Mardi 26 mai

Certains énoncés ont été transmis par des candidats aux concours en 2025 - merci à eux.

Autant que possible, j'ai veillé à établir des énoncés mathématiquement corrects, faites-moi signe si vous rencontrez des points litigieux.

Modules utilisés	
136-py-26	`numpy.random`
136-py-62	`numpy`, `scipy.integrate`
136-py-67	`numpy`, `numpy.random`, `scipy.integrate`, `matplotlib.pyplot`
136-py-68	`numpy`, `numpy.random`, `matplotlib.pyplot`
136-py-71	`numpy`, `numpy.random`, `matplotlib.pyplot`
Centrale25-I03	`numpy`, `numpy.linalg`

136-py-26

Soit $n \geqslant 2$. On considère l'ensemble $\mathfrak{S}_n$ des permutations de l'intervalle $[\![0, n-1]\!]$. On note $\mathcal{T}_n$, l'ensemble des transpositions de $\mathfrak{S}_n$ et $\mathfrak{A}_n$, l'ensemble des permutations paires.

[**1.**] *Une permutation $\sigma \in \mathfrak{S}_n$ est représentée par la liste*

$$\big[\sigma(0), \sigma(1), \dots, \sigma(n-1)\big].$$

[**1.a.**] *Écrire une fonction* `compose(s, t)` *dont les arguments* `s` *et* `t` *représentent deux permutations σ et τ, qui renvoie la permutation $\sigma \circ \tau$.*

[**1.b.**] *Écrire une fonction* `transposition_alea(n)` *dont l'argument est un entier $n \geqslant 1$, qui renvoie une transposition prise au hasard dans l'ensemble $\mathcal{T}_n$.*

[**1.c.**] *Écrire une fonction* `composition_transposition(n, k)` *qui renvoie la composée de k transpositions prises au hasard dans $\mathcal{T}_n$.*

[**1.d.**] *Pour $\sigma \in \mathfrak{S}_3$ et $k \in [\![31, 34]\!]$, estimer la fréquence avec laquelle l'exécution de* `composition_transposition(3, k)` *renvoie la permutation σ. Quelle conjecture peut-on en déduire ?*

On considère une suite $(T_k)_{k\geqslant 1}$ de variables aléatoires indépendantes qui suivent toutes la loi uniforme sur $\mathcal{T}_n$. On pose alors $X_0 = I$ et, pour $k \in \mathbb{N}^*$,

$$X_k = T_k \circ T_{k-1} \circ \cdots \circ T_1 \circ X_0.$$

[**2.**] *Calculer $\mathbf{P}(X_k \in \mathfrak{A}_n)$ en fonction de k.*

[**3.**] *Pour $\tau \in \mathcal{T}_n$ et $k \in \mathbb{N}^*$, calculer $\mathbf{P}(T_k = \tau)$.*

[**4.**] *Soient $\sigma \in \mathfrak{S}_n$ et $k \in \mathbb{N}$. Démontrer que*

$$\mathbf{P}(X_{k+1} = \sigma) = \frac{2}{n(n-1)} \sum_{\tau \in \mathcal{T}_n} \mathbf{P}(X_k = \tau \circ \sigma).$$

[**5.**] *Pour $k \in \mathbb{N}$, on note $U_k \in \mathfrak{M}_{n!,1}(\mathbb{R})$, la matrice ligne qui représente la loi de la variable aléatoire X_k :*

$$U_k = \big(\mathbf{P}(X_k = \sigma)\big)_{\sigma \in \mathfrak{S}_n}.$$

Démontrer l'existence d'une matrice

$$M = (M_{\sigma,\sigma'})_{(\sigma,\sigma') \in \mathfrak{S}_n \times \mathfrak{S}_n} \in \mathfrak{M}_{n!}(\mathbb{R})$$

telle que

$$\forall\, k \in \mathbb{N}, \qquad U_{k+1} = M U_k.$$

Expliciter la valeur de $M_{\sigma,\sigma'}$.

[**6.**] *Démontrer que la matrice M est diagonalisable.*

[**7.**] *Démontrer que $\mathrm{Sp}(M) \subset [-1, 1]$. Préciser les sous-espaces propres associés à 1 et à -1.*

[**8.**] *Soit $\sigma \in \mathfrak{S}_n$. Calculer les limites de $\mathbf{P}(X_{2k} = \sigma)$ et de $\mathbf{P}(X_{2k+1} = \sigma)$ quand k tend vers $+\infty$.*

[**9.**] *En utilisant la suite $(X_k)_{k\geqslant 1}$, construire une suite $(Y_k)_{k\geqslant 1}$ de variables aléatoires telles que*

$$\forall\, \sigma \in \mathfrak{S}_n, \qquad \lim_{k \to +\infty} \mathbf{P}(Y_k = \sigma) = \frac{1}{n!}.$$

[**1.a.**] Il est inutile d'itérer sur des indices, il suffit d'itérer sur les éléments de la liste `t`.

```python
def compose(s, t):
    return [s[x] for x in t]
```

[**1.b.**] Un générateur pseudo-aléatoire a vocation à être exécuté de très nombreuses fois, puisque c'est la Loi des grands nombres qui lui donne son utilité. Il faut donc chercher à produire un code qui s'exécute aussi vite que possible.

❧ **Première version.**

Une idée possible consiste à créer, *une fois pour toutes*, la liste $\mathcal{T}_n$ des transpositions de $\mathfrak{S}_n$ et de choisir ensuite un élément au hasard dans cette liste (la fonction `randint` nous permet de choisir un indice au hasard, de manière uniforme).

Cette méthode permet de limiter la complexité temporelle (= la vitesse d'exécution) au moyen d'une complexité spatiale accrue (= la quantité d'information stockée en mémoire).

On définit un dictionnaire `T` dont les clés sont les entiers n et les valeurs, les listes de listes représentant les ensembles $\mathcal{T}_n$.

```
T = {}

def lister_transpositions(n):
    L = []
    for i in range(n):
        for j in range(i+1, n):
            t = list(range(n)) # Identité
            t[i], t[j] = t[j], t[i]
            L.append(t)
    T[n] = len(L), L
```

Il n'y a alors plus qu'à piocher au hasard dans une liste, après s'être assuré que la liste est bien définie.

```
def transposition_alea(n):
    if n not in T: # s'il le faut, on initialise
        lister_transpositions(n)
    N, Tn = T[n]
    return Tn[rd.randint(N)]
```

❧ **Deuxième version.**

Une autre idée consiste à choisir deux entiers i et j dans l'intervalle $[\![0, n[\![$ et, s'ils sont bien distincts, renvoyer la transposition $(i \quad j)$.

```
def transposition_alea_var(n):
    t = list(range(n))
    i, j = rd.randint(0, n, 2)
    while i==j:
        i, j = rd.randint(0, n, 2)
    t[i], t[j] = t[j], t[i]
    return t
```

On gagne en complexité spatiale (on ne stocke plus aucune information), mais on perd en complexité temporelle : on risque d'effectuer beaucoup plus d'appels à `randint` que dans la version précédente.

Les tests montrent que le coût unitaire d'un appel à `randint` n'est pas négligeable, car l'exécution de cette deuxième fonction est sensiblement plus longue que celle de la première version.

❧ **Une vérification importante.**

Par défaut, un générateur aléatoire doit renvoyer des valeurs prises aléatoirement certes, mais surtout uniformément dans l'ensemble de toutes les valeurs possibles.

Si `randint` fait bien son métier, il est clair que les valeurs renvoyées par la fonction `transposition_alea` sont uniformément réparties dans $\mathcal{T}_n$.

Le code suivant permet de comparer les performances des deux fonctions précédentes : on exécute un certain nombre de fois un générateur aléatoire et on compte le nombre d'occurences de chaque transposition.

```
def test_uniforme(n, generateur, nb_essais=1000):
    if n not in T:
        lister_transpositions(n)
    N, Tn = T[n]
    nb_occurences = [0]*N
    for _ in range(nb_essais):
        i = Tn.index(generateur(n))
        nb_occurences[i] += 1
    return nb_occurences
```

Il apparaît que, de ce point de vue, les deux générateurs ont des performances tout à fait comparables.

✍ *Soit $\tau \in \mathcal{T}_n$. On peut modéliser le nombre d'occurences de cette transposition lors de l'exécution répétée N fois d'un générateur aléatoire par*

$$\sum_{k=1}^{N} \mathbb{1}_{[X_k=\tau]}$$

où les variables aléatoires $X_1, \dots, X_N$ sont indépendantes et suivent toutes la loi uniforme sur $\mathcal{T}_n$. Par conséquent, le nombre d'occurences suit la loi binomiale $\mathcal{B}(N,p)$ où $p = 1/\#(\mathcal{T}_n)$.

D'après l'inégalité de Bienaymé-Tchebychev,

$$\forall\, \alpha > 0, \qquad \mathbf{P}\left(\left|p - \frac{1}{N}\sum_{k=1}^{N} \mathbb{1}_{[X_k=\tau]}\right| \geqslant \alpha\right) \leqslant \frac{p(1-p)}{N\alpha^2} \leqslant \frac{1}{4N\alpha^2}.$$

Avec $N = 10^3$ et $\alpha = 10^{-1}$, le majorant est égal à $1/40$. Il y a donc au plus $2,5\%$ de chances pour que la fréquence d'apparition de τ ne soit pas comprise entre $p-0,1$ et $p+0,1$. Autrement dit, il y a au moins $97,5\%$ de chances pour que la fréquence observée soit égale à p à $\pm 0,1$ près : on ne doit donc pas être étonné que la fréquence observée ne soit pas très proche de la probabilité attendue.

✍ *Considérons une suite $(X_n)_{n\in\mathbb{N}}$ de variables aléatoires indépendantes, qui suivent toutes la loi uniforme sur $[\![0,n[\![$: ces variables aléatoires modélisent la succession des valeurs renvoyées par la fonction* `randint`.

La variable aléatoire

$$T = \min\{k \in \mathbb{N} \,:\, X_{2k} \neq X_{2k+1}\}$$

nous permet de modéliser les indices `i` *et* `j` *calculés par la fonction* `transposition_alea_var` *: l'indice* `i` *est égal à X_{2T} et l'indice* `j` *à X_{2T+1}.*

On peut démontrer que la loi du couple (X_{2T}, X_{2T+1}) est uniforme sur l'ensemble

$$E = \{(i,j) \in [\![0,n[\![^2 \,:\, i \neq j\},$$

c'est-à-dire

$$\forall\, 0 \leqslant i \neq j < n, \qquad \mathbf{P}(X_{2T} = i, X_{2T+1} = j) = \frac{1}{n(n-1)}.$$

(Il suffit de conditionner par le système complet d'évènements $([T=k])_{k\in\mathbb{N}}$ et exprimer l'évènement $[T=k]$ au moyen des variables aléatoires X_i.)

Si le générateur `transposition_alea_var` *est moins efficace, il a au moins le bon goût de renvoyer des valeurs uniformément réparties sur l'ensemble étudié (constaté sur les tests, vérifié par la théorie).*

[1.c.] Code sans mystère !

```
def composition_transposition(n, k, generateur=transposition_alea):
    if n not in T:
        lister_transpositions(n)
    s = list(range(n)) # Identité
    for _ in range(k):
        s = compose(generateur(n), s)
    return s
```

[1. d.] Le plus simple pour établir la liste des permutations de $\mathfrak{S}_3$, c'est encore de les écrire à la main : il n'y en a que six.

Le reste du code est sans mystère. J'en profite pour faire passer un **message** : choisissez des noms explicites pour vos variables, ce sera plus facile de rédiger code et aussi plus facile de le corriger si c'est nécessaire. Tout éditeur de code digne de ce nom permet de compléter les noms des variables (avec la touche Tabulation), on n'a donc aucune excuse !

```
S3 = [[0,1,2],[0,2,1],[1,0,2],[1,2,0],[2,0,1],[2,1,0]]

def test_frequence(N=1000):
    for nb_transpo in range(31, 35):
        print(f"Nombre de transpositions = {nb_transpo}")
        for s in S3:
            nb_occurences = 0
            for _ in range(N):
                t = composition_transposition(3, nb_transpo)
                if t==s:
                    nb_occurences += 1
            print(f"s = {s} --> {nb_occurences/N}")
        print() # saut de ligne
```

On constate que, selon la parité de k, certaines permutations apparaissent environ une fois sur trois et que les autres n'apparaissent pas du tout.

[2.] La signature d'une permutation est égale à -1.

En composant k transpositions, on obtient toujours une permutation dont la signature est égale à $(-1)^k$.

Si k est pair, alors $[X_k \in \mathfrak{A}_n] = \Omega$. Si k est impair, alors $[X_k \in \mathfrak{A}_n] = \varnothing$.

[3.] Une transposition est caractérisée par son support et son support est une partie de deux éléments choisis dans l'intervalle $[\![0, n[\![$. Donc le cardinale de $\mathcal{T}_n$ est égal à $\binom{n}{2}$ et, par hypothèse d'équiprobabilité,

$$\forall\, \tau \in \mathcal{T}_n,\ \forall\, k \in \mathbb{N}^*, \qquad \mathbf{P}(T_k = \tau) = \frac{1}{\binom{n}{2}} = \frac{2}{n(n-1)}.$$

[4.] Il suffit de remarquer que $X_{k+1} = T_{k+1} \circ X_k$ et d'introduire le système complet d'évènements associée à la variable aléatoire T_{k+1}, la formule des probabilités totales donne le résultat.

[5.]

[6.] La matrice M est symétrique réelle, donc elle est diagonalisable.

[7.] La matrice M est stochastique, exercice classique à propos du Théorème de Perron-Frobenius.

[8.]

[9.] Pour k "suffisamment grand", les variables aléatoires X_{2k} et X_{2k+1} donnent un résultat partiel. Il suffit de les mélanger au moyen d'une variable aléatoire de Bernoulli de loi $\mathscr{B}(1/2)$, indépendante de X_{2k} et de X_{2k+1}, qui va déterminer la signature de la permutation renvoyée.

```
def permutation_uniforme(n):
    if n not in T:
        lister_transpositions(n)
    parite = rd.binomial(1, 0.5)
    K = 20*n+parite
    return composition_transposition(n, K)
```

On peut (on doit !) effectuer quelques tests.

```
def test_uniforme_S3(N=1000):
    for s in S3:
        nb_occurences = 0
        for _ in range(N):
            t = permutation_uniforme(3)
            if t==s:
                nb_occurences += 1
        print(f"s = {s} --> {nb_occurences/N}")
```

Les résultats produits sont assez satisfaisants !

136-py-62

On note

$$\zeta(x)=\sum_{n=1}^{+\infty}\frac{1}{n^x}\qquad\text{et}\qquad\forall\,k\in\mathbb{N},\quad\varphi_k=x\mapsto\frac{\lfloor x\rfloor}{x^{k+1}}.$$

[**1.**] *Déterminer l'ensemble* K *des entiers* $k\in\mathbb{N}$ *tels que la fonction* φ_k *soit intégrable sur* $[1,+\infty[$.
[**2.**] *Pour* $n\in\mathbb{N}^*$, *on pose*

$$u_n=\frac{1}{n}-\ln\left(1+\frac{1}{n}\right).$$

[**2.a.**] *Démontrer que la série* $\sum u_n$ *est convergente.*
[**2.b.**] *On note* γ, *la somme de cette série. Calculer une valeur approchée de* γ.
[**3.a.**] *Déterminer l'ensemble de définition* D *de la fonction* ζ.
[**3.b.**] *Pour* $x\in D$, *étudier la monotonie de la fonction* $t\mapsto t^{-x}$ *sur* $\mathbb{R}_+^*$. *En déduire que*

$$\forall\,x\in D,\qquad\zeta(x)\leqslant 1+\frac{1}{x-1}.$$

[**4.**] *Démontrer que la série*

$$\sum\frac{(-1)^k\zeta(k)}{k}$$

est convergente.
[**5.**] *Pour* $k\in[\![2,10]\!]$, *calculer des valeurs numériques approchées de*

$$\frac{\zeta(k)}{\int_1^{+\infty}\varphi_k(x)\,dx}.$$

Quelle conjecture peut-on en déduire ?
[**6.**] *Pour* $n\geqslant 2$, *on pose*

$$S_n=\sum_{k=2}^{n}\frac{(-1)^k\zeta(k)}{k}.$$

[**6.a.**] *Écrire une fonction* `S(n)` *qui renvoie la valeur de* S_n.
[**6.b.**] *Afficher les valeurs de* S_n *et de*

$$\frac{1}{S_n}\cdot\int_1^{+\infty}\frac{\lfloor x\rfloor}{x^2(x+1)}\,dx.$$

pour $n\in[\![90,100]\!]$. *Quelle conjecture peut-on en déduire ?*
[**7.**] *Démontrer les conjectures faites au* **[5.]** *et au* **[6.b.]**.

[**1.**] La fonction φ_k est intégrable sur $[1,+\infty[$ si, et seulement si, l'entier k est supérieur à 2.
[**2.a.**] Un développement limité montre que $u_n=\mathcal{O}(1/n^2)$, donc la série $\sum u_n$ est absolument convergente.
[**2.b.**]

```
def u(n):
    return 1/n-np.log(1+1/n)

N = 1000
gamma = 0
for n in range(1, N+1):
    gamma += u(n)
```

$N=10^3$	$N=10^5$	$N=10^6$
$\gamma\approx 0,57671608$	$\gamma\approx 0,57721066$	$\gamma\approx 0,57721516$

[**3.a.**] La fonction ζ est définie pour $x\in\,]1,+\infty[$.

[3. b.] Comme $-x < 0$ et que ℓn est croissante, la fonction $t \mapsto t^{-x}$ est décroissante sur $\mathbb{R}_+^*$. En comparant somme et intégrale, on obtient

$$\forall\, x > 1, \qquad \zeta(x) \leqslant \frac{1}{1^x} + \int_1^{+\infty} \frac{dt}{t^x} = 1 + \frac{1}{x-1}.$$

[4.] Il est clair que $\zeta(x) \geqslant 1$ pour tout $x > 1$.

✍ *Le premier terme est égal à 1, tous les autres sont positifs.*

On déduit alors de l'inégalité précédente que

$$\zeta(x) \underset{x\to+\infty}{=} 1 + \mathcal{O}\Big(\frac{1}{x}\Big)$$

et donc que

$$\frac{(-1)^k \zeta(k)}{k} \underset{k\to+\infty}{=} \frac{(-1)^k}{k} + \mathcal{O}\Big(\frac{1}{k^2}\Big).$$

La série $\sum \frac{(-1)^k \zeta(k)}{k}$ est donc la somme d'une série semi-convergente (avec le Critère spécial des séries alternées) et d'une série absolument convergente (comparaison avec une série de Riemann), donc cette série est (semi-)convergente.

[5.]

```
def dzeta(k, N=1000):
    s = 0
    for n in range(1, N+1):
        s += 1/n**k
    return s

def phi(k):
    def phi_k(x):
        return np.floor(x)/x**k
    return phi_k

def test_1(k_min, k_max):
    for k in range(k_min, k_max+1):
        numerateur = dzeta(k)
        denominateur = integr.quad(phi(k), 1, np.inf)[0]
        print(numerateur/denominateur)
```

Au vu des résultats, on conjecture aisément que

$$\zeta(k) \underset{k\to+\infty}{\sim} k \int_1^{+\infty} \varphi_k(x)\, dx.$$

❧ On ne doit pas prendre à la légère le message `IntegrationWarning` : la fonction `quad` arrive au bout de ses possibilités, la valeur approchée de l'intégrale n'est plus considérée comme satisfaisante pour $k = 10$.

[6. a.]

```
def S(n):
    s = 0
    for k in range(2, n+1):
        s += (-1)**k*dzeta(k)/k
    return s
```

[6. b.]

```
def psi(x):
    return np.floor(x)/(x**2*(x+1))

def test_2(k_min, k_max):
    denominateur = integr.quad(psi, 1, np.inf)[0]
    for k in range(k_min, k_max+1):
        print(S(k)/denominateur)
```

Les valeurs calculées sont très proches de 1, ce qui suggère que

$$\lim_{n\to+\infty} S_n = \int_1^{+\infty} \frac{\lfloor x\rfloor}{x^2(x+1)}\,dx.$$

[7.] Conjecture du [5.]

Pour $k \geqslant 2$, la fonction φ_k est intégrable sur $[1,+\infty[$, donc on peut appliquer la relation de Chasles :

$$\int_1^{+\infty} \varphi_k(x)\,dx = \int_1^{+\infty} \frac{\lfloor x\rfloor}{x^{k+1}}\,dx = \sum_{n=1}^{+\infty}\int_n^{n+1} \frac{n}{x^{k+1}}\,dx = \sum_{n=1}^{+\infty} \frac{n}{k}\Big(\frac{1}{n^k} - \frac{1}{(n+1)^k}\Big).$$

Pour $k \geqslant 3$, on pourrait scinder la somme en faisant apparaître deux séries convergentes, mais c'est impossible pour $k = 2$ (on ferait apparaître deux séries divergentes). Il faut donc prendre les précautions habituelles pour démontrer le résultat général.

Pour tout entier $N \geqslant 1$,

$$\begin{aligned}\sum_{n=1}^{N} \frac{n}{k}\Big(\frac{1}{n^k} - \frac{1}{(n+1)^k}\Big) &= \frac{1}{k}\Big[\sum_{n=1}^{N}\frac{n}{n^k} - \sum_{n=1}^{N-1}\frac{n+1-1}{(n+1)^k} - \frac{N}{(N+1)^k}\Big]\\ &= \frac{1}{k}\Big[1 + \sum_{n=2}^{N}\frac{n-(n-1)}{n^k} - \frac{N}{(N+1)^k}\Big]\\ &= \frac{1}{k}\sum_{n=1}^{N}\frac{1}{n^k} - \frac{N}{k(N+1)^k} \xrightarrow[N\to+\infty]{} \frac{\zeta(k)}{k}.\end{aligned}$$

On a démontré que

$$\forall\, k \geqslant 2, \qquad \int_1^{+\infty} \varphi_k(x)\,dx = \frac{\zeta(k)}{k}$$

et la conjecture du **[5.]** est démontrée.

❧ **Conjecture du [6.b.]**

Pour les mêmes raisons,

$$\int_1^{+\infty} \frac{\lfloor x\rfloor}{x^2(x+1)}\,dx = \sum_{n=1}^{+\infty} n\int_n^{n+1} \frac{dx}{x^2(x+1)}.$$

On calcule la décomposition en éléments simples sans difficulté :

$$\frac{1}{x^2(x+1)} = \frac{-1}{x} + \frac{1}{x^2} + \frac{1}{x+1}$$

et on en déduit que

$$\begin{aligned}\forall\, n \geqslant 1, \qquad \int_n^{n+1} \frac{dx}{x^2(x+1)} &= \ell n\,\frac{n}{n+1} + \ell n\,\frac{n+2}{n+1} + \frac{1}{n} - \frac{1}{n+1}\\ &= \Big[\frac{1}{n} - \ell n\Big(1+\frac{1}{n}\Big)\Big] - \Big[\frac{1}{n+1} - \ell n\Big(1+\frac{1}{n+1}\Big)\Big]\end{aligned}$$

Par conséquent,

$$\int_1^{+\infty} \frac{\lfloor x\rfloor}{x^2(x+1)}\,dx = \sum_{n=1}^{+\infty} n(u_n - u_{n+1}). \qquad (\star)$$

✍ *Encore une transformation d'Abel !*

Pour tout entier $N \geqslant 1$,

$$\begin{aligned}\sum_{n=1}^{N} n(u_n - u_{n+1}) &= \sum_{n=1}^{N} nu_n - \sum_{n=2}^{N+1}(n-1)u_n\\ &= u_1 + \sum_{n=2}^{N}[n-(n-1)]u_n - Nu_{N+1} = \sum_{n=1}^{N} u_n - Nu_{N+1}.\end{aligned}$$

On sait que $u_n = \mathcal{O}(1/n^2)$ lorsque n tend vers $+\infty$ et que la série $\sum u_n$ converge, donc

$$\int_1^{+\infty} \frac{\lfloor x \rfloor}{x^2(x+1)}\,dx = \sum_{n=1}^{+\infty} u_n = \gamma.$$

✍ *Intéressant, mais cela ne nous rapproche pas vraiment de S_n...*

Partons donc de la somme partielle S_n.

La suite de terme général $\zeta(k)$ est décroissante et positive, donc la suite de terme général $\zeta(k)/k$ est décroissante et tend vers 0. D'après le Critère spécial des séries alternées, la somme S_n est donc une somme partielle d'une série convergente. Il nous reste à déterminer sa somme.

Pour tout $n \geqslant 2$,

$$\begin{aligned} S_n = \sum_{k=2}^{n} (-1)^k \frac{\zeta(k)}{k} &= \sum_{k=2}^{n} \frac{(-1)^k}{k}\big[1 + \big(\zeta(k) - 1\big)\big] \\ &= \sum_{k=2}^{n} \frac{(-1)^k}{k} + \sum_{k=2}^{n} \frac{(-1)^k}{k} \sum_{m=2}^{+\infty} \frac{1}{m^k}. \end{aligned}$$

Le premier terme du second est une somme partielle d'une série convergente (Critère spécial des séries alternées).

Considérons ensuite la famille définie par

$$\forall\, k \geqslant 2,\ \forall\, m \geqslant 2, \qquad v_{k,m} = \frac{(-1)^k}{k} \cdot \frac{1}{m^k}.$$

Pour tout $k \geqslant 2$, la sous-famille $(|v_{k,m}|)_{m \geqslant 2}$ est sommable (série de Riemann), de somme

$$0 \leqslant \sigma_k = \frac{\zeta(k) - 1}{k}.$$

Une comparaison somme/intégrale nous donne

$$\forall\, k \geqslant 2, \qquad 0 \leqslant \sum_{m=3}^{+\infty} \frac{1}{m^k} \leqslant \int_2^{+\infty} \frac{dt}{t^k} = \frac{1}{k-1} \cdot \frac{1}{2^{k-1}}.$$

Par conséquent,

$$\zeta(k) - 1 = \frac{1}{2^k} + \sum_{m=3}^{+\infty} \frac{1}{m^k} \underset{k \to +\infty}{\sim} \frac{1}{2^k},$$

ce qui prouve que la série $\sum \sigma_k$ est convergente. D'après le Théorème de Fubini, la famille $(v_{k,m})_{k \geqslant 2, m \geqslant 2}$ est sommable et de ce fait, la série

$$\sum \frac{(-1)^k}{k} \sum_{m=2}^{+\infty} \frac{1}{m^k}$$

est convergente et sa somme est égale à

$$\sum_{k=2}^{+\infty} \frac{(-1)^k}{k} \sum_{m=2}^{+\infty} \frac{1}{m^k} = \sum_{m=2}^{+\infty} \sum_{k=2}^{+\infty} (-1)^k \frac{(1/m)^k}{k} = \sum_{m=2}^{+\infty} \Big[\frac{1}{m} - \ln\Big(1 + \frac{1}{m}\Big)\Big].$$

✍ *On doit reconnaître, au signe près, le développement en série entière de $\ln(1+x) - x$ où on a pris $x = 1/m$, qui se situe bien dans l'intervalle ouvert de convergence $]-1, 1[$.*

Par conséquent, la suite (S_n) converge vers

$$\sum_{k=2}^{+\infty} \frac{(-1)^k}{k} + \sum_{k=2}^{+\infty} \Big[\frac{1}{k} - \ln\Big(1 + \frac{1}{k}\Big)\Big].$$

✍ *En changeant d'indice ($m \leftarrow k$), on se donne les moyens de deviner la suite des aventures.*

On sait (c'est dans le cours) que

$$\forall\, x \in]-1,1[, \qquad \ell n(1+x) - x = -x + \sum_{k=1}^{+\infty} \frac{(-1)^{k+1}}{k} x^k = -\sum_{k=2}^{+\infty} \frac{(-1)^k}{k} x^k. \tag{$\star\star$}$$

Pour $x = 1$, la série du second membre est encore convergente (Critère spécial des séries alternées), donc la somme de la série entière est continue sur $]-1,1]$ (Théorème d'Abel). Comme la fonction $x \mapsto -x + \ell n(1+x)$ est également continue sur $]-1,1]$, on peut prolonger l'égalité ($\star\star$) à $x = 1$ et en déduire que

$$1 - \ell n(1+1) = \sum_{k=2}^{+\infty} \frac{(-1)^k}{k}.$$

Autrement dit,

$$\begin{aligned}\lim_{n\to+\infty} S_n = \big[1 - \ell n(1+1)\big] + \sum_{k=2}^{+\infty} \Big[\frac{1}{k} - \ell n\Big(1+\frac{1}{k}\Big)\Big] &= \sum_{k=1}^{+\infty} \Big[\frac{1}{k} - \ell n\Big(1+\frac{1}{k}\Big)\Big] \\ &= \int_1^{+\infty} \frac{\lfloor x \rfloor}{x^2(x+1)}\, dx\end{aligned}$$

ainsi qu'on l'a vu plus haut ($\star$).

❧ Le point remarquable de ce calcul est d'obtenir :

$$\int_1^{+\infty} \frac{\lfloor x \rfloor}{x^2(x+1)}\, dx = \sum_{(=-\infty}^{+\infty} 1)^k \frac{\zeta(k)}{k} = \sum_{n=1}^{+\infty} u_n = \gamma.$$

136-py-67

On considère les fonctions

$$f = \left[(x,y) \mapsto \frac{1}{1-y^2}\,\ell n\,\frac{x+y}{1+xy}\right] \qquad \textit{et} \qquad F = x \mapsto \int_0^1 f(x,y)\,dy.$$

[**1.**] *Déterminer l'ensemble de définition* D *de* f. *Démontrer que* D *est un ouvert.*
[**2.**] *Soit* S_D, *l'ensemble des fonctions* $\varphi \in \mathscr{C}^1(D,\mathbb{R})$ *telles que*

$$\forall\,(x,y) \in D, \quad 2y\varphi(x,y) + (1-x^2)\frac{\partial\varphi}{\partial x}(x,y) - (1-y^2)\frac{\partial\varphi}{\partial y}(x,y) = 0.$$

Vérifier que $f \in S_D$.
[**3.**] *Démontrer que*

$$\forall\,(x,y) \in \mathbb{R}_+ \times]0,1[, \qquad \left|\ell n\,\frac{x+y}{1+xy}\right| \leqslant |\ell n\, y|.$$

En déduire que F *est bien définie sur* $\mathbb{R}_+$.
[**4.**] *Tracer la courbe représentative de* F *sur l'intervalle* $[0,50]$. *Calculer* $F(0)$ *et* $F(10^4)$. *Quelle conjecture peut-on en déduire ?*
[**5.**] *Exprimer* $F(1/x)$ *à l'aide de* $F(x)$ *et revenir sur la conjecture précédente.*

[**1.**] L'ensemble de définition D de la fonction f est clairement égal à

$$[1-y^2 \neq 0] \cap \left[\frac{x+y}{1+xy} > 0\right].$$

L'ensemble $[1-y^2 \neq 0]$ est l'image réciproque de l'ouvert $\mathbb{R}^*$ par la fonction continue $(x,y) \mapsto 1-y^2$.

✍ *Continue, car polynomiale bien entendu.*
Mais attention, on étudie ici une partie de $\mathbb{R}^2$, *il faut bien préciser que c'est une fonction polynomiale du couple* (x,y) *— et non pas une fonction polynomiale de* y *seulement.*

D'autre part, l'ensemble $[\frac{x+y}{1+xy} > 0]$ est l'image réciproque de l'ouvert $]0,+\infty[$ par une fonction continue (fonction rationnelle du couple (x,y) à nouveau).
Donc D est une partie ouverte de $\mathbb{R}^2$ en tant qu'intersection de deux parties ouvertes.

✍ *Il est clair que l'ensemble de définition* D *contient le quart de plan* $\mathbb{R}_+ \times \mathbb{R}_+$. *Mais encore ? Question de technique élémentaire, mais question redoutable néanmoins.*

❧ Fixons $y \in \mathbb{R} \setminus \{\pm 1\}$ et considérons la fonction

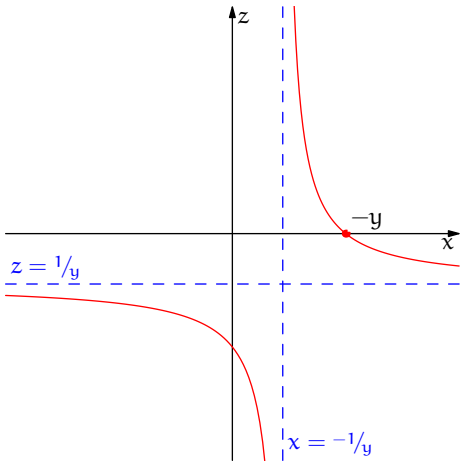
$$x \longmapsto z_y(x) = \frac{x+y}{1+xy}.$$

Nous cherchons pour quelles valeurs de x la fonction z_y prend des valeurs positives.
Le réel y étant fixé, il faut prendre conscience que la fonction z_y est une **homographie** et que son étude peut être simplifiée :

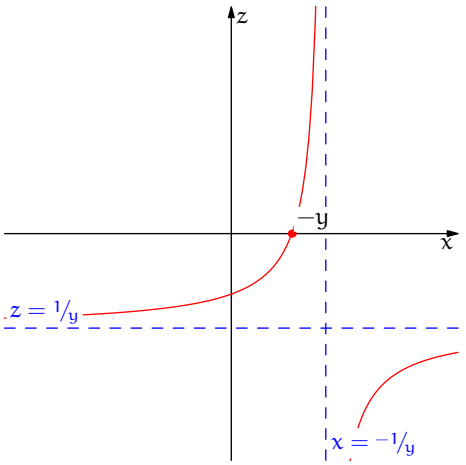
$$\forall\, x \in \mathbb{R} \setminus \{-1/y\}, \qquad z_y(x) = \frac{1}{y} + \frac{y^2-1}{y}\cdot\frac{1}{1+yx}$$
$$z_y'(x) = \frac{1-y^2}{(1+xy)^2}. \qquad (\dagger)$$

De part et d'autre de l'asymptote verticale $[x = -1/y]$, la fonction z_y est monotone :
— décroissante pour $|y| > 1$;
— croissante pour $|y| < 1$.

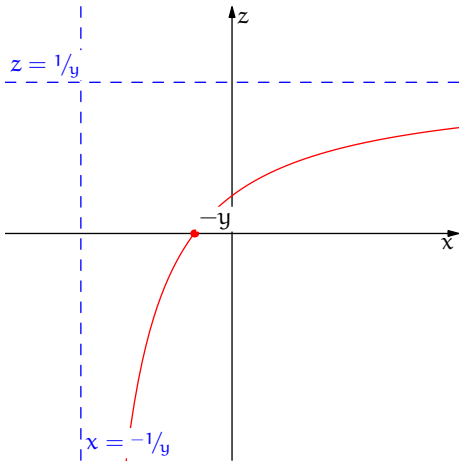
Il est par ailleurs clair que $z_y(x)$ s'annule en $x = -y$. On distingue donc quatre cas.



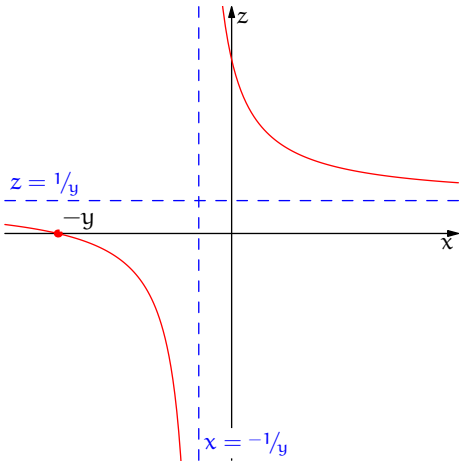
$y < -1$



$-1 < y < 0$



$0 < y < 1$



$y > 1$

— Si $y < -1$, alors $z_y(x) > 0$ si, et seulement si, $-1/y < x < -y$, c'est-à-dire

$$y < -x \qquad \text{et} \qquad y < \frac{-1}{x}$$

(puisque $x > 0$ et $y < 0$).

— Si $-1 < y < 0$, alors $z_y(x) > 0$ si, et seulement si, $-y < x < -1/y$, c'est-à-dire

$$y > -x \qquad \text{et} \qquad y > \frac{-1}{x}$$

(pour les mêmes raisons de signe).

— Si $0 < y < 1$, alors $z_y(x) > 0$ si, et seulement si, $x < -1/y$ ou $x > -y$, c'est-à-dire

$$0 < y < 1 \qquad \text{et} \qquad \left[y > -x \quad \text{ou} \quad y > \frac{-1}{x}\right]$$

(puisque $y > 0$ et, dans le second cas, $x < 0$).

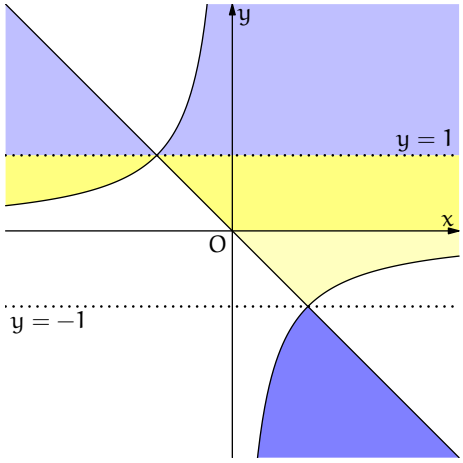
— Si $y > 1$, alors $z_y(x) > 0$ si, et seulement si, $x < -y$ ou $x > -1/y$, c'est-à-dire

$$y > 1 \qquad \text{et} \qquad \left[y < -x \quad \text{ou} \quad y < \frac{-1}{x}\right].$$

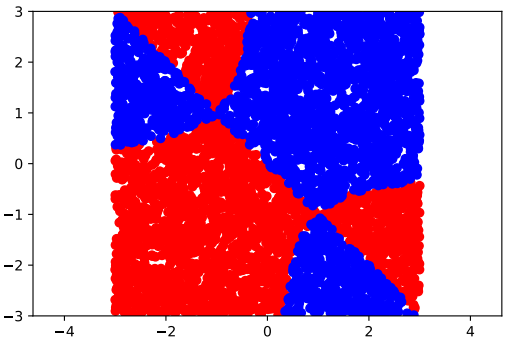
(Il faut discuter sur le signe de x...)

— Et bien entendu il reste un cinquième cas : si $y = 0$, alors $z_y(x) > 0$ si, et seulement si, $x > 0$.

L'ensemble de définition D est la partie colorée de la figure suivante.



✍ *On peut aussi déterminer* D *à l'aide de la "méthode de Monte-Carlo" : le domaine* D *est colorié en bleu et son complémentaire en rouge.*




```
def quotient(x, y):
    return (x+y)/(1+x*y)

a, N = 3, 5000
Z = a*(2*rd.random((N,2))-1)
Px, Py = [], [] # le quotient est positif
Nx, Ny = [], [] # le quotient est négatif
for z in Z:
    x, y = z
    if quotient(x, y)>0:
        Px.append(x)
        Py.append(y)
    else:
        Nx.append(x)
        Ny.append(y)
plt.plot(Nx, Ny, 'ro')
plt.plot(Px, Py, 'bo')
```

[**2.**] On déduit de (†) que

$$\frac{\partial f}{\partial x}(x,y) = \frac{1}{1-y^2}\cdot\frac{z_y'(x)}{z_y(x)} = \frac{1}{(x+y)(1+xy)}$$

et, par symétrie des rôles joués par x et y, que

$$\frac{\partial}{\partial y}\left(\frac{x+y}{1+xy}\right) = \frac{1-x^2}{(1+xy)^2}.$$

La formule de dérivation des produits nous donne alors

$$\begin{aligned}\frac{\partial f}{\partial y}(x,y) &= \frac{2y}{1-y^2}\cdot f(x,y) + \frac{1}{1-y^2}\cdot\frac{1-x^2}{(1+xy)^2}\cdot\frac{1+xy}{x+y}\\ &= \frac{1}{1-y^2}\cdot\left(2yf(x,y) + \frac{1-x^2}{(x+y)(1+xy)}\right).\end{aligned}$$

Par conséquent,

$$(1-x^2)\frac{\partial f}{\partial x}(x,y) - (1-y^2)\frac{\partial f}{\partial y}(x,y) = -2yf(x,y)$$

et cela prouve que f appartient bien à l'ensemble S_D.

[3.] On a étudié plus haut les variations de z_y : pour $0 < y < 1$, on sait donc que z_y est strictement positive et strictement croissante sur $\mathbb{R}_+$. On en déduit sans peine que $z_y(x) = 1$ si, et seulement si, $x = 1$.

❧ Si $0 \leqslant x \leqslant 1$, alors $z_y(x) \leqslant 1$, donc les deux logarithmes sont négatifs et il s'agit donc de vérifier que

$$\frac{1+xy}{x+y} \leqslant \frac{1}{y}.$$

✍ *La fonction $\ln$ est strictement croissante.*

Comme $x + y$ et y sont strictement positifs, cette inégalité équivaut à $xy(y-1) \leqslant 0$, ce qui est vrai puisque $x \geqslant 0$ et $0 < y < 1$.

❧ Si $x > 1$, alors $z_y(x) > 1$ et il s'agit cette fois de vérifier que

$$\frac{x+y}{1+xy} \leqslant \frac{1}{y}.$$

Comme $1 + xy$ et y sont strictement positifs, cette inégalité équivaut à $y^2 \leqslant 1$, ce qui est vrai.

On a ainsi démontré que

$$\forall\, x \in \mathbb{R}_+,\ \forall\, y \in]0,1[, \qquad \left|\ln \frac{x+y}{1+xy}\right| \leqslant |\ln y|.$$

❧ Pour tout $x \in \mathbb{R}_+$, la fonction $y \mapsto f(x,y)$ est continue sur $]0,1[$.

✍ *Pour $x > 0$, la fonction $y \mapsto f(x,y)$ est en fait continue sur $[0,1[$, mais cela ne simplifie pas vraiment l'étude.*

On déduit de l'encadrement précédent que

$$\forall\, y \in]0,1[, \qquad \big|f(x,y)\big| \leqslant \big|g(y)\big| \qquad \text{où} \qquad g(y) = \frac{\ln y}{1-y^2}. \tag{$\star$}$$

La fonction g est continue sur l'intervalle ouvert $]0,1[$. Au voisinage de 0, $g(y) \sim \ln y$ et on sait que la fonction $\ln$ est intégrable au voisinage de 0, donc g est intégrable au voisinage de 0. Au voisinage de 1,

$$g(y) = \frac{\ln y}{1-y} \cdot \frac{1}{1+y} \sim \frac{-1}{2},$$

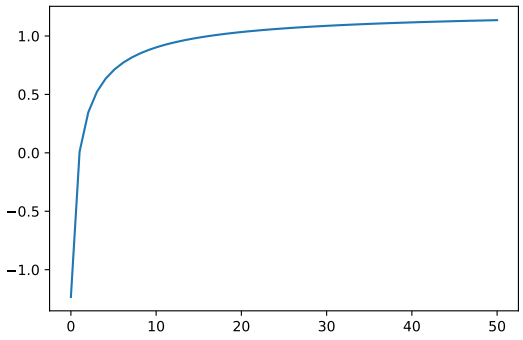
donc g est intégrable au voisinage de 1 (limite finie au voisinage d'une borne finie).

Comme la fonction g est intégrable sur $]0,1[$, la domination $(\star)$ prouve que $y \mapsto f(x,y)$ est intégrable sur $]0,1[$ et donc que $F(x)$ est bien définie pour tout $x \in \mathbb{R}_+$.

[4.] La documentation nous donne la marche à suivre.

```
def F(x):
    def f(y):
        return np.log(quotient(x, y))/(1-y**2)
    return integr.quad(f, 0, 1)[0]

plt.figure()
X = np.linspace(0, 50)
Y = [F(x) for x in X]
plt.plot(X, Y)
```



On obtient également $F(0) \approx -1,2337005501361697$ et $F(10^4) \approx 1.2326795160957926$. Quelle surprise !
Continuons sur cette belle lancée :

$$F(10^4) + F(0) \approx -0,001021034$$
$$F(10^5) + F(0) \approx -0,000125129$$
$$F(10^6) + F(0) \approx -0,000014815$$

Il semble bien que $F(0) = -F(+\infty)$.
[5.] On vérifie que $f(1/x, y) = -f(x, y)$ et on en déduit que

$$\forall\, x > 0, \qquad F(1/x) = -F(x).$$

La domination $(\star)$ permet de prouver que F est continue sur $[0, +\infty[$ et donc que

$$F(0) = F(0^+) = \lim_{x \to +\infty} F(x).$$

136-py-68

On dispose d'une urne bleue et d'une urne rouge. Initialement, l'urne bleue contient n boules bleues et l'urne rouge contient n boules rouges. À chaque étape, on choisit une boule au hasard dans chaque urne et on échange ces deux boules.
On note Z_k, le nombre de boules rouges présentes dans l'urne rouge à l'issue du k-ième échange. On a donc $Z_0 = n$ et on pose

$$\forall\, k \in \mathbb{N}, \qquad \Delta_k = Z_{k+1} - Z_k.$$

[**1.**] *Justifier que Z_k est aussi égal au nombre de boules bleues dans l'urne bleue à l'issue du k-ième échange.*
[**2.**] *Déterminer la loi, l'espérance et la variance de Z_1.*
[**3.**] *Quelles sont les valeurs possibles de Z_k ? Justifier que Z_k est d'espérance finie et admet une variance.*
[**4.**] *On dispose d'une fonction* `simul(n, k)` *qui simule les variables aléatoires Z_k. Écrire une fonction* `esp(n, k)` *qui renvoie une valeur approchée de $\mathbf{E}(Z_k)$ au moyen de 10^3 simulations.*
[**5.**] *Pour $n = 10$, 15 et 20, représenter graphiquement $\mathbf{E}(Z_k)$ pour $k \in [\![0, 30]\!]$. Quelle conjecture peut-on en déduire sur la limite de $\mathbf{E}(Z_k)$?*
[**6. a.**] *Démontrer que*

$$\mathbf{P}(\Delta_k = -1) = \sum_{i=0}^{n} \left(\frac{i}{n}\right)^2 \mathbf{P}(Z_k = i). \qquad (\dagger)$$

[**6. b.**] *Établir une formule similaire pour $\mathbf{P}(\Delta_k = 1)$.*
[**6. c.**] *En déduire une relation de récurrence pour la suite de terme général $\mathbf{E}(Z_k)$.*
[**6. d.**] *Exprimer $\mathbf{E}(Z_k)$ en fonction des paramètres n et k. Démontrer la conjecture faite en* **[5.]**.
[**7.**] *Déterminer la limite de $\mathbf{V}(Z_k)$ quand k tend vers $+\infty$.*

[**1.**] À chaque étape du processus, on échange deux boules. Par conséquent, il y a en permanence n boules dans l'urne rouge et n boules dans l'urne bleue.

S'il y a Z_k boules rouges dans l'urne rouge, alors il y aussi $n - Z_k$ boules rouges (= les autres boules rouges) dans l'urne bleue. Et comme il y a en tout n boules dans l'urne bleue, on en déduit qu'il y a exactement

$$n - (n - Z_k) = Z_k$$

boules bleues dans l'urne bleue.
[**2.**] La première étape ne doit rien au hasard : on dépose nécessairement une boule rouge dans l'urne bleue et une boule bleue dans l'urne rouge. Donc Z_1 suit une loi de Dirac :

$$\mathbf{P}(Z_1 = n - 1) = 1, \qquad \mathbf{E}(Z_1) = n - 1, \qquad \mathbf{V}(Z_1) = 0.$$

[**3.**] En général, la valeur de Z_k est comprise entre 0 (toutes les boules rouges sont passées dans l'urne bleue) et n (toutes les boules rouges se trouvent dans l'urne rouge).

Comme Z_k est une variable aléatoire bornée, elle admet des moments de tous les ordres. En particulier, c'est une variable aléatoire d'espérance finie et de variance finie.
[**4.**] Comme Z_k est une variable aléatoire de variance finie, on peut appliquer la Loi des grands nombres : si $(Z_{k,i})_{0 \leqslant i < N}$ est un échantillon i.i.d. de la loi de Z_k,

$$\frac{1}{N} \sum_{0 \leqslant i < N} Z_{k,i} \approx \mathbf{E}(Z_k).$$

On produit le code suivant en utilisant la fonction donnée.

```
def esp(n, k):
    N = 1000
    somme_valeurs = 0
    for i in range(N):
        somme_valeurs += simul(n, k)
    return somme_valeurs/N
```

✍ *On peut assez facilement coder la fonction* `simul(n, k)`.
À chaque étape, on calcule Z_k/n : c'est la proportion de boules rouges dans l'urne rouge et aussi, comme on l'a remarqué plus haut, la proportion de boules bleues dans l'urne bleue. La valeur de `R` *(resp. de* `B`*) nous indique si on tire une boule rouge dans l'urne rouge (resp. une boule bleue dans l'urne bleue).*

— *Si* R *et* B *sont vraies toutes les deux, alors on remplace une boule rouge de l'urne rouge par une boule bleue, donc* $Z_{k+1} = Z_k - 1$.
— *Si* R *et* B *sont fausses toutes les deux, alors on remplace une boule bleue de l'urne rouge par une boule rouge, donc* $Z_{k+1} = Z_k + 1$.
— *Si* R *est fausse et* B *est vraie, alors on tire une boule bleue dans chaque urne et en échangeant les deux boules, le nombre de boules rouges ne change pas :* $Z_{k+1} = Z_k$.
— *De même, si* R *est vraie et* B *est fausse, alors on échange deux boules rouges et* $Z_{k+1} = Z_k$.

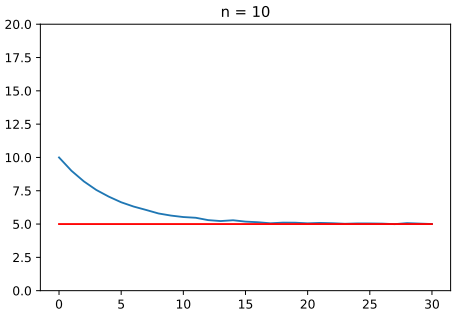
Cela étant compris, le code est transparent.

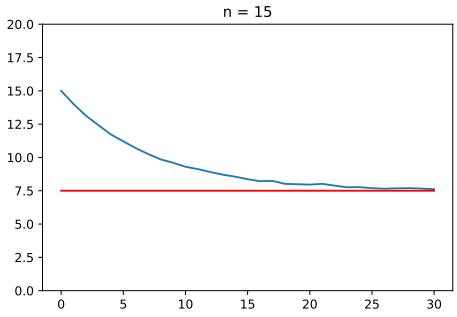
```
def simul(n, k):
    Z = n
    for i in range(k):
        seuil = Z/n
        R = rd.random()<seuil
        B = rd.random()<seuil
        if R==1 and B==1:
            Z = Z-1
        elif R==0 and B==0:
            Z = Z+1
    return Z
```

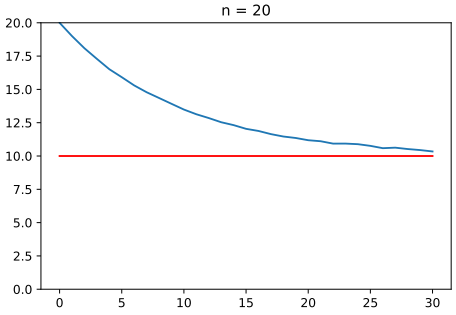
[5.]

```
for n in [10, 15, 20]:
    Z = []
    for k in range(31):
        Z.append(esp(n, k))
    plt.figure()
    plt.title(f"n = {n}")
    plt.plot(Z)
    plt.plot([0, 30], [n/2, n/2], 'r')
```

Il semble bien que $\mathbf{E}(Z_k)$ tende vers $n/2$ lorsque k tend vers $+\infty$.







[6. a.] La famille $([Z_k = i])_{0 \leqslant i \leqslant n}$ est un système complet d'évènements et

$$\mathbf{P}(\Delta_k = -1 \mid Z_k = i) = \frac{\mathbf{P}(Z_{k+1} = Z_k - 1, Z_k = i)}{\mathbf{P}(Z_k = i)} = \mathbf{P}(Z_{k+1} = i - 1 \mid Z_k = i) = \left(\frac{i}{n}\right)^2$$

puisqu'on tire une boule rouge dans l'urne rouge (qui contient n boules, dont i sont rouges) et, de façon indépendante, une boule bleue dans l'urne bleue (qui contient n boules, dont i sont bleues).

✍ *Le terme en $i = 0$ est nul, mais il est prudent de le conserver.*

[**6. b.**] Même raisonnement :

$$\mathbf{P}(\Delta_k = 1 \mid Z_k = i) = \mathbf{P}(Z_{k+1} = i+1 \mid Z_k = i) = \left(\frac{n-i}{n}\right)^2$$

puisqu'on tire cette fois une boule bleue dans l'urne rouge et, de façon indépendante, une boule rouge dans l'urne bleue. On en déduit que

$$\mathbf{P}(\Delta_k = 1) = \sum_{i=0}^{n} \left(\frac{n-i}{n}\right)^2 \mathbf{P}(Z_k = i). \qquad (\ddagger)$$

✍ *Cette fois, le terme en $i = n$ est nul, mais on le conserve quand même.*

[**6. c.**] Par linéarité de l'espérance, $\mathbf{E}(Z_{k+1}) = \mathbf{E}(Z_k) + \mathbf{E}(\Delta_k)$ et, comme le support de Δ_k est égal à $\{-1; 0; 1\}$,

$$\mathbf{E}(\Delta_k) = \mathbf{P}(\Delta_k = 1) - \mathbf{P}(\Delta_k = -1) = 1 - \frac{2}{n}\,\mathbf{E}(Z_k)$$

d'après (†) et (‡).

On en déduit que la suite de terme général $\mathbf{E}(Z_k)$ est arithmético-géométrique :

$$\forall\, k \in \mathbb{N}, \qquad \mathbf{E}(Z_{k+1}) = 1 + \left(1 - \frac{2}{n}\right)\mathbf{E}(Z_k).$$

[**6. d.**] Sachant que $\mathbf{E}(Z_0) = n$, on en déduit que

$$\forall\, k \in \mathbb{N}, \qquad \mathbf{E}(Z_k) = \frac{n}{2}\left[1 + \left(1 - \frac{2}{n}\right)^k\right]$$

et en particulier que $\mathbf{E}(Z_k)$ tend vers $n/2$ lorsque k tend vers $+\infty$, conformément à ce qu'on a observé en **[5.]**.

[**7.**] [Aucune idée qui marche !]

136-py-71

Soit $(\varepsilon_k)_{k\geqslant 1}$, une suite de variables aléatoires indépendantes qui suivent toutes la loi de Rademacher :

$$\forall\, k \geqslant 1, \quad \mathbf{P}(\varepsilon_k = 1) = \mathbf{P}(\varepsilon_k = -1) = \frac{1}{2}.$$

Pour tout $n \in \mathbb{N}^*$, on définit les variables aléatoires

$$X_n = \sum_{k=1}^{n} \frac{\ln k}{k}\, \varepsilon_k$$

et

$$M_n = \max_{1\leqslant k\leqslant n} X_k.$$

[1.] *Démontrer que X_n admet une espérance et une variance (qu'on calculera).*
[2.] *Pour $n \geqslant 1$, on pose*

$$S_n = \sum_{k=1}^{n} \frac{\ln^2 k}{k^2}.$$

Démontrer que la suite $(S_n)_{n\geqslant 1}$ converge. On notera S, sa limite.
[3.] *Démontrer que*

$$\forall\, a > 0, \qquad \mathbf{P}(X_n \geqslant a) \leqslant \frac{S}{a^2}.$$

[4.] *Écrire une fonction* `sim_X()` *qui permette de simuler X_n, puis une fonction qui estime $\mathbf{P}(M_n \geqslant a)$ en effectuant 10^4 simulations.*
[5.] *Conjecturer le signe de*

$$\mathbf{P}(M_n \geqslant a) - \frac{S_n^2}{a}$$

pour $n \in \{20;\ 30;\ 50\}$ et $a \in [\![2, 4]\!]$.
[6.] *On admet l'**inégalité maximale de Doob** :*

$$\forall\, n \in \mathbb{N}^*, \qquad \mathbf{E}(M_n^2) \leqslant 4\,\mathbf{E}(X_n^2).$$

Démontrer que, pour tout $a > 0$,

$$\lim_{n\to+\infty} \mathbf{P}\Big(\frac{M_n}{\sqrt{\ln n}} \geqslant a\Big) = 0.$$

[7.] *Soit $a > 0$.*
[7.a.] *On considère les évènements $A_1 = [X_1 \geqslant a]$ et*

$$\forall\, k \geqslant 2, \qquad A_k = [X_k \geqslant a] \cap \bigcap_{\ell<k} [X_\ell < a].$$

Exprimer l'évènement $[M_n \geqslant a]$ à l'aide des A_k.
[7.b.] *Démontrer que*

$$\forall\, k \geqslant 1, \qquad \mathbf{E}(X_k^2 \mathbb{1}_{A_k}) \geqslant a^2\, \mathbf{P}(A_k).$$

[7.c.] *Démontrer la conjecture faite en* **[5.]**.

[1.] Chaque variable aléatoire ε_k n'a que deux valeurs possibles : ± 1, donc X_n a au plus 2^n valeurs possibles. En tant que variable aléatoire bornée, la variable X_n admet des moments de tout ordre. En particulier, elle est d'espérance finie et de variance finie.
[2.] Comme

$$u_k = \frac{\ln^2 k}{k^2} = \frac{\ln^2 k}{\sqrt{k}} \cdot \frac{1}{k^{3/2}} \underset{k\to+\infty}{=} o\Big(\frac{1}{k^{3/2}}\Big),$$

la série $\sum u_k$ est absolument convergente et la suite $(S_n)_{n\geqslant 1}$ de ses sommes partielles est convergente.

✍ *Puisqu'il s'agit d'une série dont le terme général est strictement positif, la suite de ses sommes partielles est strictement croissante. En particulier, chaque somme partielle S_n est strictement inférieure à la limite S.*

[3.] Comme $a > 0$, si $X_n(\omega) \geqslant a$, alors $X_n(\omega)^2 \geqslant a^2$. Autrement dit,

$$[X_n \geqslant a] \subset [X_n^2 \geqslant a^2]$$

Comme X_n admet un moment d'ordre 2, on déduit de l'inégalité de Markov que

$$\mathbf{P}(X_n \geqslant a) \leqslant \mathbf{P}(X_n^2 \geqslant a^2) \leqslant \frac{\mathbf{E}(X_n^2)}{a^2}.$$

Chaque variable aléatoire ε_k est centrée, donc X_n est centrée (comme combinaison linéaire de variables aléatoires centrées) et $\mathbf{V}(\varepsilon_k) = \mathbf{E}(\varepsilon_k^2) = 1$. Comme les ε_k sont supposées indépendantes,

$$\mathbf{V}(X_n) = \sum_{k=1}^{n} \frac{\ell n^2\, k}{k^2} \mathbf{V}(\varepsilon_k) = S_n < S$$

puisque la suite $(S_n)_{n\geqslant 1}$ est strictement croissante.

Ainsi,

$$\forall\, a > 0, \qquad \mathbf{P}(X_n \geqslant a) \leqslant \frac{S}{a^2}.$$

[4.] Le polynôme $2X-1$ est le polynôme interpolateur (au sens de Lagrange) qui envoie 0 sur -1 et 1 sur 1. Par conséquent, si la variable aléatoire U suit la loi de Bernoulli $\mathscr{B}(1/2)$, alors la variable aléatoire $2U-1$ suit la loi de Rademacher.

✍ *On rappelle que, pour Python, la loi de Bernoulli $\mathscr{B}(1/2)$ est en fait la loi binomiale $\mathscr{B}(1, 1/2)$.*

```
def sim_X(n):
    B = 2*rd.binomial(1, 0.5, n) - 1
    S = 0
    for i in range(1, n+1):
        S += np.log(i)/i*B[i-1]
    return S
```

❧ Pour estimer $\mathbf{P}(M_n \geqslant a)$, on se fonde sur la Loi des grands nombres (= méthode de Monte-Carlo) : on simule $N = 10^4$ réalisations du vecteur aléatoire

$$(X_1, X_2, \dots, X_n),$$

on en déduit N réalisations de la variable aléatoire M_n et on conclut en rappelant que

$$\mathbf{P}(M_n \geqslant a) \approx \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{[M_n(i)\geqslant a]}.$$

```
def sim_M(n):
    echantillon = [sim_X(_) for _ in range(2, n+1)]
    return max(echantillon)

def estimation_proba(a, n):
    cptr = 0
    N = 1000
    for _ in range(N):
        cptr += sim_M(n)>=a
    return cptr/N
```

✍ *Quelques précisions sur la Loi des grands nombres...*

❧ *Si Y est une variable aléatoire admettant un moment d'ordre deux et si $(Y_k)_{k\in\mathbb{N}}$ est une suite de variables aléatoires indépendantes qui suivent toutes la loi de Y, alors*

$$\forall\, \varepsilon > 0, \quad \lim_{N\to+\infty} \mathbf{P}\left(\left|\mathbf{E}(Y) - \frac{1}{N}\sum_{k=1}^{N} Y_k\right| \geqslant \varepsilon\right) = 0.$$

Autrement dit, pour tout entier N "assez grand",

$$\frac{1}{N}\sum_{k=1}^{N} Y_k \approx \mathbf{E}(Y).$$

❧ *L'inégalité de Bienaymé-Tchebychev donne une estimation quantitative imprécise mais très simple de ce qu'est un entier N "assez grand". C'est même la raison d'être de ce théorème.*

❧ *Ici, il s'agit d'estimer la probabilité d'un évènement $A \in \mathcal{A}$. Pour cela, on considère une variable aléatoire de Bernoulli $Y = \mathbb{1}_A$, pour laquelle $\mathbf{E}(Y) = \mathbf{P}(A)$.*

❧ *Les appels successifs à la fonction `random` (ou aux fonctions qui en dérivent) sont censés reproduire un échantillon de N variables aléatoires indépendantes et de même loi. Il faut donc être sûr de disposer d'un générateur pseudo-aléatoire de bonne qualité pour utiliser l'algorithme de Monte-Carlo.*

[**5.**] On commence par calculer les valeurs de S_n.

```
def S(n):
    somme = 0
    for i in range(2, n+1):
        somme += np.log(i)**2/i
    return somme
```

On applique ce qui précède dans une double boucle `for`. L'exécution du code est de plus en plus lente, car le nombre de calculs augmente avec n (il ne dépend pas de la valeur de a).

```
for n in [20, 30, 50]:
    for a in [2, 3, 4]:
        flag = estimation_proba(a, n)<=S(n)/a**2
        print(flag)
```

On constate que la probabilité $\mathbf{P}(M_n \geqslant a)$ semble inférieure à S_n/a^2.

[**6.**] On procède comme plus et on déduit de l'inégalité de Markov que

$$\mathbf{P}\Big(\frac{M_n}{\sqrt{\ell n\, n}} \geqslant a\Big) \leqslant \frac{\mathbf{E}(M_n^2)}{a^2\,\ell n\, n}.$$

On enchaîne avec l'inégalité maximale de Doob : comme $\mathbf{E}(X_n^2) = S_n \leqslant S$,

$$\forall\, n \geqslant 1, \qquad \mathbf{P}\Big(\frac{M_n}{\sqrt{\ell n\, n}} \geqslant a\Big) \leqslant \frac{4S}{a^2\,\ell n\, n},$$

ce qui donne le résultat attendu.

[**7. a.**] L'évènement $[M_n \geqslant a]$ signifie que, parmi les variables aléatoires $X_1, \ldots, X_n$, il y en a au moins une qui dépasse le seuil a.

L'évènement A_k est "réalisé" quand la variable aléatoire X_k est la première à dépasser le seuil a. En conséquence, les évènements $A_1, \ldots, A_n$ sont deux à deux disjoints et, si $M_n(\omega) \geqslant a$, alors ω appartient à l'un des A_k.

Autrement dit :

$$[M_n \geqslant a] = \bigsqcup_{k=1}^{n} A_k.$$

[**7. b.**]

[**7. c.**] [Aucune idée !]

On définit les matrices

$$I = I_2, \quad A = \begin{pmatrix} i & 0 \\ 0 & -i \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \quad C = \begin{pmatrix} 0 & -i \\ -i & 0 \end{pmatrix}.$$

On considère l'espace vectoriel réel

$$E = \mathrm{Vect}_{\mathbb{R}}(I, A, B, C)$$

ainsi que le sous-espace $F = \mathrm{Vect}_{\mathbb{R}}(A, B, C)$. *On suppose que* E *est muni d'un produit scalaire* $\langle \cdot | \cdot \rangle$ *pour lequelle* (I, A, B, C) *est une base orthonormée de* E.
On note alors $S = \{X \in E \ : \ \|X\| = 1\}$ *et, pour toute matrice* $X \in S$, *on définit l'application*

$$\varphi_X = \left[Y \mapsto XYX^{-1}\right] \ : \ E \to E.$$

On pose enfin

$$\varphi = [X \mapsto \varphi_X] \ : \ S \to \mathrm{GL}(E).$$

[**1.**] *Démontrer que le produit scalaire des deux matrices*

$$\begin{pmatrix} a & b \\ -\overline{b} & \overline{a} \end{pmatrix} \qquad \textit{et} \qquad \begin{pmatrix} c & d \\ -\overline{d} & \overline{c} \end{pmatrix}$$

est égal à $\mathfrak{Re}(a\overline{c} + b\overline{d})$. *En déduire un lien entre la norme associée à ce produit scalaire et le déterminant.*
[**2.**] *Écrire une fonction python qui, pour* $X \in S$ *renvoie la matrice de* φ_X *relative à la base* (I, A, B, C). *Calculer* φ_I, φ_A, φ_B *et* φ_C.
[**3.**] *Démontrer que* S *est un groupe pour le produit matriciel.*
[**4.**] *Démontrer que* φ_X *est une isométrie de* E *pour tout* $X \in S$.
[**5.**] *Démontrer que le sous-espace* F *est stable par tout élément de* $\varphi_*(S)$.

✍ *Cinq autres questions non traitées.*

[**1.**] En posant $a = x + iy$ et $b = z + it$ (où x, y, z et t sont réels),

$$\begin{pmatrix} a & b \\ -\overline{b} & \overline{a} \end{pmatrix} = \begin{pmatrix} x + iy & z + it \\ -z + it & x - iy \end{pmatrix} = xI + yA + zB - tC.$$

✍ *On travaille ici avec des matrices à coefficients complexes mais avec des scalaires* réels. *Par conséquent, la dimension de* $\mathfrak{M}_2(\mathbb{C})$, *en tant qu'espace vectoriel sur* $\mathbb{K} = \mathbb{R}$ *est égale à* 8 *tandis que* E *est un sous-espace de dimension* 4.

En posant $c = x' + iy'$ et $d = z' + it'$ où x', y', z' et t' sont réels), du fait que la famille (I, A, B, C) soit orthonormée, le produit scalaire des deux matrices est égal à

$$xx' + yy' + zz' + tt'.$$

Par ailleurs,

$$a\overline{c} + b\overline{d} = (x + iy)(x' - iy') + (z + it)(z' - it') = (xx' + yy' + zz' + tt') + i(\dots).$$

Donc le produit scalaire est bien égal à $\mathfrak{Re}(a\overline{c} + b\overline{d})$.

❧ En particulier, le carré de la norme de la première matrice est égal à

$$\mathfrak{Re}(a\overline{a} + b\overline{b}) = |a|^2 + |b|^2,$$

c'est-à-dire égal au déterminant de la matrice.
[**2.**] Il faut calculer $\varphi_X(Y)$ lorsque la matrice Y parcourt la base (I, A, B, C) pour obtenir les colonnes de la matrice de φ_X.

Comme la base (I, A, B, C) est une base orthonormée de E, il est facile de décomposer un vecteur de E dans cette base :

$$\forall\, M \in E, \quad M = \langle I | M \rangle \cdot I + \langle A | M \rangle \cdot A + \langle B | M \rangle \cdot B + \langle C | M \rangle \cdot C.$$

❧ On commence donc par coder la "base canonique" de E :

```
I = np.eye(2)
A = np.diag([1j,-1j])
B = np.array([[0,1],[-1,0]])
C = np.array([[0,-1j],[-1j,0]])
```

puis le produit scalaire.

```
def ps(M, N):
    a, b = M[0,0], M[0,1]
    cc, dd = N[1,1], N[1,0]  # conjugués
    return (a*cc-b*dd).real
```

Le calcul de la matrice se fait alors de manière directe.

```
def phi(X, Y):
    return X.dot(Y).dot(alg.inv(X))

def mat_phi(X):
    M = np.zeros((4,4))
    for j, Y in enumerate([I, A, B, C]):
        phiY = phi(X, Y)     # image d'un vecteur de base
        M[:,j] = [ps(phiY, U) for U in [I,A,B,C]] # décomposition dans la BON
    return M
```

On trouve alors

$$\mathfrak{Mat}_{\mathscr{B}_{\text{can}}}(\varphi_I) = I_4, \qquad \mathfrak{Mat}_{\mathscr{B}_{\text{can}}}(\varphi_A) = \text{Diag}(1,1,-1,-1),$$
$$\mathfrak{Mat}_{\mathscr{B}_{\text{can}}}(\varphi_B) = \text{Diag}(1,-1,1,-1), \qquad \mathfrak{Mat}_{\mathscr{B}_{\text{can}}}(\varphi_C) = \text{Diag}(1,-1,-1,1).$$

[**3.**] D'après la première question, la sphère unité coïncide avec le noyau du morphisme det. Donc c'est un sous-groupe pour la multiplication matricielle.

[**4.**] Soit $X \in S$. Toujours d'après la première question,

$$\forall\, Y \in E, \quad \left\|\varphi_X(Y)\right\|^2 = \det \varphi_X(Y) = \det Y = \|Y\|^2.$$

Par conséquent, φ_X est bien une isométrie de E.

✍ *Le résultat est vrai pour tout $x \in E$, pas seulement pour $x \in S$.*

[**5.**] Comme (I, A, B, C) est une base orthonormée de E, le sous-espace $F = \text{Vect}_{\mathbb{R}}(A, B, C)$ est l'orthogonal (relatif à E) de la droite vectorielle $\mathbb{R} \cdot I$.

Pour tout $X \in S$, le vecteur I est évidemment un vecteur propre de l'isométrie $\varphi_X = \varphi(X)$. Par conséquent, son orthogonal $(\mathbb{R} \cdot I)^\perp = F$ est aussi stable par φ_X.

✍ *Pour toute isométrie u d'un espace euclidien V, si un sous-espace F est stable par u, alors son orthogonal $F^\perp$ est aussi stable par u.*