Voir 1116 centrale 2022

1117. PYTHON. Soit E les fcts $f : \mathbb{R} \to \mathbb{R}$ cies et bornées avec la norme $\|\|\|_\infty$.

Pour $f \in E$, on pose $\Phi(f) : x \mapsto \displaystyle\int_0^{+\infty} \arctan(xt)\frac{f(t)}{1+t^2}\ \mathrm{d}t$.

a) Montrer que Φ est un endomorphisme de E.

b) Soit g l'image par Φ de la fonction constante égale à 1.

Avec Python, tracer g sur le segment $[0, 5]$ et émettre une conjecture sur la limite de g.

c) Calculer la limite de g en $+\infty$.

d) Étudier la dénivabilité de g et calculer sa dérivée.

e) Calculer $g(x) + g(1/x)$. On pourra utiliser Python pour intuiter le résultat.

f) Déduire de ce qui précède la valeur de $\displaystyle\sum_0^\infty \frac{1}{(2n+1)^2}$.

Sol : Alerte rouge, exo tombé tous les ans (entre autre Dorian M).

Deux élèves seront désignés et je relirai avec grand soin !

Codes Python revus sur le cahier de prépa 2022-2023.

Sol : a) Attention à la continuité de l'image, continuité intégrales à paramètres,

domination car f bornée et arctan aussi.

On travaille sur $\mathbb{R}^+$, objet impaire.

b)Codes.

c) Convergence dominée à param continu , dedans on tend vers $\dfrac{\pi}{2(1+t^2)}$.

d) Dérivation int à param, mais il faut passer par le caractère local, sur les segments de $\mathbb{R}_*^+$

Non dérivable en 0 cf graphe python mais pas facile à dém (taux accroissement et $g(0) = 0$) .

$g'(x) = \int_0^\infty \frac{t}{(1+t^2)(1+x^2t^2)}dt$, peut-être calculable...

e) La dérivée est nulle fonct cste sur $\mathbb{R}_*^+$.

La cste se calcule en 1, par primitive directe , arctan au carré divisé par 2.

Cste égal à $\frac{\pi^2}{4}$.

```
import numpy as np
import scipy.optimize as resol
import scipy.integrate as integr
import matplotlib.pyplot as plt

def g(x) :
    def f(t) :
        return np.arctan(x*t)/(1+t*t)
    return integr.quad(f,0,np.inf)[0]

#affichage de g(x)
x=[]
y=[]
pas=0.03
abs=0
for k in range(80):
    x.append(abs)
    y.append(g(abs))
    abs=abs+pas

plt.plot(x,y)
plt.show()

#affichage de g(x) + g(1/x)
x=[]
y=[]
pas=0.01
abs=1.0
for k in range(2090):
    x.append(abs)
    y.append(g(abs)+g(1/abs))
    abs=abs+pas

#plt.plot(x,y)
#plt.axis([0.5, 5., 2., 4.])# astuce pour réguler l'écran voir feuilles Centrale 2
#plt.show()
```

f) Bien sûr les objets valent $\frac{\pi^2}{8}$, mais je n'ai pas vu d'inversion pour le prouver...

Oui oui, j'ai trouvé voir Dorian ds les retours de Rms Mines2021...

1059. PYtHoN . On définit une suite $(A_n)_{m\geqslant 0}$ de polynômes par les conditions :

$$A_0 = 1, \forall n \in \mathbb{N}, A'_{n+1} = A_n \text{ et } \forall n \in \mathbb{N}, \int_0^1 A_{n+1}(x)dx = 0.$$

a) Déterminer A_1, A_2, A_3. Écrire un code qui calcule A_n (utiliser numpy.polynomial).

b) Comparer pour plusieurs valeurs de n : $A_n(0)$ et $A_n(1)$; $A_n(X)$ et $A_n(1-X)$.

Tracer $x \mapsto \frac{x}{e^x - 1}$ et $x \mapsto \sum_{k=0}^{10} A_k(0)x^k$. Émettre des conjectures.

c) Démontrer les conjectures émises.

Sol :

a) On a directement $A_1 = X - \frac{1}{2}$ puis, par intégration et annulation de l'intégrale,

$A_2 = \frac{X^2}{2} - \frac{X}{2} + \frac{1}{12}$.

On trouve aussi $A_3 = \frac{X^3}{6} - \frac{X^2}{4} + \frac{X}{12}$.

Voici un programme qui renvoie A_n en tant qu'objet Python de type Polynomial.

```
import numpy.polynomial as pol
def A(n):
    U=pol. Polynomial ([1])
    if n==0:
        return(U)
    else:
        B=A(n-1).integ()
        C=B.integ(1,0)
        D=B-C(1)
        return(D)
```

b) On compare $A_n(0)$ et $A_n(1)$ pour les 9 premières valeurs via le programme suivant :

```
for n in range(1,10):
   print(n,A(n)(0),A(n)(1))
```

Le résultat obtenu suggère la conjecture suivante :

Conjecture 1. On a $A_n(0) = A_n(1)$ pour tout $n \geqslant 2$.

On peut également composer les polynômes

(le code de programmation de composition des polynômes est intuitif) :

```
for n in range (1,10):
    S=pol. Polynomial ([1,-1])
    print(n)
    print(A(n))
    print(A(n)(S))
    print("")
```

Au signe près, on trouve les mêmes valeurs numériques

(non facilement identifiables) pour les coefficients de $A_n(1 - X)$ et A_n.

La conjecture qui se profile est :

Conjecture 2. On a $A_n(1 - X) = (-1)^n A_n$ pour tout $n \in \mathbb{N}$.

Passons à la représentation graphique demandée.

```
import math
import matplotlib.pyplot as plt
import numpy
def u(x):
     return(x/()math.exp(x)-1)
def w(x):
    S=A(0)(0)
    for k in range(1,11):
        S+=A(k)(0)*(x**k)
    return(S)

X=numpy.linspace(-2,2,10)
Y=[u(x) for x in X]
Z=[w(x) for x in X]
plt.plot (X, Y)
plt.plot (X, Z)
plt.show ()
```

En apparence, on obtient une seule courbe... :

Si l'on remplace la ligne définissant les ordonnées Z par

```
Z=[w(x)+1 for x in X]
```

afin de translater vers le haut la courbe de la seconde fonction, alors on obtient...

II est maintenant clair que la première représentation est une superposition des 2 courbes.

Conjecture 3. On a

$$\forall x \in \mathbb{R}\backslash\{0\}, \quad \frac{x}{e^x - 1} = \sum_{k=0}^{+\infty} A_k(0) x^k$$

c) Preuve de la conjecture 1. Soit $n \in \mathbb{N}$ tel que $n \geqslant 2$. On a tout simplement :

$$A_n(1) - A_n(0) = \int_0^1 A_n'(t)\mathrm{d}t = \int_0^1 A_{n-1}(t)\mathrm{d}t = 0$$

Preuve de la conjecture 2 par récurrence.

La formule est triviale pour $n = 0$.

On suppose que l'on a $A_n(1 - X) = (-1)^n A_n$ et l'on souhaite montrer la formule

$A_{n+1}(1 - X) = (-1)^{n+1} A_{n+1}$.

Or le polynôme différence $A_{n+1}(1 - X) - (-1)^{n+1} A_{n+1}$ est clairement d'intégrale nulle de 0 à 1 (quitte à faire un changement de variable linéaire immédiat sur le terme en $1 - X$) et son polynôme dérivé est

$$-A_{n+1}'(1 - X) - (-1)^{n+1} A_{n+1}' = -A_n(1 - X) - (-1)^{n+1} A_n = 0.$$

Ainsi, $A_{n+1}(1 - X) - (-1)^{n+1} A_{n+1}$ est forcément le polynôme nul.

Preuve de la conjecture 3.

Il y a deux difficultés dans cette conjecture.

D'abord, il faut justifier que la série du second membre cv et pour quels x.

En l'occurrence, on a peut-être été trop généreux en s'autorisant à choisir x dans $\mathbb{R}^*$.

II serait déjà satisfaisant d'avoir une information pour x voisin de l'origine.

Justifions que la suite $(A_k(0))$ est bornée.

Cela prouvera, par thm de séries entières, que la série $\sum_{kk}(0)x^k$ cv sur $x \in]-1, 1[$.

(le rayon de la série entière est au moins égal à 1).

D'après la définition des polynômes A_k, l'inégalité $\sup_{[-1,1]} |A_k(x)| \leqslant 1$

s'obtient immédiatement par récurrence grâce au lemme suivant :

Lemme 1. Soit $f \in \mathcal{C}^1([0,1],R)$ vérifiant $\int_0^1 f(x)dx = 0$ et $\|f\|_\infty \leqslant 1$. Alors $\|f\|_\infty \leqslant 1$.

Preuve. Pour tout $x \in [-1,1]$, on écrit

$$f(x) = (f(x) - f(1/2)) + f(1/2) \quad (1).$$

L'inégalité des accroissements finis donne

$$|f(x) - f(1/2)| \leqslant |x - 1/2| \times \left\|f'\right\|_\infty \leqslant \frac{1}{2} \quad (2).$$

En intégrant (1) sur $[0,1]$ et en exploitant (2), on obtient

$$|f(1/2)| = \left| \int_0^1 f(x) - f(1/2) \mathrm{d}x \right| \leqslant \int_0^1 |f(x) - f(1/2)| \mathrm{d}x \leqslant \frac{1}{2} \quad (3)$$

Grâce à (2) et à (3), on conclut que $|f(x)| \leqslant 1$.

Ensuite, il faut trouver un moyen d'identifier les coefficients de deux côtés !

ll est plus simple de tenter de démontrer la formule suivante :

$$\forall x \in]-1,1[, \quad x = (e^x - 1) \sum_{k=0}^{+\infty} A_k(0) x^k \quad (4).$$

Remarque. L'étude de l'intervalle maximal sur lequel la série est convergente

(c'est-à-dire le calcul exact du rayon de convergence) est HORS programme...

D'ailleurs, si on reprend les représentations graphiques précédentes sur un intervalle strictement plus grand que $[-2\pi, 2\pi]$, on constate effectivement une explosion à l'extérieur de cet intervalle :

Revenons à la preuve de (4). Par convergence absolue des séries envisagées,

le membre droit se reformule via un produit de Cauchy :

$$\forall x \in]-1,1[, \left(\sum_{k=1}^{+\infty} \frac{x^k}{k!} \right) \times \left(\sum_{k=0}^{+\infty} A_k(0) x^k \right) = \sum_{n=1}^{+\infty} \left(\sum_{k=1}^{n} \frac{A_{n-k}(0)}{k!} \right) x^n.$$

Le coefficient d'indice $n = 1$ vaut $A_0(0) = 1$ (d'après la question a)).

La formule (4) sera donc conséquence des identités suivantes

$$\forall n \geqslant 2, \quad \sum_{k=1}^{n} \frac{A_{n-k}(0)}{k!} = 0$$

Cette formule (non évidente) le devient abordable si l'on invoque le lemme suivant :

Lemme 2. Pour tout polynôme $P \in \mathbb{R}_n[X]$ on a la formule $P(1) = \sum_{k=0}^{n} \frac{P^{(k)}(0)}{k!}$.

Preuve. II suffit d'appliquer la formule de Taylor de 0 à 1.

On applique le lemme précédent au polynôme A_n

(qui est de degré n et vérifie $A_n(k) = A_{n-k}$ par récurrence immédiate) :

$$A_n(1) = A_n(0) + \sum_{k=1}^{n} \frac{A_{n-k}(0)}{k!}$$

La conjecture 1 (prouvée ci-dessus) affirme que $A_n(1) = A_n(0)$. Donc (4) est vraie.

Math-info Centrale Duncan :

Enoncé : Epreuve bidon au sens informatique, mais pas si simple en maths.

Soit $(q_n)_{\mathbb{N}^*}$ une suite d'entiers croissante, avec $q_1 \geqslant 2$. On note $S_n = \sum_{1}^{n} \frac{1}{q_1 q_2 ... q_k}$.

1) Info : Regarder la convergence de S_{100} pour différentes suites. Voir Codes après.

Conjecture ?

2) Démontrer la conjecture ($S_\infty \in]0, 1]$).

3) On se propose de regarder la réciproque, à savoir que $\forall x \in]0, 1]$,

il peut s'obtenir comme limite de S_n pour une suite d'entiers bien choisie...

a) Montrer que $\frac{1}{q_1} < x \leqslant \frac{1}{q_1 - 1}$.

b) En déduire q_1.

c) On note $x_p = (x - S_p) \times (q_1 ... q_p)$. Trouver q_{p+1} à partir de x_p.

d) Montrer que $x_{p+1} = x_p.q_{p+1} - 1$.

e) Ecrire un programme qui trouve la suite jusqu'à 100.

Solution :

1) On conjecture la convergence et dans $]0, 1]$.

Chaque $q_k \geqslant 2$, donc le terme général positif de notre série est majoré par $\frac{1}{2^k}$ (cvte).

2) On fait une analyse, qui devra se terminer par une réciproque structurée...

Si $x = \sum_1^n \frac{1}{q_1 q_2 ... q_k}$.

a) Il est clair que $x > \frac{1}{q_1}$ qui est le premier terme de cette série strictement positive.

b) On passe à l'inverse, $q_1 - 1 \leqslant \frac{1}{x} < q_1$, donc $q_1 = 1 + \left\lfloor \frac{1}{x} \right\rfloor$.

c) $x - S_p = \frac{1}{q_1 ... q_{p+1}} + \frac{1}{q_1 ... q_{p+1} q_{p+2}} + ...$

Donc $(x - S_p) \times (q_1 ... q_p) = x_p = \frac{1}{q_{p+1}} + \frac{1}{q_{p+1} q_{p+2}} + ...$

On est revenu très exactement à la question ab), donc $q_{p+1} = 1 + \left\lfloor \frac{1}{x_p} \right\rfloor$.

Rq : pour la réciproque $q_{p+1} > 1$, et entier.

d) On remplace $x_{p+1} = \left(x - S_p - \frac{1}{q_1 ... q_{p+1}} \right) (q_1 ... q_{p+1}) = x_p.q_{p+1} - 1(**)$.

Par développement simple.

e) Les codes sont après avec des résultats cohérents.

Mais, avons nous **TOUT** ce qu'il fallait ?

Par (**), $x_{p+1} \leqslant x_p \Leftrightarrow x_p \leqslant \frac{1}{q_{p+1}}$ qui est vraie donc la suite (x_p) décroit (>0).

Donc $q_{p+1} = 1 + \left\lfloor \frac{1}{x_p} \right\rfloor$ est croissante.

Mais la série ainsi crée (cvte par 1)) converge-t-elle vers x ?

$x - S_p = \frac{x_p}{q_1 ... q_p}$ est positive décroissante vers 0, car numérateur décroit,

dénominateur croissant vers l'infini. Ai-je oublié qq chose ?

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

def S(L):
    s=1/L[0]
    P=L[0]
    for j in range (1,len(L)):
        P=P*L[j]
        s=s+1/P
    return s
#A=[rd.randint(2,10)]
#for k in range (99):
    #A.append(rd.randint(A[-1],A[-1]+100))
def G(x):
    L=[]
    y=x
    for j in range(40):
        q=1+np.floor(1/y)
        L.append(q)
        y=y*q-1
    return L

>>> S(G(0.32))
0.31999999999999995
>>> S(G(0.6772))
0.67719999999999991 # Cohérent
```

Certains codes ds rms centrale 19 en vrac à la fin. ou Centrale 2021 en vrac à la fin...

Camille Centrale math-info. Enoncé qui me semble curieux...Mais maintenant ça passe.

On pose $A_n = \begin{pmatrix} 2 & 1 & 0 & \cdots & 0 \\ 1 & 2 & 1 & \ddots & \vdots \\ 0 & 1 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 2 & 1 \\ 0 & \cdots & 0 & 1 & 2 \end{pmatrix} \in \mathcal{M}_n(\mathbb{R})$.

1.Prouver que A_n possède n valeurs propres réelles, notées $(\lambda_1^{(n)}, ..., \lambda_n^{(n)})$.

.

2. Ecrire une fonction `rectangle`(f), qui en utilisant la méthode des rectangles permet de calculer une valeur approchée de $\dfrac{1}{2\pi}\displaystyle\int_{-2}^{2} f(t+2)/\sqrt{4-t^2}dt$, avec f continue .

3. Ecrire une fonction `demicercle`(n, f) qui calcule $\frac{1}{2n}\sum_{k=1}^{n} f\left(\frac{\lambda_k^{(n)}}{1.}\right)$.

4. Afficher les résultats des 2 fcts pour $n \in \{10, 100, 1000\}$ et $f : t \mapsto 1$, $f : t \mapsto t$.

Faire une conjecture.

5. Soit $\lambda \in Sp(A_n)$ et $X = (x_1, ..., x_n)^T$ un vecteur propre associé.

Démontrer que pour $k \in [\![1, n]\!]$, $x_{k-1} + (2 - \lambda)x_k + x_{k+1} = 0$.

En posant $x_0 = x_{n+1} = 0$.

6. En déduire le spectre de A_n.

7. Conclure sur la conjecture précédente.

Sol :

```
#centrale camille 2021
from math import *
from numpy import *
from numpy.linalg import *
import numpy.linalg as alg
def rectangle(f,n):# intégrale généralisée donc rectangle milieu
#pour éviter le message d'erreur
    h=4/n
    x,s = -2.+h/2,0.
    for k in range(n):
        s += f(x+2)/sqrt(4.-x*x)
        x += h
    return h*s/2/np.pi
def f(t):
    return 1.
def g(t):
    return t+0.0
def h(t):# pour faire des tests
    return sqrt(t+3.)+0.0
def  virginie(n): # Nos matrices
    V=np.zeros((n,n))
    for i in range (0,n):
        V[i,i]=2
    for j in range(0,n-1):
        V[j,1+j]=1
    for k in range(1,n):
        V[k,k-1]=1
    return V
def vp(M): # pour visualiser le spectre
    return list(alg.eigvals(M))
def demicercle(n,f):
    L=vp(virginie(n))
    S=0
    for j in range(0,n):
        S=S+f(L[j])
    return S/n/2
```

1. A_n est symétrique réelle donc $\mathbb{R}$ dz.

2. Voir codes , attention aux erreurs. Il y a un piège pour la méthodes des rectangles !

Rq : intégrale généralisée convergente, car aux extrémités on a un dominant

de type Riemann convergente $\alpha = \frac{1}{2}$.

3. Voir codes qui valorisent les feuilles "Centrale".

4. L'égalité bien sûr.

5. Cela a été fait au tableau, matrices de Toeplitz (Centrale PSI 2018) .

Mais, ici c'est plus simple car on connait les coeff.

La relation de récurrence linéaire s'obtient par un simple développement matriciel.

6. On trouve n valeurs propres : $k \in [\![1, n]\!]$, $\lambda_k^{(n)} = 2 + 2\cos\left(\dfrac{k\pi}{n+1}\right)$.

Je refais avec les hypothèses de l'énoncé.

Equation caractéristique : $r^2 + (2 - \lambda)r + 1 = 0$. Rq utile $r \neq 0$.

Si $\Delta = 0$ racine double, $x_k = (\alpha + k\beta)r^k$, absurde.

Car $x_0 = 0$ entraine $\alpha = 0$, puis $x_{n+1} = 0$ entraine $\beta = 0$.

Donc $r_1 \neq r_2$, $x_k = \alpha r_1^k + \beta r_2^k$. Mais $x_0 = 0$ donc $\alpha = -\beta$.

$x_k = \alpha(r_1^k - r_2^k)$, mais $\alpha \neq 0$ et $x_{n+1} = 0$ donc $\dfrac{r_1}{r_2} \in \mathcal{U}_{n+1}$.

Donc $r_1 = r_2 e^{\frac{2ik\pi}{n+1}}$, mais $r_1 r_2 = 1$ par produit des racines, donc $r_2^2 = e^{\frac{-2ik\pi}{n+1}}$.

Rq : $k \in [\![1, n]\!]$, car $r_1 \neq r_2$ et racines unité.

Comme r_1 et r_2 ont des rôles symétriques, par ex $r_2 = e^{\frac{-ik\pi}{n+1}}$.

Par somme des racines $\lambda_k = 2 + r_2\left(1 + e^{\frac{2ik\pi}{n+1}}\right)$, $k : 1 \longrightarrow n$.

On remplace , c'est fini.

7. On fait le changement de classe $\mathcal{C}^1$ et strictement monotone : $t = 2\cos(\theta)$.

On aboutit à : $\dfrac{1}{2\pi}\displaystyle\int_0^{\pi} f(2 + 2\cos(\theta))d\theta$. D'où l'idée du demi-cercle.

On nous a imposé la méthode des rectangles , invitation directe aux sommes de Riemann.

On évalue : $\dfrac{1}{2n}\displaystyle\sum_{k=1}^{n} f\left(2 + 2\cos\left(\dfrac{k\pi}{n}\right)\right)$.

Elle ressemble beaucoup à celle qui est dans l'énoncé.

On regarde la somme de Riemann associée à notre intégrale,

attention on la prend au rang $n+1$ et à gauche.

$S_n = \dfrac{\pi}{2\pi(n+1)} \displaystyle\sum_{0}^{n} f\left(0 + \frac{k\pi}{n+1}\right)$.

C'est presque la nôtre , un terme en trop qui tend vers 0.

Les deux convergent bien vers la même limite car $\dfrac{1}{n+1} \sim \dfrac{1}{n}$.

Centrale 9 (2017) :

Une matrice $M = (m_{i,j}) \in \mathcal{M}_n(\mathbb{R})$ est dite à diagonale strictement dominante si et seulement si, pour tout $i \in [\![1, n]\!]$, $\sum_{j \neq i} |m_{i,j}| < |m_{i,i}|$.

1. Écrire un programme Python diagdom, qui prend en argument une matrice M et un entier n (sa taille) et teste si la matrice M est ou n'est pas à diagonale strictement dominante.

2. Trouver une matrice $M \in \mathcal{O}_2(\mathbb{R})$ qui vérifie ce test.

3. L'ensemble des matrices à diagonale strictement dominante.

Est-il stable pour la multiplication des matrices.

4. Soit X un vecteur colonne de $\mathcal{M}_{n,1}(\mathbb{R})$ de norme $\|X\|_\infty$ le maximum des valeurs absolues de ses composantes.

Si M est à diagonale strictement dominante, montrer que $MX = 0$ entraîne $X = 0$.

Que peut-on en déduire pour M ?

5. Écrire une fonction d'argument M et n qui renvoie $\Gamma_M = \text{In}\, f \left\{ |m_{i,i}| - \sum_{j \neq i} |m_{i,j}|, i \in [\![1, n]\!] \right\}$.

6. Montrer que l'ensemble des matrices à diagonale strictement dominante est un ouvert de $\mathcal{M}_n(\mathbb{R})$ (on utilisera la norme $\|\|_\infty$ sur ces matrices.

Sol :

Une matrice $M = (m_{i,j}) \in \mathcal{M}_n(\mathbb{R})$ est dite à diagonale strictement dominante

si et seulement si, pour tout $i \in [\![1, n]\!]$, $\sum_{j \neq i} |m_{i,j}| < |m_{i,i}|$.

2. Si l'on choisit une rotation, exprimée en cos sin, il faut que le sinus, soit en valeur absolue strictement plus petit que la valeur absolue du cosinus. Ceci correspond aux angles de mesure dans $]-\frac{\pi}{4}, \frac{\pi}{4}[$ et $]\frac{3\pi}{4}, \frac{5\pi}{4}[$ à 2π près.

3. En choisissant deux rotations d'angle $\frac{\pi}{6}$,

on trouvera une rotation d'angle $\frac{\pi}{3}$ qui n'est donc plus à diagonale strictement dominante..

Donc cet ensemble n'est pas stable pour la multiplication des matrices.

4. Voir exercice 987...

6. Si i est fixé, l'application ϕ_i qui à une matrice M associe le réel $|m_{i,i}| - \sum_{j \neq i} |m_{i,j}|$

est continue (par somme, composition avec la valeur absolue, puis différence).

Ainsi l'ensemble $\phi_i^{-1}(]0, \infty[)$ est l'image réciproque d'un ouvert de $\mathbb{R}$,

c'est-à-dire un ouvert de $\mathcal{M}_n(\mathbb{R})$.

Par intersection finie d'ouverts, pour $i \in [\![0, n-1]\!]$,

l'ensemble étudié est donc un ouvert de $\mathcal{M}_n(\mathbb{R})$.

Le code :

```
def dd(M,n):
    b,i=True,0
    while b and i<n:
        s=0
        for j in range(n):
            if j!= i:
                s+=abs(M[i,j])
        b=(abs(M[i,i])-s>0)
        i +=1
    return (b)
def rot(t):
    a=np.cos(t)

    M=np.identity(2)
    M=a*M
    M[0,1]=-np.sin(t)
    M[1,0]= np.sin(t)
    return (M)
# M=rot(pi/6)
#dd(np.dot(M,M),2)

def gamma(M,n):
    mm=abs (M[ 0 , 0 ] )
    for i in range ( n ) :
        s=0
        for j in range (n):
            if j!= i:
                s+=abs(M[i,j])
        m=abs(M[i,i])-s
        mm=min(m,mm)
    return (mm)
```

Centrale 2 : exo 4

Les parties informatique et Mathématique sont entremêlées .

Soit $I_n = \int_0^1 (1-t)^n \sin(\pi t) dt$.

1.a Montrer que I_n est bien définie.

1.b Calcul des 50 premiers termes, les tracer.

1.c Faire une conjecture et la démontrer.

2.a Chercher une relation entre I_{n+2} et I_n. La vérifier avec Python.

2.b En déduire un équivalent de I_n.

3.a Nature de la série $\sum I_n$.

3.b Exprimer la somme de cette série sous forme d'une intégrale.

Sol :

Centrale 2 Soit $I_n = \int_0^1 (1-t)^n \sin(\pi t) dt$.

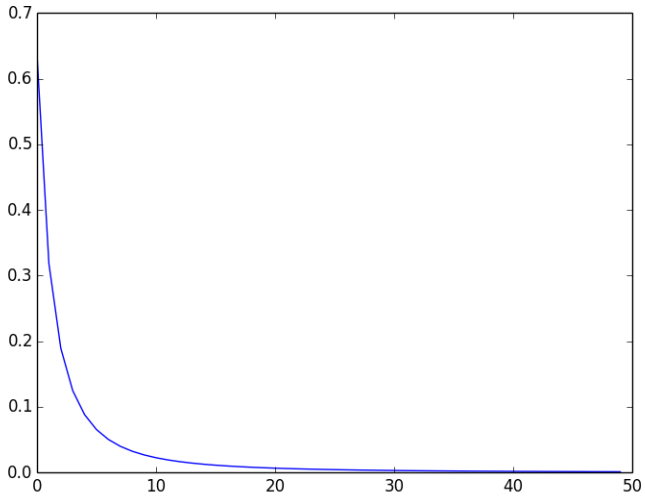
1.a Pour tout entier n fixé, la fonction $t \mapsto (1-t)^n \sin(\pi t)$ est continue sur le segment [0,1].

```
import scipy .integrate as si
import matplotlib.pyplot as plt
from math import *
import numpy as np

def f(n):
    return(lambda t: (1-t)**n*sin(pi*t))
def I(n):
    return(si.quad(f(n) ,0,1)[0])

def question_1 ():
    s=[]
    for k in range(50):
        s.append(I(k))
    return(s)
print( question_1 ())
x=range(50)
y= question_1 ()
plt.plot(x,y)
```

On a :



Ce qui nous fait "conjecturer" que la suite décroît et converge vers 0.

Pour tout entier n, (intégrale d'une fonction négative) :

$$I_{n+1} - I_n = -\int_0^1 t(1-t)^n \sin(\pi t) dt \leqslant 0.$$

Donc la décroissance est établie.

De plus $|I_n| \leqslant \int_0^1 (1-t)^n dt = \dfrac{1}{n+1} \mapsto 0$ donc par encadrement la suite converge bien vers 0.

2.a Pour tout entier n fixé. En intégrant deux fois par parties l'intégrale qui définit I_n

(on dérive à chaque fois la fonction trigonométrique).

$$I_n = \int_0^1 (1-t)^n \sin(\pi t) dt = \frac{\pi}{(n+1)(n+2)} - \frac{\pi^2}{(n+1)(n+2)} \int_0^1 \sin(\pi t)(1-t)^{n+2} dt$$
$$I_n = \frac{\pi}{(n+1)(n+2)} (1 - \pi I_{n+2})$$

Le petit script ajouté au programme déjà écrit permet de vérifier si les calculs sont justes.

```
def verif ():
    b,k=True ,0
    while b and (k <48):
        b=(y[k]-pi /((k+1)*(k+2))*(1-pi*y[k+2]) <10**( -8))
        k+=1
    return(b)

print(verif ())
```

(ça affiche "True")...

2.b Comme $I_{n+2} = o(1)$ dans la formule précédente on obtient directement :

$$I_n \sim \frac{\pi}{(n+1)(n+2)} \sim \frac{\pi}{n^2}$$

3.a Par comparaison à la série de Riemann pour $\alpha = 2 > 1$,

appliquée pour les séries à termes positifs, la série $\sum I_n$ converge donc.

3.b Si on autorise la commutation des symboles on obtient pour la somme $S = \int_0^1 \dfrac{\sin(\pi t)}{t} dt$.

Il reste à vérifier le théorème de Fubini pour la suite de fonction $u_n : t \mapsto (1-t)^n \sin(\pi t)$.

- Pour tout n la fonction u_n est continue (par morceaux) sur $]0, 1[$.

- Pour tout t fixé dans $]0,1[$, la série (géométrique) de raison $0 < 1-t < 1$ converge vers $\frac{1}{t}$ donc la série $\sum u_n$ converge simplement sur $]0,1\,[$ vers $f : \frac{\sin(\pi t)}{t}$.

- La fonction f est clairement cpm sur l'ouvert $]0,1[$ (elle l'est même sur le fermé).

- Comme la fonction à intégrer est positive, le β_n cöncide avec I_n et il est donc la série des β_n converge (question précédente).

Centrale 25 :

Soit $f_n : x \mapsto \cos(n\pi x)x(1-x)$. On pose $I_n = n\displaystyle\int_0^1 f_n(t)dt$.

1. Tracer les fonctions f_n pour quelques valeurs de n sur $[0,1]$.

Que remarquez vous ? Le démontrer.

2. Montrer que si n est impair I_n est nulle.

3. Calculer la suite des I_{2n} pour $n \in \{10, 100, 1000\}$. Que conjecturer ?

On admet désormais que I_{2n} est équivalent à $\dfrac{K}{\pi^2}\dfrac{1}{n^\alpha}$ (avec α entier).

Comment évaluer grâce à Python les constantes manquantes

(on utilisera pour n les nombres précédemment donnés).

4. Soit $\sum I_n x^n$. Déterminer son rayon R.

Sol :

Soit $f_n : x \mapsto \cos(n\pi x)x(1-x)$. On pose $I_n = n\displaystyle\int_0^1 f_n(t)dt$.

On a $f_n(1-x) = \cos(n\pi(1-x))(1-x)x = (-1)^n f_n(x)$.

Donc si n est pair la représentation graphique de la fonction admet un axe de symétrie (d'équation $x = \dfrac{1}{2}$) et si n est impair la somme $f_n\left(\dfrac{1}{2}-h\right) + f_n\left(\dfrac{1}{2}+h\right) = 0$

donc le point de coordonnées $\left(\dfrac{1}{2}, 0\right)$ est centre de symétrie.

2. Les graphes montrent des symétries possibles.

On fait à l'intérieur de I_n (intégrale d'une fonction continue sur le segment pas de problème) le changement de variable $v = 1-t$.

Quand n est impair ceci permet de démontrer que l'intégrale est nulle (ce qui était évident du point de vue géométrique).

Il semble que la suite tende vers 0.

Les résultats nous inclinent à penser que $\alpha = 1$ et $K = -\frac{1}{2}$.

On admet désormais que $I_{2n} \sim \dfrac{-1}{2\pi^2}\dfrac{1}{2n}$.

4. Tous les termes impairs de la série sont nuls. $F(x) = \sum I_{2n}x^{2n}$.

On a donc $\left|I_{2n}x^{2n}\right| \sim \dfrac{x^{2n}}{2\pi^2}\dfrac{1}{2n}$ et par exemple par le critère de d'Alembert on trouve $R = 1$.

Le code :

```
import numpy as np
import scipy .integrate as si
import matplotlib.pyplot as plt
from math import *
def f(n):
    return(lambda x: x*(1-x)*cos(n*pi*x))
def dessin(ens):
    x=np.arange(0, 1, 0.005)
    for z in ens:
        y=[f(z)(u) for u in x]
        plt.plot(x,y)
    plt.show ()
dessin([2 ,3 ,5 ,8 ,11])

def I(n):
    return(n*si.quad(f(n) ,0,1)[0])
print(I(5),I(10),I(100))
#I(10)/I(100) alpha =1...
#I(100)*100*np.pi*np.pi attention , si n grand...Aie
```

Centrale 30 :

Soit (a, b) fixés dans $\mathbb{R}^2$ et $n \in \mathbb{N}^*$.

On définit l'application $f : P \mapsto (X - a)(X - b)P' - nXP$ où $P \in \mathbb{R}_n[X]$.

1. Montrer que f définit ainsi un endomorphisme de $\mathbb{R}_n[X]$.

2. Exhiber la matrice M de f dans la base canonique.

3. Créer une fonction prenant en argument n, a, b et renvoyant la matrice M correspondante.

4. Créer une fonction prenant (n, a, b) en argument et renvoyant les valeurs propres et les vecteurs propres de la matrice M.

Sol :

Soit (a, b) fixés dans $\mathbb{R}^2$ et $n \in \mathbb{N}^*$.

On définit l'application $f : P \mapsto (X-a)(X-b)P' - nXP$ où $P \in \mathbb{R}_n[X]$.

1. La linéarité se fait sans problème. À l'arrivée on obtient bien un polynôme.

Le terme de plus haut degré fournit par e calcul est na_nX^{n+1} à cause du premier terme

(si a_nX^n commence P).

Celui du second est le même. Donc le résultat a un degré inférieur ou égal à n et ainsi

f définit bien un endomorphisme de $\mathbb{R}_n[X]$.

2. Pour $X^0 = 1, f(1) = -nX$.

Pour X^k(avec $k \leqslant n-1$) on a $f\left(X^k\right) = k(X-a)(X-b)X^{k-1} - nX^{k+1}$ donc

$$f\left(X^k\right) = X^{k+1}(k-n) - kX^k(b-a) + kabX^{k-1}$$

Enfin

$$f\left(X^n\right) = -nX^n(b-a) + nabX^{n-1}$$

Le code :

```
np.set_printoptions(precision=1)
def M(n,a,b):
    A=np.zeros ((n+1,n+1))
    A[1 ,0]= -n
    A[n,n]=-n*(b-a)
    A[n-1,n]=n*a*b
    for i in range(1,n):
        A[i,i+1]=(i-n)
        A[i,i]=-i*(b-a)
        A[i-1,i -1]=n*a*b
    return(A)
def propre(n,a,b):
    A=M(n,a,b)
    return(np.linalg.eig(A))#attention à la lecture des vectpro...
```

Centrale 37 :Non non non voir Baude retour 23 bien mieux rédigé !

(Avec préparation et ordinateur) (30 minutes de préparation, 30 minutes d'oral) :

1. Soit $(a_n)_{n\geqslant 1}$ une suite telle que $a_1 = 1$ et pour tout $n > 1, a_n = \sum_{k=1}^{n-1} a_k a_{n-k}$.

a. Calculer les 10 premiers termes de la suite.

b. Soit $S_n = \sum_{k=1}^{n} \frac{a_k}{4^k}$.

Tracer S_n en fonction de n pour $1 \leqslant n \leqslant 10$.

Que peut-on conjecturer ? On admettra la conjecture.

c. Compte tenu de la conjecture, la série entière converge pour $x = \frac{1}{4}$,

donc le rayon de convergence $R \geqslant \frac{1}{4}$.

Que peut-on dire du rayon de la série entière $\sum a_n x^n$?

2. Soit la suite $(b_n)_{n\geqslant 1}$ telle que $b_1 = 1$ et pour tout $n > 1, b_n = \frac{-n}{2(n-1)} \sum_{k=1}^{n-1} b_k b_{n-k}$.

a. Calculer les 10 premiers termes de la suite.

b. Que peut-on dire du rayon de la série entière $\sum b_n x^n$?

ddl 119 F6 exo 23 voir Baude retours 2023

Sol :

1. Soit $(a_n)_{n\geqslant 1}$ une suite telle que $a_1 = 1$ et pour tout $n > 1$, $a_n = \sum_{k=1}^{n-1} a_k a_{n-k}$.

Plutôt que de créer une fonction qui calcule a_k (par exemple par récursivité),

je préfére écrire un script qui renvoie toute la liste des valeurs,

ce qui évite de recalculer plusieurs fois les termes qui servent à chaque cran.

On a préféré que le terme de rang i, soit directement a_i sans décalage,

et pour ce faire on a ajouté un premier terme fictif.

Calculer les 10 premiers termes de la suite.

La courbe semble asymptote à la droite d'équation $y = \dfrac{1}{2}$, ce qui sous entendrait que $S_n \sim \frac{1}{2}$.

c. Compte tenu de la conjecture, la série entière converge pour $x = \dfrac{1}{4}$,

donc le rayon de convergence $R \geqslant \frac{1}{4}$.

2. Soit la suite $(b_n)_{n\geqslant 1}$ telle que $b_1 = 1$ et pour tout $n > 1, b_n = \dfrac{-n}{2(n-1)} \sum_{k=1}^{n-1} b_k b_{n-k}$.

Note on a amélioré la fonction dessin, pour pouvoir prendre des entiers plus grands.

Sinon le re-calcul de tous les termes coute trop cher.

On voit que la suite $(|b_n|) \leqslant a_n$, le rayon est donc supérieur à 1/4.

Le code :

```
from math import *
import numpy as np
import matplotlib.pyplot as plt
from random import *
def liste_a(n):
    l=[0 ,1]
    for i in range(2,n+1):
        s=0
        for j in range(1,i):
            s+=l[j]*l[i-j]
        l.append(s)
    return(l)
def a(n):
    l=liste_a(n)
    return(l[n])
print(a(10))
def S(n):
    l=liste_a(n)
    s=0
    p=4
    for k in range(1,n):
        s+=l[k]/p
        p=4*p
    return(s)

print(S(10))
def dessin(n):
    x=range(1,n)
    y=[S(xk) for xk in x]
    plt.plot(x,y)
    plt.show ()
dessin (200)
def liste_b(n):
    l=[0 ,1]
    for i in range(2,n+1):
        s=0
        for j in range(1,i):
            s+=l[j]*l[i-j]
        l.append(-i/(2*(i -1))*s)
    return(l)

def b(n):
    l=liste_b(n)
    return(l[n])

def dessin_b(n):
    x=range(0,n+1)
    y=liste_b(n)
    plt.plot(x,y)
    plt.show ()
```

Centrale 50 :

1.Que renvoie cette fonction ? Comment réduire le nombre de tests ?

```
def fonction(n):
    d=0
    if n==1:
        res=False
    else:
        d=2
        res=True
        while d<n:
            if n%d==0:
                res=False
            d+=1
    return res
```

2. Un nombre d est diviseur propre de n si d divise n et $d < n$.

Créer une fonction d'argument n qui renvoie la liste des diviseurs propres de n.

3. Créer une fonction d'argument n qui renvoie la somme des diviseurs propres de n.

Créer une fonction d'argument n qui renvoie le nombre de diviseurs propres de n.

4. Un nombre est AA si la somme de ses diviseurs propres est divisible par le nombre

de diviseurs propres .

Créer une fonction d'argument n qui renvoie un booléen indiquant si n est AA ou non.

5. Renvoyer tout les nombres AA inférieurs à 200 .

Renvoyer tout les nombres AA et premiers inférieurs à 200.

30

Sol :

1. La fonction renvoie vrai si et seulement si l'entier n supérieur à 1 est premier.

En effet on le divise par tous les entiers à partir de 2 .

Dès qu'un reste est nul, la variable booléenne res prend la valeur False.

Comme on la renvoie à la sortie, elle ne reste vraie que si et seulement si

aucune division n'est exacte.

Pour réduire le nombre de tests deux voies sont possibles. On arrête les diviseurs à $\sqrt{n}$.

On peut aussi sortir de la boucle dès que res est False (boucle conditionnelle) .

2) On peut réduire fortement la complexité , car en arrêtant à $\sqrt{n}$,

on a pu récupérer au fur et à mesure, les diviseurs manquant qui sont les $n//i$.

Attention à ne pas compter deux fois si le nombre est un carré parfait ex : 121...

Le code

```
def fonction2(n):
    d=0
    if n==1:
        res=False
    else:
        d=2
        res=True
        while d<= sqrt(n) and res:
            if n%d==0:
                res=False
            d+=1
    return res
def liste (n):#on peut faire plus econome avec racine de n...
    l=[1]
    for i in range(2,n):
        if n%i==0:
            l.append(i)
    return(l)
def somme (n):# mieux en modulaire de la variante econome...
    s=1
    for i in range(2,n):
        if n%i==0:
            s+=i
    return(s)
def nombre(n):# mieux en modulaire ou len()...
    nb=1
    for i in range(2,n):
        if n%i==0:
            nb+=1
    return(nb)
def AA(n):
    return(somme (n)%nombre(n)==0)
def InfAA (n):
    l=[1]
    for k in range(2,n+1):
        if AA(k):
            l.append(k)
    return(l)

def InfAAPrem(n):
    l=[1]
    for k in range(2,n+1):
        if AA(k) and fonction2(k):
            l.append(k)
    return(l)
```

Centrale 2 (16)

A. 1. Soit $f(x) = \sum_{n\in\mathbb{Z}} x^{n^2}$.

Déterminer le domaine D de définition de f.

2. Écrire une fonction Python de paramètre x et n et renvoyant $\sum_{k=-n}^{n} x^{k^2}$.

3. Déterminer n tel que $\sum_{k=-n}^{n} \left(\frac{1}{2}\right)^{k^2}$ approche $f\left(\frac{1}{2}\right)$ à 10^{-5} près.

4. Tracer sur un même graphe cette approximation de f et la fonction $g : x \mapsto \sqrt{\frac{\pi}{1-x}}$.

Que conjecturez-vous ?

5. Trouver un équivalent de f en 1 on donne $\left(\int_{-\infty}^{+\infty} e^{-\alpha t^2} dt = \sqrt{\frac{\pi}{\alpha}}\right)$.

Sol :

Le code :

```
import numpy as np
import scipy .integrate as si
import matplotlib.pyplot as plt
from math import *
def S(n):
    def g(x):
        u,s=1,1
        for k in range(1,n+1):
            s+=x**(k**2)
        return (2*s-1)
    return(g)
def krobar (f):
    x=np.arange(-1,1,0.01)
    y=[f(elt) for elt in x]
    plt.plot(x,y)
def h(x):
    return (sqrt(pi/(1-x)))
krobar(h)
krobar(S(200))
plt.show()
```

1. Le référentiel est très étrange.Il était peut-être précisé dans le texte distribué.

On dit qu'une série $\sum_{n\in\mathbb{Z}} u_n$ si et seulement si les deux séries numériques

$\sum_{n\in\mathbb{N}} u_n$ et $\sum_{n\in\mathbb{N}^*} u_{-n}$ convergent et, dans ce cas, la somme est la somme des sommes.

Ceci nous renvoie alors au programme de notre classe. Soit $f^+(x) = \sum_{n\in\mathbb{N}} x^{n^2}$.

Il s'agit d'une série entière lacunaire. Si $x = 1$ la série diverge grossièrement.

Donc $R \leqslant 1$. Si $|x| < 1, |x|^{n^2} \leqslant |x|^n$ donc la série converge. Le rayon vaut 1 .

Comme $f(0) = 1$ on a, pour $x \in]-1, 1[, f(x) = 2f^+(x) - 1$. En résumé $D =]-1, 1[$.

3. Pour que $\sum_{k=-n}^{n} \left(\frac{1}{2}\right)^{k^2}$ approche $f\left(\frac{1}{2}\right)$ à 10^{-5} près,

il faut et il suffit que le reste $\sum_{k=n+1}^{\infty} \left(\frac{1}{2}\right)^{k^2}$ soit inférieur à $\frac{1}{2}10^{-5}$.

Pour que ceci soit vrai, il suffit que

$\left(\frac{1}{2}\right)^n = \left(\frac{1}{2}\right)^{n+1} \frac{1}{1-\frac{1}{2}} = \sum_{k=n+1}^{\infty} \left(\frac{1}{2}\right)^k$ soit inférieur à $\frac{1}{2}10^{-5}$,

par l'inégalité du 1. En passant au logarithme on trouve $n = 18$.

Au voisinage de 1 les deux courbes sont tangentes.

5. On fixe $x \in]0,1[$. La fonction $\varphi : t \mapsto x^{(t^2)}$ est décroissante sur $[1, \infty[$

(la dérivée est $2x^{t^2} t \ln(x) < 0$, donc par comparaison série intégrale, $k \geqslant 3$

$$\varphi(k+1) \leqslant \int_k^{k+1} \varphi(t)dt \varphi(k)$$

pour $k \geqslant 1$

$$\int_k^{k+1} \varphi(t)dt \leqslant x^{(k^2)} \leqslant \int_{k-1}^{k} \varphi(t)dt$$

donc en sommant

$$\int_1^{+\infty} \varphi(t)dt \leqslant f^+(x) \leqslant \int_0^{+\infty} \varphi(t)dt.$$

Or

$$\int_0^{+\infty} \varphi(t)dt = \int_0^{+\infty} x^{(t^2)}dt = \int_0^{+\infty} e^{t^2 \ln x}dt = \frac{1}{2}\sqrt{\frac{\pi}{-\ln x}} \underset{x\mapsto 1}{\simeq} \frac{1}{2}\sqrt{\frac{\pi}{1-x}} \mapsto \infty.$$

Donc l'inégalité précédente s'écrit

$$\int_0^{+\infty} \varphi(t)dt + O(1) \leqslant f^+(x) \leqslant \frac{1}{2}\sqrt{\frac{\pi}{1-x}} + o\left(\sqrt{\frac{1}{1-x}}\right)$$

ou

$$\int_0^{+\infty} \varphi(t)dt + o\left(\sqrt{\frac{1}{1-x}}\right) \leqslant f^+(x) \leqslant \frac{1}{2}\sqrt{\frac{\pi}{1-x}} + o\left(\sqrt{\frac{1}{1-x}}\right)$$

et en passant au quotient, par encadrement

$$f^+(x) \sim \frac{1}{2}\sqrt{\frac{\pi}{1-x}}$$

Et par conséquent

$$f(x) \underset{1}{\sim} \sqrt{\frac{\pi}{1-x}}$$

Exo 22.

Soit f définie si $x \leqslant 0$ par $f(x) = 0$ et sinon $f(x) = e^{-\frac{1}{x}}$.

1. a. Représenter f sur $]-2, 2[$.

b. Montrer que pour tout entier n, il existe un polynôme P_n tel que

$$\forall x \in]0, \infty[, \quad f^{(n)}(x) = \frac{P_n(x)}{x^{2n}} f(x).$$

Calculer les cinq premiers polynômes.

c. Étudier la classe de la fonction f sur $\mathbb{R}$.

d. La fonction f est-elle DSE ?

2. On pose $g(x) = f(x)f(1-x)$ et $h(x) = \dfrac{1}{\alpha}\displaystyle\int_0^x g(t)dt$ où $\alpha = \displaystyle\int_0^1 g(t)dt$.

Indiquer la classe des fonctions g et h et les représenter sur un intervalle à déterminer.

Sol :

Soit f définie si $x \leqslant 0$ par $f(x) = 0$ et sinon $f(x) = e^{-\frac{1}{x}}$.

b. On le montre par récurrence, en trouvant, au passage la transition

$$P_{n+1}(x) = x^2 P'_{n+1}(x) + P_n(x)(1 - 2nx)$$

c. On peut montrer par récurrence (en prime) que le degré du polynôme est $n - 1$

et que le terme de plus bas degré est $(-1)^{n+1}$.

Ainsi la fonction est équivalente en 0 à droite à $\frac{(-1)^n}{x^{2n}} e^{-\frac{1}{x}}$ qui tend vers 0.

Il en va de même à gauche.

On rédige alors une récurrence à l'aide du théorème dangereux pour montrer que la fonction est de classe $\mathcal{C}_k$ en 0 pour tout k (voir le cours sur la fonction plate).

On en déduit que f est de classe $\mathcal{C}_\infty$ sur $\mathbb{R}$.

d. La fonction f n'est pas DSE, car si elle l'était son développement serait au moins

sur $]-\rho, \rho[(\rho > 0)$ le développement de Taylor, c'est-à-dire le développement nul.

Donc la fonction serait nul sur un petit intervalle ce qui est exclu.

2. On a posé $g(x) = f(x)f(1 - x)$ et $h(x) = \frac{1}{\alpha} \int_0^x g(t)dt$ où $\alpha = \int_0^1 g(t)dt$.

Par produit la fonction g est $\mathcal{C}_\infty$ sur $\mathbb{R}$. Si $x \in]0, 1[$, la fonction g est strictement positive,

et cie par produit, donc si l'intégrale qui définit α était nulle g serait nulle c'est exclu.

Ainsi $\alpha \neq 0$ et h est bien définie (primitive d'une fonction continue).

La fonction h est primitive d'une fonction $\mathcal{C}_\infty$ et donc elle l'est aussi.

Le code :

```
import numpy as np
import scipy .integrate as si
import matplotlib.pyplot as plt
from math import *
from numpy.polynomial import Polynomial
import scipy.integrate as integr
def f(x):
    if x>0:
        return (exp(-1/x))
    else:
        return(0)
def desss(a,b,f):
    C=np.arange(a,b,0.001)
    Y=[f(e) for e in C]
    plt.plot(C,Y)
#desss(-2,2,f)
#plt.show()
def P(n):
    if n==0:
        return(Polynomial([1]))
    else:
        return(P(n-1).deriv()*Polynomial([0,0,1])+P(n-1)*Polynomial([1,-2*n+2]))

for i in range(5):
    print(P(i))
def g(x):
    return(f(x)*f(1-x))
def h(x):
    alpha=integr.quad(g,0,1)[0]
    return(1/alpha*integr.quad(g,0,x)[0])
desss(-2,2,g)
desss(-2,2,h)
plt.show()
```

Centrale 2 (31)

$$\forall x \in \mathbb{R}, F(x) = \sum_{n=1}^{\infty} \frac{x}{n^2 + x^2}$$

1. Domaine de définition de F. Parité ?

2. La fonction F est-elle continue sur son domaine de définition ? Est-elle $\mathcal{C}1$?

3. Écrire une fonction Python qui prend en argument un réel x et qui renvoie $F(x)$ avec une précision à 10^{-5} près.

4. Tracer F sur $[-2, 2]$. On pose

$$\forall x \in \mathbb{R}, \quad G(x) = \int_0^{+\infty} \frac{\sin(xt)}{e^t - 1} dt$$

5. Ensemble de définition de G. Parité ?

6. Montrer que G est C_∞ sur son domaine de définition.

7. Tracer sur un même graphe F et G sur $[-2, 2]$. Conjecture ?

8. Donner le développement en série entière de $t \mapsto \frac{1}{1-t}$. En déduire celui de $t \mapsto \frac{1}{e^t - 1}$.

Sol :

$$\forall x \in \mathbb{R}, \quad F(x) = \sum_{n=1}^{\infty} \frac{x}{n^2 + x^2}$$

On posera $u_n(x) = \dfrac{x}{n^2 + x^2}$.

1. Pour x fixé non nul, la valeur absolue du terme général est un $O\left(\dfrac{1}{n^2}\right)$

(et même pour $x = 0$!), donc par comparaison la série est absolument convergente,

donc convergente et le domaine de définition est donc $\mathbb{R}$.

Il est symétrique par rapport à O et l'on vérifie que F est impaire.

2. Pour $0 < x \leqslant b$ on a $0 \leqslant u_n(x) \leqslant \dfrac{b}{n^2} = \alpha_n$ et cette inégalité reste vraie quand $x = 0$.

Comme $\sum \alpha_n$ converge, la série converge normalement sur tout segment $[0, b]$,

et comme u_n est continue (le dénominateur ne s'annule jamais) sur cet intervalle,

F est continue sur tout $[0, b]$, et donc sur $[0, \infty[$, puis par parité sur $\mathbb{R}$ entier.

On remarque que u_n est $\mathcal{C}_1$ sur $\mathbb{R}$ (fraction rationnelle sans pôle).

Pour tout x réel $u_n'(x) = \dfrac{(n - x)(n + x)}{(n^2 + x^2)^2}$.

Sur $[0, b]$, on a $\left|u_n'(x)\right| \leqslant \dfrac{n^2 + x^2}{n^4} \leqslant \dfrac{n^2 + b^2}{n^4}$ et comme précédemment

on montre que la série des dérivées converge normalement sur tout segment $[0, b]$.

En écrivant les trois hypothèses du théorème de tranfert, on montre que F est $\mathcal{C}_1$ sur $\mathbb{R}$.

3. Pour pouvoir écrire la fonction demandée, il faut évaluer l'erreur commise.

Cette erreur correspond au reste de la série.

Comme $y \mapsto \dfrac{x}{y^2 + x^2}$ est > 0 et décroissante ($x > 0$), on compare le reste à une intégrale.

$$\sum_{k=n+1}^{\infty} \frac{x}{k^2 + x^2} \leqslant \int_n^{+\infty} \frac{x}{y^2 + x^2} dy = \arctan\left(\frac{x}{n}\right)$$

4. Vu la question précédente.On pose

$$\forall x \in \mathbb{R}, \quad G(x) = \int_0^{+\infty} \frac{\sin(xt)}{e^t - 1} dt.$$

5. Fonction à intégrer continue sur $]0, \infty[$.

En 0 équivalente à $\frac{xt}{t}$ donc prolongeable par continuité.

À l'infini, $O\left(e^{-t}\right)$. Fonction impaire.

6. Il faut mettre en œuvre une récurrence basée sur le théorème de Leibniz :

$$\mathcal{P}_k : \quad G \in \mathcal{C}_k(\mathbb{R}), \text{ et, } \forall x \in \mathbb{R}, G^{(k)}(x) = \int_0^{+\infty} t^k \frac{\sin\left(xt + k\frac{\pi}{2}\right)}{e^t - 1} dt.$$

Au rang 0, c'est le théorème de continuité, $v(x,t) = \frac{\sin(xt)}{e^t - 1}$

est continue par morceaux par rapport à t continue par rapport à x.

$$\forall (x,t) \in [a,b] \times]0, \infty[|v(x,t)| = \frac{|\sin(xt)|}{e^t - 1} \leqslant \frac{t|x|}{e^t - 1} \leqslant \frac{tm}{e^t - 1}.$$

avec $m = \max(|a|, |b|)$ la fonction majorante étant intégrable sur $]0, \infty[$

pour les mêmes raisons que précédemment. Le théorème de continuité s'applique.

Si l'on suppose que $\mathcal{P}_k$ est vraie. Soit $w(x,t) = t^k \frac{\sin\left(xt + k\frac{\pi}{2}\right)}{e^t - 1}$.

Par hypothèse de récurrence, cette fonction est intégrable sur $]0, \infty[$ par rapport à t,

et elle admet une dérivée partielle par rapport à x de sorte que

$$\frac{\partial w}{\partial x}(x,t) = t^{k+1} \frac{\sin\left(xt + (k+1)\frac{\pi}{2}\right)}{e^t - 1}.$$

On vérifie la continuité par rapport à chacune des variables,

et la majoration est en tout point semblable à celle que nous venons d'effectuer.

Donc $\mathcal{P}_{k+1}$ est vraie. Ainsi G est C_∞ sur son domaine de définition

On n'a pas pu programmer la borne "np.inf", car l'exponentielle déclenchait une erreur.

On intègre donc jusqu'à 100 , ce qui devrait être suffisant, puisque $e^{100} \cong 0,3\,10^{44}$.

Le graphique obtenu :

Les codes :

```
def F(x):
    n=1
    s=0
    while atan(x/n)>10**(-5):
        s+=1/(n**2+x**2)
        n+=1
    s+=1/(n**2+x**2) #le dernier terme
    return(x*s)
def dessin(a,b,f):
    C = np. arange (a, b, 0.0 1)
    K=[ f ( c ) for c in C]
    plt.plot (C,K)
dessin(-2,2,F)
plt.show()

def G(x):
    def g(t):
        if t!=0:
            return(sin(x*t)/(exp(t)-1))
        else:
            return(x)
    return(integr.quad(g, 0, 100)[0])

dessin(-2,2,F)
dessin(-2,2,G)
plt.show()
```

Sur les positifs, les deux courbes coincident bien.

Elles diffèrent un peu de l'autre côté, ce qui n'est pas grave puisque

les deux fonctions sont impaires.

8. Avec un rayon valant $1\dfrac{1}{1-t} = \displaystyle\sum_{n=0}^{\infty} t^n$. Donc par substitution si $t > 0$,

$$\frac{1}{e^t - 1} = \frac{e^{-t}}{1 - e^{-t}} = \sum_{n=1}^{\infty} e^{-nt}$$

9. On écrit la variante du théorème de convergence dominée (x est un réel fixé).

Soit $S_n = \displaystyle\sum_{k=1}^{n} \sin(xt)e^{-nt}$.

La fonction S_n est continue par morceaux sur $]0,\infty[$.

Elle converge simplement sur cet intervalle vers la fonction $t \mapsto \dfrac{\sin(xt)}{e^t-1}$,

comme nous venons de le voir.

La limite est continue par morceaux.

$$|S_n| \leqslant \sum_{k=1}^{n} |\sin(xt)|e^{-nt} = |\sin(xt)| \sum_{k=1}^{n} e^{-nt} \leqslant |\sin(xt)| \sum_{k=1}^{\infty} e^{-nt} = \frac{|\sin(xt)|}{e^t-1}$$

la majorante étant intégrable sur $]0,\infty[$.

Par le théorème de convergence dominée, on peut permuter les lim et $\int$ si bien que

$$G(x) = \int_0^{+\infty} \lim_n S_n(t)dt = \lim_n \int_0^{+\infty} S_n(t)dt = \lim_n \int_0^{+\infty} \sum_{k=1}^{n} \sin(xt)e^{-nt}dt = \lim_n \sum_{k=1}^{n} \int_0^{+\infty} \sin(xt)e^{-nt}.$$

En passant par les complexes $\displaystyle\int_0^{+\infty} \sin(xt)e^{-nt}dt = \frac{x}{x^2+n^2}$ donc finalement $F(x) = G(x)$

.

Centrale 2 (34)

Soit $A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 4 \\ 1 & -1 & 3 \end{bmatrix}$. On note C_i la ième colonne de A.

1. Montrer de la manière la plus simple possible la réponse à la question suivante :

La famille (C_1, C_2, C_3) est-elle une base de $\mathbb{R}^3$?

2. Trouver une famille orthonormée qui orthonormalise la famille (C_1, C_2, C_3).

On la notera (U, V, W).

3. Soit H tel que $H^T = (\alpha, \beta) \in \mathcal{M}_{1,2}(\mathbb{R})$ soit non nul.

Trouver $S \in SO_2(\mathbb{R})$ telle que $SH = t\ (\gamma, 0)$. On exprimera γ en fonction de α et β.

4. Trouver Q orthogonale, telle que QA soit triangulaire supérieure.

Sol :

Soit $A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 4 \\ 1 & -1 & 3 \end{bmatrix}$. On note C_i la ième colonne de A.

1. Par exemple :

La famille (C_1, C_2, C_3) est libre (determinant 6) .

3. On prend $S = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$, et par calcul, $SH = \begin{bmatrix} \cos(\theta)\alpha - \sin(\theta)\beta \\ \sin(\theta)\alpha + \cos(\theta)\beta \end{bmatrix}$.

On prend un vecteur unitaire de $H^\perp$ que l'on peut noter $(\sin\theta, \cos\theta)$.

Il répondra à la question.

Plus explicitement, si $\alpha \neq 0$ on prend $\theta = -\arctan\left(\frac{\beta}{\alpha}\right)$.

Ceci donne

$$\gamma = \alpha \frac{1}{\sqrt{1+\frac{\beta^2}{\alpha^2}}} + \beta^2\alpha^{-1}\frac{1}{\sqrt{1+\frac{\beta^2}{\alpha^2}}} = \sqrt{\alpha^2+\beta^2}\ .$$

4. La matrice de passage de la base canonique à la base (C_1, C_2, C_3) est A,

celle de la base (C_1, C_2, C_3) à la base orthonormée (U, V, W) est T (triangulaire inversible).

Directement on passe de la base canonique à la base (U, V, W) par une matrice Q

(orthogonale, car de BON vers BON).

Ainsi pour tout vecteur X dans la base canonique de coordonnées Y

dans la base (U, V, W) on a $X = QY = TAY$.

Comme ceci est vrai pour tout Y on a $Q = TA$ où $T^{-1} = Q^T A$ et comme l'inverse

d'une matrice triangulaire est encore triangulaire, on a répondu à la question [4].

Le code :

```
import numpy as np
import scipy .integrate as si
import matplotlib.pyplot as plt
from math import *
from numpy.polynomial import Polynomial
import scipy.integrate as integr
A=np.zeros((3,3))
for i in range(3):
    A[0,i]=1
    A[1,i]=i+2
    A[2,i]=(-1)**i
A[2,2]=3
#print(np.linalg.det(A))
def C(j):
    M=np.zeros((3,1))
    for i in range(3):
        M[i,0]=A[i,j]
    return(M)
def ps(X,Y):
    s=0
    for k in range(3):
        s+=X[k,0]*Y[k,0]
    return(s)
U=1/sqrt(ps(C(0),C(0)))*C(0)
VV=C(1)-ps(C(1),U)*U
V=1/sqrt(ps(VV,VV))*VV
WW=C(2)-ps(C(2),U)*U-ps(C(2),V)*V
W=1/sqrt(ps(WW,WW))*WW
print(ps(U,V),ps(U,W),ps(W,V),ps(U,U),ps(V,V),ps(W,W))
```

Rappel : Code QR

```
import numpy as np
np.set_printoptions(precision = 3) #pour un affichage raisonnable

from math import sqrt

def pscal(u, v):
    n = len(u)
    s = 0
    for k in range(n):
        s += u[k]*v[k]
    return s

def norme(u):
    return sqrt(pscal(u, u))

def QR(A): #A est une matrice carrée inversible
    n = len(A)
    e = np.transpose(A);
    R = np.zeros((n, n))
    epsilon = np.zeros((n, n))
    for j in range(n):
        v = e[j]
        for k in range(j):
            R[k, j] = pscal(epsilon[k], e[j])
            v = v - R[k, j]*epsilon[k]
        R[j, j] = norme(v)
        epsilon[j] = v/R[j, j]
    return np.transpose(epsilon), R

L = [[1, 2, 3], [4, 5, 6], [8, 9, 7]]
A = np.array(L)
Q, R = QR(A)
print('A :')
print(A)
print()
print('Q :')
print(Q)
print()
print('tQQ :')
print(np.dot(np.transpose(Q), Q))
print()
print('R :')
print(R)
print()
print('QR :')
print(np.dot(Q, R))
```

992. PYTHON Pour tout entier $n \in \mathbb{N}^*$, on note $\omega_n = e^{2i\pi/n}$ et

$F_n \in M_n(\mathbb{C})$ la matrice $\left(\omega_n^{(k-1)(l-1)}\right)_{1\leqslant k,l\leqslant n}$.

a) Ecrire une fonction Python qui prend un entier n en argument et renvoie F_n.

Afficher plusieurs matrices F_n.

b) Calculer avec Python le produit $F_n\overline{F_n}$.

c) Écrire une fonction Python qui prend un entier n en argument et renvoie F_n^{-1}.

Que peut-on conjecturer ?

d) Ecrire une fonction Python qui prend deux entiers n et k en arguments et renvoie F_n^k.

Que peut-on conjecturer ?

e) Démontrer les conjectures précédentes.

f) Déterminer les valeurs propres de F_n. Cette matrice est-elle diagonalisable ?

Sol : Exo utile... Voilà les codes :

Attention à la manipulation des complexes, donc attention aux dtypes...
Voir code disk dur centrale 992

```
import numpy as np
import numpy.linalg as alg
from numpy.polynomial import Polynomial
from math import *

#pour un affichage raisonnable
#on peut limiter le nombre de décimales
np.set_printoptions(precision=2)
# et ici ce n'est pas du luxe...
def F(n):#je ne m'occupe pas de diminuer la complexité
    omega=np.exp(2*1j*np.pi/n)
    F=np.zeros((n,n),dtype=complex)# attention aux types...
    for i in range(n):
        for k in range(n):
            F[i,k]=np.exp(2*1j*np.pi*i*k/n)
    return F/sqrt(n)# la racine carrée facilite la lecture
#for s in range(2,5):
    print(np.dot(F(s),np.conj(F(s))))
def FI(n):
    G=alg.matrix_power(F(n),-1)
    return G
def FP(n,k):
    HH=F(n)
    H=np.identity(n,dtype=complex)#ou np.eye
    if k==0:
        return H
    elif k>0:
        L=H
        for s in range(k):
            L=np.dot(L,HH)#ou méthode dot(voir Centrale.py)
        return L
    else:
        L=H
        for s in range(k):
            L=np.dot(L,HH)
        return alg.matrix_power(L,-1)
#alg.eigvals(F(5))
```

1) Voir code .

2) (np.dot(F(3),np.conj(F(3)))), on conjecture que c'est $n.I_n$.

conj comme conjugué !

Il nous vient l'idée du $\sqrt{n}$ qui améliore bcp la lecture...

3) Voir code. Il semble que $F_n^{-1} = \frac{1}{n}\overline{F_n}$. Je note $G_n = \frac{1}{\sqrt{n}}F_n$...

4) Voir code. Fonction FP.

On conjecture que $G_n^4 = I_n$, d'où les puissances de F_n.

Rq : On conjecture aussi $G_n^2 = \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & S_{n-1} \end{array}\right)$.

Avec S_{n-1} qui n'a des 1 que sur la "mauvaise" diagonale.

e) Calculs de première année :

Il faut poser les produits matriciels qui finiront par confirmer que $G_n\overline{G_n} = I_n$ et $G_n^2 = ...$

L'essentiel dans ce calcul est de maitriser que $\sum_{j=1}^{n}(\omega_n^s)^j$,

vaut n quand s multiple de n et 0 sinon.

Ceci se démontre facilement avec des sommes géométriques. (attention à la raison).

Le conjugué dans les calculs revient à multiplier l'exposant de ω_n par -1.

Pour $G_n^4 = (G_n^2)^2$, il ne faut pas passer par les coefficients,

mais regarder l'application linéaire associée à G_n^2...

f) Oui, c'est diagonalisable. Mais ça ne vient pas du thm spectral , car ici symétrique mais $\mathbb{C}$.

On un polynôme annulateur scindé simple pour G_n, $X^4 - 1$.

Les valeurs propres sont donc incluses dans : $\{1, -1, i, -i\}$, pour G_n.

Ne pas oublier de remultiplier par $\sqrt{n}$.

Les valeurs propres sont dans la dernière ligne du code.

Le détail mathématique ne me semble pas raisonnable...

993. PYTHON Soit n un entier naturel. On considère la matrice $A_{n+1} \in \mathcal{M}_{n+1}(\mathbb{R})$ telle que $a_{j-1,j} = j-1, a_{j+1,j} = n+1-j$ pour tout j, et dont tous les autres coefficients sont nuls.

a) Écrire une fonction Python qui prend un entier n en argument et renvoie A_{n+1}

b) Déterminer avec Python les valeurs propres de A_{n+1}. Que peut-on conjecturer ?

c) Soit u l'endomorphisme de $\mathbb{R}_n[X]$ canoniquement associé à la matrice A_{n+1}.

Montrer qu'il existe un polynome Q ne dépendant pas de n tel que,

pour tout $P \in \mathbb{R}_n[X], u(P) = QP' + nXP$.

d) En déduire les éléments propres de u.

e) La matrice A_{n+1} est-elle diagonalisable ?

Code disk dur centrale 993

```
import numpy as np
import numpy.linalg as alg
from numpy.polynomial import Polynomial
from math import *
#pour un affichage raisonnable
#on peut limiter le nombre de décimales
np.set_printoptions(precision=2)

def A(n):# attention aux index...
    B=np.zeros((n+1,n+1))
    for j in range (0,n+1):
        B[j-1,j]=j
    for j in range (0,n):
        B[j+1,j]=n-j
    return B
#alg.eigvals(A(s))
```

b) On peut conjecturer à la lecture des résulats Python que les valeurs propres

sont 2 à 2 distinctes et au nombre de $n+1$ et valent $2k-n$ pour $k \in \{0, ..., n\}$.

Bref $\{n, n-2, n-4, ..., 2-n, -n\}$.

c) En regardant l'image de la base canonique, on conjecture que $Q = 1 - X^2$.

On regarde l'endomorphisme ainsi crée et ça marche pour toute la base canonique.

d) On gère le problème comme une EDL homogène du premier ordre (première année).

Soit $\lambda \in \{n, n-2, n-4, ..., 2-n, -n\}$, l'EDL est : $P' + \dfrac{nX - \lambda}{1 - X^2} P = 0$.

On intègre ...$P = \mu(X^2-1)^{n/2}.\left(\dfrac{X+1}{X-1}\right)^{(2k-n)/2} = \mu(X+1)^{n-k}(X-1)^k$.

Pour $k : 0 \to n$, on a posé $\lambda = n - 2k$.

Elles sont toutes validées, les peureux peuvent calculer leurs images par u.

Ca sort tout seul, Anthony avait le bon résultat.

On trouve bien des polynômes de $\mathbb{R}_n[X]$! ! ! Penser à simplifier les quotients.

Penser à vérifier les degrés, le cas n pair amène à la non-injectivité.

e) Bref, ça diagonalise.

994. PYTHON On considère la matrice $A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \end{pmatrix}$.

a) Calculer le polynôme caractéristique de A. La matrice A est-elle diagonalisable ?

Préciser le module de ses valeurs propres.

b) On considère la suite $(u_n)_n$ définie par $u_0 = 2, u_1 = 3, u_2 = 8, u_3 = 4, v_4 = 11$ et,

pour tout $n \in \mathbb{N}, u_{n+5} = \dfrac{1}{5}(u_n + u_{n+1} + u_{n+2} + u_{n+3} + u_{n+4})$.

i) Écrire une fonction Python qui prend en argument un entier n et renvoie

les $n + 1$ premiers termes de cette suite.

ii) Afficher sur un graphique les 25 premiers termes de la suite.

Que peut-on conjecturer concernant la convergence de u_n?

iii) Réécrire la relation de récurrence a l'aide de la matrice A.

iv) Montrer que la suite de matrice $(A^n)_n$ converge vers la matrice d'un projecteur

dont on précisera les éléments caractéristiques.

v) Trouver un vecteur non nul X tel que $A^T X = X$.

Sol : Mail 13 mai 2023.

(a) On définit la matrice A.

Puis tous les codes...

```
import matplotlib. pyplot as plt
import numpy as np
import numpy. linalg as alg
A=np . zeros ((5,5))
for i in range (4):
     A[i, i+1]=1
for i in range(5):
A[4, i]=1/5

chiA=np.poly(A)# feuille Centrale

SPA=alg.eigvals(A)# feuille Centrale

moduvp=[np.abs(vp) for vp in SPA]# feuille Centrale

#comment améliorer la lisibilité de l'écran?

def u(n):
    Lu=[2,3,8,4,11]
    for i in range (5, n+1):
        ui=0
        for k in range (i-5,i):
            ui+=Lu[k]
        Lu.append (ui / 5)
    return Lu

plt.plot (u(24),'o')# feuille Centrale

P=alg.eig(A)[1]# feuille Centrale les ...
pdiag=np.zeros((5,5))
pdiag [0,0]=1
pro=np.dot(np.dot(P,pdiag),alg.inv(P))# feuille Centrale

(pro*15) . real# feuille Centrale

B=np.eye(5)# feuille Centrale

for i in range(1,5):
    B+=(i+1)*alg.matrix_power(A,i)# feuille Centrale
```

i. Matrice compagnon, donc ... On connait le résultat...

$$\chi_A = X^5 - \frac{X^4 + X^3 + X^2 + X + 1}{5}$$

Attention ! Différent du module Polynomial, les coefficients du polynôme sont ici donnés selon les puissances décroissantes.

Le polynôme caractéristique est unitaire, le premier coefficient est donc 1.

a) Dur de factoriser ce polynôme à la main (on remarque que 1 est racine).

On constate que 1 est la seule vp réelle de A, les autres sont complexes et 2 à 2 conjuguées.

La matrice A n'est donc pas diagonalisable dans $\mathbb{R}$ mais elle est bien diagonalisable dans $\mathbb{C}$

(puisque $A \in \mathcal{M}_5(\mathbb{C})$ possède 5 vp distinctes).

On voit par ailleurs que les modules des vp complexes sont < 1 .

(b) On calcule les u_n de manière simple en réécrivant la relation de récurrence sous une forme plus simple (facile à coder).

$$\forall i \geqslant 5, \quad u_i = \frac{u_{i-5} + u_{i-4} + u_{i-3} + u_{i-2} + u_{i-1}}{5}$$

En traçant la suite $(u_n)_{n\in\mathbb{N}}$, on voit qu'elle converge (vers une limite inférieure à 6).

La relation de récurrence peut aussi s'écrire

$$\forall n \in \mathbb{N}, \quad X_{n+1} = AX_n \quad \text{avec} \quad X_n = \begin{pmatrix} u_n \\ u_{n+1} \\ u_{n+2} \\ u_{n+3} \\ u_{n+4} \end{pmatrix}$$

ce qui nous donne

$$\forall n \in \mathbb{N}, \quad X_n = A^n X_0.$$

On a constaté plus haut qu'il existait une matrice $P \in GL_5(\mathbb{C})$ telle que

$$\Delta = P^{-1}AP = \text{Diag}(1, \alpha, \bar{\alpha}, \beta, \bar{\beta})$$

avec $|\alpha| < 1$ et $|\beta| < 1$.

On en déduit que

$$P^{-1}A^nP = \left(P^{-1}AP\right)^n = \text{Diag}\left(1, \alpha^n, \bar{\alpha}^n, \beta^n, \bar{\beta}^n\right)$$

tend vers $\text{Diag}(1, 0, 0, 0, 0)$, qui est une matrice de projection (de rang 1).

Comme l'application $\left[M \mapsto \text{PMP}^{-1}\right]$ est cie (linéaire endimension finie),

on déduit du théorème de composition que

$$\Pi = \lim_{n\to+\infty} A^n = P\,\mathrm{Diag}(1,0,0,0,0)P^{-1}$$

qui est aussi une matrice de projection.

Comme

$$\chi_A = \frac{(X-1)\left(5X^4+4X^3+3X^2+2X+1\right)}{5}$$

on déduit du théorème de ""décomposition des noyaux "" (et du Théorème de Cayley-Hamilton) que

$$\mathbb{R}^5 = \mathrm{Ker}\,(A-I_5) \oplus \mathrm{Ker}\left(5A^4+4A^3+3A^2+2A+I_5\right)$$

et on conjecture que Π est la projection sur $\mathrm{Ker}\,(A-I_5)$ parallèlement au sous-espace $\mathrm{Ker}\left(5A^4+4A^3+3A^2+2A+I_5\right)$. (La formule du changement de base nous assure que Π est une projection sur la droite propre associée à la valeur propre 1 .)

Concrètement Π.

Pour clarifier , on vire les $i\mathbb{R}$ (erreurs d'arrondi) : car Π est réelle) et après analyse...

on voit que :

Il s'agit donc de la projection sur la droite

$$\mathbb{R}\cdot\begin{pmatrix}1\\1\\1\\1\\1\end{pmatrix} = \mathrm{Ker}\,(A-I_5)$$

parallèlement à l'hyperplan d'équation

$$x_1+2x_2+3x_3+4x_4+5x_5 = 0.$$

On vérifie. On constate que $B = 15\Pi$.

En passant à la limite dans $A^{n+1} = A^n$. On obtient $\Pi = \Pi A$, et donc $A^\top\Pi^\top = \Pi^\top$.

Chaque colonne de $\Pi^\top$ nous donne donc un vecteur X tel que $A^\top X = X$ et,

d'après ce qui précède, chaque colonne de Π^{T} est égale à

$$\begin{pmatrix}1\\2\\3\\4\\5\end{pmatrix}$$

1009. PYTHON. On pose, pour $(x,y)\in\mathbb{R}^2, N(x,y)=\displaystyle\int_0^1 |x-ty|\ \mathrm{dt}$.

a) Vérifier que N est bien une norme sur $\mathbb{R}^2$.

b) Tracer la courbe de la fonction $x\mapsto N(x,1)$ pour $x\in[-1/2,3/2]$ avec Python.

c) Calculer $N(x,1)$ pour tout $x\in\mathbb{R}$.

d) En déduire la valeur de $N(x,y)$ pour tout $(x,y)\in\mathbb{R}^2$.

e) Soit C le cercle unité d'équation $x^2+y^2=1$. Tracer la courbe de $x\mapsto N\left(x,\sqrt{1-x^2}\right)$ pour $x\in[-1,1]$ avec Python. Estimer les valeurs de sup $N(C)$ et inf $N(C)$ a l'aide du tracé.

f) Déterminer la valeur exacte de sup $N(C)$.

Sol :

```
from math import *
from scipy import *
import scipy.integrate as integr
from matplotlib.pyplot import *
import matplotlib.pyplot as plt
from numpy import *

T = np.linspace(-0.5, 1.5, 300)
TT= np.linspace(-1.0, 1.0, 1300)

def N(x,y):
    def f(t):
        return abs(x-y*t)
    return integr.quad(f,0,1)[0]
#G= [N(x,1) for x in T]
#plt.plot(T,G)
#plt.show()
GG= [N(x,sqrt(1-x*x)) for x in TT]
plt.plot(TT,GG)
plt.show()
```

Les maths :

a) La preuve de la norme est clasique, on fait attention à ne rien oublier.

Pour l'inégalité triangulaire : $\displaystyle\int_0^1 |(x+x')-t(y+y')|dt=\int_0^1 |(x-ty)+(x'-ty')|dt\leqslant$.

$N(x,y)+N(x',y')$.

Pour c) $\displaystyle\int_0^1 |x-y|dt$, il faut séparer les cas en x pour virer la valeur absolue.

De plus si $x\in[0,1]$, il faut séparer le segment d'intégration en 2...

Résultats après calculs faciles mais pas drôles :

- $x\geqslant 1$, $0.5*(2x-1)$.

- $x\leqslant 0$, $0.5*(1-2x)$.

- $x\in[0,1]$, $x^2-x+0.5$.

Rq : parfaitement cohérent avec le graphe Python.

d) Si $y=0$, on a $|x|$.

Sinon, $|y|.N\left(\dfrac{x}{y},1\right)$, et on applique les différents cas précédents...

e) Voir graphe Python, il semblerait que le sup vaille 1.115 et l'inf 0.222.

f) D'abord on peut se restreindre au cas $y\geqslant 0$, car $N(x,y)=N(-x,-y)$, pb symétrique...

On pourrait tout calculer au d) voir que la fonction est $\mathcal{C}^1$ etc...

Calculs lourds , fastidieux , points crtiques etc...

Soyons malins , il est clair que le max est atteint avec x entre -1 et -0.8.

Je ne calcule le d) que dans ce cas, $y>0$, $x<0$...le deuxième du c...

Il vient : le max de $\dfrac{-2x+\sqrt{1-x^2}}{2}$.

On dérive, tableau de variations, on otpimise en $x=-\sqrt{\dfrac{4}{5}}$, cohérent !

Le max, on remplace, et c'est cohérent...

1002. (2021) PYTHON . Soient $x_0,\dots,x_{n-1}$ des réels distincts, (ordre croissant).

$L_0,\dots,L_{n-1}$ les polynômes de Lagrange associés.

Soient $k\in[\![0,n-1]\!]$, $A_k=L_0+\cdots+L_k$, $P_k=A_k-\displaystyle\sum_{i\neq k}\frac{A_k'(x_i)}{\Lambda_i'(x_i)}\Lambda_i$. où,

pour $i \in [\![0, n-1]\!], \Lambda_i = (X - x_i)(X - x_k) \prod_{j \neq i, j \neq k} (X - x_j)^2$.

a) Écrire une fonction Python Lambda $(i, \mathrm{k}, \mathrm{X})$ où $X = (x_0, \ldots, x_{n-1})$ et qui renvoie Λ_i.

b) Vérifier que P est bien défini.

Calculer $P_k(x_j)$ pour tout $j \in [\![0, n-1]\!]$ et $P_k'(x_j)$ pour tout $j \in [\![0, n-1]\!]$ distinct de k.

Montrer que P_k est l'unique polynôme de degré $2n-2$ vérifiant ces conditions.

c) En étudiant les racines de P_k' et les variations de P_k, mq, pour $t \leqslant x_k$, on a $P_k(t) \geqslant 1$.

Sol : Ici, seul le module Polynomial a de l'importance, niveau info ...

a)

```
from numpy.polynomial import *
import matplotlib.pyplot as plt
def monome(a):# on declare X-a
    return Polynomial([-a,1])
def Lagrange(L,k): # L= [x0,x1,...,xn-1], les abcisses

    P = Polynomial([1])
    for i in range(0,len(L)):
        if i != k:# on enlève l'index concerné
            P *= monome(L[i]) / (L[k] - L[i])
    return P # attention aux indices...
def Lambd(i,k,L):#attention à i,k...
    P = Polynomial([1])
    if i!=k:
        for j in range(len(L)):
            P=P*monome(L[j])
        for j in range(len(L)):
            if j != k and j !=i:
                P=P*monome(L[j])
    else:
        for j in range(len(L)):
            P=P*monome(L[j])
        P=P*P


    return P
```

b) Pour $i \neq k$, x_i est racine simple de Λ_i donc pas racine de sa dérivée.

P_k est bien défini.

Pour $P_k\left(x_j\right)$, Λ_i s'annule pour toutes nos abcisses.

Donc $P_k\left(x_j\right) = 1$ si $j \leqslant k$, et 0 sinon,

en utilisant nos connaissances sur les polynômes de Lagrange.

Pour $P_k'\left(x_j\right)$, il faut se méfier des confusions d'indices...

La plupart des $\Lambda_i'(x_j)$ sont nuls grâce aux racines doubles.

Pour $i = j$ on a un téléscopage, il reste $P_k'\left(x_j\right) = A_k'(x_j) - A_k'(x_j) = 0$.

Rq $j \neq k$.

Maintenant on regarde avec soin les degrés.

Λ_i est de degré $2n-2$, donc P_k de degré au plus $2n-2$.

Donc la dérivée de degré au plus $2n-3$.

Soit Q_k un autre client, on regarde $P_k - Q_k$.

On a déjà récolté $n-1$ racines de cette dérivée juste avant,

mais en appliquant Rolle sur chaque $[x_j, x_{j+1}]$.

On en récolte $n-1$ autres, trop de racines, $(P_k - Q_k)' = 0_{\mathbb{R}[X]}$.

Il est donc constant, bref nul car il a des racines. Unicité.

c) C'est long, je donne les idées, pour des raisons de degrés,

les racines de P_k' sont simples(*).

P_k' n'est pas nul car P_k non constant.

Donc (*) sont des extrémas locaux.

On fait un tableau de signe, avant x_k.

On va de 1 en 1, en étant toujours du même côté, (cf extrémas locaux).

Donc , il reste à savoir lequel, la réponse est en $-\infty$.

La limite y est $+\infty$.

Ça vient du fait que le coeff en X^{2n-2} est $-\sum_{i\neq k}\frac{A'_k(x_i)}{\Lambda'_i(x_i)} > 0$.

Pour le prouver il faut utiliser (comme plus haut aussi).

$\left(\prod_1^s P_i\right)' = \sum_{k=1}^{s}\left(\left(\prod_{i\neq k}^{s} P_i\right)(P_k)'\right)$. Stop.

Une autre sol

b) Dans cet exercice k est fixé. Le polynôme Λ_i a par définition x_i comme racine simple.

Son polynôme dérivé ne s'annule pas sur cette valeur, et par conséquent, P est bien défini.

On remarque que le polynôme Λ_i a tous les points de la subdivision comme racine,

simplement, certaines sont doubles.

- Soit $j \in [\![0, k]\!]$ fixé.

$$A(x_j) = L_j(x_j) = 1$$
$$\Lambda_i(x_j) = 0$$

donc

$$P(x_j) = 1$$

Si, de plus, j est différent de k et de i on a

$$\Lambda'_i(x_j) = 0$$

alors que

$$\Lambda'_i(x_i) \neq 0$$

donc

$$P'(x_j) = A'(x_j) - \frac{A'(x_j)}{\Lambda'_j(x_j)}\Lambda'_j(x_j) = 0$$

- Soit $j \in [\![k+1, n-1]\!]$.

$$A(x_j) = 0$$

donc

$$P(x_j) = 0$$

et le calcul est identique pour la dérivée.

Ainsi P est un polynôme de degré $2n-2$ (le premier terme est de degré $n-1$) qui vaut 1 sur k valeurs, 0 sur $n-k-1$ valeurs, et de dérivée nulle sur $n-1$ valeurs.

L'application

$$P \mapsto \left(P(x_0), \dots, P(x_{n-1}), P'(x_0), \dots, P'(x_k), \dots, P'(x_{n-1})\right.$$

est linéaire de $\mathbb{R}_{2n-2}[X]$ dans $\mathbb{R}^{2n-1}$.

Son noyau est constitué par les polynômes dont tous les éléments de la subdivision (à part k qui correspond à une racine simple) sont racines doubles, donc par le polynôme nul, ou des polynômes de degré au moins $n+n-1 = 2n-1$ qui ne sont pas dans l'ensemble de départ. L'application est donc injective, et par égalité des dimensions, il s'agit d'un isomorphisme. Le polynôme qui est l'antécédent du $2n-1$ uplet $(1, 1, 1 \dots, 1, 0, \dots, 0)$ est donc unique.

c) Nous avons trouvé $n-1$ ravines de P'.

De plus le théorème de Rolle appliqué à P entre tous les points de la subdivision où il prend des valeurs égales, fournit de nouveau $n-2$ annulations différentes (on rédige Rolle, et l'annulation se fait sur l'ouvert). En tout $2n-3$ racines donc exactement le degré de P' qui est donc scindé simple sur $\mathbb{R}$. Il a donc un signe qui alterne entre les points de la subdivision. Comme P a un terme dominant pair, en $-\infty$ il est positif, donc P' commence à décroître jusqu'à 1 , puis il recroît à nouveau etc. Et ceci est vrai pour tous les intervalles de la subdivision qui sont de rang inferieur ou égal à k.

En résumé, pour $t \leqslant x_k$, on a $P(t) \geqslant 1$.

1006. PYTHON. Soit $n \in \mathbb{N}^*$.

On s'intéresse au problème $FL(n)$ suivant : étant donnés $a_0, \dots, a_{n-1}, \lambda_0, \dots, \lambda_{n-1} \in \mathbb{C}$, existe-t-il une matrice dont les coefficients diagonaux sont $a_0, \dots, a_{n-1}$ et les valeurs propres sont $\lambda_0, \dots, \lambda_{n-1}$?

a) Écrire une fonction Python prenant en entrée des complexes $a_0, \dots, a_{n-1}, \mu_0, \dots, \mu_{n-1}, \alpha$ et renvoyant la matrice

$$\begin{pmatrix} a_0 & 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{n-1} & 1 \\ \mu_0 & \cdots & \cdots & \mu_{n-1} & \alpha \end{pmatrix}.$$

b) Montrer que le problème $FL(2)$ admet une solution si et seulement si $a_0 + a_1 = \lambda_0 + \lambda_1$.

Préciser une solution sous cette condition.

c) $FL(3)$ admet-il une solution pour $a_0 = 1, a_1 = -1, a_2 = 0, \lambda_0 = 1, \lambda_1 = 2$ et $\lambda_2 = -3$?

d) Soient $a_0, \dots, a_{n-1} \in \mathbb{C}$. On pose , pour $k \in [\![0, n-1]\!], P_k = \prod_{i=0}^{k-1} (X - a_i)$.

i) Justifier que la famille $(P_0, \dots, P_n)$ est une base de $\mathbb{R}_n[X]$

puis qu'il existe $\mu_0, \dots, \mu_{n-1}$ tels que $\prod_{k=0}^{n-1} (X - \lambda_k) = P_n - \sum_{k=0}^{n-1} \mu_k P_k$.

ii) Écrire une fonction Python prenant $a_0, \dots, a_{n-1}$ et renvoyant la liste $[P_0, \dots, P_n]$.

iii) Écrire une fonction Python prenant $a_0, \dots, a_{n-1}, \lambda_0, \dots, \lambda_{n-1}$ et renvoyant les μ_k.

Conclure.

Sol : 30 mn MDR .

a) On crée une np de taille n remplie de zéros avec "eye",

mais attention il y a un piège à c...Voir 992 (2019).

Puis une boucle pour les 1 .

Puis deux autres pour ...

```
def mat (Mu ,Ad ,alpha ,n):
    N=np. zeros ((n+1,n+1),...992)
    for i in range (n):
        N[i,i]= Ad[i]
        N[n,i]= Mu[i]
        N[i,i +1]=1
    N[n-1,n]=1
    N[n,n]= alpha
    return (N)
#comparer au mien
def listeP (Ad ,n):
    pi =Polynomial ([1])
    l=[ pi]
    for i in range (n):
        pi=pi* Polynomial ([-Ad[i] ,1])
        l. append (pi)
    return (l)
```

```
def prod_l (Lambda ,n):
    pi =Polynomial ([1])
    for i in range (n):
        pi=pi* Polynomial ([- Lambda [i] ,1])
    return (pi)
def passage (Ad ,n):
    P=np.zeros ((n+1,n +1))
    fP =listeP (Ad ,n)
    j=0
    while (j<n+1):
        for x in fP:
            for i in range (j):
                P[i,j]=(x. coef)[i]
                j+=1
    return (P)
```

b) La condition est nécessaire pour des raisons de traces.

On cherche une solution avec la "bonne" diagonale,

le polynôme caractéristique $X^2 - (a_0 + a_1)X + a_0a_1 - \alpha\beta$.

Si il s'annule en λ_0 c'est suffisant car après trigonalisation,

la trace donnera l'autre annulation.

On remplace $\alpha\beta = \lambda_0^2 - (a_0 + a_1)\lambda_0 + a_0a_1$.

On peut imposer $\beta = 1$ pour commencer à respecter a).

c) On impose la "bonne" diagonale, on cherche une solution sur le modèle de a).

On écrit le poly caract ss développer, on force l'annulation en 1,

ça donne $\mu_0 = 0$, puis on force en 2, ça donne $\beta = 6$.

La troisième racine est obligatoire par la trace après trigonalisation(*).

d) i) Bon cardinal et échelonnée en degrés.

$P_n - \prod_{k=0}^{n-1} (X - \lambda_k) \in \mathbb{R}_{n-1}[X]$ donc de décomposition unique sur notre base.

Rq personelle qui va prendre de l'importance dans la conclusion :

Comme on va imposer $\sum_0^{n-1} a_i = \sum_0^{n-1} \lambda_i$.

On est dans $\mathbb{R}_{n-2}[X](***)$.

ii) Un p'tit coup de module polynomial, on remplit la liste qui sera retournée

au fur et à mesure et en multipliant par le dernier monôme.

On commence à faire attention à ne pas décaler les indices...

iii) Mal rédigé (selon moi), on cherche une solution de type a).

Il y bcp de travail...

Je pense que l'examinateur veut le plan .

Je vais essayer de suivre l'énoncé...

3 pbs se posent : Pourquoi les μ_k sont les bons ?

Comment est la solution qui va avec ?

Comment les récupérer par Python ?

Ok , c'est la dernière question, ...

Je guide, un p'tit calcul de déterminant pour commencer.

les décalages d'indices sont agaçants à gérer.

Celui de : $\begin{pmatrix} X-a_0 & -1 & ... & 0 \\ 0 & X-a_1 & \ddots & 0 \\ \vdots & & \ddots & -1 \\ -\mu_0 & ... & -\mu_{n-2} & X-a_{n-1} \end{pmatrix}$.

On développe suivant la dernière ligne, méthode crayons...

Le miracle a lieu : $\prod_0^{n-1}(X-a_i) - \left(\sum_0^{n-2}\mu_i P_i\right)$. Well done .

On constate avec un certain plaisir, que 2 choses sont éclaircies .

En ouvrant bien les yeux, la matrice $\begin{pmatrix} X-a_0 & -1 & ... & 0 \\ 0 & X-a_1 & \ddots & 0 \\ \vdots & & \ddots & -1 \\ -\mu_0 & ... & -\mu_{n-2} & X-a_{n-1} \end{pmatrix}$.

fera l'affaire (CN) .

Car d'abord comme on cherche une solution de type (a), et on exige $\sum_0^{n-1} a_i = \sum_0^{n-1} \lambda_i$.

Donc $\chi_M = P_n - \sum_0^{n-2} \mu_i P_i = \prod_0^{n-1} (X - \lambda_k) + \sum_0^{n-2} \mu_k P_k - \sum_0^{n-2} \mu_i P_i$.

Attention au piège, on a utilisé $\mu_{n-1} = 0 (***)$.

Il vient que le polynôme caractéristique est nul en tous les λ_i

Donc d)i) nous garanti que les $(\lambda_i)_0^{n-1}$ sont toutes solutions.

La matrice que nous cherchions est celle dont on a calculé le χ.

Les μ_k se récupèrent , à la remontada.

Par une double boucle, on évalue d)i) en a_0, on en sort μ_0.

Puis on évalue en a_1, on attrape μ_1.

Etc ...

Une autre sol avec des codes à relire.

b) Par invariance de la trace la condition $a_0 + a_1 = \lambda_0 + \lambda_1$ est clairement nécessaire.

Réciproquement, si $a_0 + a_1 = \lambda_0 + \lambda_1$, soit t le complexe défini par $t = a_0 a_1 - \lambda_0 \lambda_1$.

La matrice $M = \begin{pmatrix} a_0 & t \\ 1 & a_1 \end{pmatrix}$ a pour polynôme caractéristique :

$\zeta^2 - \mathrm{Tr}(M)\zeta + \det(M) = \zeta^2 - (\lambda_0 + \lambda_1)\zeta + \lambda_0\lambda_1 = (\zeta - \lambda_0)(\zeta - \lambda_1)$

convient et le problème $FL(2)$ admet une solution (non unique).

c) Le problème $FL(3)$ admet-il une solution pour

$$a_0 = 1, a_1 = -1, a_2 = 0, \lambda_0 = 1, \lambda_1 = 2 \text{ et } \lambda_2 = -3 \text{ avec } A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 6 & 0 \end{bmatrix}.$$

Comment procède-t-on ?

On suit l'indication du texte et l'on construit la matrice

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \\ m0 & m1 & 0 \end{bmatrix}$$

qui a bien les bons coefficients sur la diagonale.

On calcule alors le polynôme caractéristique de cette matrice $X^3 - Xm1 - X + m1 - m0$ et l'on ajuste les coefficients pour que ce polynôme soit $(X-1)(X-2)(X+3) = X^3 - 7X + 6$ on trouve $m_0 = 0$ et $m_1 = 6$.

d) Soient $a_0, \dots, a_{n-1} \in \mathbb{C}$. On pose , pour $k \in [\![0, n]\!]$, $P_k = \prod_{i=0}^{k-1} (X - a_i)$.

i) La famille $(P_0, \dots, P_n)$ est échelonnée en degré, elle est donc libre, et comme elle a le bon cardinal ($P_0 = 1$ par convention) c'est une base de $\mathbb{R}_n[X]$.

Il existe donc $\mu_0, \dots, \mu_{n-1}, \mu_n$ tels que $\prod_{k=0}^{n-1} (X - \lambda_k) = \mu_n P_n - \sum_{k=0}^{n-1} \mu_k P_k$.

Comme le polynôme produit est unitaire, on a $\mu_n = 1$.

On utilise la matrice de changement de base, en prenant garde à ne pas mélanger les types matrice (numpy) et polynôme.

1035. PYTHON. On considère l ' équation différentielle (E) : $\left(1 + x^2\right) y'' + xy' - y/4 = 0$.

a) Existe-t-il une solution y de (E) sar $]-1, 1[$ vérifiant $y(0) = 0$ et $y'(0) = \sqrt{2}$?

Si oui, est-elle unique ?

b) Si une telle solution existe, la représenter graphiquement à l'aide de Python.

c) Déterminer les solutions de (E) développables en série entière.

d) Soit $f_1 : x \mapsto \sum_{n=0}^{+\infty} a_n x^n$ une solution de (E) développable en série entière
telle que $a_0 = 0$ et $a_1 = \sqrt{2}$. Quel est le rayon de convergence de cette série ?

Sol : Latex clé usb noire.

(a) L'équation

$$\left(1 + x^2\right) y''(x) + xy'(x) - \frac{1}{4} y(x) = 0$$

est une équation différentielle linéaire et homogène du second ordre.

Les coefficients sont continus sur $\mathbb{R}$ et le coefficient de $y''(x)$ ne s'annule jamais.

On peut appliquer le Thm de Cauchy-Lipschitz : quels que soient les réels x_0, a et b,

il existe une, et une seule, solution f de (E) de classe $\mathbb{C}^2$ sur $\mathbb{R}$ telle que

$$f(x_0) = a \text{ et } f'(x_0) = b.$$

En particulier, il existe une, et une seule, solution sur]-1, 1[telle que $y(0) = 0$ et $y'(0) = \sqrt{2}$.

(b) La résolution numérique d'une équation différentielle demande qu'on l'écrive sous forme résoluble. L'équation (E) devient alors

$$\forall x \in \mathbb{R}, \quad Y'(x) = \begin{pmatrix} y'(x) \\ \frac{1}{1+x^2} \cdot \left(\frac{1}{4}y(x) - xy'(x)\right) \end{pmatrix} = f(Y(x), x).$$

c) La résolution numérique ne demande alors plus que de choisir un intervalle de temps

$(x_i)_0 \leqslant i < n$ et une condition initiale (y_0, v_0).

L'array$Y$ dooné par odeint est de la forme $(Y_{i,j})_{0\leqslant i<n, 0\leqslant j<2}$ où $Y_{i,0} \approx y(x_i)$ et $Y_{i,1} \approx y'(x_i)$.

- Mais l'approximation de la sol obtenue n'est définie que sur $[0, 1]$ et pas sur $[-1, 1]$!

Une première possibilité consiste à calculer l'équation vérifiée par $z = [x \mapsto y(-x)]$.

Comme $z(x) = y(-x), z'(x) = -y'(-x)$ et $z''(x) = y''(-x)$ et que y est une solution de (E)

sur l' intervalle $[x_1, x_2]$, alors z est une solution de (E) sur l'intervalle $[-x_2, -x_1]$:

$$\begin{aligned} \left(1+x^2\right) z''(x) + xz'(x) - \frac{1}{4}z(x) & \\ &= \left(1+x^2\right) y''(-x) - xy'(-x) - \frac{1}{4}y(-x) \\ &= \left[1+(-x)^2\right] y''(-x) + (-x)y'(-x) - \frac{1}{4}y(-x) \\ &= 0. \end{aligned}$$

Par conséquent, y est solution sur $[-1, 0]$ ssi, z est sol sur $[0, 1]$ avec la CI

$(z(0), z'(0)) = (y(0), -y'(0)) = (0, -\sqrt{2}) = -(0, \sqrt{2})$

REMARQUE.- En général, les fonctions y et z ne vérifient pas la même équa diff...

- On peut procéder de manière un peu différente en tirant parti de la structure particulière de l'équation différentielle.

Considérons la solution y de (E) définie sur $\mathbb{R}$ associée à la condition initiale $(a, b) = (0, b)$ et la fonction $z = [x \mapsto -y(-x)]$. On vérifie (comme plus haut) que z est une solution de (E) associée à la même condition initiale : comme le Théorème de Cauchy-Lipschitz s'applique, on en déduit que

$$\forall x \in \mathbb{R}, \quad z(x) = -y(-x) = y(x).$$

Autrement dit, la solution y est impaire et son graphe admet donc l'origine pour centre de symétrie et il est donc inutile d'invoquer deux fois la fonction odeint !

```
import...
def f(Y, x):
    y, y_prime = Y[0], Y[1]
    y_seconde = (y/4-x*y_prime)/(1+x**2)
    return np.array([y_prime, y_seconde])


X = np.arange(0, 1.01, 0.01)
CI = np.array([0, np.sqrt(2)])
Y = odeint(f, CI, X)
positions, vitesses = Y[:,0], Y[:,1]
plt.plot(X, positions)

Z = odeint(f, -CI, X) # on résout sur [0,1]
plt.plot(-X, Z[:,0]) # on trace sur [-1,0]

plt.figure()
plt.plot(X, positions)
plt.plot(-X, -positions)
```

Num 1009 (2021)

On munit $\mathbb{R}^4$ de sa structure euclidienne canonique.

Soit σ la réflexion de $\mathbb{R}^4$ par rapport à l'hyperplan d'équation $x + y + 3z + t = 0$.

a) Soit $x \in \mathbb{R}^4$. Montrer que $\sigma(x) = x - 2\langle x, u\rangle u$ où $u = \dfrac{1}{2\sqrt{3}}(1, 1, 3, 1)^T$.

En déduire la matrice P_σ de σ dans la base canonique de $\mathbb{R}^4$.

b) Soit $A = \begin{pmatrix} 13 & 3 & 0 & 2 \\ 3 & 5 & 6 & 4 \\ 0 & 6 & 9 & 3 \\ 2 & 4 & 3 & 9 \end{pmatrix}$. Montrer que P_σ est une matrice de passage vers une base de diagonalisation de A. En déduire les valeurs propres de A.

Sol :

a) Le vecteur $u = \frac{1}{2\sqrt{3}}(1,1,3,1)^T$ est le vecteur unitaire normal à l'hyperplan H

d'équation d'équation $x + y + 3z + t = 0$.

C'est donc une bon de H et par thm le proj $\perp$ d'un vecteur $x \in \mathbb{R}^4$ est $y = \langle x, u\rangle u$.

Si z est le proj $\perp$ sur H, on a $x = z + y$ et $\sigma(x) = z - y$ donc $\sigma(x) = x - 2\langle x, u\rangle u$.

On trouvera :

```
In [14]: print (m_sigma ())
[[ 0.83333333 -0.16666667 -0.5        -0.16666667]
 [-0.16666667  0.83333333 -0.5        -0.16666667]
 [-0.5        -0.5        -0.5        -0.5       ]
 [-0.16666667 -0.16666667 -0.5         0.83333333]]
```

On voit que réduire le nombre de décimales est ...

b) Soit $A = \begin{pmatrix} 13 & 3 & 0 & 2 \\ 3 & 5 & 6 & 4 \\ 0 & 6 & 9 & 3 \\ 2 & 4 & 3 & 9 \end{pmatrix}$.

On a par ex :

```
A=np. array ([
[13 , 3 , 0 , 2],[ 3 , 5, 6 , 4], [0, 6, 9, 3],[ 2 , 4 , 3 , 9]])
print (A)
print (msigma())
```

On trouve

$$\begin{pmatrix} 12 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 18 & 0 \\ 0 & 0 & 0 & 6 \end{pmatrix}$$

et les valeurs propres sur la ...

```
import...
a =1/2/3**(1/2)
```

```
u=np. array ([[ a],[a] ,[3*a],[a]])
print (" _____________________________ ")
print (u)
print (" _____________________________ ")
def ps(x,y):
    s=0
    for i in range (4):
        s+=x[i ,0]*y[i,0]
    return (s)
def sigma (x):
    return (x -2* ps(x,u)*u)
def base_c ():
    b=[ np.array ([[1] ,[0] ,[0] ,[0]])]
    b.append (np. array ([[0] ,[1] ,[0] ,[0]]))
    b.append (np. array ([[0] ,[0] ,[1] ,[0]]))
    b.append (np. array ([[0] ,[0] ,[0] ,[1]]))
    #b=[]
    #I=np.eye (4)
    #for i in range (4):
    # b.append (I[0:3 , i :i])
    return (b)# attention aux types.................
def m_sigma ():
    b=base_c ()
    M=sigma (b [0])
    for i in range (3):
        M=np. concatenate((M,sigma (b[i+1])) , axis =1)
    return (M)
def simp(M):# set option !!
    for i in range (4):
        for j in range (4):
            if abs (M[i,j ]) <0.001:
                M[i,j]=0
    return (M)
```

Num 1018 (2021) ? ?

Soit, pour $n \in \mathbb{N}^*$, $f_n : x \mapsto \left(1 + \dfrac{x}{n}\right)^n$.

a) Tracer $f_{10}, f_{100}, f_{1000}$ et exp ; que peut-on conjecturer ? Établir cette conjecture.

b) Avec Python, illustrer que la cv de la suite (f_n) n'est pas uniforme sur $\mathbb{R}^+$; le prouver.

c) Montrer que, pour $x \in \mathbb{R}^+$ et $n \in \mathbb{N}^*, x \geqslant \ln(1+x) \geqslant x - \dfrac{x^2}{2}$ et $e^x \geqslant \left(1 + \dfrac{x}{n}\right)^n$.

d) Soit $[a, b]$ un segment de $\mathbb{R}^+$. Montrer que la suite (f_n) converge uniformément

sur $[a,b]$ vers exp et, plus précisément, que $\|f_n - \exp\|_{\infty,[a,b]} = O\left(\frac{1}{n}\right)$.

Sol : A l'écran les courbes se chevauchent.

Graphiquement on peut conjecturer que la suite de fonction converge simplement vers l'exponentielle (la suite de fonctions étant croissante).

En effet si x est un réel fixé, on a

$$f_n(x) = \exp\left(n\ln\left(1+\frac{x}{n}\right)\right) = \exp\left(n\left(\frac{x}{n}+o\left(\frac{x}{n}\right)\right)\right) = \exp(x+o(x)) \mapsto e^x.$$

b) En allant chercher un peu plus loin (d'où l'utilité de la fonction "Dessin" qui donne la fenêtre sur l'axe des abscisses) on voit que l'écart entre les courbes peut s'éloigner fortement.

La convergence de la suite (f_n) n'est pas uniforme sur $\mathbb{R}^+$.

On a

$$|f_n(n) - e^n| = |e^n - 2^n| \cong e^n \mapsto \infty$$

A n fixé, $\{|f_n(x) - e^x|, x \in \mathbb{R}^+\}$contient $|e^n - 2^n|$ donc sa borne sup (dans $\overline{\mathbb{R}}$)

$$\|f_n - \exp\|_\infty \geqslant |e^n - 2^n|$$

qui tend vers l'infini.

c) Par étude de fonction classique on montre que, pour $x \in \mathbb{R}^+$ et $n \in \mathbb{N}^*$,

$x \geqslant \ln(1+x) \geqslant x - \frac{x^2}{2}$.

Mq $e^x \geqslant \left(1+\frac{x}{n}\right)^n$ est équivalent à $x \geqslant n\ln\left(1+\frac{x}{n}\right)$ soit $\frac{x}{n} \geqslant \ln\left(1+\frac{x}{n}\right)$ c'est fait.

d) Soit $[a,b]$ un segment de $\mathbb{R}^+$.

$$\frac{x}{n} - \frac{x^2}{2n^2} \leqslant \ln\left(1+\frac{x}{n}\right) \leqslant \frac{x}{n}$$

$$x - \frac{x^2}{2n} \leqslant n\ln\left(1+\frac{x}{n}\right) \leqslant x$$

$$e^{x-\frac{x^2}{2n}} \leqslant \left(1+\frac{x}{n}\right)^n \leqslant e^x$$

L'écart

$$|\exp(x) - f_n(x)| \leqslant e^x - e^{x-\frac{x^2}{2n}} = e^x\left(1-e^{-\frac{x^2}{2n}}\right) \leqslant e^b\left(1-e^{-\frac{a^2}{2n}}\right)$$

donc

$$\|\exp - f_n\|_\infty \leqslant e^b \left(1 - e^{-\frac{a^2}{2n}}\right)$$

le majorant tend vers 0 ce qui prouve que la convergence est uniforme sur $[a, b]$. Or

$$1 - e^{-\frac{a^2}{2n}} = \left[e^t\right]_{-\frac{a^2}{2n}}^0 = \int_{-\frac{a^2}{2n}}^0 e^t dt \leqslant \int_{-\frac{a^2}{2n}}^0 dt = \frac{a^2}{2n} = \mathcal{O}\left(\frac{1}{n}\right)$$

Voilà , et les codes :

```
import...
def f(n):
    def g(x):
        return ((1+x/n)** n)
    return (g)

tabcouleur=['green','blue','magenta','yellow']
def Dessin (a,b,N,f,col ):
    X=np. linspace (a,b,N)
    Y=[f(X[i]) for i in range (N)]
    plt .plot(X,Y,color =col )
Dessin (-1,4,20, exp , tabcouleur [0])
for i in range (3):
    Dessin (-1,4,20, f (10**( i+1)) , tabcouleur[i+1])
plt .show()
#plt . close ()
def F(x):
    def g(t):
        return (exp (-x*t)/( t +1))
    return ( integr . quad(g, 0, np.inf )[0])

def Dessin (a,b,N,f,col ):
    X=np. linspace (a,b,N)
    Y=[f(X[i]) for i in range (N)]
    plt .plot(X,Y,color =col )

Dessin (0.1 ,5 ,100 , F, tabcouleur [0])
```

Num 1026 (2021) ? ?

Soit $F : x \longmapsto \displaystyle\int_0^{+\infty} \frac{e^{-xt}}{t+1} dt$.

a) Tracer la courbe représentative de F à l'aide de Python.

Que peut-on conjecturer sur le domaine de définition, les variations et limites de F ?

b) Démontrer ces conjectures.

c) Montrer que F est solution d'une équation différentielle d'ordre 1 à coefficients constants.

En déduire que F est de classe $\mathcal{C}^\infty$ sur son domaine de définition.

Sol :

- La fonction F est définie sur $]0, \infty[$.

Si $x > 0$ est fixé, $t \mapsto \dfrac{e^{-xt}}{1+t}$ est continue sur $]0, \infty\,[$,

positive et $\dfrac{e^{-xt}}{1+t} \leqslant e^{-xt}$ qui est une fonction intégrable sur cet intervalle.

Si $x = 0$ la fonction $t \mapsto \dfrac{1}{1+t}$ non intégrable au voisinage de l'infini par équivalence à $1/t$ sur l'échelle de Riemann.

Si $x < 0$ alors $\dfrac{e^{-xt}}{1+t} > \dfrac{1}{1+t}$ redonne le même résultat.

- La fonction F est C_1 sur $]0, \infty[$.

$\diamond$ la fonction $t \mapsto \dfrac{e^{-xt}}{1+t}$ est intégrable sur $]0, \infty[$.

$\diamond$ la fonction $x \mapsto \dfrac{e^{-xt}}{1+t}$ admet une dérivée partielle relativement à $x\dfrac{-te^{-xt}}{1+t}$ qui elle même est continue par morceaux par rapport à t et continue par rapport à x.

De plus $\left|\dfrac{-te^{-xt}}{1+t}\right| \leqslant e^{-xt}$ qui est intégrable.

Par le théorème de Leibniz, la fonction F est bien $\mathcal{C}^1$.

- De plus la dérivée est l'intégrale de la dérivée partielle, négative, donc F est décroissante.

Rq : On peut très bien le démontrer directement.

- La limite en l'infini fait 0, donc asymptote horizontale, l'axe des abscisses.

Pour le démontrer on prend une suite (x_n) qui tend vers l'infini.

On choisit $x_n > 1$ ce qui n'est pas restrictif. On pose $u_n(t) = \dfrac{e^{-x_{nt}}}{1+t}$, suite de fonctions,

◇ La fonction u_n est clairement continue par morceaux sur $]0,\infty[$.

◇ Elle converge simplement vers la fonction nulle.

◇ Cette dernière étant elle-même continue par morceaux.

$$|u_n(t)| \leqslant e^{-x_n t} \leqslant e^{-t}$$

la majorante étant intégrable. Par le théorème de convergence dominée la suite des intégrales converge donc vers l'intégrale de la limite, c'est-à-dire 0.

- Au voisinage de $0, F$ tend vers l'infini, asymptote verticale, l'axe des ordonnées.

F étant décroissante, au $\mathcal{V}(0)$, soit elle est majorée, ou bien elle admet $+\infty$ comme limite.

Comme $t \leqslant \dfrac{1}{x}$ donne $-xt \geqslant -1$

$$F(x) \geqslant \int_0^{\frac{1}{x}} \frac{e^{-xt}}{t+1}dt \geqslant \frac{1}{e}\int_0^{\frac{1}{x}} \frac{1}{t+1}dt = \frac{1}{e}[\ln(1+t)]_0^{\frac{1}{x}} = O\left(\ln\left(1+\frac{1}{x}\right)\right)$$

elle n'est pas bornée.

c) On a montré (par anticipation ?)

$$F'(x) - F(x) = \int_0^{+\infty} \frac{-te^{-xt}}{1+t}dt - \int_0^{+\infty} \frac{e^{-xt}}{1+t}dt = \int_0^{+\infty} e^{-xt}dt = \frac{1}{x}\left[e^{-xt}\right]_{+\infty}^0 e^{-xt} = \frac{1}{x}.$$

Une récurrence aisée montre alors que F est de classe C^∞ sur son domaine de définition.

Codes :

```
import...
Voir 1117
```

Num 1032 (2021) ?, 1123 (2022)

Soient $f : (x,y) \in \mathbb{R}^2 \longmapsto \left(2x^2+3y^2\right)e^{-\left(x^2+y^2\right)}$ et, pour $R>0, D_R = \left\{(x,y) \in \mathbb{R}^2, x^2+y^2 \leqslant R\right\}$.

a) Tracer la surface représentative de la fonction f sur $[-2,2]^2$.

b) Montrer que f admet un minimum sur $\mathbb{R}^2$;

déterminer sa valeur et le point en lequel il est atteint.

c) Soit $a>0$. Montrer qu'il existe $R_a>0$ tel que, pour tout $(x,y) \in \mathbb{R}^2 \backslash D_{R_a}, f(x,y) \leqslant a$.

d) Montrer que f admet un maximum M sur $\mathbb{R}^2$. Calculer M.

e) Pour $c \in [0, M]$, soit $\Gamma_c = \big\{(x, y) \in \mathbb{R}^2, f(x, y) = c\big\}$.

Tracer Γ_c pour $c \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.75, 0.8, 0.9]$

Sol : les codes :

```
import matplotlib . pyplot as plt
import numpy as np
from mpl_toolkits. mplot3d import Axes3D
from scipy. integrate import odeint
from math import *
from random import *
import numpy. random as rd
from numpy. polynomial import Polynomial
import numpy. linalg as alg
import scipy. integrate as integr
ax = Axes3D (plt . figure ())
def f(x,y):
    return ((2*x **2+3* y**2)* np.exp(-x**2- y **2))

def Dessin2 (a,b,c,d,pas ,f):
    X = np. arange (a,b, pas )
    Y = np. arange (c, d, pas )
    X, Y = np. meshgrid (X, Y)
    Z = f(X, Y)
    ax. plot_surface(X, Y, Z)

Dessin2 (-2,2,-2,2,0.02 , f)

plt .show()

#plt . close ()

def Niveau (a,b,c,d,pas ,f, liste ):
    X = np. arange (a,b, pas )
    Y = np. arange (c, d, pas )
    X, Y = np. meshgrid (X, Y)
    Z = f(X, Y)
    #plt. axis(' equal ')
    plt .contour (X, Y, Z, liste)
#Dessin2 (-2,2,-2,2,0.02, f)

liste =[0.3 ,0.4 ,0.5 ,0.6 ,0.7 ,0.75 ,0.8 ,0.9]
a=2
```

```
Niveau (-a,a,-a,a ,0.02 , f, liste )
plt .show()
```

b) La fonction f, positive, admet en $(0,0)$ un minimum nul sur $\mathbb{R}^2$.

c) Soit $a > 0$.

En passant en coordonnées polaires on a

$$|f(x,y)| = \rho^2 \left(2\cos^2\theta + 3\sin^2\theta\right) e^{-\rho^2} \leqslant 5\rho^2 e^{-\rho^2}$$

qui tend vers 0 quand ρ tend vers l'infini, autrement dit, il existe $R_a > 0$ tel que,

pour tout $(x,y) \in \mathbb{R}^2 \backslash D_{R_a}, f(x,y) \leqslant a$.

d) On recherche les points critiques, en tenant compte de la double parité sur les variables qui permet de se ramener au cas où toutes les variables sont positives.

Les dérivées partielles sont

$$\left[-2xe^{-x^2-y^2}\left(-2+2x^2+3y^2\right), -2ye^{-x^2-y^2}\left(-3+2x^2+3y^2\right)\right]$$

et elles s'annulent en $(0,0),(0,1),(1,0)$, pour des valeurs de $f : 0, \dfrac{3}{e}, \dfrac{2}{e}$.

On applique la question précédente avec $a = \dfrac{3}{2e}$.

Dans la boule fermée de centre 0 et de rayon R_a, la fonction f continue admet des extrema qu'elle atteint, elle contient également $(0,1)$. Le maximum pour cette boule ne peut pas être pris au bord, car l'image de ce point est supérieure à $f(0,1)$ et à l'exterieur de la boule sont les points dont les valeurs valent la moitié de $f(0,1)$, et si ce point était sur le bords on l'atteindrait par une suite de points dont les images valent la moitié de ce qui doit être obtenu par limite, et c'est absurde.

Donc le maximum est un point critique, qui ne peut être que $(0,1)$.

Comme à l'extérieur de la boule et au bord, les valeurs images sont plus petites, ce maximum sur la boule est aussi un maximum absolu sur tout le plan.

e) Pour $c \in [0,M]$, soit $\Gamma_c = \left\{(x,y) \in \mathbb{R}^2, f(x,y) = c\right\}$.

Il s'agit des lignes de niveau de la surface à z constant.

Num 1035 (2021) ?

On dispose de n urnes $U_1, \ldots, U_n$ et n boules numérotées de 1 à n, que l'on place indépendamment dans les urnes, chaque boule ayant la probabilité $\frac{1}{n}$ d'être placée dans l'urne U_i pour chaque $i \in [\![1,n]\!]$. On note X_n le nombre d'urnes vides après le placement des n boules.

a) Écrire une fonction Python "different" qui prend une liste / et qui renvoie le nombre d'éléments distincts de I.

Par exemple, "different" ($[1, 2, 3, 1, 2]$) renvoie 3 .

b) Écrire une fonction Python simulX qui prend un entier n et renvoie une simulation de X_n.

c) Pour tout $i \in [\![1, n]\!]$, on note Y_i la variable aléatoire valant 1 si l'urne U_i est vide, 0 sinon. Détermine la loi de Y_i, son espérance et sa variance.

d) Montrer que $X_n = Y_1 + \cdots + Y_n$. Est-ce que la loi de X_n est binomiale ?

e) Calculer l'espérance de X_n.

f) Écrire une fonction Python esperanceX qui prend en entrée un entier n et renvoie une valeur approchée de $\mathrm{E}(X_n)$.

Sol : les codes :

```
import...
l= rd.randint (50, size=(n))

def different (l):
    n=len (l)
    m=max (l)
    t=[0 for i in range(m +1)]
    compteur =0
    for i in range (n):
        if t[l[i ]]==0:
            compteur +=1
            t[l[i]]=1
    return ( compteur )
def simul (n):
    tab_des_urnes =[[] for i in range (n)]
    for i in range (n):
        tirage =rd. randint (1,n)
        tab_des_urnes[ tirage ]. append (i)
    x=0
    for i in range (n):
        if tab_des_urnes[i]==[]:
            x+=1
    print ( tab_des_urnes)
    return (x)
def esperanceX (N,n):
    S=0
    for k in range (N):
```

```
        S+= simul (n)
    return (S/N)

def esperanceT (n):
    return ((n*(1 -1/ n )**(n)))

#for i in range (4):
    #n =10**( i+1)
    #for j in range (6):
        #print ([i,esperanceX (10**( j+1), n), esperanceT (n)])
```

c) Pour tout $i \in [\![1, n]\!]$, on note $Y_i(\Omega) = \{0, 1\}$, donc la variable aléatoire suit une loi de Bernoulli de paramètre p, d'espérance p et de variance $p(1-p)$ avec p la probabilité que cette urne soit vide. En raisonnant par dénombrement, le cardinal des possibles est celui des applications de l'ensemble des boules sur celui des urnes, avec un cardinal de n^n. L'ensemble des possibles est celui des applications de l'ensemble des boules sur celui des urnes privé de Y_i, avec un cardinal de $(n-1)^n$.

$$p = \left(1 - \frac{1}{n}\right)^n$$

d) De manière évidente on a $X_n = Y_1 + \cdots + Y_n$ mais la loi de X_n n'est pas binomiale car les variables de Bernoulli qui ont bien le même paramètre ne sont pas indépendantes.

e) Par linéarité,

$$E(X_n) = np = n\left(1 - \frac{1}{n}\right)^n$$

On choisit un nombre fixé d'urnes (variable grâce à i) et l'on simule un grand nombre d'expérience (variable grâce à j). On fait la moyenne des résultats positifs que l'on compare à l'espérance théorique.

$[0, 4.2, 3.486784401000001]$
$[0, 3.75, 3.486784401000001]$
$[0, 3.79, 3.486784401000001]$
$[0, 3.7749, 3.486784401000001]$
$[0, 3.77106, 3.486784401000001]$
$[0, 3.771536, 3.486784401000001]$
$[1, 35.9, 36.60323412732292]$
$[1, 36.85, 36.60323412732292]$
$[1, 36.935, 36.60323412732292]$
$[1, 36.868, 36.60323412732292]$
$[1, 36.85439, 36.60323412732292]$
$[1, 36.867499, 36.60323412732292]$
$[2, 369.1, 367.6954247709637]$
$[2, 367.97, 367.6954247709637]$
$[2, 367.974, 367.6954247709637]$
$[2, 367.8001, 367.6954247709637]$
$[2, 367.97215, 367.6954247709637]$

Num Non 2017 ? ?

a) Soit $A \in \mathcal{S}_n^+(\mathbb{R})$. Montrer qu'il existe $M \in \mathcal{S}_n^+(\mathbb{R})$ telle que $M^2 = A$.

b) Soit $A \in S_n^+(\mathbb{R})$. On définit la suite $(U_p)_{p\in\mathbf{N}}$ d'éléments de $\mathcal{M}_n(\mathbb{R})$

par $U_0 = I_n$ et $\forall p \in \mathbb{N}, \quad U_{p+1} = \frac{1}{2}\left(U_p + AU_p^{-1}\right)$.

En commençant par le cas où $n = 1$,

montrer que la suite $(U_p)_{p\in\mathbf{N}}$ est bien définie et qu'elle converge vers $\sqrt{A}$.

c) Implémenter le calcul de U_p pour tout entier naturel p et pour $A \in \mathcal{S}_n^+(\mathbb{R})$.

Tester avec une matrice A donnée par l'énoncé.

d) Soit $A \in GL_n(\mathbb{R})$. Montrer que $A^TA \in S_n^+(\mathbb{R})$.

En déduire qu'il existe un unique couple $(U, S) \in O_n(\mathbb{R}) \times S_n^{++}(\mathbb{R})$ tel que $A = US$.

Sol : Les codes :

```
import...
n=3
```

```
I=np.eye(n)
A = np. random . randint (10, size=(n,n))
A=A+np. transpose (A)

def car (M):
    return (np .dot (M,M))
def U(p):
    S=I
    for q in range (1,p +1):
        S =1/2*( S+np.dot(A,alg .inv (S)))
    return (S)
def U_rec (p):
    if p==0:
        return (I)
    else:
        return (1/2*( U(p -1)+ np.dot (A,alg.inv(U(p -1)))))

print (A)
for p in range (5):
    print (car (U(2* p)))
    print ('_____________________')
```

a) Classique. On diagonalise A, orthogonalement semblable via P à $D = \delta^2$,

et l'on pose $M = P\delta P^\top$.

b) Soit $A \in S_n^+(\mathbb{R})$. Par récurrence $U_p \in \mathcal{S}_n^{++}(\mathbb{R})$.

Attention , il en manque mais ça va aller !

Si c'est vrai au rang p, alors pour tout vecteur X non nul on a

$$X^\top 2U_{p+1}X = X^\top U_p X + X^\top A U_p^{-1} X$$

Or $X^\top U_p X > 0$ par hypothèse de récurrence, et

$$X^\top A U_p^{-1} X = X^\top M^2 U_p^{-1} X = X^\top M^\top M U_p^{-1} X = Y^\top U_p^{-1} Y \geqslant 0$$

avec $Y = MX$ et $U_p^{-1} \in S_n^+(\mathbb{R})$.

Donc $X^\top 2U_{p+1}X > 0$ et $U_{p+1} \in \mathcal{S}_n^+(\mathbb{R})$.

La suite $(U_p)_{p\in\mathbb{N}}$ est bien définie.

Les oublis : le produit de 2 sym est sym ssi elles commutent.

L'inverse d'une sym def positive l'est aussi, ses vp sont les inverses...

Il faut auusi expliquer que A et U_p commutent par rec.

Il faut aussi expliquer que U_p est sym par rec.

Si A est diagonale (sinon par le théorème spectral on s'y ramène), alors on montre par récurrence que U_p l'est et que sur la diagonale, les coefficients diagonaux sont régis par la suite récurrente $2u_{p+1} = u_p + \dfrac{\lambda_i}{u_p}$.

Suite récurrente classique...La méthode de Héron, intervalle stable $[\sqrt{\lambda}, +\infty[$.

On y arrive au pire au 2 ème coup.

La fonction y est croissante, on descend minoré , TLM, cie un seul pt fixe $\sqrt{\lambda}$.

En passant à la limite M par continuité on a $2M = M + AM^{-1}$, donc $M^2 = A$.

Cette suite converge vers $\sqrt{A}$, oui oui composante par composantes...

c) Voir codes.

d) On montre facilement que A^TA est symétrique et que pour tout

$XX^TA^TAX = \|AX\|^2 \geqslant 0$ et donc que $A^TA \in S_n^+(\mathbb{R})$ et même dans $S_n^{++}(\mathbb{R})$.

Ainsi il existe S dans $S_n^{++}(\mathbb{R})$ telle que $A^TA = M^2 = S^TS$, donc $U = AS^{-1}$ est orthogonale et il existe un couple $(U, S) \in O_n(\mathbb{R}) \times S_n^{++}(\mathbb{R})$ tel que $A = US$.

Si, avec les mêmes conditions, $US = U'S', U^{-1}U'$ est orthogonale donc ses valeurs propres sont dans $\mathbb{U}$, et aussi dans $\mathcal{S}_n^{++}(\mathbb{R})$, donc ses valeurs propres sont toutes égales à 1 . Comme $U^{-1}U' = SS^{-1}$, cette matrice symétrique qui n'a que 1 comme valeur propre est l'identité d'où $S = S'$ et $U = U'$, l'unicité est démontrée.

Num ? ? ?

Soient $A = (X_{i,j})_{1\leqslant i,j\leqslant n}$. une matrice dont les coeff sont des VA indé et $D = \det A$.

a) Avec PYTHON, estimer les valeurs de $E(D)$ et $\mathbf{V}(D)$ pour différentes valeurs de n lorsque les $X_{i,j}$ sont à valeurs dans $\{-1, 1\}$ avec $\mathbf{P}(X_{i,j} = -1) = \mathbf{P}(X_{i,j} = 1) = 12$. Conjectures ?

b) Montrer les conjectures dans le cas $n = 1$ puis $n = 2$.

c) Montrer que $\mathrm{E}(D) = \det(\mathrm{E}(X_{i,j}))_{1\leqslant i,j\leqslant n}$.

d) Soit $x \in \mathbb{R}$. Calculer $\mathbf{E}(X_A(x))$ dans le cas où les $X_{i,j}$ ont la même loi.

e) On suppose les $X_{i,j}$ centrées réduites.

i) Soient σ et T deux permutations de $[[1,n]]$. Montrer que :

$$\mathrm{Cov}\left(\prod_{i=1}^{n} X_{\sigma(i),i}, \prod_{i=1}^{n} X_{\tau(i),i}\right) = \begin{cases} 1 & \text{si } \sigma = \tau \\ 0 & \text{sinon.} \end{cases}$$

ii) Que vaut $\mathbf{V}(D)$?

$\diamond$ On admettra que le détd'une matrice $A = (a_{i,j}$ de taille n est (MPSI)

$$\det(A) = \left(\sum_{\sigma \in S_n} (-1)^{s(\sigma)} \prod_{i=1}^{n} a_{\sigma(i),i}\right)$$

où S_n désigne le groupe des permutations de $[\![1,n]\!]$.

Sol : les codes .

```
import...
n=20
l= rd.randint (50, size=(n))

def Bern_pm ():
    return (2* rd. binomial (1, 0.5) -1)

def Mat_al (n):
    M=np. zeros ((n,n))
    for i in range (n):
        for j in range (n):
            M[i,j]= Bern_pm ()
    return (M)

def D(n):
    return (alg .det (Mat_al (n)))

def esp (n,N):
    E=0
    V=0
    for k in range (N):
        E+=D(n)
        V+=D(n )**2
    return ([E/N,V/N-(E/N )**2])

def essai ():
    for n in [2 ,3 ,5 ,7]:
        print (esp(n ,10000))
```

```
essai ()

p=0.25
n=100

def Ph(a):
    if a==1:
        def g(t):
           return (1/4/t/t)
        return (g)
    else:
        def g(t):
            return (2* exp (-2* t*t))
        return (g)

def u(n,p):
    def g(t):
        S=rd. binomial (n, p, 100)
        pi =0
        for x in S:
            if abs (x-p)>t*sqrt(n):
                pi +=1
        return (pi /100)
    return (g)
def Dessin (a,b,N,f,col ):
    X=np. linspace (a,b,N)
    Y=[f(X[i]) for i in range (N)]
    plt .plot(X,Y,color =col )

plt .xlim (0 ,3)
plt .ylim (0 ,50)

Dessin (0,3,100, Ph (1), tabcouleur [0])
Dessin (0,3,100, Ph (2), tabcouleur [1])
Dessin (0,3,100, u(n,p), tabcouleur [2])

plt .show()
#plt . close ()
```

La moyenne serait nulle, et la variance la factorielle de la dimension ?

b) $\diamond$ Si $n = 1$, $E(D) = E(X) = 0$ et comme $X^2 = 1, \mathbf{V}(D) = 1$.

$\diamond$ Si $n = 2$.

Si X et Y sont deux variables de Bernoulli indépendantes, $XY(\Omega) = \{-1, 1\}$,

et par incompatibilité, puis par indépendance

$$P[XY=1]=P[X=1,Y=1]+P[X=-1,Y=-1]=P[X=1]P[Y=1]+P[X=-1]P[Y=-1]=\frac{1}{4}+\frac{1}{4}=$$

donc XY a la même loi que X par exemple.

Ainsi

$$E(D)=E\left(X_{1,1}X_{2,2}-X_{1,2}X_{2,1}\right)=E\left(X_{1,1}X_{2,2}\right)-E\left(X_{1,2}X_{2,1}\right)=0-0=0$$

d'après ce qui précède.

Comme les couples de variables qui définissent le déterminant sont indépendants (car définis à partir de systèmes de variables mutuellement indépendantes)

$$\mathbf{V}(D)=\mathbf{V}\left(X_{1,1}X_{2,2}-X_{1,2}X_{2,1}\right)=\mathbf{V}\left(X_{1,1}X_{2,2}\right)+\mathbf{V}\left(-X_{1,2}X_{2,1}\right)=1+1=2$$

c) À partir de cette question les composantes ne suivent plus des lois de Bernoulli.

Démontrons par récurrence sur n que $\mathrm{E}(D)=\det\left(\mathrm{E}\left(X_{i,j}\right)\right)_{1\leqslant i,j\leqslant n}$.

C'est évident pour $n=1$.

Si on suppose la relation vraie pour toute matrice aléatoire de taille $n-1$, et si l'on prend M alétoire de taille n.

En développant avec la première colonne $M_{i,j}$ étant la matrice

obtenue en ôtant la ième ligne et la jème colonne : sur laquelle on pourra appliquer l'hypothèse de récurrence par linéarité

$$\mathrm{E}(D)=E\left(\sum_{i=1}^{n}(-1)^{i+1}X_{i,1}\det\left(M_{i,1}\right)\right)=\sum_{i=1}^{n}(-1)^{i+1}E\left(X_{i,1}\det\left(M_{i,1}\right)\right)$$

Comme $X_{i,1}$ est indépendante des variables qui définissent $\det\left(M_{i,1}\right)$, et donc de ce déterminant ;

$$\mathrm{E}(D)=\sum_{i=1}^{n}(-1)^{i+1}E\left(X_{i,1}\right)E\left(\det\left(M_{i,1}\right)\right)$$

et avec l'hypothèse de récurrence

$$\mathrm{E}(D)=\sum_{i=1}^{n}(-1)^{i+1}E\left(X_{i,1}\right)\det\left(\mathrm{E}\left(X_{k,j}\right)\right)_{k\neq i;2\leq j\leq n}$$

où l'on retrouve la définition du déterminant de la matrice des espérances, développé grâce à la première colonne

$$\mathrm{E}(D)=\det\left(\mathrm{E}\left(X_{i,j}\right)\right)_{1\leq i,j\leq n}$$

item [d)] Soit $x \in \mathbf{R}$.

On applique la question précédente

$$\mathbf{E}\left(X_A(x)\right) = \det\left(\mathrm{E}\left(x\delta_{i,j} - X_{i,j}\right)\right)_{1\leq i,j\leq n}.$$

Comme les variables ont la même loi on peut se concentrer sur deux cas (μ l'espérance commune) :

$$\left(\mathrm{E}\left(x\delta_{1,1} - X_{1,1}\right)\right) = \left(\mathrm{E}\left(x - X_{1,1}\right)\right) = x - \mathrm{E}\left(X_{1,1}\right) = x - \mu$$
$$\left(\mathrm{E}\left(x\delta_{1,2} - X_{1,2}\right)\right) = \left(\mathrm{E}\left(-X_{1,1}\right)\right) = -\mathrm{E}\left(X_{1,2}\right) = -\mu$$

Il s'agit donc du polynôme caractéristique de la matrice μJ (J classiquement remplie avec des 1).

$$\mathbf{E}\left(X_A(x)\right) = x^{n-1}(x - n\mu).$$

e) On suppose les $X_{i,j}$ centrées réduites.

i) Soient σ et T deux permutations de $[\![0, n]\!]$.

On a par indépendance

$$\mathbf{E}\left(\prod_{i=1}^{n} X_{\sigma(i),i}\right) = 0$$

, et de même pour l'autre variable.

Donc

$$\mathrm{Cov}\left(\prod_{i=1}^{n} X_{\sigma(i),i}, \prod_{i=1}^{n} X_{\tau(i),i}\right) = \mathbf{E}\left(\prod_{i=1}^{n} X_{\sigma(i),i} \prod_{i=1}^{n} X_{\tau(i),i}\right)$$

si $\sigma \neq \tau$, alors il existe i dans $\in [\![1, n]\!]$ tel que $\sigma(i) \neq \tau(i)$

$$\mathrm{Cov} = \mathbf{E}\left(\prod_{\sigma(i)=\tau(i)} X^2_{\sigma(i),i}\right) \mathbf{E}\left(\prod_{\sigma(i)\neq\tau(i)} X_{\sigma(i),i} X_{\tau(i),i}\right)$$

et comme les systèmes de variables indépendantes sont disjoints,

$$\mathrm{Cov} = \mathbf{E}\left(\prod_{\sigma(i)=\tau(i)} X^2_{\sigma(i),i}\right) \prod_{\sigma(i)\neq\tau(i)} \mathbf{E}\left(X_{\sigma(i),i}\right) \mathbf{E}\left(X_{\tau(i),i}\right) = 0$$

puisque le second référentiel n'est pas vide.

$\diamond$ si $\sigma = \tau$, de nouveau par indépendance

$$\text{Cov} = \mathbf{E}\left(\prod_{i=1}^{n} X_{\sigma(i),i}^2\right) = \prod_{i=1}^{n} \mathbf{E}\left(X_{\sigma(i),i}^2\right) = 1$$

ii) Les variables étant centrées par la question c) la moyenne du déterminant est nulle, et de plus, par définition du déterminant

$$\mathbf{V}(D) = \mathbf{E}\left(\left(\sum_{\sigma}(-1)^{s(\sigma)}\prod_{i=1}^{n} X_{\sigma(i),i}\right)\left(\left(\sum_{\tau}(-1)^{s(\tau)}\prod_{i=1}^{n} X_{\tau(i),i}\right)\right) = \sum_{\sigma,\tau}(-1)^{s(\sigma)}(-1)^{s(\tau)}E\left(\prod_{i=1}^{n} X_{\sigma(i),i}\right)\prod_{i=1}^{n}\right.$$

et avec ce qui précède

$$\mathbf{V}(D) = \sum_{\sigma} 1 = n!$$

ceci démontre la conjecture du début.

Maintenant oraux stainer centrale bis (la réf)...

Enoncés

Code disk dur (tous ?) o18,o19...

Exo 11 de la première série : voir annexe 17(20212022).

Relire enorme pb géométrique exo 11 !!!!!

Ca devrait etre correct maintenant...

1) *(Centrale)* Soit χ_n le poly caractéristique de $A_n = \begin{pmatrix} 0 & 0 & \cdots & 0 & 1/n \\ 1 & \ddots & \ddots & \vdots & \vdots \\ 0 & \ddots & \ddots & 0 & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & 1 & 1/n \end{pmatrix} \in \mathcal{M}_n(\mathbb{R})$.

   (1) Écrire une fonction Python prenant pour paramètre n et renvoyant les coefficients de χ_n sous forme d'un tableau `numpy` (cf. `numpy.poly`).

   (2) Afficher les coefficients de χ_n pour $n \in [\![2, 8]\!]$ et conjecturer la valeur de χ_n. Démontrer ce résultat.

   (3) Afficher les modules des racines de χ_n pour $n \in [\![2, 8]\!]$. Que peut-on conjecturer ? Démontrer ce résultat.

2) *(Centrale)* Programmer en Python le calcul de $u_n = \displaystyle\int_0^1 \left(3x^2 - 2x^3\right)^n \mathrm{d}x$.

Écrire u_n comme une somme et en déduire une autre implémentation du calcul de u_n.

Calculer u_n pour $n \in [\![0, 50]\!]$ par les deux méthodes et commenter.

Montrer que (u_n) converge vers 0.

Tracer les valeurs de $\sqrt{n}u_n$ pour $n = 100k,\ k \in [\![0, 100]\!]$. Conjecturer.

Montrer que $u_n \sim \dfrac{A}{\sqrt{n}}$ où $A = \displaystyle\int_0^{+\infty} e^{-3t^2}\mathrm{d}t$.

(on pourra utiliser le changement de variable $u = 1 - x$).

3) *(Centrale)* Donner un code Python permettant de calculer

$$u_n = \frac{1}{n}\sum_{k=1}^{n} \cos\left(\frac{k\pi}{n}\right) \arctan\left(\frac{k}{n}\right).$$

(1) Tracer les u_n pour $n \in [\![1, 100]\!]$. Que remarque-t-on ? Le démontrer.

(2) On note ℓ la limite de (u_n) et $v_n = \dfrac{n}{\pi}(\ell - u_n)$.

Tracer les v_n pour $n \in [\![1, 100]\!]$. Conjecture ?

(3) Soit g une fonction de classe $\mathcal{C}^2$ sur $[0, 1]$ et $M = \max\limits_{[0,1]} |g''|$.

Montrer que, pour $n \geqslant 2$:

$$\forall k \in [\![1, n]\!] \quad \forall t \in \left[\frac{k-1}{n}, \frac{k}{n}\right] \quad \left|g(t) - g\left(\frac{k}{n}\right) - g'\left(\frac{k}{n}\right)\left(t - \frac{k}{n}\right)\right| \leqslant \frac{M}{2n^2}.$$

En déduire :

$$\int_0^1 g(t)\,\mathrm{d}t - \frac{1}{n}\sum_{k=1}^{n} g\left(\frac{k}{n}\right) = \frac{g(0) - g(1)}{2n} + o\left(\frac{1}{n}\right).$$

4) *(Centrale)* Pour $n \geqslant 3$, montrer que l'équation (E_n) $e^x = x^n$ admet une unique solution u_n dans $[1, 2]$.

Afficher avec Python les u_n pour $n = 10k,\ k \in [\![1, 100]\!]$.

Conjecture ? Démontrer le résultat.

On note ℓ la limite de (u_n). Avec Python, conjecturer les valeurs a, b telles que

$$u_n - \ell = \frac{a}{n} + \frac{b}{n^2} + o\left(\frac{1}{n^2}\right).$$

Démontrer le résultat.

5) *(Centrale)* Justifier que $\langle P, Q\rangle = \displaystyle\int_{-1}^{1} PQ$ définit un produit scalaire

et $N_\infty(P) = \max\limits_{[-1,1]} |P|$ une norme sur $E = \mathbb{R}[X]$.

Soit $F = \mathbb{R}_5[X]$; déterminer la base orthonormale $(E_0, \ldots, E_5)$ déduite

de la base canonique par l'algorithme de Gram-Schmidt.

Tracer les graphes des E_i et déterminer les valeurs de t où $N_\infty(E_i)$ est atteinte.

Montrer que, si $P \in F$ vérifie $\|P\| = 1$, alors $N_\infty(P) \leqslant 3\sqrt{2}$. Quand a-t-on égalité ?

Trouver a et b tels que, pour tout P de F, $a\|P\| \leqslant N_\infty(P) \leqslant b\|P\|$.

Donner des exemples de P pour lesquels il y a égalité à droite ou à gauche.

6) *(Centrale)* Intégrer $xy'' - y' + 4x^3y = 0$ à l'aide du changement de variable $t = x^2$.

On pose $a = \sqrt{\pi/2}$.

On trouve que la solution f vérifiant $f(a) = f'(a) = 1$ est $x \mapsto \sin\left(x^2\right) - \dfrac{\cos\left(x^2\right)}{2a}$.

Comparer les graphes sur $[a, 4]$ de cette solution exacte, de la solution approchée fournie par la méthode d'Euler (que l'on programmera) et de celle fournie par la fonction `scipy.integrate.odeint`.

7) *(Centrale)* Une particule se déplace sur une surface comportant 4 positions possibles :

A_0 et A_3 qui sont des puits, A_1 et A_2 qui sont des positions intermédiaires.

À chaque étape :

– si la particule est dans un puits, elle y reste avec une probabilité de 1.

– si elle est en A_1, elle va en A_0 avec une proba p, en A_2 avec une probabilité $1-p$.

– si elle est en A_2, elle va en A_1 avec une proba p, en A_3 avec une probabilité $1-p$.

On note x_n la variable aléatoire donnant la position de la particule à l'étape n : $x_n(\Omega) = [\![0, 3]\!]$.

(1) Écrire une fonction Python qui donne x_{n+1} en fonction de x_n et de p, puis une fonction renvoyant x_n en fonction de n, x_0 et p.

(2) Tracer l'histogramme des x_n obtenues durant N essais (cf. `matplotlib.pyplot.bar`).

(3) Soit $X_n = \begin{pmatrix} P(x_n = 0) \\ P(x_n = 1) \\ P(x_n = 2) \\ P(x_n = 3) \end{pmatrix}$.

Trouver $A \in \mathcal{M}_4(\mathbb{R})$, indépendante de n, telle que $X_{n+1} = AX_n$ pour tout n.

(4) Montrer que A est diagonalisable si et seulement si $p \in]0, 1[$.

(5) Pour $p = 1/2$, diagonaliser A avec Python et calculer $\lim_{n\to\infty} X_n$. Comparer avec les résultats précédents.

N.B. Prendre garde que, si `A` est un tableau `numpy` carré et `n` un entier naturel, `A**n` se contente d'élever chaque coefficient de `A` à la puissance `n` ! ! Cf. `numpy.linalg.matrix_power` pour un calcul efficace des puissances d'une matrice carrée, ou programmer soi-même une exponentiation rapide...

8) *(Centrale)* On donne $n+1$ entiers naturels $a_0 < a_1 < \cdots < a_n$ et, pour $P \in \mathbb{R}_n[X]$,

on pose

$$\|P\|_A = \max_{i \in [\![0,n]\!]} |P(a_i)| \quad \text{où} \quad A = (a_0, \ldots, a_n).$$

Écrire une fonction Python `N(A,P)` qui renvoie $\|P\|_A$.

Montrer que $\|\cdot\|_A$ est une norme sur $\mathbb{R}_n[X]$.

On veut évaluer d_n, distance de X^n à $\mathbb{R}_{n-1}[X]$.

Pour $A = (0, 1)$, calculer le minimum des $\|X + a\|_A$ pour a décrivant $\mathbb{R}$.

Pour $A = (0, 1, 2)$, calculer le minimum des $\left\|X^2 + aX + b\right\|_A$ pour (a, b) décrivant $\mathbb{R}^2$ *(on pourra utiliser la fonction* `fmin` *de* `scipy.optimize` *qui prend comme argument une fonction f et un point initial et renvoie le minimum de f).*

Dans le cas général, pour $P \in \mathbb{R}_{n-1}[X]$, montrer qu'il existe une unique famille de

réels $(b_0, \ldots, b_n)$ telle que

$$X^n + P(X) = \sum_{k=0}^{n} b_k \prod_{j \in [\![0,n]\!] \setminus \{k\}} (X - a_j).$$

Calculer $\sum_{k=0}^{n} b_k$ et montrer que

$$\forall k \in [\![0,n]\!] \qquad \prod_{j \in [\![0,n]\!] \setminus \{k\}} |a_k - a_j| \geqslant \frac{n!}{\binom{n}{k}}.$$

En déduire que $\|X^n + P(X)\|_A \geqslant \dfrac{n!}{2^n}$.

Calculer d_n lorsque $A = (0, 1, \ldots, n)$.

9) *(Centrale)* Pour p dans $\mathbb{N}^*$, on définit le polynôme

$$Q_p(X) = (2p+2)X^{2p+1} - 2pX^{2p-1} - (2p-1)X^{2p-2} - \cdots - 2X - 1.$$

En utilisant le module `Polynomial`, écrire une fonction Python renvoyant Q_p.

Calculer les racines de Q_p pour $p \in \{3, 4, 5, 10\}$;

quelle conjecture peut-on faire sur les racines réelles ?

Tracer les racines des Q_p pour $p < 30$ ainsi que le cercle unité ;

quelle conjecture peut-on faire sur les racines complexes ?

Écrire une fonction Python renvoyant le minimum des modules des racines de Q_p.

Montrer que $R_p(X) = -X^{2p+1} Q_p(1/X)$ admet une unique racine réelle positive.

En déduire qu'il en est de même de Q_p.

Montrer que $Q_p(X) = (2p+2)X^{2p+1} - \dfrac{1 - X^{2p+1}}{(1-X)^2} + \dfrac{(2p+1)X^{2p}}{1-X}$.

En déduire qu'à partir d'un certain rang $3/2 < \alpha_p < 2$,

où α_p est la racine réelle positive de Q_p.

10) *(Centrale)* Dans $\mathbb{R}^n$ on désigne par $\|\cdot\|$ et $\langle \cdot, \cdot \rangle$ la norme euclidienne et le produit scalaire canoniques. Pour A et B dans $\mathcal{M}_n(\mathbb{R})$, on note (N) la propriété

$$\forall X \in \mathbb{R}^n \quad \|AX\| = \|BX\|.$$

Montrer que A et B vérifient (N) si et seulement si

$$\forall (i,j) \in [\![1,n]\!]^2 \quad \langle C_i(A), C_j(A)\rangle = \langle C_i(B), C_j(B)\rangle$$

où $C_j(M)$ désigne le j-ième vecteur colonne de la matrice M.

Coder en Python $d(A,B) = \sum_{i=1}^{n}\sum_{j=1}^{n}\left(\langle C_i(A), C_j(A)\rangle - \langle C_i(B), C_j(B)\rangle\right)^2$, puis une fonction booléenne renvoyant `True` si et seulement si A et B vérifient (N).

Montrer que, si A est inversible, A et B vérifient (N) si et seulement s'il existe une matrice orthogonale Ω telle que $B = \Omega A$.

11) *(Centrale)* On fixe un entier $N \geqslant 2$, un réel p dans $]0,1[$ et l'on pose $q = 1-p$.

Une infinité de joueurs $(J_k)_{k\in\mathbb{N}}$ participent à un tournoi : le match 1 oppose J_0 à J_1 et J_0 l'emporte avec la probabilité p. Puis, pour $k \geq 2$, J_k affronte le vainqueur du match $k-1$ avec la probabilité p de gagner. Chaque perdant est définitivement éliminé.

Le tournoi s'achève lorsqu'un même joueur parvient à enchaîner N victoires

consécutives, ledit joueur étant alors déclaré vainqueur.

On note $T_{N,p}$ le nombre de matchs nécessaires pour qu'un joueur soit vainqueur.

Pour $n \in \mathbb{N}^*$ on note A_n l'événement "il n'y a pas de vainqueur au match n".

Écrire une fonction `T(N,p)` qui renvoie $T_{N,p}$ suite à une simulation d'un tournoi.

Écrire une fonction `moyenne(N,p)` qui renvoie la moyenne des $T_{N,p}$ obtenus

lors de 10^4 tournois.

Que donnent `moyenne(3,0.3)` et `moyenne(6,0.5)` ?

Montrer que

$$\forall n \in [\![1, N-1]\!] \quad P(A_n) = 1 \quad \text{et} \quad P(A_N) = 1 - q^{N-1}.$$

Que représente l'événement $A_{n-1}\backslash A_n$? Établir

$$\forall n \geqslant N+1 \quad P(A_{n-1}) - P(A_n) = pq^{N-1}P(A_{n-N}).$$

Montrer l'existence et calculer les valeurs de $\lim_{n\to\infty} P(A_n)$ et $\sum_{n=1}^{\infty} P(A_n)$.

En déduire l'espérance de $T_{N,p}$ et comparer avec les résultats des simulations.

Corrigés

Il est essentiel de s'approprier les commandes les plus importantes

de la bibliothèque fournie par Centrale.

Code disk dur o19info01

1) (1) Je crée une matrice nulle de la bonne taille, puis j'installe les coefficients non nuls et enfin j'appelle `np.poly` qui donne la liste des coefficients de χ_n **en commençant par le coefficient dominant** !

```
import numpy as np
from numpy.polynomial import Polynomial

#on peut limiter le nombre de décimales
pour rendre plus lisible
np.set_printoptions(precision=3)

def khi(n):
    A=np.zeros((n,n))
    for j in range(n-1): A[j+1,j]=1
    for i in range(n): A[i,n-1]=1/n
    return np.poly(A)

def modules(n):
    P=khi(n)
    P=Polynomial([P[n-k] for k in range(n+1)])
     #pour renverser, attention aux décalages d'indices...
    return np.abs([r for r in P.roots()])

for n in range(2,9): print(khi(n))
print('Modules des valeurs propres')
for n in range(2,9): print(modules(n))
)
```

(2) Le résultat fourni par Python est le suivant :

```
[ 1.  -0.5 -0.5]
[ 1.    -0.333 -0.333 -0.333]
[ 1.   -0.25 -0.25 -0.25 -0.25]
[ 1.  -0.2 -0.2 -0.2 -0.2 -0.2]
[ 1.    -0.167 -0.167 -0.167 -0.167 -0.167 -0.167]
[ 1.    -0.143 -0.143 -0.143 -0.143 -0.143 -0.143 -0.143]
[ 1.    -0.125 -0.125 -0.125 -0.125 -0.125 -0.125 -0.125 -0.125]
```

Cela permet de conjecturer à la surprise générale que $\chi_n = X^n - \frac{1}{n}\sum_{k=0}^{n-1} X^k$.

C'est un résultat limite programme mais classique

(*matrice compagnon* d'un polynôme) démo ds le cours...

(3) Pour les modules des racines, on n'aura valeurs numériques approchées...
`P.roots()` renvoie une liste des valeurs approchées des racines de `P`, si toutefois il a été construit par la commande `Polynomial` du module du même nom (cf. la doc de Centrale). **Attention !** les coefficients doivent ici être fournis par ordre de degré croissant, il faut donc **renverser** la liste fournie par `np.poly` !!

Noter que `[abs(r) for r in P.roots()]` renverrait le même résultat, à ceci près que le nombre de décimales imposé à `numpy` ne serait pas pris en compte...
Le résultat fourni par Python est le suivant :

```
[ 0.5  1. ]
[ 0.577  0.577  1.   ]
[ 0.606  0.642  0.642  1.   ]
[ 0.646  0.646  0.692  0.692  1.   ]
[ 0.67   0.683  0.683  0.73   0.73   1.   ]
[ 0.696  0.696  0.714  0.714  0.76   0.76   1.   ]
[ 0.715  0.72   0.72   0.74   0.74   0.785  0.785  1.   ]
```

On peut conjecturer qu'il y a une valeur propre de module 1 et que les autres sont de module strictement inférieur.

En fait, il est clair que 1 est valeur propre (car racine de χ_n d'après **b)**). On peux même s'assurer qu'elle est simple, car

$$\chi_n'(1) = n - \frac{n-1}{2} \neq 0.$$

Pour les autres valeurs propres, j'utilise plutôt $B = A_n^T$, qui a le même polynôme caractéristique et se trouve être stochastique (la somme des coefficients de chaque ligne vaut 1). Il en résulte que, pour tout vecteur colonne X, $N_\infty(BX) \leqslant N_\infty(X)$ (clair) écrire le produit si besoin, d'où, si λ est valeur propre de B, $|\lambda| \leqslant 1$.
Supposons un instant λ de module 1, distincte de 1 ; λ étant racine de χ_n, j'ai

$$n\lambda^n = \frac{1-\lambda^n}{1-\lambda} \quad \text{soit} \quad \lambda^n\left[n(1-\lambda)+1\right] = 1$$

d'où

$$|n+1-n\lambda| = 1$$

donc $n\lambda$ appartient au cercle de centre $n+1$ (sur la droite réelle) et de rayon 1. Or $n\lambda$ appartient aussi au cercle de centre 0 et de rayon n et ces deux cercles sont tangents au point d'affixe n, qui est donc leur seul point d'intersection. Par conséquent $n\lambda = n$, d'où une contradiction. En conclusion

1 est valeur propre simple de A_n et toutes ses autres valeurs propres sont de module strictement inférieur à 1.

2) Quoi qu'on fasse Python ne pourra donner qu'une valeur numérique approchée de u_n pour une valeur numérique de n. Donc, autant demander directement une valeur approchée à `scipy.integrate.quad` (cf. la doc fournie par Centrale).

Pour l'expression "comme une somme", j'utilise la formule du binôme et j'intègre terme à terme (somme finie !), ce qui donne :

$$u_n = \sum_{k=0}^{n} (-1)^k \binom{n}{k} \frac{3^{n-k}2^k}{2n+k+1},$$

que je calcule avec Python grâce à la fonction `sum` qui renvoie la somme des termes d'une liste. Si l'examinateur insiste, il n'est pas très compliqué de programmer une boucle...

On constate qu'avec le calcul approché direct, les valeurs semblent décroître doucement, tandis que la somme de grosses valeurs de signes variés conduit assez vite à des résultats fantaisistes, cela en raison du nombre trop faible de chiffres significatifs.

Comme la fonction $x \mapsto 3x^2 - 2x^3$ croît strictement de 0 à 1 sur $[0,1]$, le théorème de convergence dominée permet de fontrer facilement que (u_n) converge vers 0.

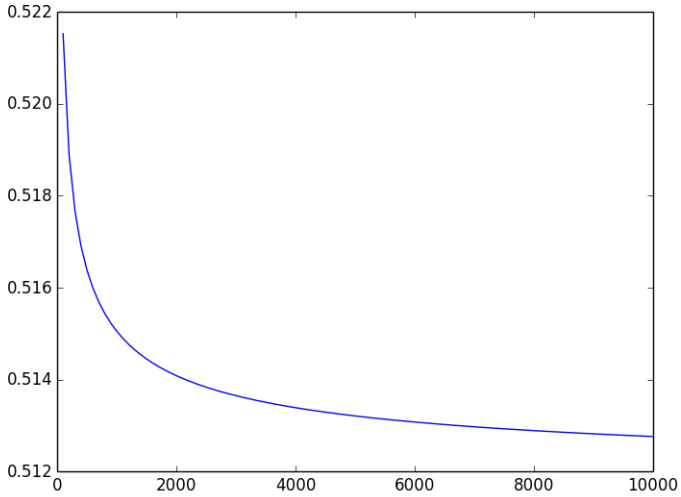
Il vaut mieux utiliser la première fonction pour étudier le comportement de $\sqrt{n}u_n$! Le tracé se fait naturellement en construisant la liste des valeurs de n et la liste correspondante des $\sqrt{n}u_n$.

```
import scipy.integrate as integr
import matplotlib.pyplot as plt
import numpy as np
from math import *
from scipy.special import binom
#Il y autre chose dans random...
def u(n):
    return integr.quad(lambda x: (3*x**2-2*x**3)**n,0,1)[0]

def u2(n):
    return sum([(-1)**k*binom(n,k)*3**(n-k)*2**k/(2*n+k+1) for k in range(n+1)])

print([u(n) for n in range(51)])
print([u2(n) for n in range(51)])
print(integr.quad(lambda x: np.exp(-3*x**2),0,np.inf))
N=[100*k for k in range(1,101)]
plt.plot(N,[sqrt(n)*u(n) for n in N])
plt.show()
```

On obtient le graphique suivant, or il se trouve que $\displaystyle\int_0^{+\infty} e^{-3t^2}\mathrm{d}t \approx 0,511$!



1.png

Le changement de variable suggéré donne

$$u_n = \int_1^0 \left(3\,(1-u)^2 - 2\,(1-u)^3\right)^n (-\mathrm{d}u) = \int_0^1 \left(1-3u^2+2u^3\right)^n \mathrm{d}u$$

et en posant $t = \sqrt{n}u$ j'obtiens

$$\sqrt{n}u_n = \int_0^{\sqrt{n}} \left(1 - \frac{3t^2}{n} + \frac{2t^3}{n^{3/2}}\right)^n \mathrm{d}t = \int_0^{+\infty} f_n(t)\,\mathrm{d}t \quad \text{où} \quad f_n(t) = \begin{cases} \left(1 - \dfrac{3t^2}{n} + \dfrac{2t^3}{n^{3/2}}\right)^n & \text{si } t \leqslant \sqrt{n} \\ 0 & \text{si } t > \sqrt{n} \end{cases}.$$

Là encore, le théorème de convergence dominée permet de montrer classiquement que

$$\sqrt{n}u_n \underset{n\to\infty}{\longrightarrow} \int_0^{+\infty} e^{-3t^2}\mathrm{d}t$$

d'où l'équivalent souhaité.

3) (1) Calculs et tracés classiques :

```
from math import cos,atan,pi
import matplotlib.pyplot as plt
import scipy.integrate as integr

def u(n):
    return sum([cos(k*pi/n)*atan(k/n) for k in range(1,n+1)])/n

def v(n):
    global L
    return n/pi*(L-u(n))

L=integr.quad(lambda x:cos(pi*x)*atan(x),0,1)[0]# attention au tuple...
print(L)
N=list(range(1,101))
plt.plot(N,[u(n) for n in N],'b:',linewidth=2,label='$u_n$')
plt.plot(N,[v(n) for n in N],'r--',linewidth=2,label='$v_n$')
plt.legend(loc=0)
plt.grid()
plt.show()
```

On conjecture que (u_n) converge, vers une valeur voisine de $-0,16$. La démonstration est immédiate, puisqu'il s'agit de sommes de Riemann, pour la fonction continue $f : x \mapsto \cos(\pi x)\arctan(x)$ sur le segment $[0,1]$. Par conséquent

$$\boxed{(u_n) \text{ converge vers } L = \int_0^1 \cos(\pi x)\arctan(x)\,\mathrm{d}x.}$$

Pas de primitive "simple" pour f, nous en restons donc à la valeur numérique approchée fournie par Python...

Noter que, curieusement, la fonction arctan (notation officielle en maths) s'appelle bien `arctan` dans le module `numpy`, mais `atan` dans le module maths !??

(2) Le code pour le tracé demandé a été donné au **a)**. On conjecture que (v_n) converge vers une valeur voisine de $0,125$ (penser à zoomer dans la fenêtre créée par `matplotlib` !).

(3) Cette question (de maths !) permet de démontrer le résultat précédent. La majoration est fournie par l'inégalité de Taylor-Lagrange, qui découle immédiatement de la formule de Taylor avec reste intégral, appliquée entre $\dfrac{k}{n}$ et $\dfrac{k-1}{n}$

(attention à l'ordre !) :

$$\forall t \in \left[\frac{k-1}{n}, \frac{k}{n}\right] \quad g(t) - g\left(\frac{k}{n}\right) - g'\left(\frac{k}{n}\right)\left(t - \frac{k}{n}\right) = \int_{k/n}^{(k-1)/n} \left(\frac{k-1}{n} - t\right) g''(t)\,\mathrm{d}t$$

d'où

$$\forall k \in [\![1,n]\!] \quad \forall t \in \left[\frac{k-1}{n}, \frac{k}{n}\right] \quad \left|g(t) - g\left(\frac{k}{n}\right) - g'\left(\frac{k}{n}\right)\left(t - \frac{k}{n}\right)\right| \leqslant \frac{M}{2n^2},$$

grâce à l'inégalité de la moyenne, du fait que

$$\int_{k/n}^{(k-1)/n}\left(\frac{k-1}{n}-t\right)\mathrm{d}t=\left[-\frac{1}{2}\left(\frac{k-1}{n}-t\right)\right]_{k/n}^{(k-1)/n}=\frac{1}{2n^2}.$$

De même,

$$\int_{(k-1)/n}^{k/n}\left(\frac{k}{n}-t\right)\mathrm{d}t=\frac{1}{2n^2},$$

donc, en utilisant la relation de Chasles et la majoration précédente, j'obtiens

$$\begin{aligned}\left|\int_0^1 g(t)\,\mathrm{d}t-\frac{1}{n}\sum_{k=1}^{n}g\left(\frac{k}{n}\right)+\frac{1}{2n^2}\sum_{k=1}^{n}g'\left(\frac{k}{n}\right)\right| &= \left|\sum_{k=1}^{n}\int_{(k-1)/n}^{k/n}\left[g(t)-g\left(\frac{k}{n}\right)-g'\left(\frac{k}{n}\right)\left(t-\frac{k}{n}\right)\right]\mathrm{d}t\right| \\ &\leqslant n\cdot\frac{1}{n}\cdot\frac{M}{2n^2}=\frac{M}{2n^2}\end{aligned}$$

d'où

$$n\left(\int_0^1 g(t)\,\mathrm{d}t-\frac{1}{n}\sum_{k=1}^{n}g\left(\frac{k}{n}\right)\right)\underset{n\to\infty}{=}-\frac{1}{2n}\sum_{k=1}^{n}g'\left(\frac{k}{n}\right)+o(1)$$

or je reconnais une nouvelle somme de Riemann, pour la fonction g' sur $[0,1]$, donc

$$-\frac{1}{2n}\sum_{k=1}^{n}g'\left(\frac{k}{n}\right)\underset{n\to\infty}{\longrightarrow}-\frac{1}{2}\int_0^1 g'(t)\,\mathrm{d}t=\frac{g(0)-g(1)}{2}.$$

En conclusion, en divisant par n :

$$\boxed{\int_0^1 g(t)\,\mathrm{d}t-\frac{1}{n}\sum_{k=1}^{n}g\left(\frac{k}{n}\right)\underset{n\to\infty}{=}\frac{g(0)-g(1)}{2n}+o\left(\frac{1}{n}\right).}$$

Ce résultat général donne dans notre cas particulier, où $g(0)=0$ et $g(1)=-\pi/4$,

$$\boxed{\frac{n}{\pi}(u_n-\ell)\underset{n\to\infty}{\longrightarrow}\frac{1}{8}=0,125\,!\,!}$$

4) Notons que $x>0$ est solution de (E_n) si et seulement si $x-n\ln x=0$. Or une étude rapide de $\varphi_n : x\mapsto x-n\ln x$ montre qu'elle décroît de $+\infty$ à $n(1-\ln n)$ sur $]0,n]$, puis croît de $n(1-\ln n)$ à $+\infty$ sur $[n,+\infty[$. Pour $n\geqslant 3$, $n(1-\ln n)<0$, donc φ_n s'annule exactement deux fois sur $\mathbb{R}^{+*}$, en vertu du théorème de la bijection monotone, une fois sur $]0,n[$ et une fois sur $]n,+\infty[$. De plus, $\varphi_n(1)=1>0$ et $\varphi_n(2)=2-n\ln 2<0$ pour $n\geqslant 3$, donc la solution dans $]0,n[$ appartient en fait à $]1,2[$; on la note u_n.

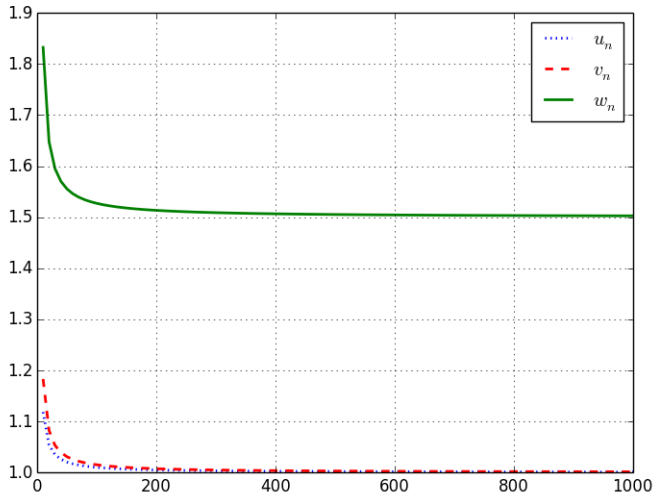
Voici un graphique permettant de visualiser le comportement de (u_n), où l'on conjecture que (u_n) converge vers 1, puis l'on examine (v_n), où $v_n=n(u_n-1)$, qui semble aussi converger vers 1, et enfin on regarde $\big(n(v_n-1)\big)$.

En effet, si $u_n\underset{n\to\infty}{=}\ell+\dfrac{a}{n}+\dfrac{b}{n^2}+o\left(\dfrac{1}{n^2}\right)$, alors

$$v_n=n(u_n-\ell)\underset{n\to\infty}{=}a+\frac{b}{n}+o\left(\frac{1}{n}\right)\underset{n\to\infty}{\longrightarrow}a$$

et

$$w_n = n\,(v_n - a) \underset{n\to\infty}{=} b + o\,(1) \underset{n\to\infty}{\longrightarrow} b$$



2.png

Et voici le code ayant fourni ledit graphique :

```
import scipy.optimize as resol
import matplotlib.pyplot as plt
from math import exp

def u(n):
    return resol.fsolve(lambda x: exp(x)-x**n,1)[0]# attention au retour de resol...
def v(n):
    return n*(u(n)-1)
def w(n):
    return n*(v(n)-1)

N=[10*k for k in range(1,101)]
plt.plot(N,[u(n) for n in N],'b:',linewidth=2,label='$u_n$')
plt.plot(N,[v(n) for n in N],'y--',linewidth=2,label='$v_n$')
plt.plot(N,[w(n) for n in N],'g-',linewidth=2,label='$w_n$')
plt.legend(loc=0)
plt.show()
```

Reste à prouver les conjectures qui sont apparues !

– Nous avons vu que, pour tout $n \geqslant 3$, φ_n est strictement positive sur $[1, u_n[$ et strictement négative sur $]u_n, 2]$. Or $u_{n+1} = (n+1)\ln u_{n+1}$, donc $\varphi_n\,(u_{n+1}) = \ln u_{n+1} > 0$; il en résulte que $u_{n+1} \in]1, u_n[$, c'est-à-dire que la suite (u_n) est strictement décroissante. Comme elle est minorée par 1, elle converge vers une limite $\ell \geqslant 1$. Enfin nécessairement $\ell = 1$, car si $\ell > 1$, alors $u_n = n \ln u_n$ tend vers

$+\infty$. Donc

$$\boxed{(u_n) \text{ converge vers } 1.}$$

– Il en résulte que $\ln u_n = \dfrac{u_n}{n} \sim \dfrac{1}{n}$; or $\ln x \underset{x\to 1}{\sim} x-1$, d'où $\ln u_n \sim u_n - 1$ puisque (u_n) converge vers 1. Ainsi, par transitivité, $u_n - 1 \sim \dfrac{1}{n}$, c'est-à-dire $n\,(u_n-1) \underset{n\to\infty}{\longrightarrow} 1$ (cf. les valeurs de v_n ci-dessus).

Cela signifie que : $u_n \underset{n\to\infty}{=} 1 + \dfrac{1}{n} + o\left(\dfrac{1}{n}\right)$.

– Je note alors $\varepsilon_n = u_n - 1$, de sorte que la relation $u_n = n \ln u_n$ s'écrit

$$\ln(1+\varepsilon_n) = \frac{u_n}{n} \underset{n\to\infty}{=} \frac{1}{n} + \frac{1}{n^2} + o\left(\frac{1}{n^2}\right).$$

Par ailleurs, comme $\varepsilon_n \underset{n\to\infty}{\longrightarrow} 0$,

$$\ln(1+\varepsilon_n) \underset{n\to\infty}{=} \varepsilon_n - \frac{\varepsilon_n^2}{2} + \varepsilon_n^2 z_n \quad \text{où} \quad z_n \underset{n\to\infty}{\longrightarrow} 0.$$

Or $\varepsilon_n \sim \dfrac{1}{n}$ d'après ce qui précède, donc $\varepsilon_n^2 \sim \dfrac{1}{n^2}$ et donc

$$\ln(1+\varepsilon_n) \underset{n\to\infty}{=} \varepsilon_n - \frac{1}{2n^2} + o\left(\frac{1}{n^2}\right)$$

d'où finalement

$$\varepsilon_n \underset{n\to\infty}{=} \frac{1}{n} + \frac{3}{2n^2} + o\left(\frac{1}{n^2}\right) \quad \text{soit} \quad u_n = 1 + \frac{1}{n} + \frac{3}{2n^2} + o\left(\frac{1}{n^2}\right)$$

5) Pour plus de modularité, je programme le calcul (approché !) du produit scalaire de deux polynômes, créés grâce au module `Polynomial`, l'intégrale étant approchée par `scipy.integrate.quad` :

Reconnaissez-vous certains codes ?

```
import scipy.integrate as integr
import scipy.optimize as opt
import matplotlib.pyplot as plt
import numpy as np
from numpy.polynomial import Polynomial
#pour un affichage raisonnable :
np.set_printoptions(precision=3)

def pscal(u,v):
    P,Q=Polynomial(u),Polynomial(v)
    return integr.quad(lambda t:P(t)*Q(t),-1,1)[0]#tuple...

def norme(u):
    return sqrt(pscal(u,u))

def GS(e): #e est une liste de vecteurs (famille libre)
    n=len(e)
    v=np.zeros((n,n))
    eps=np.zeros((n,n))
    v[0]=e[0]
    eps[0]=v[0]/norme(v[0])
    for k in range(1,n):
        v[k]=e[k]
        for j in range(k):
            v[k]=v[k]-pscal(eps[j],e[k])*eps[j]
        eps[k]=v[k]/norme(v[k])
    return eps #[eps[k] for k in range(n)]

e=GS(np.eye(6))
print(np.array([[pscal(e[i],e[j]) for j in range(6)] for i in range(6)]))
E=[Polynomial(e[i]) for i in range(6)]
X=np.linspace(-1,1,100)
plt.plot(X,E[5](X))
plt.show()
```

Le code ci-dessus applique l'algorithme de Gram-Schmidt, les vecteurs des familles initiale (`e`) et finale (`eps`) étant stockés dans les lignes des matrices du même nom. `numpy` traite une ligne de matrice comme un vecteur...

Je trace aussi le graphe de E_5. Pour les autres graphes, il suffit de remplacer le 5...

On constate que chaque $|E_i|$ semble atteindre son maximum en -1 et en 1.

La norme avec les doubles barres de l'énoncé est (implicitement !) la norme euclidienne associée au produit scalaire considéré. Supposons $P = \sum_{i=0}^{5} x_i E_i \in F$. J'utilise

les propriétés de la norme N_∞ et l'inégalité de Cauchy-Schwarz dans $\mathbb{R}^6$:

$$N_\infty(P) \leqslant \sum_{i=0}^{5} |x_i| \, N_\infty(E_i) \leqslant \sqrt{\sum_{k=0}^{5} x_i^2} \sqrt{\sum_{i=0}^{5} N_\infty(E_i)^2},$$

avec égalité si et seulement si le vecteur $(|x_i|)_{0\leqslant i\leqslant 5}$ est colinéaire au vecteur $\left(N_\infty(E_i)\right)_{0\leqslant i\leqslant 5}$.

Le cas particulier où $\|P\| = 1$, c'est-à-dire où $\sum_{k=0}^{5} x_i^2 = 1$ fournit le majorant $\sqrt{\sum_{i=0}^{5} N_\infty(E_i)^2}$.

Quant à la valeur dudit majorant, on peut en obtenir une valeur approchée grâce aux résultats fournis par la programme précédent, en utilisant le fait que $N_\infty(E_i)$ est la somme de ses coefficients (sa valeur en 1, résultat constaté expérimentalement...). On obtient bien approximativement $3\sqrt{2}$!

Les calculs exacts nous entraîneraient un peu loin... mais c'est un sujet classique ! Les E_i sont (à un coefficient près) les polynômes de Legendre et l'on peut montrer (via l'unicité de la famille fournie par Gram-Schmidt) que

$$E_i = \frac{1}{\|L_i\|}.L_i \quad \text{où} \quad L_i = \left[\left(X^2-1\right)^i\right]^{(i)}.$$

En effet, ces polynômes sont de degrés échelonnés et forment une famille orthogonale (cela se montre à l'aide d'intégrations par parties itérées...).

Enfin, nous venons de montrer que $b = 3\sqrt{2}$ convient et nous avons fourni des polynômes vérifiant l'égalité.

De l'autre côté, je constate que

$$\int_{-1}^{1} P^2 \leqslant 2N_\infty(P)^2$$

et donc $a = \sqrt{2}$ convient, avec égalité par exemple pour les polynômes constants.

6) Pour la partie résolution mathématique, c'est très classique, attention à la gomposition des fonctions.

On pose $g(t) = y(x)$, soit $g(x^2) = y(x)$.

On arrive à $g" + g = 0$. On utilise les conditions initiales.

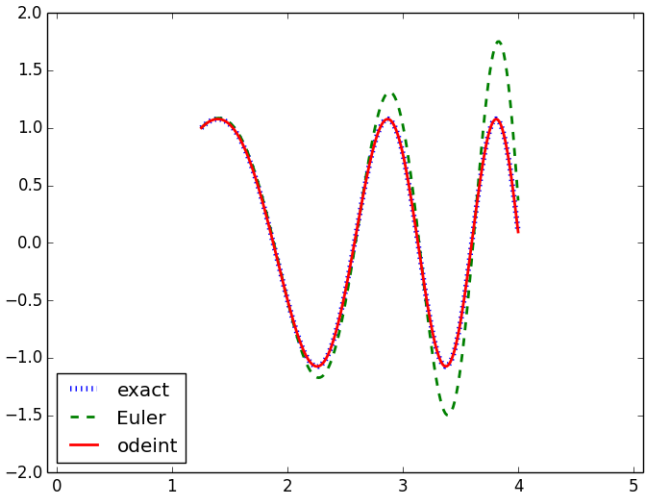
Pour une équation **scalaire d'ordre 2** $y'' = G(y, y', t)$, la méthode d'Euler comme `scipy.integrate.odeint` s'appliquent à l'équation **vectorielle d'ordre 1**

$$Y' = F(Y,t) \quad \text{où} \quad Y = \left[y, y'\right] \quad \text{et} \quad F(Y,t) = \left[Y[1], G(Y[0], Y[1], t)\right].$$

Pour la programmation "maison" on peut choisir, mais `odeint` impose l'ordre (Y, t) pour la fonction F. Vu en td , l'ordre des arguments de `odeint`...

On constate que la méthode (rudimentaire) d'Euler a du mal à converger.

Voici les graphiques avec 201, puis 501 points si vous le tapez, et enfin le code Python.



4.png

```
from math import pi,sin,sqrt
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

a=sqrt(pi/2)

#solution exacte#
def f(x):
    global a
    return np.sin(x**2)-np.cos(x**2)/(2*a)

X=np.linspace(a,4,201)# 201
plt.plot(X,f(X),'b:',linewidth=4,label='exact')

def F(Y,t):
    return np.array([Y[1],Y[1]/t-4*t**2*Y[0]])
N=201
T=np.linspace(a,4,N)

#Euler "maison"#
Y=np.zeros((N,2))
Y[0,:]=[1,1]
h=(4-a)/(N-1)
for k in range(N-1):
    Y[k+1,:]=Y[k,:]+h*F(Y[k,:],T[k])
Y0=Y[:,0]
plt.plot(T,Y0,'g--',linewidth=2,label='Euler')

#scipy.integrate.odeint#
Y=odeint(F,[1,1],T)[:,0]
plt.plot(T,Y,'r-',linewidth=2,label='odeint')

plt.axis('equal')
plt.legend(loc=0)
plt.show()
```

7) (1) Pour simuler une variable de Bernoulli, j'utilise `numpy.random.random()` (la fonction a le même nom que le module!!), qui renvoie un flottant (pseudo-)aléatoire de $[0, 1[$, c'est-à-dire que la répartition est uniforme. Ainsi la probabilité que le résultat soit inférieur à p vaut p. D'où les deux fonctions :

```
import numpy.random as rd

def suivant(x,p):
    if x in [0,3]: return x
    alea=rd.random()
    if alea<p:
        return x-1
    else:
        return x+1

def arrivee(n,x0,p):
    x=x0
    for k in range(n): x=suivant(x,p)
    return x
```

(2) Voici un exemple de code permettant d'engendrer un histogramme :

```
import numpy as np
import numpy.random as rd
import matplotlib.pyplot as plt
import numpy.linalg as alg
#pour un affichage raisonnable
#on peut limiter le nombre de décimales
np.set_printoptions(precision=3)

A=np.array([[1,0.5,0,0],[0,0,0.5,0],[0,0.5,0,0],[0,0,0.5,1]])

def suivant(x,p):
    if x in [0,3]: return x
    alea=rd.random()
    if alea<p:
        return x-1
    else:
        return x+1

def arrivee(n,x0,p):
    x=x0
    for k in range(n): x=suivant(x,p)
    return x

def essais(N,n,p):
    #loi uniforme pour le point de départ
    X0=rd.randint(0,4,N)
    X=[0,1,2,3]
    Y=[0,0,0,0]
    for k in range(N):
        Y[arrivee(n,X0[k],p)]+=1
    plt.bar(X,Y)
    plt.xticks(X)
    plt.show()

essais(1000,50,0.5)

D, P = alg.eig(A)
print(D)
print(P)
print(alg.inv(P).dot(A).dot(P))
```

J'utilise `numpy.random.randint` pour engendrer une liste aléatoire de positions initiales, puis j'utilise les fonctions du **a)** pour calculer le point d'arrivée à partir de chacune de ces positions intiales, après n itérations.

(3) La formule des probabilités totales donne classiquement

$$\boxed{A = \begin{pmatrix} 1 & p & 0 & 0 \\ 0 & 0 & p & 0 \\ 0 & 1-p & 0 & 0 \\ 0 & 0 & 1-p & 1 \end{pmatrix}.}$$

(4) Il vient $\chi_A(x) = (x-1)^2\left(x^2 - p(1-p)\right)$. De plus on constate facilement que $rg(A-I) = 2$, donc $E_1(A)$ est de dimension $4-2=2$. Par conséquent, si $p \in]0,1[$, A est diagonalisable puisqu'elle admet, outre la valeur propre 1, deux valeurs propres distinctes $\pm\sqrt{p(1-p)}$, donc la somme des dimensions des sous-espaces propres vaut bien 4.

En revanche, si $p \in \{0,1\}$, A admet 0 comme valeur propre de multiplicité 2, tandis que A est de rang 3, donc $E_0(A)$ n'est que de dimension 1 : A n'est pas diagonalisable dans ce cas.

$$\boxed{A \text{ est diagonalisable si et seulement si } p \in]0,1[.}$$

(5) Pour $p = 1/2$, Python permet la diagonalisation approchée avec `numpy.linalg.eig`, qui renvoie le vecteur des valeurs propres, ainsi que la matrice de passage vers une base de vecteurs propres (cf. la doc fournie par Centrale). Pour calculer les puissances de A, on peut utiliser cette diagonalisation ou bien directement `numpy.linalg.matrix_power`.

D'une manière ou d'une autre, Python permet de conjecturer que

$$A^n \underset{n\to\infty}{\longrightarrow} \begin{pmatrix} 1 & 2/3 & 1/3 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1/3 & 2/3 & 1 \end{pmatrix} \quad \text{d'où} \quad X_n \underset{n\to\infty}{\longrightarrow} \begin{pmatrix} p_0 + 2p_1/3 + p_2/3 \\ 0 \\ 0 \\ p_1 + 2p_2/3 + p_3 \end{pmatrix}$$

où p_k désigne $P(X_0 = k)$.

On peut vérifier expérimentalement le résultat en déterminant (ou en imposant !) les fréquences des valeurs 0, 1, 2, 3 dans la liste de valeurs initiales.

8) Je considère A fournie sous forme d'une liste et P par la liste de ses coefficients (terme constant en tête), ce qui me permet d'utiliser le module `Polynomial` pour l'évaluation. Je me permets d'utiliser la fonction `max` de Python :

```
from numpy.polynomial import Polynomial

def N(A,P):
    polyP = Polynomial(P)
    return max([abs(polyP(a)) for a in A])
```

Pour $A = (0,1)$ et a réel, j'ai

$$\|X+a\|_A = \max(|a|, |1+a|) = \begin{cases} -a > 1 & \text{si } a < -1 \\ 1+a > 1 & \text{si } a > 0 \\ \max(-a, 1+a) & \text{si } a \in [-1,0] \end{cases}.$$

Il en résulte que le minimum est atteint pour $a = -1/2$:

$$\boxed{\min_{a\in\mathbb{R}} \|X + a\|_A = 1/2.}$$

Pour $A = (0, 1, 2)$, l'étude à la main serait plus longue, d'où l'idée d'utiliser `fmin`. Attention, contrairement à ce qui est écrit dans l'ODLT, la fonction Python doit être une fonction d'un variable, vectorielle si besoin. D'où le code, en complément de `N(A,P)` déjà définie :

```
from scipy.optimize import fmin

def f(v):
    return N([0,1,2], [v[1], v[0], 1])
```

Bizarrement, le point initial $(0, 0)$ donne un résultat aberrant, alors que tous les autres essais m'ont donné un minimum voisin de 0.5, obtenu pour un couple (a, b) proche de $(-2, 0.5)$.

Notons que ces deux premiers résultats correspondent à la recherche de d_1 et d_2. La fin de l'énoncé permet de démontrer le résultat général.

Posons, pour tout k dans $[\![0, n]\!]$,

$$B_k = \prod_{j\in[\![0,n]\!]\setminus\{k\}} (X - a_j)\,.$$

Les B_k sont $n + 1$ polynômes de $\mathbb{R}_n[X]$ et ils forment une famille libre : pour s'en convaincre (cf. les polynômes de Lagrange !), supposons une famille $(b_0, \ldots, b_n)$ de scalaires tels que $\sum_{i=0}^{n} b_i B_i$ soit le polynôme nul. En évaluant en a_k j'obtiens $b_k B_k(a_k) = 0$, puisque par construction a_k est racine de B_i pour tout i distinct de k. Or $B_k(a_k)$ est non nul puisque par hypothèse les a_j sont distincts deux à deux. D'où $b_k = 0$, cela pour tout k.

$(B_0, \ldots, B_n)$ est donc une base de $\mathbb{R}_n[X]$ (famille libre de $n+1$ vecteurs, $n+1$ étant la dimension de $\mathbb{R}_n[X]$!).

Pour P fixé dans $\mathbb{R}_{n-1}[X]$, les coordonnées de $X^n + P$ dans cette base forment donc l'unique famille demandée dans l'énoncé.

Comme tous les B_k sont unitaires et de degré n, $\sum_{k=0}^{n} b_k$ n'est autre que le coefficient de X^n dans $\sum_{k=0}^{n} b_k B_k$, ainsi par définition

$$\boxed{\sum_{k=0}^{n} b_k = 1.}$$

Regardons maintenant de plus près $|B_k(a_k)|$, pour k fixé. Comme $a_0 < \cdots < a_n$ j'ai (un produit vide valant 1)

$$|B_k(a_k)| = \prod_{j=0}^{k-1} (a_k - a_j) \times \prod_{j=k+1}^{n} (a_j - a_k)\,.$$

De plus, les a_j étant des entiers distincts, l'écart entre deux valeurs consécutives de la famille est au moins égal à 1 ! Une récurrence immédiate montre alors que

$$0 \leqslant p \leqslant q \leqslant n \Rightarrow a_q - a_p \geqslant q - p.$$

Il en résulte

$$|B_k(a_k)| \geqslant \prod_{j=0}^{k-1} (k-j) \times \prod_{j=k+1}^{n} (j-k) = k!\,(n-k)!$$

Autrement dit

$$\boxed{|B_k(a_k)| \geqslant \frac{n!}{\binom{n}{k}}.}$$

Je note $Q = X^n + P$. Pour tout k dans $[\![0, n]\!]$, j'ai par définition des B_k

$$|Q(a_k)| = |b_k B_k(a_k)| \geqslant |b_k| \frac{n!}{\binom{n}{k}}$$

d'où par définition de $\|\cdot\|_A$

$$n!\,|b_k| \leqslant \binom{n}{k} \|Q\|_A$$

cela pour tout k, d'où par sommation

$$n! \sum_{k=0}^{n} |b_k| \leqslant 2^n \|Q\|_A .$$

Or l'inégalité triangulaire donne

$$1 = \left| \sum_{k=0}^{n} b_k \right| \leqslant \sum_{k=0}^{n} |b_k|$$

d'où finalement

$$n! \leqslant 2^n \|Q\|_A$$

et en conclusion

$$\boxed{\|X^n + P\|_A \geqslant \frac{n!}{2^n}.}$$

Cela étant vrai pour tout P de $\mathbb{R}_{n-1}[X]$, j'en déduis que $d_n \geqslant \dfrac{n!}{2^n}$.

Dans le cas $A = (0, 1, \ldots, n)$ (*i.e.* $a_k = k$ pour tout k), il suffirait de trouver P permettant d'atteindre ce minorant, qui serait alors le plus petit élément ! Essayons avec

$$P = \sum_{k=0}^{n} b_k B_k - X^n \quad \text{où} \quad b_k = \frac{\binom{n}{k}}{2^n} \quad \text{pour tout } k \text{ de } [\![0, n]\!].$$

Par construction $P \in \mathbb{R}_{n-1}[X]$ et les minorations ci-dessus sont des égalités, du fait que $a_k = k$:

$$\forall k \in [\![0, n]\!] \quad |k^n + P(k)| = \frac{\binom{n}{k}}{2^n} |B_k(k)| = \frac{n!}{2^n}$$

d'où $\|X^n + P\|_A = \dfrac{n!}{2^n}$. Donc dans ce cas $d_n \leqslant \dfrac{n!}{2^n}$ d'où finalement

$$\boxed{\text{Lorsque } A = (0, 1, \ldots, n),\ d_n = \frac{n!}{2^n}.}$$

9) Voir la documentation sur le module `Polynomial` et bien penser à mettre le terme constant en tête de liste (alors que `np.poly` renvoie les coefficients du polynôme caractéristique avec le coefficient dominant en tête...).

```
from numpy.polynomial import Polynomial

def Q(p):
    L = [-k-1 for k in range(2*p)]
    L.append(0)
    L.append(2*p+2)
    return Polynomial(L)

for p in [3,4,5,10]:
    print(Q(p).roots())
```

On peut conjecturer l'existence d'une unique racine réelle, qui semble de plus positive.

Avec `pyplot` je trace le cercle unité comme un arc paramétré et je place un point pour chaque racine par un simple `plot`, en récupérant les parties réelles comme abscisses et les parties imaginaires comme ordonnées. Ne pas oublier l'option `linestyle=''` pour empêcher le tracé d'une ligne brisée ! L'option `marker` permet de choisir le symbole affiché pour chaque point représenté.

```
import matplotlib.pyplot as plt
import numpy as np

T = np.linspace(0, 2*np.pi, 100)
plt.axis('equal')
plt.plot(np.cos(T), np.sin(T))
for p in range(30):
    R = Q(p).roots()
    plt.plot(np.real(R), np.imag(R), color='r', marker='.', linestyle='')
plt.show()
```

Pour le minimum des modules, j'utilise la forme vectorisée `np.abs` qui applique `abs` à tous les éléments d'une liste (ou d'un tableau `numpy`) et je me permets d'utiliser la fonction `min` de Python. J'utilise `round` pour éviter d'encombrer l'affichage avec des décimales superflues.

```
def module_min(p):
    return min(np.abs(Q(p).roots()))

print([round(module_min(p), 3) for p in range(1, 30)])
```

On peut conjecturer que ce minimum est croissant et majoré par 1 (la limite 1 n'est pas évidente...). Cela explique l'allure du graphique précédent. Si tu le veux, tu tapes !

Les fractions rationnelles n'étant pas au programme, je vais plutôt manipuler les fonctions polynomiales. Fixons p dans $\mathbb{N}^*$.

$$\forall x \in \mathbb{R}^* \quad -x^{2p+1}Q_p\left(1/x\right) = x^{2p+1} + 2x^{2p} + \cdots + 2px^2 - (2p+2).$$

Cette expression définit donc bien un polynôme, noté R_p. La fonction polynomiale associée est clairement strictement croissante sur $\mathbb{R}^+$. Or $R_p\left(0\right) < 0$ et $R_p\left(x\right) \underset{x\to+\infty}{\longrightarrow} +\infty$, donc le théorème de la bijection montre que R_p admet une unique racine dans $\mathbb{R}^{+*}$. Il en est donc de même de $Q_p : t \mapsto -t^{2p+1}R_p\left(1/t\right)$:

$$\boxed{Q_p \text{ admet une unique racine dans } \mathbb{R}^{+*}.}$$

L'expression de $Q_p\left(x\right)$ fait penser à une dérivée :

$$2px^{2p-1} + (2p-1)\,x^{2p-2} + \cdots + 2x + 1 = \frac{\mathrm{d}}{\mathrm{d}x}\left(\sum_{k=0}^{2p} x^k\right)$$

d'où pour $x \neq 1$

$$\begin{aligned} 2px^{2p-1} + (2p-1)\,x^{2p-2} + \cdots + 2x + 1 &= \frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{1-x^{2p+1}}{1-x}\right) \\ &= \left(1-x^{2p+1}\right)\cdot\frac{1}{(1-x)^2} - (2p+1)\,x^{2p}\cdot\frac{1}{1-x} \end{aligned}$$

ce qui donne bien

$$\boxed{\forall x \neq 1 \quad Q_p\left(x\right) = (2p+2)\,x^{2p+1} - \frac{1-x^{2p+1}}{(1-x)^2} + \frac{(2p+1)\,x^{2p}}{1-x}.}$$

En particulier

$$Q_p\left(2\right) = (2p+3)\,2^{2p+1} - 1 - (2p+1)\,2^{2p} \underset{p\to\infty}{\sim} p2^{2p+1} \underset{p\to\infty}{\longrightarrow} +\infty$$

et

$$Q_p\left(\frac{3}{2}\right) = (2p+6)\left(\frac{3}{2}\right)^{2p+1} - 4 - 2\,(2p+1)\left(\frac{3}{2}\right)^{2p} \underset{p\to\infty}{\sim} -p\left(\frac{3}{2}\right)^{2p+1} \underset{p\to\infty}{\longrightarrow} -\infty.$$

Par conséquent, à partir d'un certain rang $Q_p\left(3/2\right) < 0$ et $Q_p(2) > 0$, d'où

$$\boxed{\alpha_p \in \left]\frac{3}{2}, 2\right[\text{ à partir d'un certain rang.}}$$

Numériquement, il apparaît que c'est à partir du rang 7.

Noter que l'on n'a rien prouvé concernant d'éventuelles racines réelles négatives, même si les résultats numériques laissent penser qu'il n'y en a pas...

10) Notons $E = \mathbb{R}^n$. Pour le premier résultat théorique, il suffit de remarquer que (N) équivaut à

$$\forall\left(X,Y\right) \in E^2 \quad \langle AX, AY\rangle = \langle BX, BY\rangle$$

(le sens non trivial s'obtenant grâce à une identité de polarisation...). Puis par bilinéarité, en notant $\mathcal{B} = (e_1, \dots, e_n)$ la base canonique de E, j'en déduis que (N) équivaut à

$$\forall (i,j) \in [\![1,n]\!]^2 \quad \langle Ae_i, Ae_j \rangle = \langle Be_i, Be_j \rangle .$$

Or pour $M \in \mathcal{M}_n(\mathbb{R})$, Me_k n'est autre que $C_k(M)$, k-ième vecteur colonne de M. Ainsi

$$\boxed{(N) \Leftrightarrow \forall (i,j) \in [\![1,n]\!]^2 \quad \langle C_i(A), C_j(A) \rangle = \langle C_i(B), C_j(B) \rangle .}$$

Pour la (petite) partie Python, j'utilise la fonction `vdot` de `numpy`, qui effectue le produit scalaire (canonique) de deux vecteurs `numpy`. Reste à se rappeler que `len(A)` renvoie le nombre de lignes d'une matrice `numpy`, qui correspond donc à la taille de la matrice lorsqu'elle est carrée, et que `A[:,j]` permet d'extraire la colonne `j` **sous forme de vecteur** (alors que `A[:,[j]]` renvoie une matrice à `n` lignes et une colonne, comme expliqué dans la documentation fournie par Centrale...).

```
import numpy as np

def d(A, B):
    s = 0
    n = len(A)
    for i in range(n):
        for j in range(i+1,n):
            s += (np.vdot(A[:,i], A[:,j]) - np.vdot(B[:,i], B[:,j]))**2
    return s

def verifientN(A, B):
    return abs(d(A, B)) < 1e-5
```

On proscrit bien sûr le test `==0` pour un flottant !!

Pour la dernière question, il suffit de remarquer que, lorsque A est inversible, en posant $Y = AX$

$$\|AX\| = \|BX\| \Leftrightarrow \|Y\| = \left\|BA^{-1}Y\right\|$$

donc A et B vérifient (N) si et seulement si BA^{-1} est une matrice orthogonale, autrement dit

$\boxed{A \text{ et } B \text{ vérifient } (N) \text{ si et seulement s'il existe } \Omega \in O_n(\mathbb{R}) \text{ telle que } B = A\Omega.}$

11) Pour la simulation d'un tournoi, j'utilise une boucle `while` (en conjecturant qu'il est presque impossible qu'il n'y ait jamais de vainqueur... On pourrait ajouter un "garde-fou" pour forcer la sortie de la boucle lorsque le vainqueur se fait attendre "trop longtemps"). J'initialise à 1 le nombre de matchs disputés et le nombre de victoires accumulées par le vainqueur du match précédent, puis je mets à jour en fonction du résultat de chacun des matchs suivants.

Le calcul de moyenne est banal. Attention tout de même : `1e4` est un flottant et ne peut donc être utilisé tel quel dans `range`... Voir le code page suivante.

Deux essais :

$\boxed{\text{Arrondi à 2 décimales, } \texttt{moyenne(3, 0.3)} \text{ renvoie } \texttt{4.47} \text{ et } \texttt{moyenne(6, 0.5)} \text{ renvoie } \texttt{63.3}.}$

Par définition, il ne peut pas y avoir de vainqueur avant le match N et il y a un vainqueur au match N si et seulement si le vainqueur du match 1 remporte les $N-1$ matchs suivants ; d'où par passage au complémentaire :

$$\boxed{\forall n \in [\![1, N-1]\!] \quad P(A_n) = 1 \quad \text{et} \quad P(A_N) = 1 - q^{N-1}.}$$

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

def T(N, p):
    nbm, nbv = 1, 1#nombre de matchs, de victoires consécutives
    while nbv < N:
        if rd.random() <= 1-p:
            nbv += 1
        else:
            nbv = 1
        nbm += 1
    return nbm

def moyenne(N, p):
    s = 0
    for k in range(10000):
        s += T(N, p)
    return s*1e-4
```

Après rectification de l'énoncé de l'ODLT, je considère l'événement $A_{n-1} \backslash A_n$, qui est constitué des issues pour lesquelles il n'y a pas de gagnant au match $n-1$ et il y a un gagnant au match n. En fait, par définition le tournoi s'arrête lorsqu'il y a un vainqueur

$\boxed{A_{n-1} \backslash A_n \text{ n'est autre que l'événement “le tournoi se termine au match } n\text{”}.}$

De plus, toujours par définition, s'il n'y a pas de vainqueur au match n, nécessairement il n'y avait pas non plus de vainqueur au match $n-1$, c'est-à-dire que $A_n \subset A_{n-1}$. Par conséquent

$$P(A_{n-1} \backslash A_n) = P(A_{n-1}) - P(A_n).$$

Or pour $n > N$, cet événement $A_{n-1} \backslash A_n$ peut aussi se décrire ainsi : “il n'y a pas de vainqueur au match $n-N$ et ce match est remporté par le joueur J_{n-N} et ledit joueur remporte les $N-1$ matchs suivants”. C'est en effet la seule possibilité pour qu'il y ait un gagnant exactement au match n. D'où par indépendance :

$$\boxed{\forall n > N \quad P(A_{n-1}) - P(A_n) = pq^{N-1} P(A_{n-N}).} \tag{1}$$

Comme $P(A_{n-N}) \in [0, 1]$, j'en déduis que $P(A_{n-1}) - P(A_n)$ est un $O\left(q^N\right)$; ainsi, par comparaison à une série géométrique, puisque $q \in]0, 1[$:

$\boxed{\text{La série de terme général } P(A_{n-1}) - P(A_n) \text{ est absolument convergente.}}$

Or la convergence de cette série équivaut à la convergence de la suite de terme général $P(A_n)$:

$$\boxed{\text{La suite de terme général } P(A_n) \text{ converge.}}$$

Notons ℓ sa limite. La relation (1) donne, à la limite quand n tend vers l'infini, $0 = pq^{N-1}\ell$: nécessairement $\ell = 0$. Or (1) donne également en remplaçant n par $N+k$:

$$\forall k > 0 \quad P(A_k) = \frac{1}{pq^{N-1}} \left[P(A_{k+N-1}) - P(A_{k+N}) \right]$$

d'où (somme télescopique) :

$$\forall K > 0 \quad \sum_{k=1}^{K} P(A_k) = \frac{1}{pq^{N-1}} \left[P(A_N) - P(A_{K+N}) \right].$$

Je retrouve ainsi la convergence de la série et j'obtiens la valeur de sa somme (en faisant tendre K vers l'infini).

$$\boxed{P(A_n) \underset{n\to\infty}{\longrightarrow} 0 \quad \text{et} \quad \sum_{n=1}^{\infty} P(A_n) = \frac{1}{pq^{N-1}} P(A_N) = \frac{1}{pq^{N-1}} \left[1 - q^{N-1}\right].}$$

Or, en admettant que le numéro $T_{N,p}$ du match ou le vainqueur du tournoi est proclamé est une variable aléatoire, j'ai par définition $A_n = (T_{N,p} > n) = (T_{N,p} \geqslant n+1)$. D'où

$$\forall n \geqslant 2 \quad P(T_{N,p} \geqslant n) = P(A_{n-1}).$$

Un (fameux) théorème du cours nous indique alors que $T_{N,p}$ est d'espérance finie et que

$$E(T_{N,p}) = \sum_{n=1}^{\infty} P(T_{N,p} \geqslant n) = 1 + \sum_{n=2}^{\infty} P(A_{n-1}) = 1 + \frac{1}{pq^{N-1}} \left[1 - q^{N-1}\right].$$

En conclusion

$$\boxed{E(T_{N,p}) = 1 + \frac{1}{p}\left(\frac{1}{q^{N-1}} - 1\right).}$$

On obtient avec plaisir (cf. les simulations)

$$E(T_{3,0.3}) \approx 4.47 \quad \text{et} \quad E(T_{6,0.5}) = 63.$$

Enoncés II

1. (Centrale) On définit (f_n) sur $[0,1]$ par

$$f_0(x) = 1 \quad \text{et} \quad f_{n+1}(x) = 2\int_0^x \sqrt{f_n(t)}\mathrm{d}t$$

Calculer f_1 et f_2 puis montrer que $f_n(x)$ est de la forme $\alpha_n x^{\beta_n}$.

Déterminer les relations de récurrence entre les termes des suites (α_n) et (β_n).

Calculer β_n et donner la limite de la suite β_n.

Écrire un programme Python qui retourne les 30 premiers termes de la suite (α_n) ;

conjecturer la limite.

Montrer que $\ln \alpha_n = -\sum_{k=1}^{n} 2^{-k+1} \ln\left(1 - 2^{-n-1+k}\right)$ et en déduire la limite de α_n.

En déduire que (f_n) converge uniformément sur $[0,1]$.

2. (Centrale) On pose $P_0 = 1, P_1 = 2X$ et $P_{n+1}(X) = 2XP_n(X) - P_{n-1}(X)$.

Avec Python, calculer P_n pour $n \in [\![2,8]\!]$.

Conjecturer la parité, le degré et le coefficient dominant de P_n; démontrer ces conjectures.

Montrer que $(P,Q) \mapsto (P \mid Q) = \displaystyle\int_{-1}^{1} \sqrt{1-t^2}P(t)Q(t)\mathrm{d}t$ définit un produit scalaire sur $E = \mathbb{R}_8[X]$.

Avec Python, calculer $(P_i \mid P_j)$ pour $0 \leqslant i,j \leqslant 8$. Qu'en déduire pour (P_n) ?

Exprimer la matrice de $\phi : P \mapsto 3XP' - P''$ dans la base des P_n.

3. (Centrale) À une liste $L = [a_0, \ldots, a_{n-1}]$ on associe la matrice $M = (m_{i,j})_{1 \leqslant i,j \leqslant n}$ vérifiant

$$m_{i,j} = 1 \quad \text{si } i = j+1, \quad m_{i,n} = -a_{i-1} \quad \text{et} \quad m_{i,j} = 0 \quad \text{sinon.}$$

Créer un programme Python Matrice(L) qui renvoie la matrice M liée à la liste $L = [a_0, \ldots, a_{n-1}]$

Calculer le polynôme caractéristique de M si L est une liste aléatoire de 8 entiers de $[\![1,10]\!]$ on pourra utiliser la fonction poly du module numpy).

Que peut-on conjecturer ? Calculer le polynôme caractéristique de M dans le cas général.

Montrer que si P est un polynôme de degré strictement inférieur à n, alors $P(M) \neq 0$.

En déduire une condition nécessaire et suffisante pour que M soit diagonalisable.

4. (Centrale) On note $\sigma(n)$ le cardinal de $H_n = \left\{(p,q) \in \mathbb{N}^2 / 2p + 3q = n\right\}$.

Justifier l'existence de $\sigma(n)$ pour tout entier naturel n. Expliciter $\sigma(0), \sigma(1)$ et $\sigma(2)$; montrer que, pour $n \geqslant 3, \sigma(n) \geqslant 1$. Écrire une fonction Python qui calcule $\sigma(n)$ et donner les valeurs pour $n \in [\![0, 25]\!]$. Calculer le rayon de convergence de $\sum \sigma(n)x^n$ et montrer que, en cas de convergence, on a

$$\sum_{n=0}^{\infty} \sigma(n)x^n = \frac{1}{(1-x^2)(1-x^3)}$$

Écrire une fonction Python basée sur cette série entière permettant de retrouver les valeurs précédentes.

5. (ENSAM) Tracer le graphe de $f : x \mapsto \dfrac{1}{1+x^2}$ sur $[-5, 5]$.

Donner la définition des polynômes de Lagrange.

Écrire une fonction coef (h, a) renvoyant la liste des coefficients du polynôme d'interpolation de la fonction h associé à la liste a des points d'interpolation (on pourra déterminer numériquement les coefficients à l'aide de la commande solve du module numpy. Iinalg). Écrire une fonction evalP(L, x) qui évalue en x le polynôme dont les coefficients sont donnés dans la liste L.

Écrire une fonction affiche (j) qui affiche les graphes sur $[-5, 5]$ de f et du polynôme de Lagrange associé pour une liste de $j+1$ valeurs équiréparties dans $[-5, 5]$, puis tester cette fonction pour $j \in \{5, 10, 15, 20\}$. Commenter Tester avec la fonction $f : x \mapsto |x|$. Commenter.

6. (ENSAM) Une balle se déplace dans une série de $n+1$ cases ; au début de l'expérience, elle se trouve dans la première et, à chaque étape, avance d'une case ou reste immobile avec équiprobabilité. Écrire une fonction tirer(n) qui renvoie la position finale de la balle (on pourra utiliser random, du module random). On réitère N fois l'expérience ; donner une fonction Python qui prend en arguments n, N et qui renvoie la liste indexée par k de la proportion de balles étant à la case k en position finale. Tracer la courbe donnant cette proportion en fonction de k. On prendra $n = 10$ et $N = 100$. Tracer la courbe théorique, après avoir écrit une fonction renvoyant $\begin{pmatrix} n \\ k \end{pmatrix}$. La probabilité que la balle avance d'une case est désormais p; modifier le programme et faire quelques essais.

7. (ENSAM) On veut générer la liste des nombres premiers. Créer une liste L30 de 31 True.

Écrire l'algorithme modifier (L, p) tel que, si $p = 0$, le premier terme de L est mis à False, si $p = 1$ le deuxième terme de L est mis à False et sinon le k -ième terme de L est mis à False

dès que $k > p$ est un multiple de p.

Le tester sur L30 pour quelques valeurs de p.

Générer la liste des nombres premiers jusqu'à n, en écrivant une fonction premiers (n).

Dire si 823417 est premier.

8. (ENSAM) On dispose d'un fichier texte Pi.txt contenant les 10000 premières décimales de π.

Lire le fichier et le convertir en chaine de caractères.

Afficher les 10 premières et les 10 dernières décimales.

Écrire un programme permettant de vérifier si une chaine de caractères motif est incluse dans une chaine source et qui retourne l'indice du premier caractère si c'est le cas. Voir si les chaines $000, 0000, 123, 1234, 314$, appartiennent aux 10000 premières décimales de π.

9. (ENSAM) Écrire une fonction chiffres qui retourne la liste de chacun des chiffres d'un entier n, ordonnée selon l'ordre des chiffres du nombre. Un nombre de p chiffres est dit narcissique si la somme de ses chiffres élevée à la puissance p vaut le nombre lui-même ; par exemple 81 est narcissique. Écrire une fonction d'argument n qui répond en booléen à la question : n est-il narcissique ?

Retourner tous les nombres narcissiques entre 0 et 10000 .

10. (Centrale) On pose $u_n = \left(e - \sum_{k=0}^{n-1} \frac{1}{k!}\right)^{1/n}$ pour $n \in \mathbb{N}^*$.

Avec Python, donner une valeur approchée de nu_n, pour $10 \leqslant n \leqslant 30$. Que constate-t-on ?

Pour améliorer la précision des calculs, on peut utiliser le module mpmath : après son chargement par from mpmath import mp et l'instruction mp. dps = 50, les calculs sont faits avec 50 chiffres significatifs, à condition d'utiliser les versions multiprécision des fonctions usuelles : mp.exp, mp.factorial, etc. Montrer que :

$$\forall n \in \mathbb{N}^* \quad \exists c_n \in [0,1] \quad e - \sum_{k=0}^{n-1} \frac{1}{k!} = \frac{e^{c_n}}{n!}$$

À l'aide de la formule de Taylor avec reste intégral, déterminer un développement

asymptotique à deux termes de c_n.

En déduire un développement asymptotique à trois termes de u_n.

11. (Centrale) Soit $u_n = \sin\left(\pi(2+\sqrt{3})^n\right)$.

Avec Python, calculer une valeur approchée des 30 premiers termes de $S_n = \sum_{k=0}^{n} u_k$.

Que peut-on conjecturer quant à la série de terme général u_n?

Étudier cette conjecture en comparant u_n à $v_n = \sin\left(\pi(2-\sqrt{3})^n\right)$.

12. ($ENSAM$) On veut tracer la courbe $\mathcal{C}$ d'équation $f(x,y) = 0$,

où $f(x,y) = x^4 + 2y^2 + 2xy - 1$.

Écrire une fonction def f(u), où u est un vecteur numpy de composantes x, y, renvoyant $f(x,y)$ Écrire de meme une fonction def nablaf (u) qui renvoie grad $f(x,y)$.

Écrire une fonction def points (A0, pas, n) qui renvoie la liste des points A_k construits par récurrence à partir de $A_0 \in \mathcal{C}$ par

$$\forall k \in [0, n-1]\!] \quad A_{k+1} = A_k + pas * \vec{v}$$

où $\vec{v}$ est approximativement un vecteur tangent à $\mathcal{C}$ en A_k, lui-meme approximativement sur $\mathcal{C}$. En déduire un tracé approximatif de $\mathcal{C}$.

Comment améliorer la précision en utilisant le développement limité à l'ordre 1 de f au voisinage de A_{k+1}?

13. (Centrale) La probabilité d'obtenir Pile en lançant une pièce est $p \in]0,1[$; on note E_n l'événement : ne jamais obtenir deux Pile d'affilée au cours des n premiers lancers $\geqslant$ et p_n sa probabilité. Écrire un programme qui prend n et p comme paramètres et renvoie True ou False selon que E_n est réalisé ou pas. Montrer que : $p_{n+2} = (1-p)p_{n+1} + p(1-p)p_n$ et en déduire que l'événement " obtenir deux Pile d'affilée sur une infinité de lancers est presque sûr".

Programmer le calcul par récurrence de p_n et comparer le résultat à celui d'une simulation. On note T la variable aléatoire qui donne le plus petit $n \geqslant 2$ tel que l'on obtienne Pile aux lancers de rangs $n-1$ et n. Donner la fonction génératrice de T. Montrer que T admet une espérance, la calculer et vérifier grâce à une simulation.

14. Dés de Sicherman : on lance deux dés cubiques équilibrés, dont les faces portent respectivement les valeurs $(1,2,2,3,3,4)$ et $(1,3,4,5,6,8)$. On note S la variable aléatoire prenant pour valeur la somme des deux résultats fournis par lesdits dés (on pourra utiliser le module Polynomial).

a) Déterminer la loi de probabilité de S. Commenter.

b) Existe-t-il d'autres valeurs dans $\mathbb{N}^*$, pour les faces des deux dés,

conduisant à la même loi pour la somme ?

On remarquera que

$$\sum_{k=1}^{6} x^k = x(x+1)\left(x^2+x+1\right)\left(x^2-x+1\right)$$

et l'on caractérisera les polynômes correspondant à la fonction génératrice du résultat donné par un dé équilibré dont les faces portent un entier naturel non nul.

Corrigés II

1) (Centrale) Le début, c'est des maths... Il vient immédiatement

$$f_1(x) = 2x \text{ et } f_2(x) = 2\int_0^x \sqrt{2t}\ \mathrm{d}t = 2\sqrt{2}\frac{1}{1/2+1}x^{1/2+1}$$

et

$$\text{si } f_n(x) = \alpha_n x^{\beta_n}, \text{ alors } f_{n+1}(x) = 2\sqrt{\alpha_n}\frac{1}{\beta_n/2+1}x^{\beta_n/2+1}\ .$$

Je définis donc les deux suites (α_n) et (β_n) par

$$\alpha_0 = 1, \beta_0 = 0 \text{ et } \forall n \in \mathbb{N} \quad \alpha_{n+1} = \frac{2\sqrt{\alpha_n}}{\beta_n/2+1} \text{ et } \beta_{n+1} = \beta_n/2+1$$

Une récurrence immédiate (compte tenu du calcul précédent) montre alors que

$$\forall n \in \mathbb{N} \quad \forall x \in [0,1] \quad f_n(x) = \alpha_n x^{\beta_n}$$

La suite (β_n) est arithmético-géométrique. La recherche d'un point fixe t tel que $t = t/2+1$ donne $t = 2$ et donc (par soustraction membre à membre) la suite $(\beta_n - 2)$ est géométrique, de raison $1/2$ et de premier terme -2. D'où

$$\forall n \in \mathbb{N} \quad \beta_n = 2 - 1/2^{n-1} \cdot (\beta_n) \text{ converge vers } 2$$

Je peux maintenant réécrire la définition de (α_n) :

$$\alpha_0 = 1 \text{ et } \forall n \in \mathbb{N}^* \quad \alpha_n = \frac{\sqrt{\alpha_{n-1}}}{1-1/2^n} \text{ puisque } \beta_{n-1}/2+1 = 2-1/2^{n-1}$$

Une boucle Python permet alors d'obtenir les valeurs demandées : je construis une liste contenant les valeurs successives, en ajoutant à chaque étape la nouvelle valeur à l'aide de la méthode append.

```
from math import sqrt

def a(n):
    v=[1.]
    for k in range(1,n):
        v.append(sqrt(v[k-1])/(1-2**(-k)))
    return v

print(a(30))
```

On peut conjecturer que (α_n) converge vers $1\ldots$ La fin, c'est à nouveau des maths ! D'après ce qui précède, les α_n sont tous strictement positifs et

$$\forall n \in \mathbb{N}^* \quad -\ln \alpha_n = -\frac{1}{2}\ln \alpha_{n-1} + \ln\left(1-2^{-n}\right)$$

La relation de l'énoncé s'en déduit par récurrence. J'utilise ensuite la majoration classique

$$\forall t > 0 \quad \ln t \leqslant t-1$$

pour écrire

$$\forall k \quad 0 \leqslant \ln \frac{1}{1-2^{-n-1+k}} \leqslant \frac{1}{1-2^{-n-1+k}} - 1 = \frac{2^{-n-1+k}}{1-2^{-n-1+k}}$$

d'où

$$0 \leqslant \ln \alpha_n \leqslant \sum_{k=1}^{n} \frac{2^{-n}}{1-2^{-n-1+k}} = \sum_{k=1}^{n} \frac{1}{2^n - 2^{k-1}} \leqslant \frac{n}{2^{n-1}}$$

puisque pour $k \leqslant n, 2^n - 2^{k-1} \geqslant 2^n - 2^{n-1} = 2^{n-1}$.

En conclusion, par croissances comparées et théorème d'encadrement, $\ln \alpha_n$ tend vers 0, autrement dit (α_n) converge vers 1.

Il résulte de ce qui précède que la suite de fonctions (f_n) converge simplement vers $f : x \mapsto x^2$. Pour étudier la convergence uniforme, je fixe $n > 0$ et j'écris, pour tout x dans $[0, 1]$,

$$|f_n(x) - f(x)| = \left|\alpha_n x^{\beta_n} - x^2\right| = \left|(\alpha_n - 1)\, x^{\beta_n} + x^{\beta_n} - x^2\right| \leqslant |\alpha_n - 1| + \left|x^{\beta_n} - x^2\right|$$

Comme $\beta_n < 2$, une étude rapide montre que la fonction $\phi_n : x \mapsto x^{\beta_n} - x^2$ est à valeurs positives sur $[0, 1]$ et atteint son maximum M_n en x_n défini par

$$\ln x_n = 2^{n-1} \ln\left(1 - \frac{1}{2^n}\right) \sim -\frac{1}{2}$$

J'en déduis que $(\beta_n \ln x_n)$ converge vers -1 et donc que

$$M_n = \phi_n(x_n) = x_n^{\beta_n} - x_n^2 \underset{n\to\infty}{\longrightarrow} e^{-1} - e^{-1} = 0$$

Or d'après la majoration précédente $\sup |f_n - f| \leqslant |\alpha_n - 1| + M_n$.

Donc par encadrement (f_n) converge uniformément vers 0 sur $[0, 1]$.

2) On peut utiliser de simples vecteurs numpy pour représenter chaque polynôme par la liste de ses coefficients. La multiplication par X est alors un simple décalage. Mais le calcul des produits de polynômes ou des valeurs prises par une fonction polynomiale sera plus pénible à programmer. Je vais donc plutôt utiliser le module Polynomial que les examinateurs de Centrale semblent apprécier... Je construis à l'aide d'une boucle une liste P telle que P[n] soit le polynôme P_n. On conjecture et l'on montre aisément par récurrence (sans Python !) que P_n est de 2^n degré n, de la parité de n et de coefficient dominant 2^n.

À nouveau des maths pour justifier (classiquement) que l'application de l'énoncé définit bien un produit scalaire... Comme le module Polynomial permet d'utiliser les polynômes ainsi construits comme des fonctions, je peux calculer directement les produits scalaires demandés (en tout cas des valeurs numériques approchées !).

```
1import numpy as np
from numpy.polynomial import Polynomial
import scipy.integrate as integr
X=Polynomial([0,1])
P=[]
P.append(Polynomial([1]))
P.append(2*X)
for k in range(2,9):
    P.append(2*X*P[k-1]-P[k-2])

def pscal(A,B):
    return integr.quad(lambda t: sqrt(1-t**2)*A(t)*B(t),-1,1)[0]

G=np.zeros((9,9))
for i in range(9):
    for j in range(9):
        G[i,j]=round(pscal(P[i],P[j]),3)
print(G)
```

La matrice obtenue, arrondie au millième, est diagonale (avec des 0. et des $-0.$(!) en dehors de la diagonale). Et sur la diagonale, que des 1.571 (ce qui fait penser à $\pi/2\ldots$. On peut apparemment conjecturer que la famille (P_n) est orthogonale. "En déduire" qu'elle l'est à partir de valeurs approchées serait abusif ! Elle l'est en réalité, car il s'agit des (classiques) polynômes de Tchebychev de seconde espèce, qui vérifient

$$\forall n \in \mathbb{N}, \quad \forall \theta \in]0,\pi[,\ P_n(\cos\theta) = \frac{\sin(n+1)\theta}{\sin\theta}.$$

Le changement de variable $t = \cos\theta$ ($\mathcal{C}^1$ bijectif strictement décroissant de $]0,\pi[$ dans $]-1,1[$) permet de démontrer le résultat (et aussi que $(P_n \mid P_n) = \pi/2$ pour tout n). Pour déterminer la matrice de ϕ dans la base $\mathcal{B} = (P_0,\ldots,P_8)$, je pourrais utiliser le fait que $\mathcal{C} = (Q_0,\ldots,Q_8)$ est orthonormale, où $Q_k = \sqrt{2/\pi}P_k$, et utiliser la formule classique $(Q_i \mid \phi(Q_j))$ pour remplir la matrice de ϕ dans $\mathcal{C}$ et en déduire la matrice dans $\mathcal{B}$. Mais il est aussi simple de déterminer directement les coordonnées de $\phi(P_j)$ dans $\mathcal{B}$, en écrivant grâce à l'orthogonalité de $\mathcal{B}$:

$$\phi(P_j) = \sum_{k=0}^{8} a_{k,j}P_k \quad \text{d'où} \quad (P_i \mid \phi(P_j)) = a_{i,j}(P_i \mid P_i) \quad \text{d'où} \quad a_{i,j} = \frac{2}{\pi}(P_i \mid \phi(P_j))$$

```
from math import pi

def phi(A):
    return 3*X*A.deriv(1)-A.deriv(2)

M=np.zeros((9,9))
for i in range(9):
    for j in range(9):
        M[i,j]=round(2/pi*pscal(P[i],phi(P[j])),3)
print(M)
```

On obtient une matrice apparemment triangulaire supérieure, à coefficients entiers. Ceci peut se retrouver grâce aux relations entre les polynômes de Tchebychev de 1^{re} et 2^{nde} espèce et aux équations différentielles qu'ils vérifient (voir par exemple la page Wikipedia intitulée Polynômes de Tchebychev).

3) (Centrale) Pas de grosse difficulté, attention tout de même aux indices :

il est plus commode avec Python de numéroter lignes et colonnes de 0 à $n-1$.

Code disk dur o18info03

```
import numpy as np

def Matrice(L):
    n=len(L)
    M=np.zeros((n,n))
    for j in range(n-1):
        M[j+1,j]=1
    for i in range(n):
        M[i,n-1]=-L[i]
    return(M)

L=np.random.randint(1,10,8)
print(L)
print(np.poly(Matrice(L)))
```

Noter qu'il y a dans Python (au moins) deux fonctions randint : avec celle du module random, randint (a, b) renvoie un entier aléatoire de $[\![a, b]\!]$(b compris), tandis qu'avec celle du module numpy . random (que j'utilise ici) on obtient un entier de $[\![a, b[\![$(b exclu !). L'avantage de cette dernière est de fournir directement une vecteur aléatoire de taille n avec la syntaxe randint (a, b, n) ou une matrice aléatoire par randint $(a, b, (n, p))$ On peut conjecturer que les coefficients du polynôme caractéristique sont, après le 1 du terme dominant, ceux de la liste fournie, lus de droite à gauche... C'est un résultat montré dans le D.L.2 (matrice compagne d'un

polynôme), la démonstration est dans le corrigé dudit D.L. ! La dernière question apporte un résultat complémentaire. Notons $\mathcal{B} = (e_1, \dots e_n)$ la base canonique de $E = \mathbb{K}^n$ et $u = \operatorname{Can} M$. Par construction et grâce à une récurrence immédiate,

$$\forall j \in [\![1, n-1]\!] \quad u(e_j) = e_{j+1} \quad \text{d'où} \quad \forall k \in [\![0, n-1]\!] \quad u^k(e_1) = e_{k+1}$$

Donc, si $P = \sum_{k=0}^{n-1} b_k X^k$, alors $P(u)(e_1) = \sum_{k=0}^{n-1} b_k e_{k+1} = \sum_{j=1}^{n} b_{j-1} e_j$. Comme $\mathcal{B}$ est une famille libre (!), il en résulte que, si P est un polynôme non nul de degré au plus $n-1$, alors $P(u) \neq 0$ (puisqu'il ne s'annule pas en e_1.). Par contraposée, j'ai montré que tout polynôme annulateur non nul de u est de degré au moins égal à n.

Notons $A = X^n + \sum_{k=0}^{n-1} a_k X^k$. D'après ce qui précède, $A = \chi_M = \chi_u$. Montrons que M est diagonalisable si et seulement si A admet n racines simples : - si M est diagonalisable, alors $P = \prod_{\lambda \in \operatorname{Sp} u} (X - \lambda)$ est un polynôme annulateur (non nul !) de u, ce qui implique d'après le résultat ci-dessus que u admet n valeurs propres distinctes et donc que A a admet n racines distinctes, nécessairement simples ; - réciproquement, c'est un résultat du cours : si A admet n racines simples, alors M admet n sousespaces propres, qui sont en somme directe et sont donc n droites dont la somme est E! Noter qu'il n'est pas nécessaire d'invoquer le théorème de Cayley-Hamilton...

4) (Centrale) Il s'agit de montrer que, pour n fixé dans $\mathbb{N}$, H_n est un ensemble fini. Or, si un couple (p, q) d'entiers naturels vérifie $2p + 3q = n$, j'ai nécessairement $2p \leqslant n$ et $3q \leq n$, donc $p \leqslant \lfloor n/2 \rfloor$ et $q \leqslant \lfloor n/3 \rfloor$. Plus précisément, pour q fixé entre 0 et $\lfloor n/3 \rfloor$, il y a 0 ou 1 élément de H_n de la forme (p, q), selon la parité de $n - 3q$! Cela prouve que H_n est fini, de cardinal au plus égal à $\lfloor n/3 \rfloor + 1$, lui-même (grossièrement) majoré par n : $\sigma(n)$ existe bien et $\sigma(n)$ est un $O(n)$.

Pour $n \leqslant 2$, on a vite fait le tour des solutions dans $\mathbb{N}^2$ de l'équation $(E_n) \quad 2p + 3q = n$!

$$\sigma(0) = 1; \sigma(1) = 0; \sigma(2) = 1$$

Pour $n \geqslant 3$, je remarque (habilement) que n et $n-3$ sont deux entiers naturels de parités différentes, donc l'un des deux est pair ! Ce qui me donne une solution de (E_n), puisque je peux écrire soit $n = 2p$ (auquel cas $(p, 0) \in H_n$), soit $n = 2p + 3$ (auquel cas $(p, 1) \in H_n$). Ainsi

$$\text{Si } n \geqslant 3, \text{ alors } \sigma(n) \geqslant 1$$

Il n'est pas clair de conjecturer une formule générale pour $\sigma(n) \dots$ Python fournit les valeurs (pour n "petit"), grâce à une double boucle (en testant tous les couples (p, q) évoqués au début), ou plus efficacement à l'aide d'une simple boucle, puisque pour q fixé entre 0 et $\lfloor n/3 \rfloor$, il suffit de tester la parité de $n - 3q$(cf. la remarque du début). D'où le programme, avec une complexité linéaire :

```
def sigma(n):
    nb=0
    for q in range(1+n//3):
        if (n-3*q)%2==0: nb+=1
    return nb

print([sigma(k) for k in range(26)])
```

Ce qui donne : $[1, 0, 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 3, 2, 3, 3, 3, 3, 4, 3, 4, 4, 4, 4, 5, 4]$.

L'encadrement déjà établi, pour $n \geqslant 3, 1 \leqslant \sigma(n) \leqslant n$ donne (sachant que les deux séries entières $\sum x^n$ et $\sum nx^n$ ont pour rayon de convergence 1) :

Le rayon de convergence de $\sum \sigma(n)x^n$ vaut 1.

Plus précisément, puisque $\sigma(n)$ ne tend pas vers 0, l'ensemble de définition de $S : x \mapsto \sum_{n=0}^{\infty} \sigma(n)x^n$ est $]-1, 1[$ (divergence grossière en ± 1). Pour établir la relation de l'énoncé, j'utilise les sommes des séries géométriques classiques, pour x fixé dans $]-1, 1[$:

$$f : x \mapsto \frac{1}{1-x^2} = \sum_{p=0}^{\infty} x^{2p} = \sum_{i=0}^{\infty} a_i x^i \quad \text{et} \quad g : x \mapsto \frac{1}{1-x^3} = \sum_{q=0}^{\infty} x^{3q} = \sum_{j=0}^{\infty} b_j x^j$$

où a_i (resp. b_j) vaut 1 si i (resp. j) est multiple de 2 (resp. 3) et vaut 0 sinon. Le produit de Cauchy de ces deux séries absolument convergentes s'écrit :

$$f(x)g(x) = \sum_{n=0}^{\infty} c_n x^n \quad \text{où} \quad \forall n \in \mathbb{N} \quad c_n = \sum_{i+j=n} a_i b_j$$

Comme $a_i b_j$ vaut 0 ou 1, c_n est le nombre de couples (i, j) de $\mathbb{N}^2$ tels que ($i + j = n$ et $a_i b_j = 1$), c'est-à-dire ($i + j = n$ et i pair et j impair). Autrement dit $c_n = \sigma(n)$! En conclusion

$$\forall x \in]-1, 1[, \sum_{n=0}^{\infty} \sigma(n)x^n = \frac{1}{1-x^2} \cdot \frac{1}{1-x^3}.$$

Pour retrouver les valeurs de $\sigma(n)$ pour $n \leqslant 25$, il suffit d'obtenir les termes de degré au plus 25 dans le développement de $f(x)g(x)$. Ils sont fournis par le développements des sommes partielles, qui sont des polynômes. Le module Polynomial permet de les obtenir facilement :

```
from numpy.polynomial import Polynomial

def a(k,d):
    if k%d==0:
        return 1
    else:
        return 0

A=Polynomial([a(k,2) for k in range(26)])
B=Polynomial([a(k,3) for k in range(26)])

print((A*B).coef[:26])
```

5) (ENSAM) Les polynômes de Lagrange ne sont plus au programme en PSI... Rappelons tout de même que, pour toute famille $(a_k)_{0\leqslant k\leqslant n}$ de $n+1$ scalaires distincts et toute famille $(b_k)_{0\leqslant k\leqslant n}$ de scalaires (non nécessairement distincts), il existe un unique polynôme P de $\mathbb{K}_n[X]$ vérifiant : $\forall k \in [\![0,n]\!] \quad P(a_k) = b_k$ (cf cours). Pour le calcul des coefficients, on peut comme le suggère l'énoncé résoudre brutalement le système linéaire dont les coefficients de P sont les solutions. Je remplis donc la matrice (de Vandermonde !) dudit système et le second membre, avant d'appeler numpy. linalg. solve.

```
from numpy.polynomial import Polynomial
import matplotlib.pyplot as plt
import numpy as np
import numpy.linalg as alg

def f(x):
    return 1/(1+x**2)

def coef(h,a):
    n=len(a)
    A=np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            A[i,j]=a[i]**j
    X=alg.solve(A,[h(a[i]) for i in range(n)])
    return list(X)

def evalP(L,x):
    n=len(L)-1
    y=L[n]
    for i in range(1,n+1):
        y*=x
        y+=L[n-i]
    return y

def evalPoly(L,x):
    return Polynomial(L)(x)

def affiche(f,j):
    X=np.linspace(-5,5,j+1)
    L=coef(f,X)
    X=np.linspace(-5,5,500)
    Y=[evalP(L,x) for x in X]
    plt.plot(X,Y,'b')
    X=np.linspace(-5,5,100)
    plt.plot(X,f(X),'r')
    plt.show()

affiche(f,10)
```

Il manque un schéma.

Le résultat de solve étant un vecteur numpy, je le convertis en liste. Pour tracer le graphe de P, je programme la fonction polynomiale associée en utilisant, soit le module Polynomial, soit l'algorithme de Hörner, avec le graphe pour $j = 10$ et les commandes pour l'avoir.

Où l'on voit que l'approximation est loin d'être "uniforme" : le graphe de P passe bien par les 11 points requis, mais P prend par endroit des valeurs très éloignées de celles prises par $f\dots$ Et cela ne s'arrange pas lorsque j augmente ! Pour constater le phénomène, ne pas oublier d'utiliser deux linspace : le premier pour définir les points d'interpolation, le second pour tracer les graphes, avec beaucoup plus de points pour une précision convenable...

6) (ENSAM) Il s'agit ici de "marche aléatoire" sur une droite, mais sans retour en arrière ! La position finale de la balle, après n étapes, est déterminée par un entier de $[\![0,n]\!]$. Pour la simulation numérique, j'écris directement les fonctions avec un paramètre p correspondant à la probabilité p d'avancer d'une case. Pour la fonction tirer (n,p), il suffit de répéter n fois le tirage d'un flottant aléatoire de $[0,1[$ (ce que fait la fonction random du module random, aussi bien que la fonction random du module numpy.random !) et de compter le nombre de valeurs inférieures à p obtenues. J'en déduis une fonction d'en-tete simul (n,N,p), j'initialise un tableau de $n+1$ zéros qui contiendra le nombre de fois que chaque position aura été atteinte durant les N appels de tirer. Il n'y a plus qu'à le diviser par N pour obtenir les fréquences (utiliser un vecteur numpy et non une liste, qui ne supporterait pas la division par $N\dots$) :

```
import numpy as np

def tirer(n,p):
    r=0
    for k in range(n):
        if np.random.random()<p:
            r+=1
    return r

def simul(n,N,p):
    L=np.zeros(n+1)
    for k in range(N):
        L[tirer(n,p)]+=1
    return L/N
```

La "courbe théorique" évoquée par l'énoncé est bien sur l'histogramme de la loi binomiale.

D'où l'intérêt de calculer les coefficients du binôme ! Le programme le plus court est donné par la formule $\dfrac{n!}{k!(n-k)!}$, qui ne pose pas de problème pour de petites valeurs, mais qu'il vaut mieux éviter... Utiliser plutôt la formule simplifiée

$$\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k!} = \prod_{j=0}^{k-1}\left(\frac{n-j}{j+1}\right)$$

après avoir pris soin de remplacer k par $n-k$ lorsque $n-k<k$! Noter qu'avec Python, cette formule sera efficace mais renverra un flottant. Si l'on a le temps et que l'on veut la valeur

exacte de l'entier, on calcule les deux entiers $\prod_{j=0}^{k-1}(n-j)$ et $k!$, puis le quotient (forcément exact !) avec un double slash... Pour tracer le graphe les flottants suffiront.

```
import matplotlib.pyplot as plt

def binom(n,k):
    if n-k<k: k=n-k
    if k<0:
        return 0
    else:
        f=1
        for j in range(k): f*=(n-j)/(j+1)
        return f

def graphe(n,N,p):
    L1=simul(n,N,p)
    L2=[binom(n,k)*p**k*(1-p)**(n-k) for k in range(n+1)]
    X=[k for k in range(n+1)]
    plt.plot(X,L1,linewidth=2,label='Simulation')
    plt.plot(X,L2,linewidth=2,label='Loi binomiale')
    plt.legend()
    plt.show()
```

Même s'il s'agit a priori d'histogrammes, le tracé des lignes brisées permet de mieux comparer les valeurs obtenues. Page suivante les graphiques obtenus avec graphe $(10, 100, 0.5)$ demandé par l'énoncé, puis avec graphe $(10, 10000, 0.5)$ où l'on apprécie la "convergence en loi" ...

Il manque 2 schémas...

7) (ENSAM) Il s'agit de l'algorithme historique du crible d'Ératosthène (savant grec et éclectique du III siècle avant J.-C., algorithme apprécié à l'époque car il pouvait s'exécuter aisément sans le moindre ordinateur ! L'idée est de barrer dans la liste des entiers de 0 à n, supposés rangés dans l'ordre et dans la liste L, successivement 0,1, puis les multiples de 2 (à partir de 4), les multiples de 3 (à partir de 6), etc. Informatiquement, le fait de "barrer" l'entier k se traduira par le passage de True à False de la valeur $L[k]$. Pour la fonction d'en-tête modifier (L,p), après avoir traité les cas où $p < 2$, je mets à False tous les multiples de p à partir de $2p$ et jusqu'à la fin de L (plus efficace que de balayer tous les indices et de tester s'ils sont multiples de p!). Pour la fonction d'en-tete premiers (n), le principe du crible est d'initialiser une liste L indexée de 0 à n et remplie de True, puis d'exécuter modifier (L, p) pour des valeurs successives de p, de sorte que les derniers True dans L correspondent aux nombres premiers. La première idée (naive) et de balayer tous les p de 2 à n. Classiquement, on peut s'arrêter à $\sqrt{n}$, ce qui est une optimisation importante ! En effet, si $m \leqslant n$ n'est pas premier, il admet nécessairement un diviseur au plus égal à $\sqrt{m}$ (donc inférieur ou égal à $\sqrt{n}$!),

puisque si $q > \sqrt{m}$ divise m, alors $m/q < \sqrt{m}$!! Autre optimisation non négligeable : il suffit d'exécuter modifier (L,p) pour les p tels que $L[p]$ vaut True puisque sinon les multiples de p sont déjà "barrés" (un multiple d'un multiple est un multiple...).

```
from math import sqrt

def modifier(L,p):
    if p<2: L[p]=False
    else:
        for k in range(2*p,len(L),p): L[k]=False
    return L

def premiers(n):
    L=[True for k in range(n+1)]
    p=0
    for p in range(int(sqrt(n))+1):
        if L[p]: L=modifier(L,p)
    return [k for k in range(n+1) if L[k]]
```

Pour répondre à la dernière question, la liste générée étant très longue, j'exécute premiers (823417)[−1], qui me donne le plus grand nombre premier inférieur ou égal à 823417 : c'est 823399, donc 823417 n'et pas premier.

8) (ENSAM) Petit rappel sur la lecture d'un fichier texte : ouvrir avec open, lire la ligne (unique dans ce cas) avec readline et fermer le fichier avec close, c'est un exercice d'anglais...

Noter que Pyzo est capable de trouver le fichier Pi.txt dans le répertoire courant, à condition d'utiliser la commande "Exécuter/Démarrer le script", qui redémarre le shell et exécute le script courant.

Raccourci clavier Ctrl+F5 ou Maj+Ctrl+E. Sinon, il peut s'avérer nécessaire d'indiquer le chemin complet vers le fichier ; dans ce cas, penser - pour le paramètre de open - à faire précéder la chaine contenant ledit chemin d'un r (pour rawstring, chaine brute), afin d'empêcher Python d'interpréter d'éventuels caractères spéciaux ; par exemple

```
open (r'U: \Python \backslashOral.py')
```

Une fois la chaine chargée, penser au slicing pour obtenir les 10 premiers caractères ([0 : 10] ou [: 10] pour les indices de 0 à 9) et les 10 derniers ([−11 :] ou [n − 11 : n] à condition que n contienne la longueur de la chaine...).

Pour la recherche d'un motif, une version naive suffira. Je profite là encore du slicing pour comparer d'un coup le motif avec une sous-chaine de la source (sinon il suffit d'une boucle pour comparer caractère par caractère...). On pourrait aussi utiliser les fonctions de Python,

mais pas sur que ce soit au gout du jury ! L'appel source.find(motif) renvoie la même chose que pos(motif, source), à savoir l'indice de sa première occurrence si motif est une sous-chaine de source et -1 sinon.

```
f=open('Pi.txt')
chaine=f.readline()
f.close()

print('10 premières : ',chaine[:10])
print('10 dernières : ',chaine[-11:])

def pos(motif,source):
    p,n=len(motif),len(source)
    k=0
    while k<=n-p:
        if source[k:k+p]==motif: return k
        k+=1
    return -1
```

Il apparait que '000' se trouve à partir de la 601^{e} décimale mais que ' 0000 ' est absente. De même '123' est à partir de la 1924^{e} décimale, '1234' est introuvable. Enfin '314' se trouve à partir de la 2120^{e} décimale (et pas au début puisqu'on a exclu la partie entière !).

9) (ENSAM) Le transtypage permet de faire faire le travail à Python avec un minimum de programmation ! Attention tout de même au type des objets manipulés : list(str(n)) renvoie la liste des chiffres composant n, mais chaque chiffre y figure en tant que caractère ! D'où cette solution (j'utilise la fonction sum sans trop de scrupules, si besoin une boucle peut la remplacer...).

```
def chiffres(n):
    return [int(c) for c in list(str(n))]

def narcissique(n):
    L=chiffres(n)
    p=len(L)
    return sum(L)**p==n

print([n for n in range(10001) if narcissique(n)])
```

Le résultat est [0,1,2,3,4,5,6,7,8,9,81,512,2401]

10) (Centrale) Lorsqu'on a besoin de précision dans ce type de calcul de somme, il vaut mieux commencer par ajouter à 0 les termes les plus petits, afin que les retenues puissent se reporter sur les plus gros termes (sinon les plus petits termes sont purement et simplement considérés comme nuls).

Alors [n*u(n) for n in range (10, 31)] renvoie une liste dont les 13 derniers termes sont nuls. Mais en augmentant la précision à 50 décimales (mp comme multiprécision), à condition de remplacer factorial par mp.factorial et exp par mp.exp...

```
from math import exp,factorial
from mpmath import mp
mp.dps=50

def u(n):
    s=0
    for k in range(n-1,-1,-1): s+=1/mp.factorial(k)
    return (mp.exp(1)-s)**(1/n)


for n in range(10,31):
    print(n, n*u(n))
```

on obtient des valeurs qui croissent doucement, $30u_{30}$ valant environ 2.49. Où l'on voit les limites du calcul numérique...

La formule de Taylor avec reste intégral à l'ordre $n+2$, appliquée entre 0 et 1 à la fonction exp, donne

$$e = \sum_{k=0}^{n+2} \frac{1}{k!} + \int_0^1 \frac{(1-t)^{n+2}}{(n+2)!} e^t \, \mathrm{d}t$$

d'où

$$e - \sum_{k=0}^{n+2} \frac{1}{k!} = \frac{1}{n!}\left(1 + \frac{1}{n+1} + \frac{1}{(n+1)(n+2)} + \int_0^1 \frac{(1-t)^{n+2}}{(n+1)(n+2)} e^t \, \mathrm{d}t\right)$$

donc

$$c_n = \ln\left(1 + \frac{1}{n+1} + \frac{1}{(n+1)(n+2)} + \int_0^1 \frac{(1-t)^{n+2}}{(n+1)(n+2)} e^t \, \mathrm{d}t\right) \text{ convient}$$

et en majorant classiquement la dernière intégrale

$$c_n = \ln\left(1 + \frac{1}{n+1} + \frac{1}{(n+1)(n+2)} + O\left(\frac{1}{n^3}\right)\right)$$

Or

$$\frac{1}{n+1} + \frac{1}{(n+1)(n+2)} = \frac{2}{n+1} - \frac{1}{n+2} = \frac{2}{n}\left(1 - \frac{1}{n}\right) - \frac{1}{n}\left(1 - \frac{2}{n}\right) + O\left(\frac{1}{n^3}\right) = \frac{1}{n} + O\left(\frac{1}{n^3}\right)$$

d'où

$$c_n = \frac{1}{n} - \frac{1}{2n^2} + O\left(\frac{1}{n^3}\right)$$

En particulier (c_n) converge vers 0, d'où grâce à la formule de Stirling

$$\ln u_n = \frac{1}{n}\left(c_n - \ln n!\right) = \frac{1}{n}\left(-n\ln n + n - \frac{\ln n}{2} - \frac{\ln(2\pi)}{2} + o(1)\right)$$

et

$$u_n = \exp \ln u_n = \frac{e}{n} \exp\left(-\frac{\ln n}{2n} - \frac{\ln(2\pi)}{2n} + o\left(\frac{1}{n}\right)\right)$$

soit finalement

$$u_n = \frac{e}{n}\left(1 - \frac{\ln n}{2n} - \frac{\ln(2\pi)}{2n} + o\left(\frac{1}{n}\right)\right)$$

Où l'on voit que (nu_n) converge vers e, ce qui n'était pas évident au vu des valeurs approchées.

11) (Centrale) Même genre de surprise que dans l'exercice précédent... Pour le calcul des sommes partielles successives, il faut les stocker (ou les afficher) au fur et à mesure et non pas les recalculer l'une après l'autre à partir de rien. Avec la précision par défaut, les premières valeurs semblent converger vers une limite proche de -1.052 puis se mettent à s'en éloigner... Avec 50 décimales, la convergence apparait de façon plus convaincante. Et en effet la formule du binôme montre (après simplifications) que $(2+\sqrt{3})^n + (2-\sqrt{3})^n$ est un entier pair, d'où $u_n = -v_n$ or $\sum v_n$ est absolument convergente par comparaison à une série géométrique, d'où la convergence absolue de $\sum u_n$ malgré les premières apparences. Le calcul numérique de u_n devient vite imprécis car l'argument du sinus tend vers l'infini, alors que pour v_n il tend très vite vers 0.

```
from math import pi,sin,sqrt
from mpmath import mp
def u(n):
    return mp.sin(mp.pi*(2+mp.sqrt(3))**n)

def S(p):
    s=0
    for n in range(p+1):
        s+=u(n)
        print(n,s)
mp.dps=30
print(S(30))
```

Noter que pour bénéficier de la multiprécision il faut utiliser mp.pi, mp.sin et mp.sqrt à la place de la constante et des fonctions habituelles !

12) (ENSAM) Pour obtenir un vecteur tangent, je fais subir un quart de tour au vecteur normal fourni par le gradient. Comme les expressions ici sont assez simples, je programme les formules exactes des dérivées partielles (on pourrait les approcher par un taux de variation...). Voir les figures ci-dessous et les programmes sur la page suivante. Je prends comme point de départ $A_0 = (1, 0)$ et j'obtiens une espèce de spirale déformée... (à gauche). Pour améliorer la précision, je considère le développement limité vérifié par f, qui est de classe $\mathcal{C}^1$:

$$f(a+h) \underset{h\to 0}{=} f(a) + (\nabla f(a) \mid h) + o(h)$$

En prenant pour point a le point déplacé le long de la tangente (1re ligne de la boucle for), je choisis pour h un terme correctif le long de la normale en a à la courbe (dirigée par $\nabla f(a)$), en faisant en sorte de compenser le terme $f(a)$, qui n'est probablement par nul ! Il est immédiat que $h = -\frac{f(a)}{\|\nabla f(a)\|^2} \cdot \nabla f(a)$ convient. Avec cette amélioration, j'obtiens une courbe fermée... (à

droite).

Il manque 2 schémas.

```
import numpy as np
import matplotlib.pyplot as plt

def f(u):
    x, y = u
    return x**4 + 2*y**2 + 2*x*y - 1

def nablaf(u):
    x, y = u
    return np.array([4*x**3 + 2*y, 4*y + 2*x])

def rot(v):
    x, y = v
    return np.array([-y, x])

def n2(v):
    x, y = v
    return x**2 + y**2

def points(A0, pas, n):
    L = [A0]
    A = A0
    for k in range(n):
        A = A + pas*rot(nablaf(A))
        G = nablaf(A)
        A = A - (f(A)/n2(G))*G
        L.append(A)
    return L

A0 = np.array([1, 0])
A = points(A0, 0.01, 500)
X = [P[0] for P in A]
Y = [P[1] for P in A]
plt.plot(X, Y)
plt.axis('equal')
plt.show()

# def points(A0, pas, n):
#     L = [A0]
#     A = A0
#     for k in range(n):
#         A = A + pas*rot(nablaf(A))
#
#         L.append(A)
#     return L
```

13) (Centrale) Un piège : il faut observer 2 tirages consécutifs, mais après avoir examiné les 2 premiers, il ne faut surtout pas refaire le 2^e tirage pour le comparer au 3^e !!

Une solution pourrait être de remplir au départ un vecteur avec n résultats de tirages,

mais c'est évidemment de la place mémoire et du temps perdus...

La relation de récurrence s'obtient par un raisonnement classique.

$A_{i,j}$ l'événement "jamais obtenir 2 Pile d'affilée du tirage i inclus à j exclu" (à la Python...).

Ainsi $p_n = P(A_{0,n})$ (en numérotant les tirages à partir de 0).

L'indépendance mutuelle des tirages fait que $P(A_{i,j})$ ne dépend que du nombre $j-i$ de tirages. Soit F_i l'événement "le tirage i donne Face".

J'utilise le SCE $\left(F_0, \overline{F_0} \cap F_1, \overline{F_0} \cap \overline{F_1}\right)$ pour calculer p_{n+2} à l'aide de la FPT, pour $n \in \mathbb{N}$:

$$\begin{aligned} P(A_{0,n+2}) &= P(F_0)\, P_{F_0}(A_{0,n+2}) + P\left(\overline{F_0} \cap F_1\right) P_{\overline{F_0}\cap F_1}(A_{0,n+2}) + P\left(\overline{F_0} \cap \overline{F_1}\right) F_{\overline{F_0}\cap\overline{F_1}}(A_{0,n+2}) \\ &= (1-p)P(A_{1,n+2}) + p(1-p)P(A_{2,n+2}) + 0 \end{aligned}$$

soit, grâce à la remarque précédente : $p_{n+2} = (1-p)p_{n+1} + p(1-p)p_n$

(ce qui permet d'expliciter (p_n) sachant que $p_0 = 1$ et $p_1 = 1-p^2$).

Nous avons donc une relation de récurrence linéaire double ; les solutions de l'équation caractéristique sont les racines du polynóme $Q = X^2 - (1-p)X - p(1-p)$. Q est de degré 2 et

$$Q(-1) = 2 - 2p + p^2 = 1 + (1-p)^2 > 0, \quad Q(0) = -p(1-p) < 0 \quad \text{et} \quad Q(1) = p^2 > 0$$

donc Q admet une racine dans $]-1, 0[$ et une dans $]0, 1[$.

En particulier, la suite (p_n) est une comb lin de 2 suites géométriques qui cv vers 0, donc (p_n) cv vers 0. Or l'événement A : "ne jamais obtenir 2 Pile d'affilée" n'est autre que $\bigcap A_{0,n}$ et la suite $(A_{0,n})$ est décroissante par construction.

Ainsi, grâce à la continuité décroissante de la probabilité, $P(A) = 0$; autrement dit :

L'événement presque certain.

Pour évaluer p_n, j'effectue N simulations et je calcule la fréquence de l'événement $E_n = A_{0,n}$.

D'ailleurs le calcul de p_n (relation de réc ci-dessus) s'effectue à l'aide d'une boucle banale.

(attention aux programmes récursifs de complexité exponentielle...).

```
import numpy as np
def E(n, p):
    if n < 2:
        return True
    else:
        P1 = np.random.rand() < p
        k = 1
        while k<n:
            P0, P1 = P1, np.random.rand() < p
            if P0 and P1: return False
            k += 1
        return True

def freqE(n, p, N):
    nb = 0
    for K in range(N):
        if E(n, p): nb += 1
    return nb/N

def pE(n, p):
    p0, p1 = 1, 1
    for k in range(1, n):
        p0, p1 = p1, p*(1-p)*p0 + (1-p)*p1
    return p1
```

La comparaison est probante pour N assez grand...

Avec les idées précédentes, T prend ses valeurs dans $\mathbb{N}^*$ et, par construction, pour $n \geqslant 1$

$P(T = n) = P(T > n-1) - P(T > n) = p_{n-1} - p_n$

d'où la valeur en x de la fonction génératrice (au moins pour $x \in [-1, 1] \dots$)

$G_T(x) = \sum_{n=1}^{\infty} (p_{n-1} - p_n)\, x^n$

Par le résultat précédent, $\sum p_n$ est abs cv, puisque c'est une comb lin de 2 séries géom cv.

Je peux donc couper la somme en deux, pour x fixé dans $[-1, 1]$:

$G_T(x) = \sum_{n=1}^{\infty} p_{n-1} x^n - \sum_{n=1}^{\infty} p_n x^n.$

Par ailleurs, en reprenant la relation de récurrence du début, en multipliant par x^{n+2}

et en sommant (toutes les séries sont absolument convergentes) j'obtiens :

$$\sum_{n=0}^{\infty} p_{n+2}x^{n+2} = (1-p)\sum_{n=0}^{\infty} p_{n+1}x^{n+2} + p(1-p)\sum_{n=0}^{\infty} p_n x^{n+2}$$

En notant $g(x) = \sum_{n=0}^{\infty} p_n x^n$, sachant que $p_0 = 1$ et $p_1 = 1 - p^2$, grâce à des réindexations,

$$g(x) - 1 - \left(1-p^2\right)x = (1-p)x[g(x)-1] + p(1-p)x^2 g(x)$$

c'est-à-dire

$$g(x)\left[1 - (1-p)x - p(1-p)x^2\right] = 1 + p(1-p)x$$

$\varphi(x) = 1-(1-p)x(1+px)$ est un pôly de deg 2 admettant $-\infty$ pour lim en $\pm\infty$ et vérifiant

$$\varphi(-1) = 1 + (1-p)^2 > 0 \quad \text{et} \quad \varphi(1) = p^2 > 0$$

Donc φ admet une racine α dans $]-\infty, -1[$, et β dans $]1, +\infty[$ et φ est strict pos sur $[-1, 1]$.

Ainsi $\forall x \in]\alpha, \beta[\quad g(x) = \dfrac{1 + p(1-p)x}{\varphi(x)}$.

or après réindexation, compte tenu de $p_0 = 1$

$G_T(x) = xg(x) - [g(x) - 1] = 1 - (1-x)g(x)$ soit finalement

$$\forall x \in]\alpha, \beta[\quad G_T(x) = 1 - \frac{(1-x)(1+p(1-p)x)}{1-(1-p)x(1+px)}.$$

Or $1 \in]\alpha, \beta[, G_T$ est dérivable en 1 donc T est d'espérance finie et (inutile d'expliciter g' !)

$$E(T) = G_T'(1) = g(1) = \frac{1+p(1-p)}{p^2} \text{ soit } E(T) = \frac{1+p(1-p)}{p^2}$$

Pour les simulations, je reprends l'idée du début, mais en comptant le nombre de lancers jusqu'à l'obtention de deux Pile consécutifs.

```
def T(p):
    n = 1
    P0 = np.random.rand() < p
    P1 = np.random.rand() < p
    while not (P0 and P1):
        P0, P1 = P1, np.random.rand() < p
        n += 1
    return n

def moyT(p, N):
    s = 0
```

```
    for K in range(N):
        s += T(p)
    return s/N

def espT(p):
    return (1 + p*(1-p))/p**2

print(espT(0.3), moyT(0.3, 10000))
```

Là encore, résultats probants !

14) a) Calculons la fonction génératrice de S, par indépendance des deux lancers :

$$\begin{aligned} G_S(x) &= \frac{1}{6}\left(x + 2x^2 + 2x^3 + x^4\right) \cdot \frac{1}{6}\left(x + x^3 + x^4 + x^5 + x^6 + x^8\right) \\ &= \frac{1}{36}\left(x^2 + 2x^3 + 3x^4 + 4x^5 + 5x^6 + 6x^7 + 5x^8 + 4x^9 + 3x^{10} + 2x^{11} + x^{12}\right) \end{aligned}$$

cela tous calculs faits, par exemple à l'aide du module Polynomial !). On reconnait la loi triangulaire, la même que pour le lancer de deux dés "classiques", dont les faces sont numérotées de 1 à 6!! Sa fonction génératrice est en effet

$$\begin{aligned} G_T(x) &= \left[\frac{1}{6}\left(x + x^2 + x^3 + x^4 + x^5 + x^6\right)\right]^2 \\ &= \frac{1}{36}\left(x^2 + 2x^3 + 3x^4 + 4x^5 + 5x^6 + 6x^7 + 5x^8 + 4x^9 + 3x^{10} + 2x^{11} + x^{12}\right) \end{aligned}$$

b) Le résultat précédent vient de deux répartitions possibles des facteurs irréductibles dans

$$X^2(X+1)^2\left(X^2+X+1\right)^2\left(X^2-X+1\right)^2$$

En effet

$$X(X+1)\left(X^2+X+1\right) = X + 2X^2 + 2X^3 + X^4$$

et

$$X(X+1)\left(X^2+X+1\right)\left(X^2-X+1\right)^2 = X + X^3 + X^4 + X^5 + X^6 + X^8$$

Pour savoir si d'autres répartitions sont possibles, il suffit de tester les 3^4 répartitions possibles...

En laissant les $\frac{1}{6}$ en facteur, je remarque qu'un produit correspond à la fonction génératrice associée à un dé équilibré à 6 faces portant chacune un entier naturel non nul si et seulement si son coefficient constant est nul et ses autres coefficients sont des entiers naturels de somme 6 .

Voici une version naive avec une quadruple boucle (qui permet d'accepter des 0 sur les faces des dés, en otant la condition que le coefficient soit nul dans le test d'un produit...).

```
from numpy.polynomial import Polynomial

L = [[0, 1], [1, 1], [1, 1, 1], [1, -1, 1]]
P = [Polynomial(lst) for lst in L]
nf = 6

def coeffs(P):
    return [int(c) for c in P.coef]

def test(P, nf):
    L = coeffs(P)
    if L[0] == 0 and sum(L) == nf:
        for c in L:
            if c < 0: return False
        return True
    else:
        return False

def prod(P, L):
    produit = Polynomial([1])
    for k in range(len(L)):
        produit = produit * P[k]**L[k]
    return produit


Lok = []
n = len(L)

for i0 in range(3):
    for i1 in range(3):
        for i2 in range(3):
            for i3 in range(3):
                La = [i0, i1, i2, i3]
                Lb = [2-i for i in La]
                if Lb not in Lok:
                    A = prod(P, La)
                    B = prod(P, Lb)
                    if test(A, nf) and test(B, nf):
                        Lok.append(La)
                        print(La, A)
                        print(Lb, B)
                        print('-'*20)


# L = [[0, 1], [1, 1], [1, 0, 1]]
# P = [Polynomial(lst) for lst in L]
# nf = 4

# L = [[0, 1], [1, 1], [1, 0, 1], [1, 0, 0, 0, 1]]
# P = [Polynomial(lst) for lst in L]
# nf = 8

# L = [[0, 1], [1, 1], [1, 1, 1, 1, 1], [1, -1, 1, -1, 1]]
# P = [Polynomial(lst) for lst in L]
```

Dans les sources Python, le lecteur acharné trouvera une généralisation récursive pour d'autres types de dés, avec les factorisations similaires à celles ci-dessus. Il en existe pour les dés "classiques" à $4, 6, 8, 10, 12$ et 20 faces !

```
import matplotlib . pyplot as plt
import numpy as np
from mpl_toolkits. mplot3d import Axes3D
from scipy. integrate import odeint
from math import *
from random import *
import numpy. random as rd
from numpy. polynomial import Polynomial
import numpy. linalg as alg
import scipy. integrate as integr
def pour_X (n):
    r=rd. random (n)
    for i in range (n):
        s=0
        for j in range (i +1):
            s=s+r[j]
        r[i]=s
    return (r)
U = Polynomial ([1])

def Lambda (i,k,X):
    Pi=U
    for j in range (n):
        F= Polynomial ([-X[j], 1])
        if j!= i and j!=k:
            Pi =Pi*F*F
        else:
            Pi =Pi*F
    return (Pi)
#comparer avec le mien, attention aux "n"
n=5
Mu = np.random .randint (10, size=(n))
Ad= np. random . randint (10, size=(n))
alpha =randint (1 ,10)

def Def (Mu ,Ad ,alpha ,n):
    N=np. zeros ((n+1,n+1))
    for i in range (n):
        N[i,i]= Ad[i]
        N[n,i]= Mu[i]
        N[i,i +1]=1
    N[n-1,n]=1
    N[n,n]= alpha
    return (N)
#comparer au mien
def favilleP (Ad ,n):
    pi =Polynomial ([1])
```

```
    l=[ pi]
    for i in range (n):
        pi=pi* Polynomial ([-Ad[i] ,1])
        l. append (pi)
    return (l)
def prod_l (Lambda ,n):
    pi =Polynomial ([1])
    for i in range (n):
        pi=pi* Polynomial ([- Lambda [i] ,1])
    return (pi)
def passage (Ad ,n):
    P=np.zeros ((n+1,n +1))
    fP =familleP (Ad ,n)
    j=0
    while (j<n+1):
        for x in fP:
            for i in range (j):
                P[i,j]=(x. coef)[i]
                j+=1
    return (P)
#comparer...
a =1/2/3**(1/2)
u=np. array ([[ a],[a] ,[3*a],[a]])
print (" ____________________________ ")
print (u)
print (" ____________________________ ")
def ps(x,y):
    s=0
    for i in range (4):
        s+=x[i ,0]*y[i,0]
    return (s)
def sigma (x):
    return (x -2* ps(x,u)*u)
def base_c ():
    b=[ np.array ([[1] ,[0] ,[0] ,[0]])]
    b.append (np. array ([[0] ,[1] ,[0] ,[0]]))
    b.append (np. array ([[0] ,[0] ,[1] ,[0]]))
    b.append (np. array ([[0] ,[0] ,[0] ,[1]]))
    #b=[]
    #I=np.eye (4)
    #for i in range (4):
    # b.append (I[0:3 , i :i])
    return (b)
def m_sigma ():
    b=base_c ()
    M=sigma (b [0])
    for i in range (3):
```

```
        M=np. concatenate((M,sigma (b[i+1])) , axis =1)
    return (M)
def simp(M):
    for i in range (4):
        for j in range (4):
            if abs (M[i,j ]) <0.001:
                M[i,j]=0
    return (M)
def f(n):
    def g(x):
        return ((1+x/n)** n)
    return (g)

tabcouleur=['green','blue','magenta','yellow']
def Dessin (a,b,N,f,col ):
    X=np. linspace (a,b,N)
    Y=[f(X[i]) for i in range (N)]
    plt .plot(X,Y,color =col )
Dessin (-1,4,20, exp , tabcouleur [0])
for i in range (3):
    Dessin (-1,4,20, f (10**( i+1)) , tabcouleur[i+1])
plt .show()
#plt . close ()
def F(x):
    def g(t):
        return (exp (-x*t)/( t +1))
    return ( integr . quad(g, 0, np.inf )[0])

def Dessin (a,b,N,f,col ):
    X=np. linspace (a,b,N)
    Y=[f(X[i]) for i in range (N)]
    plt .plot(X,Y,color =col )

Dessin (0.1 ,5 ,100 , F, tabcouleur [0])

ax = Axes3D (plt . figure ())
def f(x,y):
    return ((2*x **2+3* y**2)* np.exp(-x**2- y **2))

def Dessin2 (a,b,c,d,pas ,f):
    X = np. arange (a,b, pas )
    Y = np. arange (c, d, pas )
    X, Y = np. meshgrid (X, Y)
    Z = f(X, Y)
    ax. plot_surface(X, Y, Z)

Dessin2 (-2,2,-2,2,0.02 , f)
```

```
plt .show()

#plt . close ()

def Niveau (a,b,c,d,pas ,f, liste ):
    X = np. arange (a,b, pas )
    Y = np. arange (c, d, pas )
    X, Y = np. meshgrid (X, Y)
    Z = f(X, Y)
    #plt. axis(' equal ')
    plt .contour (X, Y, Z, liste)
#Dessin2 (-2,2,-2,2,0.02, f)

liste =[0.3 ,0.4 ,0.5 ,0.6 ,0.7 ,0.75 ,0.8 ,0.9]
a=2
Niveau (-a,a,-a,a ,0.02 , f, liste )
plt .show()

n=20

l= rd.randint (50, size=(n))

def different (l):
    n=len (l)
    m=max (l)
    t=[0 for i in range(m +1)]
    compteur =0
    for i in range (n):
        if t[l[i ]]==0:
            compteur +=1
            t[l[i]]=1
    return ( compteur )
def simul (n):
    tab_des_urnes =[[] for i in range (n)]
    for i in range (n):
        tirage =rd. randint (1,n)
        tab_des_urnes[ tirage ]. append (i)
    x=0
    for i in range (n):
        if tab_des_urnes[i]==[]:
            x+=1
    print ( tab_des_urnes)
    return (x)
def esperanceX (N,n):
    S=0
    for k in range (N):
```

```
        S+= simul (n)
    return (S/N)

def esperanceT (n):
    return ((n*(1 -1/ n )**(n)))

#for i in range (4):
    #n =10**( i+1)
    #for j in range (6):
        #print ([i,esperanceX (10**( j+1), n), esperanceT (n)])

n=3
I=np.eye(n)
A = np. random . randint (10, size=(n,n))
A=A+np. transpose (A)

def car (M):
    return (np .dot (M,M))
def U(p):
    S=I
    for q in range (1,p +1):
        S =1/2*( S+np.dot(A,alg .inv (S)))
    return (S)
def U_rec (p):
    if p==0:
        return (I)
    else:
        return (1/2*( U(p -1)+ np.dot (A,alg.inv(U(p -1)))))

print (A)
for p in range (5):
    print (car (U(2* p)))
    print ('_____________________')

n=20
l= rd.randint (50, size=(n))

def Bern_pm ():
    return (2* rd. binomial (1, 0.5) -1)

def Mat_al (n):
    M=np. zeros ((n,n))
    for i in range (n):
        for j in range (n):
            M[i,j]= Bern_pm ()
    return (M)
```

```
def D(n):
    return (alg .det (Mat_al (n)))

def esp (n,N):
    E=0
    V=0
    for k in range (N):
        E+=D(n)
        V+=D(n )**2
    return ([E/N,V/N-(E/N )**2])

def essai ():
    for n in [2 ,3 ,5 ,7]:
        print (esp(n ,10000))
essai ()

p=0.25
n=100

def Ph(a):
    if a==1:
        def g(t):
           return (1/4/t/t)
        return (g)
    else:
        def g(t):
            return (2* exp (-2* t*t))
        return (g)

def u(n,p):
    def g(t):
        S=rd. binomial (n, p, 100)
        pi =0
        for x in S:
            if abs (x-p)>t*sqrt(n):
                pi +=1
        return (pi /100)
    return (g)
def Dessin (a,b,N,f,col ):
    X=np. linspace (a,b,N)
    Y=[f(X[i]) for i in range (N)]
    plt .plot(X,Y,color =col )

plt .xlim (0 ,3)
plt .ylim (0 ,50)

Dessin (0,3,100, Ph (1), tabcouleur [0])
```

```
Dessin (0,3,100, Ph (2), tabcouleur [1])
Dessin (0,3,100, u(n,p), tabcouleur [2])

plt .show()
#plt . close ()
```