
    return M
```

Attention à la différence de numérotation des tableaux en Python et des matrices du cours de maths.

2) *A vérifier*

3) Valeurs propres via `numpy`.

Plus dur que prévu

```
def valeurs_propres(n):
    M = matrice_K(n)
    vp = np.linalg.eigvals(np.array(M))    # liste -> tableau numpy
    print("n = ", n)
    print('parties réelles', [round(v.real, 2) for v in vp])
    print('parties imaginaires',[round(v.imag, 2) for v in vp])

for n in range(1, 7):
    valeurs_propres(n)
```

4) Conjecture : les valeurs propres de K_n semblent être les imaginaiores purs $k\,i$ où k est un entier de même parité que n et vérifiant $|k| \leqslant n$

5) Matrice de Gram.

```
def gram(famille):
    n = len(famille)
    G = [ [0 for j in range(n)] for i in range(n)]
    for i in range(n):
        for j in range(n):
            G[i][j] = sum( [ famille[i][k] * famille[j][k]
                        for k in range(len(famille[i])) ] )
    return G
```

Thème 4 — Polynômes représentés par des listes

Ex 4 : *Personne n'a abordé cet exercice.*

Partie B — Exercices flash (style concours)

Ex 5 : Séquence ADN aléatoire.

```
def gene(n):
    bases = ['A', 'C', 'G', 'T']
    T = ''
    for i in range(n):
        k = rd.randint(0, 3)
        T = T + bases[k]
    return T
```

Revoir la manipulation des chaines de caractères.

```
def nbAC(g):
    count = 0
    for i in range(len(g) - 1):     # s'arrete a l'avant-dernier caractere
        if g[i] == 'A' and g[i+1] == 'C':
            count += 1
    return count

# Verification : nbAC('GAGCACCCTACTTGGCGCGA') -> 2
```

Ex 6 : Élément le plus fréquent.

Cette fonction rassemble deux situations plusieurs fois rencontrées cette année :

- Calculer dans un tableau le nombre d'apparitions des valeurs d'une liste

(Voir l'Exercice 3 dans Feuille_info_18)

- Recherche de "la" position du maximum dans une liste.

(Voir l'Exercice 1 dans Feuille_info_18)

```
def plus_frequent(L, k):
    effectifs = [0] * (k + 1)       # compteur pour chaque valeur de 0 a k
    for x in L:
        effectifs[x] += 1
                        # effectifs contient le nb d'occurences dans L
                        # des valeurs de 0 a k
    best = 0
    for i in range(1, k + 1):
        if effectifs[i] > effectifs[best]:
            best = i
    return best                                 # ou effectifs[best]

# plus_frequent([4, 0, 4, 0, 1, 4], 4) ->  4
```