Thème : Python

Exercice 1. ✧ ✎ **Deux programmes de première année** # PY144

1. Écrire un programme qui prend en argument une fonction f, un entier naturel non nul n et deux réels a, b (avec $a < b$) et renvoie la somme de Riemann d'ordre de f :

$$S_n(f) = \frac{b-a}{n}\sum_{k=0}^{n-1} f\left(a + k\frac{b-a}{n}\right).$$

2. Donner une approximation de $\sqrt{2}$ par l'algorithme de dichotomie à 10^{-5} près.

Exercice 2. ✧
Soit $X \hookrightarrow \mathscr{U}(]0;1])$ et $Y = -\ln(X)$

1. Donner la loi de Y.
2. Quelle est la loi simulée par le programme ci-contre ?

Editeur

```
def simulation(n): # n est un entier > 1
    u=np.random.rand(n)
    p=np.prod(u) # effectue le produit
        des coefficients de u
    return -np.log(p)
```

PY143

Exercice 3. ✦✦ La série $\sum \frac{1}{n^3}$ est convergente. Écrire un programme qui calcule les termes successifs de la suite $(S_n)_{n\in\mathbb{N}^*}$ des sommes partielles de cette série jusqu'à ce que $S_n - S_{n-1} < 10^{-10}$ et renvoie le dernier S_n calculé. # PY141

Exercice 4. ✦ On considère X, Y deux variables de Poisson indépendantes de paramètre $\lambda \in \mathbb{R}_*^+$. # PY142

1. Écrire une fonction d'en-tête **def estime(Lambda) :** qui, prenant en argument un réel Lambda strictement positif, simule un grand nombre de fois les variables aléatoires X et Y, et renvoie une estimation de $\mathbf{P}([X = Y])$.
*Rappel. La commande **np.random.poisson(mu,n)** renvoie un tableau de nombres de taille n dont chaque coefficient suit une loi de Poisson d'espérance **mu**.*
2. Adapter le programme pour X et Y suivant une loi géométrique de paramètre p (vérifier par un calcul théorique le résultat obtenu).

Exercice 5. ✦ **Triangle de Floyd** # PY132

1. Écrire une fonction python qui prend en argument n et renvoie, sous forme d'une matrice, les n premières lignes du triangle

$$\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 & 0 \\ 4 & 5 & 6 & 0 & 0 \\ 7 & 8 & 9 & 10 & 0 \\ 11 & 12 & 13 & 14 & 15 \\ & & \cdots & & \end{array}$$

2. On note T_n, le dernier coefficient de la n-ième ligne du triangle.
   a) Conjecturer une formule de récurrence entre T_{n+1} et T_n.
   b) En admettant que la conjecture est vérifiée, proposer un programme qui prend en argument un entier $n \geqslant 1$ et renvoie T_n.
3. En utilisant le programme précédent, à quelle ligne se trouve le nombre 666 ?

Exercice 6. ✦

Compléter le script ci-contre pour qu'il calcule et affiche, à l'aide de la méthode de Monte-Carlo, une valeur approchée de l'intégrale $\int_0^1 \ln\left(1+t^2\right) \mathrm{d}t$:

Editeur

```
import numpy as np
U=rd.random(100000)
V=np.ln(1+U**2)
I=   ...
     ...
```

PY140

Exercice 7. ✦ Soit $n \in \mathbb{N}^*$. On dit qu'une matrice est de type croix s'il existe une ligne et une colonne dont les coefficients valent 1, le reste étant 0. Exemple : # PY128

$$\mathrm{M} = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & 0 & 0 & \cdots & 0 \end{bmatrix} \in \mathscr{M}_n(\mathbb{R}).$$

Écrire un programme qui prend en argument n et renvoie une matrice croix de $\mathscr{M}_n(\mathbb{R})$ dont la ligne et la colonne non nulles sont choisies au hasard.

Exercice 8. ✦✦ **Moments factoriels d'une loi de Poisson** # PY124

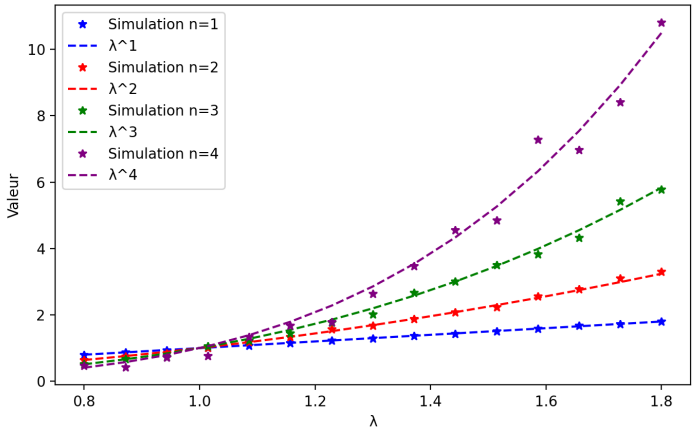
Soit X une variable aléatoire suivant la loi de Poisson de paramètre $\lambda \in \mathbb{R}_*^+$. On pose :

$$\mathrm{Y}_0 = 0 \quad \text{et} \quad \forall n \in \mathbb{N}^*, \qquad \mathrm{Y}_n = \mathrm{X}(\mathrm{X}-1)\cdots(\mathrm{X}-n+1).$$

1. Écrire une fonction python, d'arguments n et `lbda`, qui renvoie une réalisation de Y_n.
2. Commenter, conjecturer puis prouver un résultat à partir du code et résultats suivants :

Editeur

```
Lbda = np.linspace(0.8, 1.8, 15)
m = 10000
for n in range(1, 5):
    M = np.zeros(len(Lbda))
    for i in range(len(Lbda)):
        E = np.zeros(m)
        for k in range(m):
            E[k] = simuY(Lbda[i], n)
        M[i] = np.mean(E)
    plt.plot(lbda, M, '*')
    plt.plot(lbda, lbda**n,'--')
plt.show()
```



3. a) Proposer une fonction python, d'arguments n et `lbda`, qui donne une approximation de $\mathrm{Cov}(\mathrm{X}, \mathrm{Y}_n)$.
   b) Calculer $\mathrm{Cov}(\mathrm{X}, \mathrm{Y}_n)$, puis en déduire que $\mathrm{Var}(\mathrm{Y}_n) \geqslant n^2\lambda^{2n-1}$.

Exercice 9. ✦✦ Pour tout entier $n \geqslant 2$, on définit le polynôme # PY125

$$\mathrm{P}_n = x^n - \left(x^{n-1} + x^{n-2} + \cdots + x + 1\right).$$

1. Justifier que P_n admet une unique racine strictement positive.
   On pourra remarquer que

$$x^n = x^{n-1} + x^{n-2} + \cdots + x + 1 \iff 1 = \frac{1}{x^n} + \frac{1}{x^{n-1}} + \cdots + \frac{1}{x}.$$

Dans la suite, on note α_n, l'unique racine positive de P_n.

2. Afficher sur le même graphe les courbes de P_n pour $n \in [\![2;6]\!]$ sur $[0;2.2]$. Que peut-on conjecturer sur les variations et la convergence de la suite $(\alpha_n)_n$?
Pour avoir un meilleur affichage, on pourra rajouter les commandes `plt.ylim(-10,2)` et `plt.grid()`.
3. Écrire un programme python qui prend en argument $n \in \mathbb{N}^*$ et renvoie une approximation à 10^{-3}-près de α_n. Tester le programme avec $n = 2$ en comparant avec la solution théorique.
4. Afficher aussi la suite $(\alpha_n)_n$ pour $n \in [\![2;16]\!]$.

Exercice 10. ✦✦ **Nombres de Fibonacci aléatoires** # PY129

La suite de Fibonacci $(F_n)_{n\in\mathbb{N}}$ est définie par $F_0 = 0$, $F_1 = 1$, et la relation de récurrence

$$F_n = F_{n-1} + F_{n-2} \quad \text{pour} \quad n \geqslant 2.$$

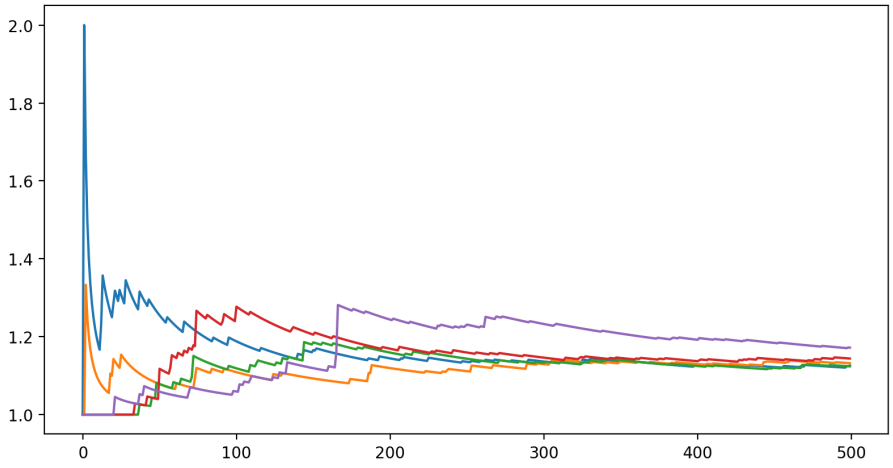
Soit N, une variable aléatoire de loi géométrique de paramètre p. On définit la variable Y par $Y = F_N$.

1. Écrire un programme qui prend en argument $p \in]0;1[$ et renvoie une simulation de Y.
On proposera deux solutions, l'une utilisant `rd.geometric`, *l'autre n'utilisant que* `rd.random()`.
2. Soient $(Y_i)_{i\in\mathbb{N}^*}$, une suite de variables aléatoires toutes indépendantes et de même loi que Y. On pose

$$\overline{Y_n} = \frac{1}{n}\sum_{i=1}^{n} Y_i.$$

a) Donner un programme qui prend en argument p et affiche une réalisation de la séquence $\left(\overline{Y_1}, \overline{Y_2}, \ldots, \overline{Y_{500}}\right)$.
On pourra commencer par donner un relation de récurrence simple entre $\overline{Y_{n+1}}$ *et* $\overline{Y_n}$.

b) Voici plusieurs tests pour $p = 0,7$. Commenter.



Exercice 11. ✦ **Approximation de π** *Les questions sont indépendantes.* # PY130

1. On admet que pour tout $n \in \mathbb{N}^*$

$$0 < \pi - p_n < \frac{1}{n} \quad \text{où} \quad p_n = 2\prod_{k=1}^{n} \frac{4k^2}{4k^2-1}.$$

Écrire un programme qui prend en argument un réel strictement positif p et renvoie une approximation de π à p près.

2. *Somme de Riemann*
Soit g une fonction de classe $\mathscr{C}^1$ sur $[a;b]$. On note $S_n(g)$, la n-ième somme de Riemann de g. On admet que

$$\left| S_n(g) - \int_a^b g(t)\mathrm{d}t \right| \leqslant \frac{\max_{[a;b]} |g'|}{12n}(b-a)^2.$$

En considérant $g : t \mapsto 1/(1+t^2)$, donner un programme qui prend en entrée une précision p et donne une approximation de π à p près.

3. *Algorithme de seuil*
On définit la suite (u_n) par

$$u_0 = 4, \qquad \forall\, n \in \mathbb{N}, \quad u_{n+1} = \frac{2u_n}{1+\sqrt{1+u_n{}^2 \cdot 4^{-(n+1)}}}.$$

On admet que cette suite converge vers π. Donner le plus petit entier n tel que u_n soit une approximation de π à 10^{-4}-près.

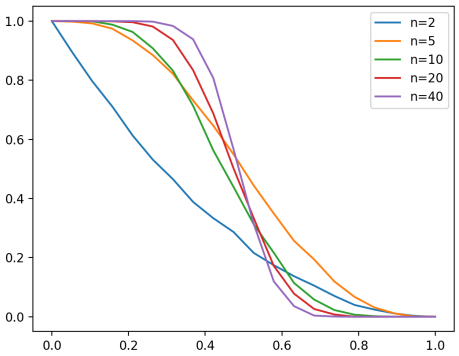
Exercice 12. ✦✦ Soit $(X_k)_{k\in\mathbb{N}^*}$ une suite de variables aléatoires indépendantes de loi de Bernoulli de paramètre p. # PY137
Pour tout $n \in \mathbb{N}^*$, on pose

$$Q_n(p) = \mathbf{P}\left(X_1 + \cdots + X_n \leqslant n/2\right).$$

1. Écrire un programme d'entête `Estim(n,p)` qui renvoie une estimation de la probabilité $Q_n(p)$.

2. **a)** Comment afficher pour différentes valeurs de p les termes $\left(Q_n(p)\right)_{n\in[\![1;20]\!]}$?

b) À l'aide du résultat ci-contre, que peut-on conjecturer sur le comportement limite de $Q_n(p)$ lorsque $n \to +\infty$?



3. **a)** Expliciter Q_n pour $n \in [\![1;3]\!]$.

b) Vérifier que Q_n est une fonction polynomiale sur $[0;1]$ de degré au plus n.

Exercice 13. ✦✦✦ # PY126

- *Préliminaires*

1. On définit la fonction f sur $\mathbb{R}$ par

$$f(x) = \begin{cases} \sqrt{\dfrac{3}{\pi}}\, x e^{-\frac{3}{16}x^4} & \text{si } x \geqslant 0 \\ 0 & \text{sinon.} \end{cases}$$

a) Justifier que f est une densité de probabilité.
On pourra faire le changement de variable $u = x^2/2$.

b) À l'aide de Python, écrire un programme python `graphe` qui prend en argument deux réels positifs a, b, vérifie que $a < b$ et dans ce cas, trace la courbe de f sur $[a;b]$.

- *Application*

2. Soit $n \in \mathbb{N}^*$, deux équipes de n joueurs s'affrontent dans une partie de paint-ball. Tant qu'il reste un joueur dans une équipe, chaque minute, un joueur (choisi au hasard parmi les joueurs restants) touche et élimine un joueur de l'équipe adverse. On suppose les tirages indépendants. On note alors N_n, la variable aléatoire donnant le nombre de joueurs restants à la fin de la partie.

a) Écrire un programme qui prend en argument n et simule N_n.

b) Tester le code suivant pour $n = 10$, $n = 20$, $n = 50$, $n = 100$ et $n = 200$. Que peut-on conjecturer ?

Editeur

```
n= ...
m=2000
Ech=np.zeros(m)
for i in range(m):
    Ech[i]=simuN(n)/n**(3/4)
plt.clf()
plt.hist(Ech,30,density=True)
graphe(np.min(Ech),np.max(Ech))
plt.show()
```

c) *Facultatif.* On suppose que $\mathbf{E}(N_n) \sim cn^{\alpha}$. Estimer α à l'aide simulations.

Exercice 14. ✦✦✦ **Python et permutations** # PY134

Soient $n \in \mathbb{N}^*$ et $E_n = [\![0; n-1]\!]$. Une permutation f de E_n est une application bijective de E_n dans E_n. On peut la coder par une matrice ligne

$$F = \begin{bmatrix} f(0) & f(1) & \dots & f(n-1) \end{bmatrix}.$$

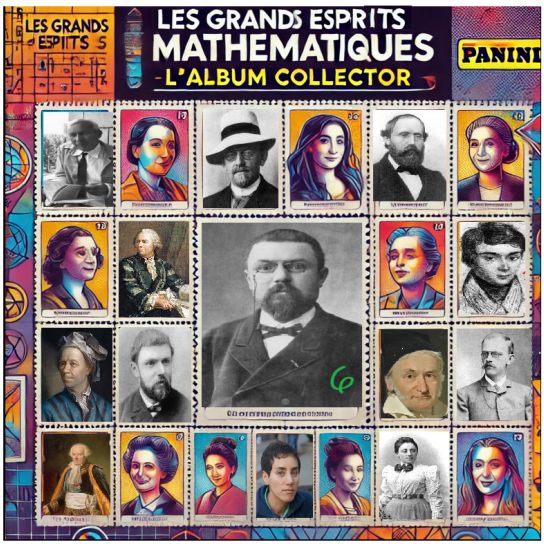
1. *Comment générer une permutation aléatoire ?*
   a) Écrire un programme python qui part d'une matrice F codant une permutation et renvoie une matrice ligne obtenue en permutant au hasard deux de ses coefficients.
   b) En partant de la matrice ligne codant id_{E_n} et en répétant un grand nombre de fois (par exemple n^2) le programme précédent, donner un programme qui prend en argument n et renvoie, au hasard, une matrice ligne codant une permutation de E_n.
2. Dans la suite, à toute permutation f de E_n, on pose

$$S_n(f) = \sum_{k=0}^{n-1} k f(k).$$

   a) Expliciter $S_n(\mathrm{id}_{E_n})$.
   b) Écrire un programme qui prend en argument une matrice ligne F codant une permutation f de E_n et renvoie la valeur de $S_n(f)$.
   c) Tester plusieurs fois votre programme en calculant $S_n(\mathrm{id}_{E_n}) - S_n(f)$ où f est une permutation de E_n choisie au hasard. En regardant le signe de cette quantité, que peut-on conjecturer ?
3. Prouver votre conjecture. On rappelle que pour $a, b \in \mathbb{R}$, $ab \leqslant (a^2 + b^2)/2$.

Exercice 15. ✦✦ **Problème du collectionneur** # PY138

La société Panini vient de créer un nouvel album d'images autocollantes à collectionner. L'album propose 10 images de mathématiciens célèbres.
Les images sont vendues à l'unité, au hasard et la répartition des images est uniforme. Monsieur F., un passionné, souhaite acquérir la collection complète tout en surveillant son budget.



Leonhard Euler, Sophie Germain, Evariste Galois, Bernhard Riemann, Andreï Kolmogorov, Henri Poincaré, Maryam Mirzakhani, Pierre-Simon de Laplace, Emilie Noether, Carl Friedrich Gauss.

1. a) Écrire un programme qui prend en argument un entier n, effectue n tirages d'images et renvoie une matrice ligne $L \in \mathcal{M}_{1,10}(\mathbb{R})$ dont le i-ème coefficient vaut soit 1 soit 0 suivant si on a tiré la i-ème image ou non.
   b) Modifier le programme précédent pour écrire une fonction `panini` sans argument qui effectue une succession de tirages et s'arrête dès que la collection des 10 images est complète. Le programme renvoie alors le nombre de tirages effectués.

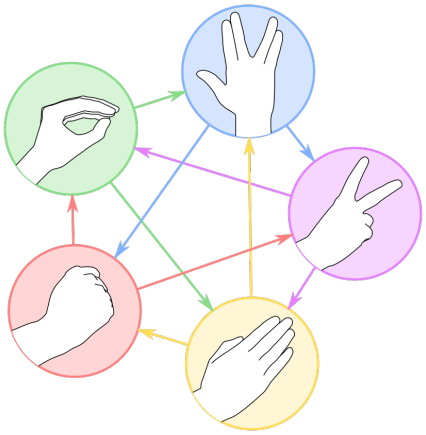
2. Estimer le prix moyen d'un album complet avec un prix de 2 euros par image. Commenter :)
3. *Bonus.* Reprendre l'estimation précédente en supposant que l'image du mathématicien Henri Poincaré est une image rare : 10 fois moins présentes que les autres.

Exercice 16. ✦✦ Pierre-feuille-ciseaux

PY139

Dans la suite, on code la pierre, la feuille et les ciseaux respectivement par 0, 1 et 2.

1. Programmer une partie de pierre-feuille-ciseaux contre la machine. C'est-à-dire, écrire un programme qui prend en argument le choix du joueur (0, 1 ou 2), tire au hasard le choix de la machine et renvoie le nom du gagnant ou l'égalité.
2. Généraliser le programme précédent avec une partie de Pierre-feuille-ciseaux-Spock-lézard.



On pourra s'aider de la matrice suivante qui résume les différentes possibilités :

$$\begin{bmatrix} 0 & -1 & 1 & -1 & 1 \\ 1 & 0 & -1 & 1 & -1 \\ -1 & 1 & 0 & -1 & 1 \\ 1 & -1 & 1 & 0 & -1 \\ -1 & 1 & -1 & 1 & 0 \end{bmatrix}$$

où les lignes 1 à 5 correspondent respectivement aux issues possibles pour les choix Pierre, feuille, ciseaux, Spock, lézard (0 égalité, 1 victoire, −1 défaite).

Pour plus de détails, cliquez pour l'extrait *The Big Bang Theory.*

3. ✦✦✦ *Facultatif*
Expliquer le fonctionnement du code suivant :

Editeur

```
def mat(n):
    if n%2!=1:
        return 'erreur n doit etre
            impair'
    p=n//2
    A=np.zeros([n,n])
    signe=1
    for k in range(1,n):
        signe=-signe
        for l in range(k):
            A[l,l-k]=signe
    A=A+np.transpose(A)
    return A
```

```
>>> mat(5)
array([[ 0.,  1., -1.,  1., -1.],
       [ 1.,  0.,  1., -1.,  1.],
       [-1.,  1.,  0.,  1., -1.],
       [ 1., -1.,  1.,  0.,  1.],
       [-1.,  1., -1.,  1.,  0.]])
```

Exercice 17. ✦✦ Mélanges de cartes

PY133

1. Créer une fonction `Carte` qui prend en argument une matrice ligne $M \in \mathcal{M}_{1,52}(\mathbb{R})$ et qui renvoie une matrice ligne $N \in \mathcal{M}_{1,52}(\mathbb{R})$ obtenue en effectuant un mélange déterministe de M de la manière suivante :
   - On sépare la matrice ligne $M \in \mathcal{M}_{1,52}(\mathbb{R})$ en deux matrices M_1, $M_2 \in \mathcal{M}_{1,26}(\mathbb{R})$ telles que si on les remettait bout-à-bout, on retrouverait la matrice M ;
   - La matrice N est construite en prenant dans l'ordre et alternativement les coefficients de M_1 et M_2 en commençant par M_2.
2. Écrire un programme qui prend en argument M et renvoie la matrice N ainsi construite.
3. Combien de mélanges sont-t-ils nécessaires pour revenir à la matrice M ?

4. Reprendre la question précédente en commençant par M_1.

Exercice 18. ✦✦✦ **Python et permutations** # PY135

Soit $n \in \mathbb{N}^*$. On note $\mathfrak{S}_n$ l'ensemble des permutations de $[\![0; n-1]\!]$ (c'est-à-dire l'ensemble des bijections de $[\![0; n-1]\!]$ dans $[\![0; n-1]\!]$). On appelle dérangement une permutation sans point fixe. On peut générer aléatoirement de manière uniforme une permutation de $\mathfrak{S}_n$ à l'aide de `rd.permutation(n)` qui renvoie un vecteur ligne composé des n entiers de $[\![0; n-1]\!]$ dans un ordre aléatoire.

1. Écrire une fonction d'entête `derang(f)` renvoyant 1 si la permutation `f` rentrée en argument est un dérangement et 0 sinon.
2. Écrire un programme qui calcule une approximation du nombre de dérangements de $\mathfrak{S}_n$.

Exercice 19. ✦✦ **Python et permutations** # PY136

Soient $n \in \mathbb{N}^*$ et $E_n = [\![0; n-1]\!]$. Une permutation f de E_n est une application bijective de E_n dans E_n. On peut la coder par une matrice ligne

$$F = \begin{bmatrix} f(0) & f(1) & \dots & f(n-1) \end{bmatrix}.$$

1. Écrire une fonction qui prend en argument une matrice ligne et permet de tester si c'est une permutation.
   On pourra utiliser la commande `np.sort` *qui permet d'ordonner la matrice ligne.*
2. Écrire une fonction python qui prend en argument deux matrices lignes F, G associées à deux permutations f, g et renvoie la matrice ligne associée à la composition $f \circ g$.
3. Écrire une fonction python qui prend une matrice ligne associée à f et renvoie la matrice ligne associée à l'application réciproque f^{-1}.

Exercice 20. ✦✦✦ Voici un extrait du livre *Probabilités pour les non-probabilistes* de Walter Appel : # PY127

Problématique

En essayant d'expliquer la répartition des niveaux d'énergie des noyaux des atomes lourds, Eugene Wigner a été amené, dans les années 1950, à étudier le spectre de matrices symétriques réelles aléatoires de grande taille. La figure 1 montre un histogramme qui représente la répartition des valeurs propres d'une matrice symétrique réelle de taille $n = 2500$ constituée de variables aléatoires indépendantes identiquement distribuées, d'espérance nulle et de variance égale à 1.

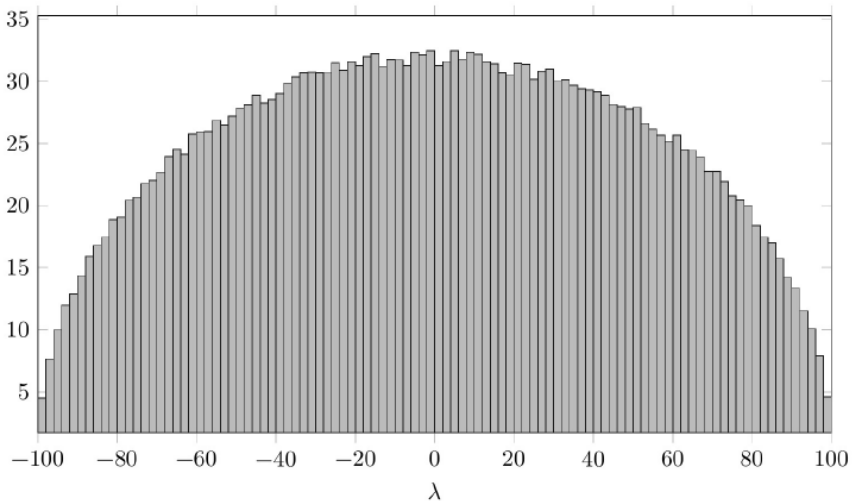


Figure 1

On voit que les valeurs propres se répartissent suivant un profil en demi-cercle de rayon $2\sqrt{n}$. En normalisant par un facteur $1/\sqrt{n}$ on obtient le profil d'un demi-cercle de rayon 2.

L'objectif est de tester numériquement le résultat précédent avec des lois uniformes et normales.

1. Écrire un programme qui prend en arguments deux matrices lignes et renvoie la matrice ligne obtenue par concaténation.

2. Soient $n \in \mathbb{N}^*$ et $p = n(n+1)/2$. Soient $X = (X_{i,j})_{1 \leqslant i \leqslant j \leqslant n}$, p variables aléatoires mutuellement indépendantes qui suivent toutes une loi normale centrée réduite. On construit alors la matrice aléatoire A_X, symétrique telle que son coefficient en position (i, j) (avec $1 \leqslant i \leqslant j \leqslant n$) soit $X_{i,j}$.
On rappelle que le spectre d'une matrice `A` est obtenue par `np.linalg.eigvals(A)` après avoir importé la bibliothèque `numpy`.

a) Écrire un programme qui donne une réalisation de la matrice A_X.

b) En déduire un programme qui teste l'affirmation.

3. Reprendre la question 2 avec une loi uniforme. *Les paramètres sont choisis afin d'avoir une variable aléatoire centrée réduite.*

Exercice 21. ✦✦✦ *Oral HEC 2025* # PY131

Editeur

```
def ST(A): # A est une matrice 3*3
    alpha=rd.normal(0,1,1000)
    beta=rd.normal(0,1,1000)
    S=0
    for i in range(3):
        for j in range(i+1):
            S+=A[i,j]**2
    dep= S - (A[0,1])**2+(A[0,2])**2+(A[1,2])**2
    min=S
    for k in range(1000):
        T=dep+(A[0,1]-alpha[k])**2+(A[0,2]-beta[k])**2+(A[1,2]-alpha[k])**2
        if T < min :
            min=T
    return min

ST(B)
```

Que vaut la valeur prise par dep ? Que calcule ST(B) ? Justifier le résultat.

$$B = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 0 & -1 \\ 0 & -1 & 1 \end{bmatrix}.$$

Exercice 5 p. 1

PY132

1. L'idée est de commencer par une matrice remplie de « 0 » puis de parcourir le triangle en rajoutant 1 à chaque étape.

```
def Floyd(n):
    A=np.zeros([n,n])
    c=1
    for i in range(n):
        for j in range(i+1):
            A[i,j]=c
            c+=1
    return A

>>> Floyd(5)
array([[ 1.,  0.,  0.,  0.,  0.],
       [ 2.,  3.,  0.,  0.,  0.],
       [ 4.,  5.,  6.,  0.,  0.],
       [ 7.,  8.,  9., 10.,  0.],
       [11., 12., 13., 14., 15.]])
```

2.a) On a $T_1 = 1$ et pour tout $n \in \mathbb{N}^*$

$$T_{n+1} = n + 1 + T_n.$$

On peut obtenir une formule explicite pour T_n :

$$T_n - T_1 = \sum_{i=1}^{n-1} T_{i+1} - T_i = \sum_{i=1}^{n-1} (i+1)$$

$$= \frac{n(n+1)}{2} - 1.$$

Comme $T_1 = 1$,

$$T_n = \frac{n(n+1)}{2}.$$

2.b) Soit on utilise la formule précédente, soit :

```
def Tn(n):
    T=0
    for i in range(1,n+1):
        T+=i
    return T
```

3. Le nombre 666 se trouve sur la ligne 36.

```
n=0
T=0
while T<666:
    n+=1
    T+=n
print(n)

>>> 36
```

Exercice 8 p. 2

PY124

1.

```
import numpy as np
import numpy.random as rd

def simuY(lbda,n):
    X=rd.poisson(lbda)
    Y=X
    for i in range(1,n):
        Y*=(X-i)
    return Y
```

2. Vérifier par la formule de transfert que

$$\mathbf{E}(Y_n) = \lambda^n.$$

3.a)

...

3.b) D'après la formule de Huygens

$$\text{Cov}(X, Y_n) = \mathbf{E}(XY_n) - \mathbf{E}(X)\mathbf{E}(Y_n)$$

Or on a aussi par la définition de Y_n :

$$XY_n = Y_{n+1} + nY_n.$$

Par linéarité de l'espérance :

$$\mathbf{E}(XY_n) = \mathbf{E}(Y_{n+1}) + n\mathbf{E}(Y_n) = \lambda^{n+1} + n\lambda^n.$$

En conclusion :

$$\text{Cov}(X, Y_n) = n\lambda^n.$$

- On utilise l'inégalité de Cauchy-Schwarz, pour avoir :

$$\text{Var}(Y_n) \geqslant \frac{\text{Cov}(X, Y_n)^2}{\text{Var}(X)}.$$

Ce qui donne l'inégalité demandée en remplaçant par les valeurs trouvées précédemment.

Exercice 10 p. 3

PY129

1.

```
def FiboGeo(p):
    F=np.array([0,1]) # Fi et Fi+1
    N=rd.geometric(p)
    for i in range(N):
        w=F[0]+F[1]
        F[0]=F[1]
        F[1]=w
    return F[0]

def FiboGeo2(p):
    F=np.array([0,1]) # Fi et Fi+1
    N=1
    while rd.random()>p:
        w=F[0]+F[1]
        F[0]=F[1]
        F[1]=w
    return F[1]
```

2.a) Vérifier que

$$\overline{Y_{n+1}} = \frac{1}{n+1} Y_{n+1} + \frac{n}{n+1} \overline{Y_n}.$$

On en déduit le code

```
def sequence(p):
    Ym=np.zeros(500)
    Ym[0]=FiboGeo(p)
    for i in range(1,500):
        Ym[i]=(FiboGeo(p)+i*Ym[i-1])/(i+1)
    return Ym
```

2.b) Pour les tests, on écrit :

```
plt.clf()
p= ...
for i in range(5):
    plt.plot(sequence(p))
plt.show()
```

- Pour $p = 0.7$, on a l'impression qu'on a une convergence, et suggère que pour $p = 0.7$, l'espérance existe.

- Pour une preuve, on peut montrer en utilisant les suites récurrentes linéaires que

$$F_n = \frac{1}{\sqrt{5}}\left(\varphi^n - \psi^n\right)$$

avec $\varphi = \dfrac{1+\sqrt{5}}{2}$ et $\psi = \dfrac{1-\sqrt{5}}{2} = -\dfrac{1}{\varphi}$.

D'après le théorème de transfert, Y admet une espérance si et seulement si la série

$$\sum F_k \mathbf{P}(N = k) = \sum p F_k q^{k-1}$$

converge absolument. La série est à termes positifs et

$$F_k q^{k-1} \sim q^{-1}(\varphi q)^k.$$

D'après les résultats sur les séries géométriques, il y a convergence si et seulement si $q\varphi < 1$. C'est-à-dire

$$p > 1 - \frac{1}{\varphi} = 1 + \psi \simeq 0{,}38.$$

Exercice 13 — p. 4

PY126

1.a) La positivité et la continuité sur $\mathbb{R}^*$ est claire. À l'aide du changement de variable $u = t^2/2$ de classe $\mathscr{C}^1$ et strictement croissant

$$\begin{aligned}\int_{-\infty}^{+\infty} f(t)\,dt &= \int_0^{+\infty} \sqrt{\frac{3}{\pi}}\, t e^{-3t^4/16}\,dt \\ &= \frac{2\sqrt{3}}{\sqrt{\pi}} \int_0^{+\infty} e^{-3u^2}\,du \\ &= \sqrt{\frac{3}{\pi}} \int_{-\infty}^{+\infty} e^{-3u^2}\,du \quad \text{(parité)} \\ &= \sqrt{2\cdot 3\sigma^2} \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-u^2/(2\sigma^2)}\,du\end{aligned}$$

$$\int_{-\infty}^{+\infty} f(t)\,dt = 1 \qquad \left(2\sigma^2 = \frac{1}{3}\right)$$

On a bien une densité de probabilité.

1.b)

```
def graphe(a,b):
    x=np.linspace(a,b,100)
    y=np.sqrt(3/np.pi) * x * np.exp(-3*x
        **4/16)
    plt.plot(x,y)
```

2.a)

```
def simuN(n):
    A=n
    B=n
    while A>0 and B>0:
        p=A/(A+B)
        u=rd.random()
        if u>p:
            A=A-1
        else :
            B=B-1
    return max(A,B)
```

2.b) On constate une convergence des histogrammes vers la courbe de la densité f, on conjecture que la suite de variables aléatoires $\left(N_n/n^{3/4}\right)_n$ converge en loi vers une variable X de densité f.

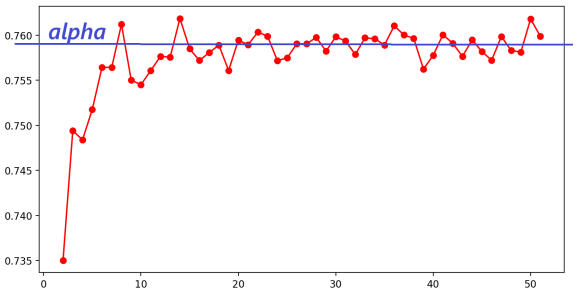
2.c) On peut dans un premier temps, écrire un programme qui donne une approximation de $\mathbf{E}(N_n)$ en s'appuyant sur la loi des grands nombres :

```
def approxEsp(n):
    m=10000
    s=0
    for i in range(m):
        s+=simuN(n)
    return s/m
```

Puis, on regarde la limite de $\ln(\mathbf{E}(N_n))/\ln(n)$ qui si l'équivalent est vérifié, vaut α.

```
N=50
suite=np.zeros(N)
for i in range(N):
    suite[i]=np.log(approxEsp(i+2))/np.log
        (i+2)
plt.clf()
plt.plot(np.arange(2,N+2),suite,'ro-')
plt.show()
```

Résultat :



Exercice 11 — p. 3

PY130

1. On remarque que pour $n \in \mathbb{N}^*$

$$p_0 = 1 \quad \text{et} \quad p_n = \frac{4n^2}{4n^2-1} p_{n-1}.$$

et si on choisit n tel que $e \leqslant 1/n$ (par exemple $n = \lfloor 1/e \rfloor + 1$), on a aussi $|\pi - p_n| \leqslant e$.

```
def approx_pi(e):
    p=1
    n=int(np.floor(1/e)+1)
    for k in range(1,n+1):
        p=p*(4*k**2)/(4*k**2-1)
    return 2*p

>>> approx_pi(10**(-4))
3.1415141265341364
>>> np.pi
3.141592653589793
```

2. Commençons par un code standard pour calculer la n-ième somme de Riemann d'une fonction f sur $[a;b]$.

```
def Riemann(a,b,f,n):
    S=0
    pas  = (b-a)/n          # Largeur du
        rectangle
    for k in range(n):
        S += f(a + k*pas) # On somme les
            hauteurs
    S=pas*S
    return S
```

On applique à la fonction continue g sur $[0,1]$ sachant que

→ $\displaystyle\int_0^1 g(t)\,\mathrm{d}t = \big[\arctan t\big]_0^1 = \pi/4.$

→ On a $\max\limits_{[0,1]} |g'| \leqslant 2$, donc $\dfrac{\max_{[0;1]} |g'|}{12n}(1-0)^2 \leqslant \dfrac{1}{12n}$

de sorte que si l'on prend $n = \lfloor 1/(3p) \rfloor + 1$, on a

$$\left|4S_n(g) - \pi\right| \leqslant \frac{4}{12n} \leqslant p.$$

D'où le code :

```
def g(t):
    return 1/(1+t**2)

def approxpi(p):
  n=int(1/(3*p))+1
  return 4*Riemann(0,1,g,n)
```

3.

```
n=0
u=4
while np.abs(u-np.pi)>10**(-4):
    n+=1
    u=2*u/(1+(1+u**2*4**(-n-1))**(1/2))

print(u,n)
>>> 3.141632080703183 7
```

Remarque.
Pour montrer la convergence de π, on peut établir

$$\tan\left(\frac{x}{2}\right) = \frac{\tan(x)}{1+\sqrt{1+\tan^2(x)}}$$

vérifier que

$$\forall\, n \in \mathbb{N}, \quad u_n = 2^n \tan(\pi 2^{-n}).$$

En utilisant un équivalant de tangente, on a

$$u_n \underset{n\to\infty}{\longrightarrow} \pi.$$

Exercice 14 — p. 5

PY134

1.a)

```
def permut(F):
    n=len(F)
    i=rd.randint(n)
    j=rd.randint(n)
    while i==j:
        i=rd.randint(n)
        j=rd.randint(n)
    # Si i=j, on retire deux entiers.
    # On recommence tant que i==j
    memoire=F[i]
    F[i]=F[j]
    F[j]=memoire
    return  F
```

- On teste en permutant plusieurs fois :

```
F=np.arange(5)
print(F)
for i in range(4):
    F=permut(F)
    print(F)

>>>
[0 1 2 3 4]
[0 3 2 1 4]
[1 3 2 0 4]
[3 1 2 0 4]
[4 1 2 0 3]
```

1.b)

```
def permuthasard(n):
    F=np.arange(n)
    for i in range(n**2):
        F=permut(F)
    return F
```

2.a)

$$S_n(\mathrm{id}_{E_n}) = \sum_{k=0}^{n-1} k^2 = \frac{n(n-1)(2n-1)}{6}.$$

2.b)

```
def sommepermut(F):
    s=0
    for i in range(len(F)):
        s+=i*F[i]
    return F
```

2.b)

```
def permuthasard(n):
    F=np.arange(n)
    for i in range(n**2):
        F=permut(F)
    return F
```

2.c)

```
n=10
F=permuthasard(n)
I=np.arange(n)
print(somme(I)-somme(F))
```

On constate que le signe est toujours positif. On conjecture que

$$S_n(f) \leqslant S_n(\mathrm{id}_{E_n})$$

pour toute permutation f de E_n.

3. Soit σ une bijection de $[\![0, n-1]\!]$. Pour tout $k \in [\![0, n-1]\!]$,

$$k\sigma(k) \leqslant \frac{k^2+\sigma(k)^2}{2}.$$

Par somme,

$$\sum_{k=0}^{n-1} k\sigma(k) \leqslant \sum_{k=0}^{n-1} \frac{k^2+\sigma(k)^2}{2} = \frac{1}{2}\sum_{k=0}^{n-1} k^2 + \frac{1}{2}\sum_{k=0}^{n-1} \sigma(k)^2.$$

L'application σ est une bijection de $[\![0, n-1]\!]$ sur lui-même. La somme $\sum_{k=0}^{n-1} \sigma(k)^2$ est la somme des carrés de tous les entiers de $[\![0, n-1]\!]$. L'ordre dans une somme n'a pas d'importance, donc

$$\sum_{k=0}^{n-1} \sigma(k)^2 = \sum_{k=0}^{n-1} k^2.$$

Puis,

$$\sum_{k=0}^{n-1} k\sigma(k) \leqslant \frac{1}{2}\sum_{k=0}^{n-1} k^2 + \frac{1}{2}\sum_{k=0}^{n-1} k^2 = \sum_{k=0}^{n-1} k^2,$$

donc $S_n(\sigma) \leqslant S_n(\mathrm{id})$.

Exercice 15 p. 5

PY138

1.a)

```
def album(n):
    L=np.zeros(10)
    for i in range(n):
        image=rd.randint(0,10)
        L[image]=1
    return L
```

1.b)

```
def panini():
    L=np.zeros(10)
    n=0
    while np.sum(L)<10:
        image=rd.randint(0,10)
        L[image]=1
        n+=1
    return n
```

2.

```
def prix():
    e=0
    for i in range(5000):
        e+=panini()
    return 2*e/5000 # 2 euro par image

>>> prix()
58.794
```

10 images de matheux pour moins de 60 euros, quelle affaire!! Il ne faut pas hésiter :) et acheter aussi des actions chez panini.

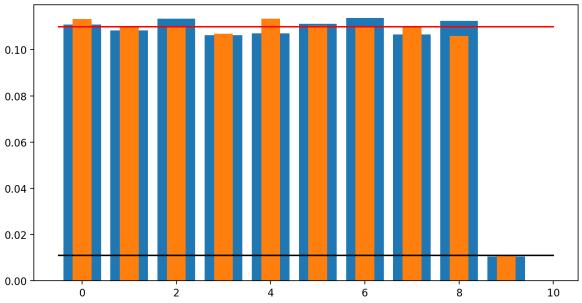
3. Dans un premier temps, on construit une loi non uniforme sur $[\![0;9]\!]$ où le 9 a dix fois moins de chance d'apparaitre que les autres.

```
# option 1
def loi_non_uniforme():
    x=rd.randint(0,91)
    return int(x/10)

# option 2
def loi_non_uniforme2():
    if rd.random()<(1/91):
        return 9
    return rd.randint(0,9)
```

- On peut tester avec un histogramme

```
plt.clf()
m=20000
Ech=np.zeros(m)
Ech2=np.zeros(m)
for i in range(m):
    Ech[i]=loi_non_uniforme()
    Ech2[i]=loi_non_uniforme2()
plt.hist(Ech,np.linspace(0,10,11)-0.5,
    rwidth=0.8,density=True)
plt.hist(Ech2,np.linspace(0,10,11)-0.5,
    rwidth=0.4,density=True)
plt.plot([-0.5,10],[10/91,10/91],'r')
plt.plot([-0.5,10],[1/91,1/91],'k')
plt.show()
```




```
def panini_rare():
    L=np.zeros(10)
    n=0
    while np.sum(L)<10:
        image=loi_non_uniforme()
        L[image]=1
        n+=1
    return n

def prix_rare():
    e=0
    for i in range(5000):
        e+=panini_rare()
    return 2*e/5000
```

On teste :

```
>>> prix_rare()
188.7828
```

Avec ce changement, on a triplé le prix moyen d'un album!

Exercice 16 p. 6

PY139

1.

```
def chifoumi(joueur):
    machine=rd.randint(3)
    if machine == joueur :
        return "égalité"
    if joueur==0 : # J fait pierre
        if machine==1 : # M fait feuille
            return "machine"
        return "joueur"
    if joueur==1 : # J fait feuille
        if machine==0 : # M fait Pierre
            return "joueur"
        return machine
    # on sait ici que J fait ciseaux (2)
    if machine==0: # M fait pierre
        return "Machine"
    return "Joueur"
```

2.

```
def megachifoumi(j):
    n=5
    m=rd.randint(n)
    A=mat(5)
    if A[i,m]==1:
        return 'victoire'
    if A[i,m]==0:
        return 'défaite'
    return 'égalité'
```

3. Le programme commence avec une matrice nulle de taille n, puis remonte les diagonales supérieures en les remplaçant par des diagonales de ± 1 en alternant le signe. Le programme termine en sommant la matrice obtenue avec sa transposée pour avoir la même chose sur les diagonales inférieures.

Exercice 17 — p. 6

PY133

1.

```
def Carte(M):
    n=len(M)
    m=n//2
    # "//" pour la division euclidienne et
        éviter
    # les problèmes si n n'est pas entier

    N=np.zeros(n)
    for i in range(m):
        N[2*i]=M[i]
        N[2*i+1]=M[m+i]
    return N
```

2. Si N et M sont deux matrices de même taille, le test `N==M` renvoie une matrice de même taille dont le coefficient en position (i, j) vaux Vrai ou Faux suivant si les coefficients de N et M sont bien égaux en position (i, j). Comme vrai s'identifie à 1 et faux à 0, pour tester si deux matrices sont égales on peut tester si la somme des coefficients de la matrice `N==M` vaut sa taille.

```
M=np.linspace(0,51,52)
N=Carte(M)
C=1
while sum(N==M)!=52:
    N=Carte(N)
    C+=1
print(C)
>>> 8
```

Dit autrement, si on fait 8 mélanges parfaits en gardant la première carte en haut, on retrouve le paquet initial.

3. Il suffit de changer les deux lignes dans la boucle `for` du premier programme par :

```
N[2*i]=M[m+i]
N[2*i+1]= M[i]
```

Le code affiche alors 52. L'ordre entre M_1 et M_2 a donc de l'importance!

Exercice 18 — p. 7

PY135

1. L'idée est de parcourir les entiers de $[\![0; n-1]\!]$. Si un point fixe apparaît ($k = f(k)$) alors f n'est pas un dérangement. Par contre, si aucun point fixe n'apparaît au bout des n essais, on peut affirmer que f est une permutation.

```
def derang(f):
    c=0 ; n=np.shape(f)[0]
    for k in range(n):
        if f[k]==k:
            return 0
    return 1


# Tests
f1=np.array([3,0,1,2])
>>> derang(f1)
1
f2=np.array([3,1,2,0])
>>> derang(f2)
0
```

2. On a le code suivant, qui calcule la fréquence d'apparition d'un dérangement pour une permutation aléatoire pour $n = 5$:

```
N=1000
n=5
P=0
for k in range(N):
    P+=derang(rd.permutation(n))/N
print(P)
```

Ce code affiche d'ailleurs 0.3760000000000003.

On peut montrer, mais c'est difficile, que la valeur exacte est $\sum_{k=0}^{n} (-1)^k/k!$ *qui tend vers* $e^{-1} \simeq 0.3679$ *lorsque* $n \to +\infty$. *Enfin Pour obtenir une approximation du nombre de dérangements, il faut multiplier par le nombre de permutations de* $[\![0; n-1]\!]$, *soit* $n!$.

Exercice 19 — p. 7

PY136

1.

```
def permutation(F):
    F1=np.sort(F)
    for i in range(len(F)):
        if F1[i]!=i:
            return False
    return True
```

```
# Tests
>>> F=np.array([0,1,3,4,2])
>>> permutation(F)
True
>>> F=np.array([0,1,3,4,1])
>>> permutation(F)
False
```

```
def compo(F,G):
    if len(F)!=len(G):
        return 'composition impossible'
    FG=np.zeros(len(F))
    for i in range(len(F)):
        FG[i]=F[int(G[i])]
    return FG
```

```
# Tests
>>> F=np.array([0,1,3,4,2])
>>> G=np.array([1,2,3,4,0])
>>> compo(F,G)
array([1., 3., 4., 2., 0.])
>>> compo(G,F)
array([1., 2., 4., 0., 3.])
```

```
def reciproque(F):
    if permutation(F)==False :
        return 'impossible'

    Fr=np.zeros(len(F))
    for i in range(len(F)):
        j=0
        while F[j]!=i:
            j+=1
        Fr[i]=j
    return Fr
```

```
# Tests
>>> F=np.array([0,1,3,4,2])
>>> Fr=reciproque(F)
>>> compo(Fr,F)
array([0., 1., 2., 3., 4.])
>>> compo(F,Fr)
array([0., 1., 2., 3., 4.])
>>> Fr
array([0., 1., 4., 2., 3.])
```

On aurait aussi la commande `np.where()` pour trouver directement l'indice.