JEUDI 4 JUIN

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.

Thèmes abordés	
136-py-07	Polynômes de Tchebychev
136-py-09	Nombres algébriques
136-py-14	Matrices
136-py-22	Calcul différentiel
136-py-39	Suite de fonctions
Centrale25-I06	Entropie d'une loi de probabilité discrète

On définit une suite de polynômes $(T_n)_{n\in\mathbb{N}}$ *en posant* $T_0 = 1$, $T_1 = X$ *et*

$$\forall\, n \in \mathbb{N}, \quad T_{n+2} = 2XT_{n+1} - T_n.$$

On admet *que* $\deg T_n = n$ *pour tout* $n \in \mathbb{N}$.
[**1.a.**] *Écrire une fonction* `poly(n)` *qui renvoie le polynôme* T_n.
[**1.b.**] *Afficher les graphes de fonctions* T_k *pour* $1 \leqslant k \leqslant 5$ *sur l'intervalle* $[-1, 1]$. *Compter leurs zéros.*
[**1.c.**] *Vérifier sur des exemples que les polynômes* T_n *commutent deux à deux pour la composition des polynômes. En va-t-il de même pour tous les polynômes ?*
[**2.**] *Démontrer que*

$$\forall\, n \in \mathbb{N},\ \forall\, \theta \in \mathbb{R}, \qquad T_n(\cos\theta) = \cos n\theta.$$

[**3.**] *Soient* $m \leqslant n$, *deux entiers naturels. Démontrer que*

$$2T_nT_m = T_{n+m} + T_{n-m} \qquad \textit{et que} \qquad T_n \circ T_m = T_{nm}.$$

[**4.**] *Vérifier que* T_n *vérifie l'équation différentielle*

$$(1-x^2)y''(x) - xy'(x) + n^2y(x) = 0.$$

[**1.a.**] On donne un identifiant simple et explicite pour les deux monômes 1 et X.

```
from numpy.polynomial import Polynomial

U = Polynomial([1])
X = Polynomial([0,1])
```

On peut alors facilement calculer T_n par récurrence.

```
def poly(n):
    T = [U, X]
    for k in range(n-1):
        T.append(2*X*T[k+1]-T[k])
    return T[n]
```

✍ *Dans ce qui suit, on aura souvent besoin d'une famille* $(T_k)_{0\leqslant k\leqslant n}$ *plutôt que d'un seul polynôme* T_n. *Une modification marginale du code précédent évitera de réaliser plusieurs fois le même calcul.*

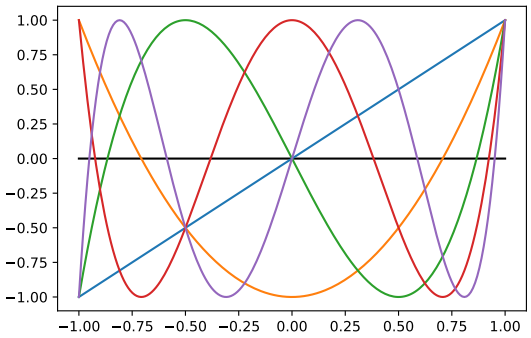
```
def Tchebychev(n):
    T = [U, X]
    for k in range(n-1):
        T.append(2*X*T[k+1]-T[k])
    return T
```

[**1.b.**]

```
t = np.linspace(-1, 1, 200)
plt.plot(t, np.zeros(len(t)), 'k')
for n in range(5):
    plt.plot(t, poly(n+1)(t))
```

✍ *On a ajouté le paramètre 200 dans* `linspace` *pour que les courbes paraissent bien lisses. Avec la valeur par défaut (c'est-à-dire 50), on voit clairement que ces courbes sont affines par morceaux et non pas polynomiales.*

C'est assez embrouillé, on n'a pas forcément en tête les instructions qui permettraient d'y voir plus clair (en définissant une légende)...



On se débrouille et on constate que P_k semble bien avoir k racines dans l'intervalle $[-1,1]$.
[1.c.] La liste `t` contient 200 réels régulièrement espacés entre -1 et $+1$.

```
def commutation(m, n):
    A, B = poly(m), poly(n)
    return A(B)(t)-B(A)(t)
```

Lorsque m et n varient en restant inférieurs à 10, on constate que la valeur renvoyée est la liste nulle, ce qui suggère bien que $T_m \circ T_n = T_n \circ T_m$.

❧ En revanche, si n est supérieur à 10, les erreurs d'arrondi deviennent prépondérantes et on constate que seule la partie centrale de la liste renvoyée est nulle (partie qui correspond aux abscisses les plus proches de 0).

✍ *Bien entendu, on a la possibilité de mener les calculs littéralement, mais ce n'est pas un attendu de l'épreuve.*

```
from sympy import poly, compose
from sympy.abc import x

def Tchebychev(n):
    T = [poly(1, x), poly(x, x)]
    for k in range(n-1):
        T.append(2*x*T[k+1]-T[k])
    return T

def commutation(n):
    T = Tchebychev(n)
    for i in range(n+1):
        P = T[i]
        for j in range(i):
            Q = T[j]
            if compose(P, Q)!=compose(Q, P):
                print(i, j)
```

❧ Avec $P = X^2$ et $Q = X^2 + 1$, on a $P \circ Q = X^4 + 2X^2 + 1$ et $Q \circ P = X^4 + 1$, donc la composition des polynômes n'est pas commutative.
[2.] La propriété est évidente pour $n = 0$ et pour $n = 1$. Si elle est vraie pour n et pour $n+1$, alors elle est aussi vraie pour $n+2$ puisque

$$\begin{aligned}\forall\,\theta \in \mathbb{R}, \qquad \cos(n+2)\theta + T_n(\cos\theta) &= \cos(n+2)\theta + \cos n\theta \\ &= \cos[(n+1)+1]\theta + \cos[(n+1)-1]\theta \\ &= 2\cos\theta\cos(n+1)\theta = 2\cos\theta \cdot T_{n+1}(\cos\theta).\end{aligned}$$

[3.] Soit $x \in [-1,1]$. Il existe donc un (unique) réel $\theta \in [0,\pi]$ tel que $x = \cos\theta$ et

$$\begin{aligned}2T_n(x)T_m(x) &= 2T_n(\cos\theta)T_m(\cos\theta) = 2\cos n\theta\cos m\theta \\ &= \cos(m+n)\theta + \cos(m-n)\theta = T_{m+n}(\cos\theta) + T_{m-n}(\cos\theta) \\ &= T_{m+n}(x) + T_{m-n}(x).\end{aligned}$$

Les polynômes $2T_nT_m$ et $T_{m+n}+T_{m-n}$ sont égaux sur une partie infinie de $\mathbb{R}$, donc ils sont en fait égaux.

❧ De même, pour tout $x \in [-1,1]$,

$$\begin{aligned}(T_n \circ T_m)(x) = T_n\big(T_m(\cos\theta)\big) = T_n(\cos m\theta) = \cos n(m\theta)\\ = T_{nm}(\cos\theta) = T_{nm}(x)\end{aligned}$$

donc $T_n \circ T_m = T_{nm}$.

[4.] On dérive une première fois l'identité $T_n(\cos\theta) = \cos n\theta$ pour obtenir :

$$\forall\, \theta \in [0,\pi], \qquad -T_n'(\cos\theta)\sin\theta = -n\sin n\theta.$$

En dérivant une seconde fois, on obtient

$$\forall\, \theta \in [0,\pi], \qquad T_n''(\cos\theta)\sin^2\theta - T_n'(\cos\theta)\cos\theta = -n^2\cos n\theta = -n^2T_n(\cos\theta).$$

En posant $x = \cos\theta$, on en déduit que

$$\forall\, x \in [-1,1], \quad (1-x^2)T_n''(x) - xT_n'(x) + n^2T_n(x) = 0.$$

Comme les deux membres de l'égalité sont polynomiaux, cette égalité est en fait vérifiée sur $\mathbb{R}$.

136-py-09

Soit $\mu \in \mathbb{R}$. Un réel $x \in \mathbb{R}_+$ est dit **μ-approchable** *s'il existe un constante $c > 0$ et une infinité de rationnels $p/q \in \mathbb{Q}_+$ tels que*

$$\left|x - \frac{p}{q}\right| \leqslant \frac{c}{q^\mu}.$$

On note $\mu(x) \in \mathbb{R}_+ \cup \{+\infty\}$, la borne supérieure de l'ensemble des réels μ tels que x soit μ-approchable. Un réel positif x est dit ***algébrique d'ordre*** *n lorsqu'il existe un polynôme $P \in \mathbb{Z}[X]$ de degré n tel que $P(x) = 0$ et que $Q(x) \neq 0$ pour tout polynôme non nul $Q \in \mathbb{Z}[X]$ de degré strictement inférieur à n.*

[**1.**] *Le* ***nombre d'or*** *est défini par*

$$\varphi = \frac{1+\sqrt{5}}{2}.$$

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

$$\forall\, n \in \mathbb{N}, \qquad F_{n+2} = F_n + F_{n+1}.$$

[**1.a.**] *Écrire une fonction* `fibonacci(n)` *qui à l'entier $n \in \mathbb{N}$ associe l'entier F_n. Que vaut F_{200} ?*

[**1.b.**] *Écrire une fonction* `test_approximation(x, p, q, c, mu)` *qui renvoie* `True` *si, et seulement si,*

$$\left|x - \frac{p}{q}\right| \leqslant \frac{c}{q^\mu}.$$

[**1.c.**] *Vérifier que*

$$\forall\, 1 \leqslant n \leqslant 12, \qquad \left|\varphi - \frac{F_{2n}}{F_{2n-1}}\right| \leqslant \frac{1}{\sqrt{5}F_{2n-1}^2}.$$

[**1.d.**] *Écrire une fonction* `meilleure_approximation(x, n)` *d'arguments $x \in \mathbb{R}_+$ et $n \in \mathbb{N}^*$ qui renvoie un couple $(p, q) \in \mathbb{N}^2$ tel que l'écriture décimale de q compte exactement n chiffres et que la distance $|x - p/q|$ soit minimale.*

[**2.**] *Soit $x \in \mathbb{R}$. Démontrer que, pour tout entier q, il existe $p \in \mathbb{N}$ tel que*

$$|qx - p| \leqslant 1.$$

En déduire que $\mu(x) \geqslant 1$.

[**3.**] *Soient a/b et p/q, deux rationnels distincts. Démontrer que*

$$\left|\frac{a}{b} - \frac{p}{q}\right| \geqslant \frac{1}{bq}.$$

En déduire que $\mu(x) = 1$ pour tout rationnel x.

[**4.**] *Démontrer que l'ensemble des nombres algébriques est dénombrable. En déduire qu'il existe au moins un nombre* ***transcendant*** *(c'est-à-dire qui n'est pas algébrique).*

[**5.**] *Soient x, un nombre algébrique d'ordre n ;*

$$P = a_0 + a_1 X + \cdots + a_n X^n \in \mathbb{Z}[X],$$

un polynôme non nul de degré n tel que $P(x) = 0$ et

$$M = \max_{0\leqslant k\leqslant n} \left|\frac{a_k}{a_n}\right|.$$

[**5.a.**] *Démontrer que P n'a pas de racine dont le module soit supérieur à $M + 1$.*

[**5.b.**] *En déduire que : si $x \in \mathbb{R}_+$ est un nombre algébrique d'ordre n, alors $\mu(x) \leqslant n$.*

[**6.**] *Démontrer que le nombre*

$$x = \sum_{n=0}^{+\infty} 10^{-n!}$$

est transcendant.

[1.a.] Le classique des classiques !

```
def fibonacci(n):
    F = [1, 1]
    for i in range(1, n):
        F.append(F[-1]+F[-2])
    return F[n]
```

On trouve $F_{200} = 453\,973\,694\,165\,307\,953\,197\,296\,969\,697\,410\,619\,233\,826$.

✍ ***Rappel hors sujet*** *: on sait que $F_n < F_{n+1}$ pour tout entier $n \geqslant 1$. Par conséquent, il est possible de lire la relation de récurrence*

$$F_{n+2} = F_{n+1} + F_n \qquad \textit{sous la forme} \qquad F_{n+1} = 1 \times F_{n+1} + F_n \quad \textit{avec} \quad 0 \leqslant F_n < F_{n+1},$$

c'est-à-dire comme une division euclidienne. Comme F_n est le reste de la division euclidienne de F_{n+2} par F_{n+1}, l'algorithme d'Euclide nous assure que le pgcd du couple (F_{n+2}, F_{n+1}) est aussi le pgcd du couple (F_{n+1}, F_n) et donc celui du couple $(F_2, F_1) = (2, 1)$. Les entiers F_n et F_{n+1} sont donc premiers entre eux, quel que soit $n \in \mathbb{N}$.

[1.b.] Difficile de faire plus simple !

```
def test_approximation(x, p, q, c, mu):
    return np.abs(x-p/q)<=c/q**mu
```

[1.c.] Heureusement que la fonction `fibonacci` n'est pas trop gourmande en calculs, on effectue ici plusieurs fois les mêmes calculs.

```
phi = (1+np.sqrt(5))/2
c = 1/np.sqrt(5)
for n in range(12):
    Fi, Fp = fibonacci(2*n+1), fibonacci(2*n+2)
    print(test_approximation(phi, Fp, Fi, c, 2))
```

Surprise ! Les onze premiers tests réussissent, le douzième échoue. (*Ultima necat ?*)

✍ *L'explication est facile à trouver, c'est évidemment une question d'arrondi. Avec la précision usuelle de python pour les flottants :*

$$\begin{aligned} \varphi &= 1.618033988749895 \\ F_{24}/F_{23} &= 1.618033988957902 \\ F_{24}/F_{23} - \varphi &= 0.000000000208007\,16704627575 \\ c/F_{23}^2 &= 0.000000000208007\,15319502499 \end{aligned}$$

Le calcul du quotient c/F_{23}^2 *est précis (une quinzaine de chiffres significatifs), alors que le calcul de la* différence $F_{24}/F_{23} - \varphi$ *n'a que six chiffres significatifs et cette imprécision fausse la comparaison.*

Complément.

La question n'est pas posée, mais on peut chercher à justifier l'encadrement proposé par un calcul littéral.

La suite de Fibonacci est une suite récurrente linéaire d'ordre deux. Les racines de l'équation caractéristique associée à cette suite :

$$X^2 - X - 1 = 0$$

sont le nombre d'or φ et son inverse φ^{-1}. La relation $\varphi^2 - \varphi = 1$ nous indique que $\varphi^{-1} = \varphi - 1$.

On vérifie sans peine que

$$\forall\, k \in \mathbb{N}, \qquad F_k = (2 - \varphi)\varphi^k + (\varphi - 1)\varphi^{-k}$$

et on en déduit (par opérations de pivot avec $\varphi^2 = \varphi + 1$, mais on pourrait vérifier ces relations par récurrence) que

$$\forall\, k \in \mathbb{N}^*, \qquad \varphi^k = F_k\varphi + F_{k-1} \qquad \text{et} \qquad \varphi^{-k} = (-1)^k(F_k - F_{k-1}\varphi).$$

On peut en déduire que

$$\forall\, n \geqslant 1, \qquad F_{2n}F_{2n-1} - \varphi F_{2n-1}^2 = (\varphi-1)(2-\varphi) + \frac{(1-\varphi)^3}{\varphi^{4n-1}} = (1-\varphi)\left[(2-\varphi) + \frac{1}{\varphi^{4n+1}}\right]$$

toujours avec la relation $\varphi^2 = \varphi + 1$.

✍ *Pour que les calculs se déroulent au mieux, il faut éviter d'utiliser l'expression "explicite" de φ et se contenter d'exploiter à fond l'équation $\varphi^2 = \varphi + 1$.*

Par conséquent,

$$\forall\, n \geqslant 1, \qquad \left|F_{2n}F_{2n-1} - \varphi F_{2n-1}^2\right| \leqslant (\varphi-1)\left(2 - \varphi + \frac{1}{\varphi^5}\right) = 2(5-3\varphi)$$

puisque $\varphi^{-5} = 5\varphi - 8$. On conclut en vérifiant facilement que

$$2(5-3\varphi) = 7 - 3\sqrt{5} \leqslant \frac{1}{\sqrt{5}}.$$

[**1.d.**] Par définition, le dénominateur q varie de 10^{n-1} à $10^n - 1$.

Si le dénominateur q est connu, alors le numérateur p est un entier tel que $|qx - p|$ soit minimale. Cela nous laisse seulement deux choix : $p = \lfloor qx \rfloor$ et $p = \lfloor qx \rfloor + 1$.

Pour déterminer le meilleur couple (p, q), on va donc tester environ 2.10^n couples : c'est beaucoup mais pas excessif.

L'algorithme dérive de l'algorithme classique de recherche du maximum d'une liste.

```
def meilleure_approximation(x, n):
    erreur = 1
    for q in range(10**(n-1), 10**n):
        p0 = int(np.floor(q*x))
        for p in [p0, p0+1]:
            if abs(x-p/q)<erreur:
                p_opt, q_opt, erreur = p, q, abs(x-p/q)
    return p_opt, q_opt, erreur
```

✍ *Je me suis écarté des spécifications de l'énoncé en retournant l'erreur d'approximation en plus du couple (p, q). À ne pas faire !*

136-py-14 [TL]

[1.] *Une matrice* $A \in \mathfrak{M}_n(\mathbb{C})$ *est dite* ***de type*** * *lorsqu'il existe un entier* $i \in [\![2, n]\!]$ *tel que* $A^i = A$.
[1.a.] *Écrire une fonction* `genere_matrice(n, a)` *qui renvoie une matrice de* $\mathfrak{M}_n(\mathbb{R})$ *dont les coefficients sont pris aléatoirement dans* $[\![-a, a]\!]$.
[1.b.] *Écrire une fonction* `type_etoile(A)` *qui renvoie* `True` *si la matrice* $A \in \mathfrak{M}_n(\mathbb{R})$ *est de type * et renvoie* `False` *sinon.*
[1.c.] *Écrire une fonction* `genere_type_etoile(n, a)` *qui renvoie une matrice* $A \in \mathrm{GL}_n(\mathbb{R})$*, de type * et dont les coefficients sont pris au hasard dans* $[\![-a, a]\!]$.
[1.d.] *Écrire une fonction* `proportion(n, a)` *qui renvoie les proportions de matrices de type * parmi les matrices de* $\mathfrak{M}_i(\mathbb{R})$ *dont les coefficients appartiennent à l'intervalle* $[\![-a, a]\!]$*, l'entier* i *variant de* 2 *à* n.
[1.e.] *Représenter sur un même graphique ces proportions pour* $2 \leqslant a \leqslant 6$ *et* $n = 5$.
[2.] *Soient* E*, un espace vectoriel complexe ;* u*, un endomorphisme de* E *et* $\ell \geqslant 2$*, un entier.*
[2.a.] *Démontrer que : si* u *est diagonalisable, alors* u^ℓ *est diagonalisable.*
[2.b.] *Donner un exemple d'endomorphisme* $u \in \mathrm{L}(E)$*, non diagonalisable, tel que* u^ℓ *soit diagonalisable pour un entier* $\ell \geqslant 2$ *convenable.*
[3.] *Soient* E*, un espace vectoriel complexe et* u*, un endomorphisme de* E*. On suppose qu'il existe un entier* $2 \leqslant \ell \leqslant n$ *tel que* u^ℓ *soit diagonalisable.*
[3.a.] *Démontrer qu'il existe des nombres complexes* $\alpha_1, \ldots, \alpha_p$ *deux à deux distincts tels que*

$$(u^\ell - \alpha_1 \mathrm{I}_E) \circ \cdots \circ (u^\ell - \alpha_p \mathrm{I}_E) = \omega_E. \qquad (\star)$$

[3.b.] *On suppose que les* α_k *sont tous différents de* 0*. Démontrer que* u *est diagonalisable.*
[3.c.] *On suppose que* $\alpha_1 = 0$.
— *Démontrer que* $\operatorname{Ker} u^\ell$ *est stable par* u.
— *Démontrer que : si* u *est diagonalisable, alors* $\operatorname{Ker} u^\ell = \operatorname{Ker} u$.
— *Démontrer que : si* $\operatorname{Ker} u^\ell = \operatorname{Ker} u$*, alors* u *est diagonalisable.*
[3.d.] *Conclure : donner une condition nécessaire et suffisante pour que* u *soit diagonalisable.*
[4.] *Le résultat est-il encore vrai dans* $\mathfrak{M}_n(\mathbb{R})$ *?*

✍ *Suit une dernière question "obscure".*

[1.a.]

```python
def genere_matrice(n, a):
    return rd.randint(-a, a+1, (n, n))
```

[1.b.] On calcule successivement $A^2, \ldots, A^n$ et on compare à la matrice A : si une comparaison réussit, la matrice A est de type *; si au contraire toutes les comparaisons échouent, la matrice A n'est pas de type *.

```python
def type_etoile(A):
    n = A.shape[0]
    B = A
    for k in range(1, n):
        B = B.dot(A) # B = A^(k+1)
        if np.array_equal(A, B):
            return True
    return False
```

[1.c.]

```python
def genere_type_etoile(n, a):
    echec = True
    while echec:
        A = genere_matrice(n, a)
        echec = (alg.det(A)==0) or not(type_etoile(A))
    return A
```

Pour $n = 3$ et $a \geqslant 5$, il ne faut pas être pressé. Pour $n \geqslant 4$ et $a \geqslant 3$, je n'en parle même pas.

✍ *Étrange question ! On considère une partie "finie" de $GL_n(\mathbb{R})$, dont le cardinal est tout de même égal à $(2a+1)^{n^2}$, ce qui n'est pas peu, et cette partie contient au moins une matrice inversible de type * (la matrice I_n).*

*Mais on conçoit aisément que les matrices de type * sont tout de même assez rares ! Dans ces conditions, compter sur le hasard pour produire une matrice de type * est plutôt optimiste.*

[**1.d.**] Pour $a = 4$ et $n = 3$, on considère un ensemble de 9^9 matrices, soit environ un milliard : il est donc hors de question de passer en revue toutes les matrices pour compter le nombre exact de matrices de type *.

❧ Une première méthode probabiliste consiste à choisir au hasard N matrices dans l'ensemble considéré et à calculer la proportion de matrices de type * parmi celles-ci.

```
def type_etoile_inversible(A, I):
    n = A.shape[0]
    B = A
    for k in range(1, n):
        if np.array_equal(B, I): # B = A^k
            return True
        B = B.dot(A)
    return False

def proportion(n, a):
    prop, N = {}, 100000
    for i in range(2, n+1):
        prop[i] = 0
        I = np.eye(i)
        for k in range(N):
            A = genere_matrice(i, a)
            prop[i] += type_etoile_inversible(A, I)
        prop[i] /= N
    return prop
```

Avec $N = 10^5$, les proportions obtenues sont de l'ordre de 10^{-3} pour $n = 2$; de l'ordre de 10^{-4} pour $n = 3$ et nulles pour $n \geqslant 4$ (= *aucune* des N matrices prises au hasard n'est de type *).

❧ Une autre méthode probabiliste consiste à modifier la fonction `genere_type_etoile` pour qu'elle renvoie le nombre de tentatives infructueuses. L'exécution répétée de cette fonction nous donne alors une valeur approchée de l'inverse de la proportion cherchée, mais comme cet inverse est très élevé, les calculs durent longtemps.

✍ *L'espérance de la loi géométrique $\mathscr{G}(p)$ (= instant du premier succès dans un schéma de Bernoulli) est égale à $1/p$.*

[**1.e.**] Toutes les proportions sont nulles ou presque. Quel intérêt de les représenter graphiquement ???

[**2.b.**] La matrice

$$N = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

est nilpotente non nulle, donc n'est pas diagonalisable. En revanche, N^2 est diagonale.

✍ *Quand on cherche un exemple de matrice non diagonalisable, toujours chercher une matrice nilpotente !*

[**3.a.**] Comme l'endomorphisme u^ℓ est diagonalisable, il admet un polynôme annulateur scindé à racines simples.

[**3.b.**] Si les $\alpha_1, \ldots, \alpha_p$ sont deux à deux distincts et non nuls, le polynôme

$$(X^\ell - \alpha_1) \cdots (X^\ell - \alpha_p)$$

est scindé à racines simples et c'est un polynôme annulateur de u.

✍ *Pour tout $\alpha \in \mathbb{C}^*$, le polynôme $X^\ell - \alpha$ admet ℓ racines complexes distinctes (qui se déduisent des racines ℓ-ièmes de l'unité).*

En revanche, si $\alpha_1 = 0$, alors le polynôme

$$X^\ell(X^\ell - \alpha_2)\cdots(X^\ell - \alpha_p)$$

est un polynôme annulateur de u, il est scindé mais 0 n'est pas une racine simple : on ne peut donc pas conclure.

[**3. c.**] Comme u^ℓ commute à u (c'est un polynôme en u), le noyau de u^ℓ est stable par u.

❧ Il est clair que $\operatorname{Ker} u^\ell = \operatorname{Ker} u$.

Réciproquement, soit $x \in \operatorname{Ker} u^\ell$. Comme u est diagonalisable, il existe une, et une seule, famille de vecteurs $(x_1, \dots, x_r)$ telle que

$$x = \sum_{k=1}^{r} x_k \qquad \text{et} \qquad \forall\, 1 \leqslant k \leqslant r, \quad u(x_k) = \lambda_k x_k.$$

✍ *On a noté, comme d'habitude, $\lambda_1, \dots, \lambda_r$, les valeurs propres (distinctes) de u et on suppose en outre que $\lambda_1 = 0$.*

Comme $x \in \operatorname{Ker} u^\ell$, on en déduit que

$$0 = u^\ell(x) = \sum_{k=1}^{r} u^\ell(x_k) = \sum_{k=2}^{r} \lambda_k^\ell x_k$$

puisque $\lambda_1 = 0$ et $\ell \geqslant 2$. Les sous-espaces propres de u sont en somme directe, donc

$$\forall\, 2 \leqslant k \leqslant r, \qquad \lambda_k^\ell x_k = 0.$$

✍ *Les sous-espaces propres d'un endomorphisme sont en somme directe, quel que soit l'endomorphisme considéré.*

Comme $\lambda_k \neq 0$ pour $k \geqslant 2$, on en déduit que $x_k = 0$ pour $k \geqslant 2$ et donc que $x = x_1 \in \operatorname{Ker} u$.

❧ Comme les scalaires α_k sont deux à deux distincts, on déduit de $(\star)$ que u admet un polynôme annulateur de la forme

$$X^\ell(X - \lambda_2)\cdots(X - \lambda_r).$$

D'après le Théorème de décomposition des noyaux,

$$E = \operatorname{Ker} u^\ell \oplus \Big(\bigoplus_{k=2}^{r} \operatorname{Ker}(u - \lambda_k \operatorname{I}_E)\Big).$$

Si $\operatorname{Ker} u^\ell = \operatorname{Ker} u$, alors

$$E = \operatorname{Ker} u \oplus \Big(\bigoplus_{k=2}^{r} \operatorname{Ker}(u - \lambda_k \operatorname{I}_E)\Big),$$

ce qui prouve que u est diagonalisable.

[**4.**] La matrice de rotation (quart de tour)

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

est inversible mais pas diagonalisable, alors que son carré (égal à $-\operatorname{I}_2$) est inversible et diagonalisable.

136-py-22 [LB]

Soit $n \geqslant 2$. On note K_n, l'ensemble des listes

$$(x_1, \dots, x_n) \in]-1,1[^n$$

telles que $x_1 < x_2 < \cdots < x_n$ et on considère la fonction $f_n : K_n \to \mathbb{R}$ définie par

$$f_n(x_1, \dots, x_n) = -\sum_{k=1}^{n} \ln(1-x_k^2) - 4 \sum_{1 \leqslant i < j \leqslant n} \ln(x_j - x_i).$$

[**1.**] *Écrire une fonction d'argument $X = [x_1, \dots, x_n]$ et calculant la valeur de $f_n(x_1, \dots, x_n)$.*
[**2.**] *Pour $1 \leqslant i \leqslant n$, on pose*

$$r_n = \left(\cos \frac{(2n-2i+1)\pi}{2n}\right)_{1 \leqslant i \leqslant n}.$$

[**2.a.**] *Vérifier que $r_n \in K_n$.*
[**2.b.**] *Calculer $f_n(r_n)$ pour $n = 3, 4, 5$ et 8.*
[**2.c.**] *Conjecturer un équivalent de $f_n(r_n)$.*
[**3.**] *L'ensemble K_n est-il ouvert ? fermé ? convexe ?*
[**4.**] *Démontrer que la fonction f_n est convexe.*
[**5.**] *La fonction f_n est-elle bornée ?*
[**6.**] *Soient $U \subset \mathbb{R}^n$, un ouvert convexe et $\varphi : U \to \mathbb{R}$, une fonction convexe de classe $\mathscr{C}^2$. On suppose que φ atteint un minimum local en $x_0 \in U$. Démontrer que ce minimum est global :*

$$\forall\, x \in U, \quad f(x) \geqslant f(x_0).$$

[**1.**] Le code de la fonction f ne présente aucune difficulté.

```
def f2(x):
    # −1 < x0 < x1 < ··· < xn−2 < xn−1 < 1
 n = len(x)
 s = 0
 for i in range(n):
     xi = x[i]
     s += np.log(1 - xi**2)
     for j in range(i+1, n):
         s += 4*np.log(x[j] - xi)
 return -s
```

Les fans de `numpy`, et eux seulement, peuvent préférer la tournure obscure.

```
def f(x):
    s1 = -np.sum(np.log(1-x**2))
    n = len(x)
    s2 = 0
    for i in range(n):
        s2 += np.sum(np.log(x[i]-x[:i]))
    return s1-4*s2
```

✍ *Le module* `numpy.random` *permet de tester facilement la fonction* `f`.

```
x = 2*rd.random((10))-1
x.sort()
```

[**2.a.**] La fonction cos est strictement décroissante de $]0, \pi[$ sur $]-1, 1[$.
[**2.b.**]

n	3	4	5	8
$f_n(r_n)$	1,726	5,545	11,632	44,361

[**2.c.**] Avant toute chose, il convient d'analyser l'expression de r_n pour simplifier les calculs. L'expression $(2n-2i+1)$ parcourt l'ensemble des entiers impairs compris entre 1 et $2n$ et il faut se souvenir que

$$\forall\, 0 \leqslant \theta \leqslant \pi, \qquad \cos(\pi - \theta) = -\cos\theta.$$

Il est donc préférable de coder comme suit.

```
def r(n):
    angles = [(2*i+1)*np.pi/(2*n) for i in range(n)]
    valeurs = [-np.cos(theta) for theta in angles]
    return np.array(valeurs)
```

Les fans de `numpy` préféreront une version courte.

```
def r(n):
    return -np.cos((2*np.arange(n)+1)*np.pi/(2*n))
```

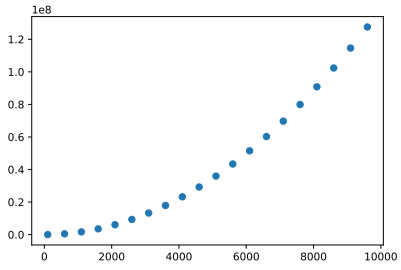
❧ Les calculs qui suivent sont assez longs : évaluer r_n effectue n appels à la fonction cos (qui coûte cher) et l'évaluation de f_n effectue $\mathcal{O}(n^2)$ appels à la fonction ℓn (qui coûté cher). L'exécution reste d'une durée raisonnable quand on sait tirer parti de `numpy` (raison pour laquelle je suis fan de ce module).

On calcule les valeurs :

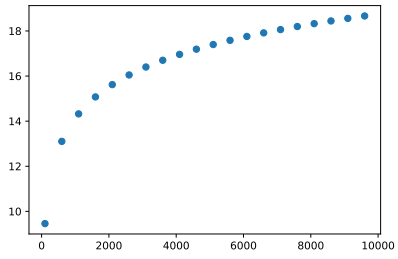
```
N = list(range(100, 10000, 500))

calculs1 = [f(r(n)) for n in N]
calculs2 = [np.log(x) for x in calculs1]
calculs3 = [np.log(x)/np.log(n) for (n,x) in zip(N, calculs1)]
calculs4 = [np.sqrt(x) for x in calculs1]
```

et on trace les graphes.

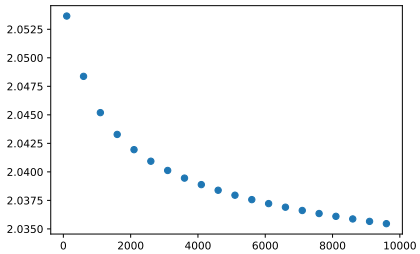
$f_n(r_n)$



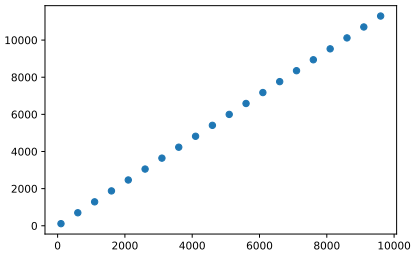
$\ell n\, f_n(r_n)$



$\dfrac{\ell n\, f_n(r_n)}{\ell n\, n}$



$\sqrt{f_n(r_n)}$



Il paraît donc légitime de conjecturer que $f_n(r_n) \sim Kn^2$ pour une constante $K > 0$ convenable (de l'ordre de 1,4).

[**3.**] L'ensemble $\Omega_n =]-1,1[^n$ est une partie ouverte de $\mathbb{R}^n$ en tant que produit cartésien d'intervalles ouverts de $\mathbb{R}$. De même, l'ensemble $O_{n-1} =]0,+\infty[^{n-1}$ est une partie ouverte de $\mathbb{R}^{n-1}$.

Une liste $(x_k)_{1\leqslant k\leqslant n} \in \Omega$ appartient à K_n si, et seulement si,

$$\forall\, 1 \leqslant k < n, \qquad x_{k+1} - x_k > 0.$$

Autrement dit, l'ensemble K_n est l'image réciproque de la partie ouverte O_{n-1} par l'application $\varphi : \Omega_n \to \mathbb{R}^{n-1}$ définie par

$$\forall\, x = (x_1,\dots,x_n) \in \Omega_n, \qquad \varphi(x) = (x_2 - x_1, x_3 - x_2, \dots, x_n - x_{n-1}).$$

L'application φ est la restriction à Ω_n d'une application linéaire de $\mathbb{R}^n$ dans $\mathbb{R}^{n-1}$. Comme $\mathbb{R}^n$ est un espace de dimension finie, l'application φ est continue, donc K_n est un ouvert de Ω_n (en tant qu'image réciproque d'une partie ouverte par une application continue sur Ω_n) et donc un ouvert de $\mathbb{R}^n$ (puisque Ω_n est lui aussi un ouvert de $\mathbb{R}^n$).

❧ Soit $x = (x_1, x_2, \dots, x_n) \in K_n$. Pour tout $k \in \mathbb{N}$,

$$u_k = \left(2^{-k}x_1 + (1-2^{-k})x_2, x_2, \dots, x_n\right)$$

appartient à K_n puisque $2^{-k} < 1$:

$$\forall\, k \in \mathbb{N}, \qquad x_1 \leqslant 2^{-k}x_1 + (1-2^{-k})x_2 < x_2$$

mais

$$\lim_{k\to+\infty} u_k = (x_2, x_2, \dots, x_n) \notin K_n.$$

Par conséquent, la partie K_n n'est pas fermée.

❧ Soient $x = (x_k)_{1\leqslant k\leqslant n}$ et $y = (y_k)_{1\leqslant k\leqslant n}$, deux éléments de K_n, et $t \in [0,1]$. Par définition de K_n, on sait que

$$\forall\, 1 \leqslant k < n, \qquad -1 < x_k < x_{k+1} < 1 \quad \text{et} \quad -1 < y_k < y_{k+1} < 1.$$

Par conséquent, pour tout $t \in]0,1[$,

$$\forall\, 1 \leqslant k < n, \qquad -1 = (1-t)(-1)+t(-1) < (1-t)x_k + ty_k < (1-t)x_{k+1} + ty_{k+1} < (1-t)\cdot 1 + t\cdot 1 = 1$$

et l'encadrement

$$-1 < (1-t)x_k + ty_k < (1-t)x_{k+1} + ty_{k+1} < 1$$

est vrai par hypothèse pour $t = 0$ et pour $t = 1$. Ainsi,

$$\forall\, t \in [0,1], \qquad (1-t)x + ty \in K_n$$

et l'ensemble K_n est une partie convexe de $\mathbb{R}^n$.

[4.] Pour que la question ait un sens, il faut déjà que l'ensemble K_n sur lequel f_n est définie soit convexe : c'est bien le cas !

❧ On reprend les notations précédentes en posant $g(u) = -\ell n(1-u^2)$. Cette fonction g est de classe $\mathscr{C}^2$ sur $]-1,1[$ et convexe puisque

$$\forall\, u \in]-1,1[, \qquad g''(u) = \frac{2(1+u^2)}{(1-u^2)^2} > 0.$$

Par conséquent, par concavité de ℓn,

$$\begin{aligned} f_n\big((1-t)x + ty\big) &= \sum_{k=1}^{n} g\big((1-t)x_k + ty_k\big) - 4\sum_{1\leqslant i<j\leqslant n} \ell n\big((1-t)(x_j-x_i) + t(y_j-y_i)\big) \\ &\leqslant (1-t)\sum_{k=1}^{n} g(x_k) + t\sum_{k=1}^{n} g(y_k) - 4\sum_{1\leqslant i<j\leqslant n} \big[(1-t)\,\ell n(x_j-x_i) + t\,\ell n(y_j-y_i)\big] \\ &\leqslant (1-t)f_n(x) + tf_n(y). \end{aligned}$$

La fonction f_n est donc convexe.

[5.] La fonction f_n n'est pas majorée : si x_1 et x_2 sont très proches l'un de l'autre, la valeur de $f_n(x)$ devient très grande.

Il est vraisemblable que f_n soit minorée (voir la question suivante).

[6.] Il faut commencer par démontrer le résultat en dimension 1. Et il me semble que cela suffit !

136-py-39

Soit $(f_m)_{m\in\mathbb{N}}$, la suite de fonctions définies par la donnée de

$$\forall\, x\in[0,1],\qquad f_0(x)=1$$

et par la relation de récurrence :

$$\forall\, n\in\mathbb{N},\ \forall\, x\in[0,1],\qquad f_{n+1}(x)=\int_0^x 2\sqrt{f_n(t)}\,dt.$$

On pose également

$$a_0=0\qquad et\qquad \forall\, n\in\mathbb{N},\quad a_{n+1}=\frac{a_n}{2}+1$$

ainsi que

$$b_0=1\qquad et\qquad \forall\, n\in\mathbb{N},\quad b_{n+1}=\frac{4\sqrt{b_n}}{a_n+2}.$$

[**1.**] *Écrire une fonction qui, pour $n\in\mathbb{N}$, renvoie le couple (a_n,b_n). Conjecturer la convergence et les limites éventuelles des suites $(a_n)_{n\in\mathbb{N}}$ et $(b_n)_{n\in\mathbb{N}}$.*
[**2.**] *Démontrer que*

$$\forall\, n\in\mathbb{N},\ \forall\, x\in[0,1],\qquad f_n(x)=b_n x^{a_n}.$$

[**3.**] *Tracer les courbes des fonctions f_2, f_3, f_7...pour $x\in[0,1]$. Quelle conjecture peut-on faire sur la suite de fonctions $(f_n)_{n\in\mathbb{N}}$?*
[**4.**] *Démontrer que la suite $(a_n)_{n\in\mathbb{N}}$ est convergente. Préciser sa limite.*
[**5.**] *Démontrer que la suite de terme général*

$$2^{n+1}\ln b_{n+1}-2^n\ln b_n$$

est bornée. En déduire que la suite $(b_n)_{n\in\mathbb{N}}$ converge et préciser sa limite.
[**6.**] *En déduire la convergence de la suite de fonctions $(f_n)_{n\in\mathbb{N}}$. La convergence est-elle uniforme ?*

[**1.**] Code sans mystère.

```
def couple_ab(n):
    a, b = 0, 1
    for i in range(n):
        a, b = a/2+1, 4*np.sqrt(b)/(a+2)
    return a, b
```

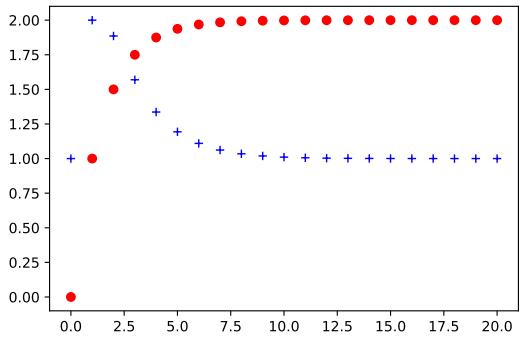
❧ Je préfère éviter de calculer plusieurs fois les mêmes valeurs : si on peut illustrer graphiquement le comportement asympotique des suites avec un complexité linéaire en n, pourquoi s'obliger à le faire avec une complexité quadratique ?

Je reprends donc le code précédent et, au lieu de renvoyer le dernier couple calculé, je renvoie l'ensemble des valeurs calculées.

```
def listes_ab(n):
    L_a, L_b = [0], [1]
    for i in range(n):
        a, b = L_a[-1], L_b[-1]
        L_a.append(a/2 + 1)
        L_b.append(4*np.sqrt(b)/(a+2))
    return L_a, L_b
```

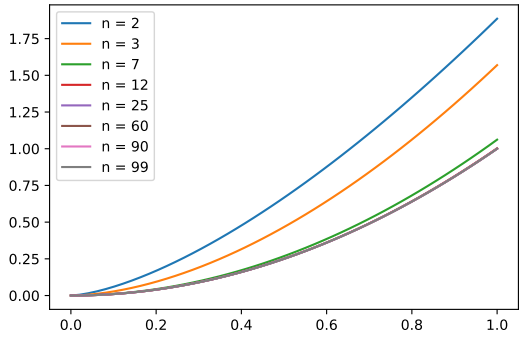
On trace le résultat et on constate que, visiblement, la suite $(a_n)_{n\in\mathbb{N}}$ converge vers 2 et que la suite $(b_n)_{n\in\mathbb{N}}$ converge vers 1.

```
def tracer_ab(n):
    L_a, L_b = listes_ab(n)
    plt.figure()
    plt.plot(L_a, 'ro')
    plt.plot(L_b, 'b+')
```



[**2.**] Récurrence sans mystère.
[**3.**] On calcule un échantillon de couples (a_n, b_n) avec des valeurs "assez" grandes de n, puis on trace le graphe de f_n pour ces valeurs de n.

```
L_a, L_b = listes_ab(100)
X = np.linspace(0, 1)
for n in [2, 3, 7, 12, 25, 60, 90, 99]:
    def fn(x):
        return L_b[n]*x**L_a[n]
    plt.plot(X, fn(X), label=f"n = {n}")
plt.legend()
```



Il semblerait que les trois dernières courbes (pour $n \geqslant 60$) soient confondues — convergence uniforme sur $[0,1]$?
[**4.**] La suite $(a_n)_{n\in\mathbb{N}}$ est une suite arithmético-géométrique de raison $1/2$ et comme $|1/2| < 1$, cette suite est convergente. Sa limite est le "point fixe" ℓ défini par $\ell = \ell/2 + 1$, c'est-à-dire $\ell = 2$.

✍ *Si $u_{n+1} = au_n + b$ et $\ell = a\ell + b$, alors la suite de terme général $v_n = u_n - \ell$ est géométrique de raison a (vérification sans difficulté). Si $|a| < 1$, la suite $(v_n)_{n\in\mathbb{N}}$ converge vers 0, donc la suite $(u_n)_{n\in\mathbb{N}}$ converge vers ℓ.*

[**5.**] Sachant que, pour tout $n \in \mathbb{N}$,

$$(a_n - 2) = \frac{1}{2^n}(a_0 - 2) \qquad \text{c'est-à-dire} \qquad a_n = 2\left(1 - \frac{1}{2^n}\right),$$

on déduit de la relation de récurrence que

$$\forall\, n \in \mathbb{N}, \qquad \ln b_{n+1} - \frac{1}{2}\ln b_n = \ln \frac{b_{n+1}}{\sqrt{b_n}} = \ln \frac{4}{2 + a_n} = -\ln\left(1 - \frac{1}{2^{n+1}}\right).$$

On en déduit que

$$2^{n+1}\ln b_{n+1} - 2^n \ln b_n \xrightarrow[n\to+\infty]{} 1$$

et donc (télescopage et sommation des relations de comparaison) que

$$\ell n\, b_n \underset{n\to+\infty}{\sim} \frac{n}{2^n}$$

et en particulier que la suite $(b_n)_{n\in\mathbb{N}}$ converge vers 1.
[**6.**] Comme b_n tend vers 1 et que a_n tend vers 2, il est clair que, pour tout $x \in [0,1]$, l'expression $b_n x^{a_n}$ tend vers x^2. La suite de fonctions $(f_n)_{n\in\mathbb{N}}$ converge donc simplement sur $[0,1]$ vers $x \mapsto x^2$.

❧ Pour tout $n \in \mathbb{N}$ et tout $x \in [0,1]$,

$$\left|f_n(x) - x^2\right| = \left|(b_n - 1)x^{a_n} + (x^{a_n} - x^2)\right| \leqslant |b_n - 1| + |x^{a_n} - x^2|.$$

La fonction Δ définie par $\Delta(x) = x^{a_n} - x^2$ est nulle en 0 et en 1. De plus,

$$\Delta'(x) = a_n x^{a_n-1} - 2x = (a_n x^{a_n-2} - 2)x$$

donc $\left|\Delta(x)\right|$ est maximum pour $x = x_0$ où $x_0^{a_n-2} = 2/a_n$. Par conséquent,

$$\max_{x\in[0,1]} \left|\Delta(x)\right| = x_0^2(x_0^{a_n-2} - 1) = x_0^2\Big(\frac{2}{a_n} - 1\Big) \leqslant \Big(\frac{2}{a_n} - 1\Big)$$

puisque $0 \leqslant x_0 \leqslant 1$.

✍ *On sait que $a_n < 2$, donc on connaît le signe de $\Delta(x)$.*
Sur les courbes qu'on a tracées plus haut, on a l'impression que l'écart $|f_n(x) - x^2|$ est maximal pour $x = 1$. Les calculs qui précèdent montrent que ce n'est pas le cas – pas tout à fait, car x_0 est quand même très proche de 1.

Bref,

$$\forall\, n \geqslant 1,\ \forall\, x \in [0,1], \qquad \left|f_n(x) - x^2\right| \leqslant |b_n - 1| + \Big(\frac{2}{a_n} - 1\Big).$$

Le majorant est indépendant de x et tend vers 0, donc la suite de fonctions $(f_n)_{n\in\mathbb{N}}$ converge uniformément sur $[0,1]$ vers $x \mapsto x^2$.

[1.a.] *Définir une fonction* `phi(x)` *qui renvoie* 0 *si* $x \leqslant 0$ *et* $x \ln x$ *si* $x > 0$. *Tracer sur une même figure le graphe de* φ *et celui de* $[x \mapsto x-1]$.
[1.b.] *Démontrer que* $\varphi(x) \geqslant x-1$ *pour tout* $x \in \mathbb{R}$, *avec égalité si, et seulement si,* $x = 1$.
[2.] *On considère une variable aléatoire* X *à valeurs dans* $[\![1,n]\!]$ *et on pose*

$$\forall\, 1 \leqslant k \leqslant n, \qquad p_k = \mathbf{P}(X=k)$$

ainsi que

$$H(X) = -\sum_{k=1}^{n} \varphi(p_k).$$

[2.a.] *Définir une fonction* `loi_alea(p)` *qui crée une variable aléatoire* X *à valeurs dans* $[\![1,n]\!]$ *telle que*

$$p = \big(\mathbf{P}(X=1), \dots, \mathbf{P}(X=n)\big).$$

[2.b.] *Définir une fonction* `H(p)` *qui calcule la valeur de* H *en fonction de la loi discrète*

$$p = (p_1, \dots, p_n).$$

[2.c.] *Tester la fonction* `H` *avec la loi binomiale* $\mathscr{B}(5, 1/2)$ *et avec la loi uniforme sur* $[\![1,6]\!]$.
[2.d.] *Démontrer que* $H(X) \geqslant 0$ *avec égalité si, et seulement si,* X *suit une loi de Dirac.*
[2.e.] *Calculer* $H(U_n)$ *où la variable aléatoire* U_n *suit la loi uniforme sur* $[\![1,n]\!]$.
[2.f.] *Démontrer que*

$$H(X) \leqslant H(U_n)$$

pour toute variable aléatoire discrète X *dont la loi est portée par* n *points au plus.*
☞ *On pourra utiliser l'inégalité établie en* ***1.b****.*

[3.] *Pour une variable aléatoire* X *dont la loi est portée par* $\mathbb{N}^*$, *on pose*

$$H(X) = -\sum_{k=1}^{+\infty} \varphi(p_k)$$

(sous réserve de convergence).
[3.a.] *Calculer* $H(X)$ *lorsque* X *suit la loi géométrique de paramètre* p.

✍ *D'autres questions, non traitées.*
Exercice sur le même thème : [136-1239].
NB : *La fonction* H *étudiée ici est l'****entropie*** *au sens des variables aléatoires discrètes.*
Parmi les lois discrètes portées par $\mathbb{N}^*$ *dont l'espérance est fixée à* $\mu > 0$, *la loi d'entropie maximale est la loi géométrique d'espérance* μ, *c'est-à-dire de paramètre* $p = 1/\mu$.

[1.a.] Le code de la fonction `phi` est sans mystère.

```
def phi(x):
    if x>0:
        return x*np.log(x)
    else:
        return 0
```

Mais ce code n'est pas compatible avec numpy, il faut donc calculer les images une par une.

```
X1 = np.linspace(-2, 0)
X2 = np.linspace(0, 3)
Y1 = [phi(x) for x in X1]
Y2 = [phi(x) for x in X2]
```

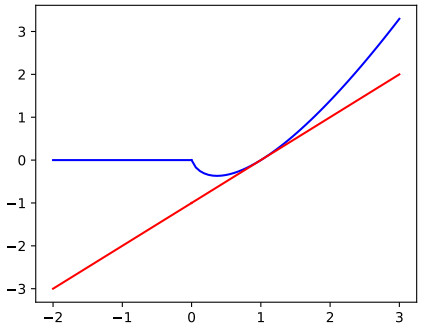
On a procédé par morceaux de telle sorte que la figure (figure de gauche, page suivante) fasse apparaître la tangente verticale en $x = 0$.

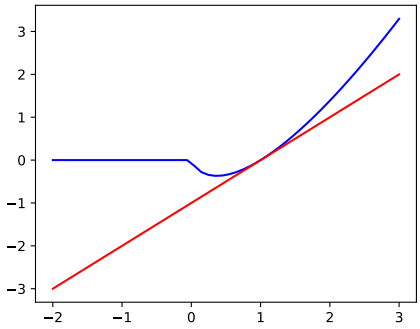
```
plt.plot(X1, Y1, 'b')
```

```
plt.plot(X2, Y2, 'b')
plt.plot(X1, X1-1, 'r')
plt.plot(X2, X2-1, 'r')
```

Une code plus court donnerait une figure légèrement fausse : sur la figure de droite, la tangente à l'origine n'est pas verticale.

```
X = np.linspace(-2, 3)
Y = [phi(x) for x in X]
plt.plot(X, Y, 'b')
plt.plot(X, X-1, 'r')
```





[**1.b.**] Pour $x \leqslant 0$, on a $\varphi(x) = 0 > -1 \geqslant x - 1$.

❧ Pour $x > 0$, la fonction φ est de classe $\mathscr{C}^\infty$ et $\varphi''(x) = 1/x > 0$, donc φ est strictement convexe sur $]0, +\infty[$.

✍ *Les figures qu'on vient de tracer nous mettent en garde : la fonction φ n'est pas convexe sur $\mathbb{R}$!*

La tangente au graphe de φ au point d'abscisse $x_0 = 1$ est la droite d'équation

$$y = \varphi(x_0) + (x - x_0)\varphi'(x_0) = x - 1.$$

Comme φ est strictement convexe, son graphe est au-dessus de chacune de ses tangentes (et en particulier au-dessus de la droite d'équation $y = x - 1$) et ne touche sa tangente au point d'abscisse x_0 qu'un ce seul point.

❧ On a donc

$$\forall\, x \in \mathbb{R}, \quad \varphi(x) \geqslant x - 1 \qquad \text{et} \qquad \varphi(x) = x - 1 \iff x = 1.$$

[**2.a.**] Question difficile, la réponse ne figure pas dans les documents d'accompagnement.

❧ Soit $p = (p_k)_{1 \leqslant k \leqslant n}$, une distribution de probabilité discrète sur l'ensemble fini $E = [\![1, n]\!]$. On définit alors les sommes partielles (aisément calculées avec la fonction `cumsum` [= sommes cumulées] du module `numpy`) :

$$s_0 = 0 \qquad \text{et} \qquad \forall\, 1 \leqslant m \leqslant n, \quad s_m = \sum_{k=1}^{m} p_k.$$

Pour $0 \leqslant x < 1$, on compte alors le nombre d'indices $0 \leqslant m < n$ tels que $x > s_m$: ce nombre d'indices est égal à $1 \leqslant k \leqslant n$ si, et seulement si, $s_{k-1} \leqslant x < s_k$, c'est-à-dire $x \in [s_{k-1}, s_{k-1} + p_k[$.

En choisissant x uniformément sur l'intervalle $[0, 1[$, le nombre d'indices prend donc la valeur $1 \leqslant k \leqslant n$ avec la probabilité p_k (= la longueur de l'intervalle précédent).

```
def loi_alea(p):
    fn_repartition = np.cumsum(p)
    return sum(rd.random()>fn_repartition)
```

[**2.b.**] Aucune difficulté cette fois-ci !

```
def H(p):
    return -np.sum([phi(x) for x in p])
```

[**2.c.**] L'entropie de la loi uniforme est proche de $1,79$.

```
uniforme = [1/6]*6
print(H(uniforme))
```

Pour calculer l'entropie de la loi binomiale $\mathscr{B}(5, 1/2)$, il est plus rapide d'écrire les coefficients binomiaux à la main que d'écrire le code pour les obtenir.

```
binomiale = [1,5,10,10,5,1]
binomiale = [cb/2**5 for cb in binomiale]
print(H(binomiale))
```

La valeur trouvée est proche de $1,52$.

[**2.d.**] Pour tout $1 \leqslant k \leqslant n$, la probabilité p_k est comprise entre 0 et 1, donc $\varphi(p_k) \leqslant 0$. Il est donc clair que $H(X) \geqslant 0$ et que $H(X) = 0$ si, et seulement si,

$$\forall\, 1 \leqslant k \leqslant n, \qquad \varphi(p_k) = 0.$$

Comme $0 \leqslant p_k \leqslant 1$, on en déduit que

$$\forall\, 1 \leqslant k \leqslant n, \qquad p_k = 0 \quad \text{ou} \quad p_k = 1.$$

Comme la somme des p_k est égale à 1, on en déduit que tous les p_k sont nuls, sauf l'un d'eux qui est égal à 1. Autrement dit, l'entropie $H(X)$ est nulle si, et seulement si, la variable aléatoire X est presque sûrement constante.

✍ *L'entropie (au sens de Shannon) mesure l'incertitude dûe au hasard.*
Pour une variable "aléatoire" de Dirac, il n'y a pas de hasard !

[**2.e.**] Dans le cas de la loi uniforme sur n points, tous les termes sont constants et la somme est évidente :

$$H(U_n) = \ln n.$$

✍ *Plus n est grand, plus l'entropie de la loi uniforme U_n est grande.*

[**2.f.**] On sait que φ est convexe sur le segment $[0,1]$ et que les p_k sont tous compris entre 0 et 1. On déduit donc de l'inégalité de Jensen que

$$\varphi(1/n) = \varphi\Big(\sum_{k=1}^{n} \frac{1}{n} p_k\Big) \leqslant \sum_{k=1}^{n} \frac{1}{n} \varphi(p_k) = \frac{-1}{n} H(X).$$

Par conséquent,

$$H(X) \leqslant -n\varphi(1/n) = H(U_n)$$

pour toute variable aléatoire X ayant un support dont le cardinal est inférieur à n.

✍ *Comme la fonction φ est* strictement *convexe sur $[0,1]$, l'entropie est maximale pour $X = U_n$ et seulement pour $X = U_n$.*

En théorie de l'information (Brillouin), on s'intéresse à la ***néguentropie*** *$-H$. Parmi les variables aléatoires discrètes qui prennent au plus n valeurs, la loi uniforme U_n est celle qui apporte le moins d'information, c'est en ce sens la plus aléatoire des lois de probabilité. Puisque cela est conforme à l'intuition qu'on a du hasard, cela valide la définition de l'information comme néguentropie.*

[**3.a.**] Si X suit la loi géométrique $\mathscr{G}(p)$, alors

$$\forall\, k \geqslant 1, \qquad \varphi(p_k) = p \ln p \cdot q^{k-1} + p \ln q \cdot (k-1) q^{k-1}.$$

Comme $0 < q < 1$, on reconnaît une combinaison linéaire de deux séries absolument convergentes (une série géométrique et sa série dérivée au sens des séries entières). Donc $H(X)$ est bien définie.

D'après les remarques précédentes,

$$\begin{aligned} H(X) &= -p \ln p \sum_{k=0}^{+\infty} q^k + p \ln q \sum_{k=1}^{+\infty} k q^k \\ &= -p \ln p \frac{1}{1-q} + q \ln q \sum_{k=1}^{+\infty} k p q^{k-1} \\ &= -\ln p + q \ln q \frac{1}{p} = \frac{-p \ln p - q \ln q}{p}. \end{aligned}$$

✍ *Puisqu'il s'agit d'un exercice de probabilité, au lieu de raisonner sur la dérivée d'une série entière, on a fait apparaître l'espérance de la loi $\mathscr{G}(p)$ — qui est égale à $1/p$ comme chacun sait.*