Kit de survie

Utilisation de Python à l'oral de Centrale

Table des matières

Extrait du rapport du jury de Centrale :

L'épreuve de mathématique 2 est un oral de 30 minutes qui succède à une préparation d'environ 30 minutes également. Le sujet est constitué d'un seul exercice comportant plusieurs questions de difficultés progressives et faisant appel, pour certaines, à l'usage de l'outil informatique. Le logiciel informatique est utilisé afin de faire des simulations du thème abordé par l'exercice à travers quelques exemples. Il permet de faire des représentations graphiques et des calculs dans le but de conjecturer ou de vérifier quantitativement les résultats attendus. Lors de la préparation, le candidat dispose d'un ordinateur sur lequel sont installés les logiciels Pyzo et Scilab, ainsi que des documents d'aide fournis à tous les candidats présentant les fonctions des bibliothèques qui pourront être utiles sans pour autant être exigibles.
À l'issue de la préparation le candidat doit présenter à l'examinateur les résultats qu'il a obtenus. Cette présentation pouvant se faire au tableau et/ou devant l'ordinateur, le candidat pouvant faire des allers-retours entre l'ordinateur et le tableau. L'examinateur évalue durant cette présentation la qualité de la pratique mathématique en regard des prestations des autres candidats. Il tient compte aussi, même si ce n'est pas le but principal de l'épreuve, de l'usage de l'outil informatique, tant du point de vue de son efficacité que de sa pertinence.

Le jury tient absolument à mettre en garde contre un temps excessif consacré au code Python pendant le temps de préparation de l'épreuve. Il s'agit avant tout d'un oral de maths, et il n'est pas raisonnable de consacrer plus de la moitié du temps de préparation à la partie informatique. Le code Python nécessaire pour démarrer l'exercice en effectuant des conjectures est normalement extrêmement simple et ne devrait pas être bloquant. Au pire, l'examinateur aidera le candidat pour corriger de petites erreurs de syntaxe. Il peut y avoir du code plus élaboré, mais il sera toujours demandé à la fin. Le jury tient également à souligner que ce n'est pas le code qui est évalué, mais l'interprétation mathématique qui en est faite.

I. Graphiques

La première chose à faire (après avoir créé et sauvegardé le fichier bien sûr !) est d'importer les modules `numpy` et `matplotlib.pyplot` :

```
import matplotlib.pyplot as plt
import numpy as np
```

Le principe est de donner la liste des abscisses et la liste des ordonnées des points à relier.

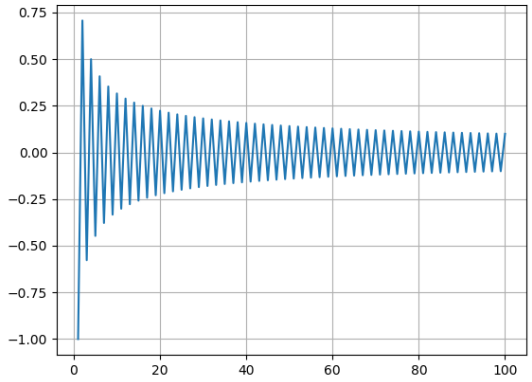
Quelques commandes utiles :

- `plt.grid()` : fait apparaître un quadrillage qui facilite la lecture.
- `plt.axis('equal')` : dessin dans un repère orthonormé
- `plt.show()` : crée une fenêtre affichant le graphe contenant toutes les données enregistrées précédemment.
- `plt.clf()` : *clear figure* : efface la fenêtre graphique en cours

I.1. Tracés de suites

Par exemple, pour tracer les 100 premiers termes de la suite $\left(\frac{(-1)^n}{\sqrt{n}}\right)$:

```
X=[n for n in range(1,101)]
Y=[(-1)**n/np.sqrt(n) for n in X]
plt.grid()
plt.plot(X,Y)
plt.show()
```



Exercice 1. Soit (u_n) la suite définie par $u_0 = u_1 = 1$ et pour tout $n \in \mathbb{N}^*$, $u_{n+1} = u_n + \frac{u_{n-1}}{n}$.

Écrire une fonction permettant de calculer u_n de manière efficace.

Vérifier graphiquement, puis démontrer que $\frac{u_{n+1}}{u_n} \underset{n\to+\infty}{\to} 1$. Que vaut le rayon deconvergence de la série entière $\sum u_n x^n$?

I.2. Tracés de fonctions

Pour définir la liste `X` des abscisses, il y a plusieurs possibilités, notamment :

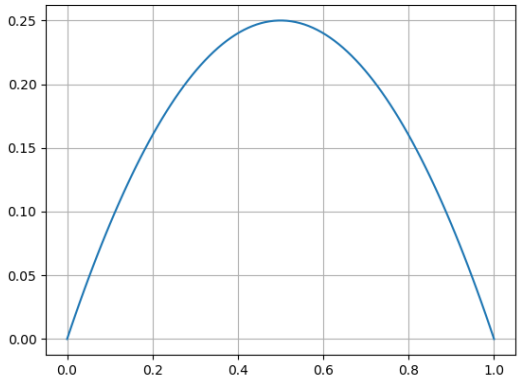
- `np.arange(a,b,h)` : renvoie la liste des $a + i * h$ contenus dans $[a, b[$
- `np.linspace(a,b,nb)` : renvoie une subdivision régulière de $[a, b]$ avec `nb` points sous forme de liste

On définit ensuite la liste `Y` des ordonnées en appliquant la fonction aux éléments de `X`.

Par exemple, pour tracer le graphe de $x \mapsto x(1-x)$

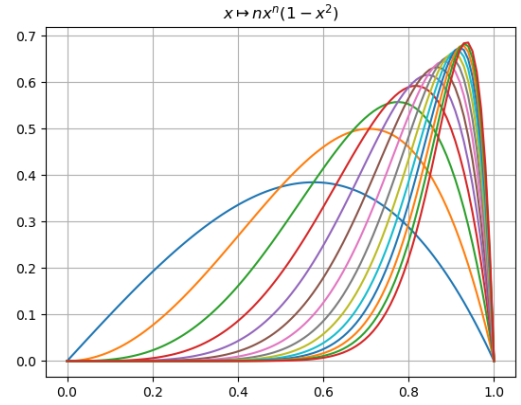
```
def f(x): return x*(1-x)

X=np.linspace(0,1,100)
Y=[f(x) for x in X]
plt.plot(X,Y)
plt.grid()
plt.show()
```



Par exemple, pour tracer les graphes des fonctions $f_n : x \mapsto nx^n(1-x^2)$ sur $[0,1]$ pour $n \in [\![1,10]\!]$:

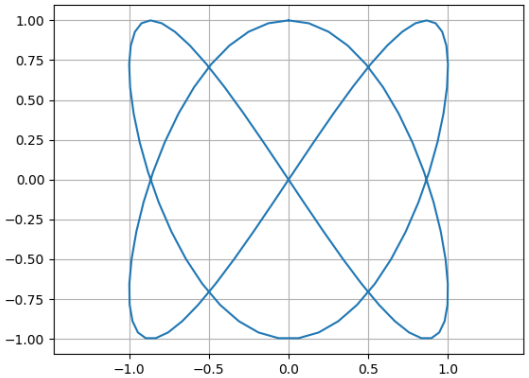
```
def f(n,x):
    return n*x**n*(1-x**2)
X=np.linspace(0,1,100)
 for n in range(11):
    Y=[f(n,x) for x in X]
    plt.plot(X,Y)
plt.grid()
plt.show()
```



I.3. Tracés d'arcs paramétrés

Exemple : allure de l'arc paramétré : $\begin{cases} x(t) = \sin(2t) \\ y(t) = \cos(3t) \end{cases}$

```
T=np.linspace(0,2*np.pi,100)
X=[np.sin(2*t) for t in T]
Y=[np.cos(3*t) for t in T]
plt.plot(X,Y)
plt.axis('equal')
plt.grid()
plt.show()
```



II. Calcul matriciel

La première chose à faire (après avoir créé et sauvegardé le fichier bien sûr !) est d'importer les modules `numpy` et `numpy.linalg` :

```
import numpy as np
import numpy.linalg as alg
```

II.1. Construction de matrices. Opérations.

On définit A à l'aide de la fonction `np.array` : A est le vecteur de ses vecteurs lignes.

On récupère

- ses coefficients par `A[i,j]`
- sa i^e ligne par `A[i,:]` **(attention les indices commencent à zéro !)**
- sa j^e colonne par `A[:,j]`

Matrices particulières :

- matrice nulle : `np.zeros(n)` ou `np.zeros((n,p))` (attention aux parenthèses)
- matrice diagonale : `np.diag([a,b,c])`
- Identité : `np.diag([1]*n)` ou bien `np.eye(n)`

Exercice 2. Écrire une fonction prenant comme entrée n et retournant $M_n = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ n & 0 & 2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & n \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix}$,

Opérations :

- somme et multiplication par un scalaire : `A+3*B`
- produit matriciel : `np.dot(A,B)` renvoie AB quand ce produit est défini.
  On peut aussi utiliser `A.dot(B)` qui est plus pratique si on veut multiplier plusieurs matrices.
- puissance d'une matrice carrée : `alg.matrix_power(A,5)` renvoie A^5.
- `np.transpose(A)` renvoie la transposée de A
- `alg.inv(A)` renvoie l'inverse de A (si elle est inversible)
- `alg.matrix_rank(A)` calcule le rang de A
- `concatenate((A,B),axis=0)` renvoie la matrice blocs $\begin{pmatrix} A \\ B \end{pmatrix}$ et `concatenate((A,B),axis=1)` renvoie $\begin{pmatrix} A & B \end{pmatrix}$

Exercice 3.

Écrire une fonction `tensoriel` prenant comme entrée A et $B \in \mathcal{M}_2(\mathbb{R})$ et retournant $\begin{pmatrix} a_{1,1}B & a_{1,2}B \\ a_{2,1}B & a_{2,2}B \end{pmatrix} \in \mathcal{M}_4(\mathbb{R})$.

Résolution d'un système linéaire :

`alg.solve(A,b)` permet de résoudre un système linéaire $Ax = b$ où $A \in \mathrm{GL}_n(\mathbb{R})$ et b est un tableau de taille n.

II.2. Réduction

- `np.trace(A)` et `alg.det(A)` renvoient respectivement la trace et le déterminant de A.

- Polynôme caractéristique : `np.poly(A)` renvoie la liste des coefficients de χ_A par degré croissant.

```
>>> A=np.array([[1,2],[-3,-4]])
>>> np.poly(A)
array([1.,  3.,  2.])
```

indique que le polynôme caractéristique de $A = \begin{pmatrix} 1 & 2 \\ -3 & -4 \end{pmatrix}$ est $\chi_A(X) = X^2 + 3X + 2$

- Spectre : `alg.eigvals(A)` renvoie le tableau des valeurs propres de A

- ⚠ Pour déterminer les éléments propres d'une matrice carrée :

  `alg.eig(A)` renvoie un couple (D, P) où

  ▷ D est le tableau des valeurs propres

  ▷ P est une matrice dont les **colonnes** sont des vecteurs propres associés aux valeurs propres précédentes.

Déterminons par exemple les éléments propres de la matrice précédente :

```
A=np.array([[1,2],[-3,-4]])
D,P=alg.eig(A)
print(D)      #réponse :  [-1. -2.]
V1=P[:,0]  #1er vecteur colonne de P
V2=P[:,1] # 2eme vecteur colonne de P

print(V1,A.dot(V1))   #On vérifie que V1 est vecteur propre associé à la valeur propre -1
#[ 0.70710678 -0.70710678] [-0.70710678  0.70710678]

#on vérifie pour le plaisir que A = P D P^{-1} :
print(P.dot(np.diag(D)).dot(alg.inv(P)))
#[[ 1.  2.]
# [-3. -4.]]
```

Exercice 4. *Centrale 18*

Soit $A_n = (a_{i,j}) \in \mathcal{M}_n(\mathbb{R})$ telle que, pour tout i, $a_{i,i} = 0$ et, pour tout $i \neq j$, $a_{i,j} = j$.

a) Écrire une fonction `M(n)` renvoyant A_n.

b) Écrire une fonction renvoyant les valeurs propres de A_n. Afficher A_n et ses valeurs propres pour n variant de 2 à 10. En déduire une conjecture sur A_n.

c) Montrer que les valeurs propres de A_n vérifient l'équation $\sum_{k=1}^{n} \frac{k}{x+k} = 1$.

d) En déduire que A_n est diagonalisable.

III. Analyse numérique

III.1. Résolution approchée d'équations

Pour résoudre une équation du type $f(x) = 0$, où f est une fonction d'une variable réelle, on utilise la fonction `fsolve` du module `scipy.optimize`. La première chose à faire (après avoir créé et sauvegardé le fichier bien sûr !) est donc

```
import numpy as np
import scipy.optimize as resol
```

La fonction `fsolve` prend deux arguments : la fonction f et une valeur initiale x_0 (point de départ de l'algorithme de résolution approchée, on choisira x_0 de manière raisonnable.) Elle renvoie la(les) solution(s) trouvée(s) sous forme de liste. Attention le résultat peut dépendre du choix de x_0 !

```
def f(x):
    return x**2-2
```

```
>>> resol.fsolve(f,1)
array([1.41421356])
>>> resol.fsolve(f,-3)
array([-1.41421356])
```

Exercice 5. *Oral Centrale*

a) Montrer que pour tout $n \in \mathbb{N}^*$, l'équation $n(x + x^3) = 1$ admet une unique solution, que l'on notera (x_n).

b) Représenter les 30 premiers termes de la suite (x_n). Conjecture ? Prouver ce résultat.

c) Trouver numériquement trois constantes a,b et c telles que $x_n = \dfrac{a}{n} + \dfrac{b}{n^2} + \dfrac{c}{n^3} + o\left(\dfrac{1}{n^3}\right)$.

d) Montrer que a, b et c existent et sont égales aux valeurs conjecturées précédemment.

III.2. Calcul approché d'intégrales

Pour calculer une valeur approchée de $\displaystyle\int_a^b f(t)\,\mathrm{d}t$, on utilise la fonction `quad` du module `scipy.integrate`.
La première chose à faire (après avoir créé et sauvegardé le fichier bien sûr !) est donc :

```
import scipy.integrate as integr
```

La fonction `quad` prend trois arguments : la fonction f, et les deux bornes d'intégration (éventuellement infinies). Le résultat un couple (v, ε), où v est une valeur approchée de l'intégrale et ε un majorant de l'erreur. Seul v nous intéresse et on appellera donc cette fonction sous la forme : `integr.quad(f,a,b)[0]`
Exemple : vérifions numériquement que $\displaystyle\int_0^{+\infty} \mathrm{e}^{-t^2}\,\mathrm{d}t = \frac{\sqrt{\pi}}{2}$

```
def f(t):
    return np.exp(-t**2)
```

```
>>> integr.quad(f,0,np.inf)[0]
  0.8862269254527579
>>> np.sqrt(np.pi)/2
  0.8862269254527579
```

Exercice 6. *Oral Centrale*
Soit $g : x \mapsto \displaystyle\int_0^{+\infty} \frac{t^{x-1}}{1+t}\,\mathrm{d}t$.

a) Montrer que le domaine de définition de g est $]0, 1[$. Étudier sa continuité.

b) Dessiner le graphe de g. Quelle propriété de symétrie semble-t-il posséder ? Le prouver.

IV. Équations différentielles

La première chose à faire (après avoir créé et sauvegardé le fichier bien sûr !) est d'importer les modules `numpy`, `scipy.integrate` et pour les dessins `matplotlib.pyplot` :

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

La fonction `odeint` du module `scipy.integrate` permet de résoudre numériquement des problèmes de Cauchy de la forme :

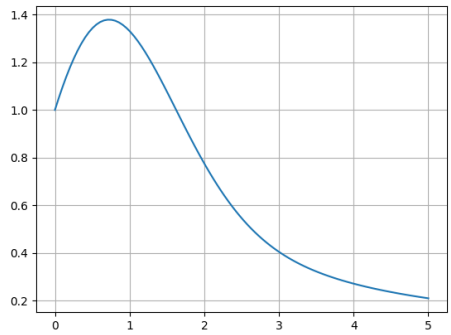
$$X'(t) = F(X(t), t) \quad \text{et} \quad X(t_0) = X_0$$

On commence par définir la fonction F (c'est le plus difficile), la valeur initiale X_0 et une liste d'instants $T = [t_0, t_1, \ldots, t_n]$. (attention : l'instant initial t_0 est le premier élément de la liste).
L'instruction `integr.odeint(F,X0,T)` renvoie le tableau des valeurs approchées des $X(t_k)$ où X est la solution du problème de Cauchy.

IV.1. ED Scalaires du 1er ordre

Exemple : allure du graphe sur $[0,5]$ de la solution de $y' = -ty + 1$ telle que $y(0) = 1$

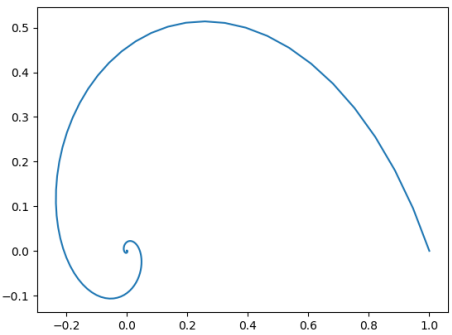
```
def f(y,t):
    return -t*y+1
T=np.linspace(0,5,100)
Y=integr.odeint(f,1,T)
plt.grid()
plt.plot(T,Y)
```



IV.2. Systèmes différentiels du 1er ordre

Exemple : allure de la trajectoire de la solution de $\begin{cases} x' = -x - 2y \\ y' = 2x - y \end{cases}$ telle que $x(0) = 1$ et $y(0) = 0$ pour $t \in [0, 10]$

```
def f(X,t):
    return np.array([-X[0]-2*X[1],2*X[0]-X[1]])
T=np.linspace(0,10,200)
X=integr.odeint(f,np.array([1,0]),T)
X=X[:,0]
Y=X[:,1]
plt.plot(X,Y)
plt.show()
```



IV.3. ED scalaires du 2e ordre. Vectorialisation.

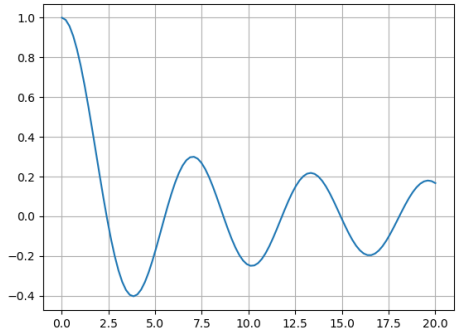
Exemple : représenter sur $[0,10]$ la solution f de l'équation différentielle $xy''+y'+xy=0$ vérifiant $f(0)=1$ et $f'(0)=0$.

(NB : on admet qu'il n'y en a qu'une, ce qui n'était pas garanti par le théorème de Cauchy)

Le plus délicat est de vectorialiser l'équation différentielle.

On pose $X=\begin{pmatrix} y \\ y' \end{pmatrix}$. y solution de (E) ssi $X'=\begin{pmatrix} y' \\ y'' \end{pmatrix}=\begin{pmatrix} y' \\ -y'/x-y \end{pmatrix}=F(X,x)$ où $F\left(\begin{pmatrix} x_0 \\ x_1 \end{pmatrix},t\right)=\begin{pmatrix} x_1 \\ -x_1/t-x_0 \end{pmatrix}$

```
def f(X,t):
    return np.array([X[1],-X[1]*1/t-X[0]])
T=np.linspace(0.01,20,100)
#j'ai pris 0.01 et non pas 0 car problème vu
#qu'on divise par t
Y=integr.odeint(f,np.array([1,0]),T)
# Y est un tableau bidimensionnel représentant
#la liste des [y(t_k),y'(t_k)]
print(Y)
Y=Y[:,0]
#on récupère seulement la liste des y(t_k)
plt.grid()
plt.plot(T,Y)
plt.savefig('bessel')
plt.show()
```



Exercice 7. Soient B et J les fonctions définies respectivement par les formules :

$$B(x)=\sum_{n=0}^{+\infty}(-1)^n\frac{x^{2n}}{4^n(n!)^2} \quad \text{et} \quad J(x)=\frac{1}{\pi}\int_0^{\pi}\cos(x\sin(t))\,\mathrm{d}t.$$

Tracer les graphes de B et J. Les comparer avec le graphe de f. Conjecture ? Démontrer le résultat conjecturé.

V. Probabilités

La première chose à faire (après avoir créé et sauvegardé le fichier bien sûr !) est d'importer la bibliothèque `numpy.random`

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

Quelques commandes utiles pour simuler les lois classiques :

- `rd.randint(a,b)` : renvoie un entier appartenant à l'intervalle entier $[a, b[$ selon la loi uniforme.

  Pour définir une v.a. qui suit la loi uniforme sur $\{-1, 1\}$: `2*rd.randint(0,2)-1`
- `rd.binomial(n,p)` : renvoie un entier compris entre 0 et n selon la loi binomiale $\mathcal{B}(n, p)$.

  En particulier `rd.binomial(1,p)` renvoie 0 ou 1 selon la loi de Bernoulli $\mathcal{B}(p)$.
- `rd.poisson(p)` : renvoie un entier naturel selon la loi de Poisson $\mathcal{P}(p)$.
- `rd.geometric(p)` : renvoie un entier de $\mathbb{N}^*$ selon la loi géométrique $\mathcal{G}(p)$.

Un troisième argument optionnel m permet de retourner directement un tableau de m valeurs.

Par exemple, `rd.binomial(1,0.5,100)` permet de simuler 100 tirages à pile ou face avec une pièce équilibrée.

Exercice 8. *Marche aléatoire*

Soit (X_i) une suite de va indépendantes suivant toutes la loi uniforme sur $\{-1, 1\}$.

On définit la suite (S_n) par $S_0 = 0$ et , pour $n \in \mathbb{N}^*$, $S_n = X_1 + X_2 + \cdots + X_n$.

Représenter $(S_n(\omega))$ pour une dizaine de valeurs de ω.

Exercice 9. *Longueur des première et deuxième séries , Centrale 2018, 2019*

On joue indéfiniment à pile ou face avec une pièce qui fait pile avec une probabilité p. Soit donc (X_n) une suite de va indépendantes suivant toutes $\mathcal{B}(p)$. On note Y la longueur de la première série de 0 ou de 1 consécutifs, et z la longueur de la deuxième série.

Par exemple, si $(X_n)_{n\in\mathbb{N}} = (1, 1, 1, 1, 0, 0, 0, 1, 1, 0, ...)$ on a $Y = 4$ et $Z = 3$.

Vu que $p \in]0, 1[$, Y est presque sûrement finie et Z est donc bien définie, elle-même à valeurs dans $\mathbb{N}^*$ (presque sûrement).

a) Écrire un programme permettant d'évaluer $\mathrm{E}(Y)$ et $\mathrm{E}(Z)$.

b) Déterminer la loi de (X, Y). En déduire la loi de Y. Calculer $\mathrm{E}(Y)$ et vérifier que $\mathrm{E}(Y) \geqslant 2$.

c) Montrer que $\mathrm{E}(Z) = 2$ (et ne dépend donc pas de p...)

VI. Polynômes

La première chose à faire (après avoir créé et sauvegardé le fichier bien sûr !) est

```
from numpy.polynomial import Polynomial
```

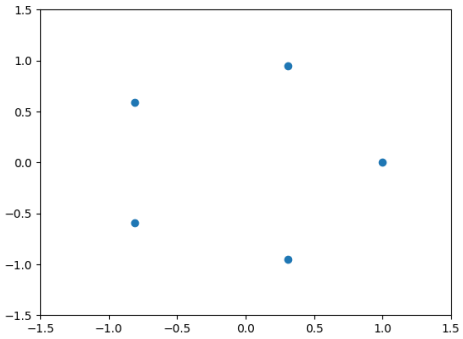
- `P=Polynomial([-3,2,0,1])` : P est le polynôme X^3+2X-3

  Il pourra être utile de définir une fois pour toutes le polynôme constant égal à 1 et le polynôme X par :

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

- `P.coef` renvoie les coefficients de P, ici `array([-3,2,0,1])`
- Pour avoir le degré et les racines complexes : `P.degree()` et `P.roots()`
- pour dériver, primitiver,et toute autre question, voir la notice de Centrale.
- Pour visualiser les racines complexes d'un polynôme :

```
P=Polynomial([-1,0,0,0,0,1])
L = P.roots()
X = [z.real for z in L]
Y = [z.imag for z in L]
plt.axis([-1.5,1.5,-1.5,1.5])
plt.plot(X,Y,"o")
plt.show()
```



Exercice 10. *Centrale 2017*

Soit $E=\mathbb{R}_4[X]$. Pour tout P et $Q\in E$, notons $\Phi(P,Q)=\int_{-2}^{2}P(t)\,Q(t)\mathrm{d}t$.

a) Montrer que Φ est un produit scalaire et l'implémenter.

b) Montrer que les sous-espaces des polynômes pairs et impairs sont supplémentaires orthogonaux dans E.

c) Déterminer une base orthonormale de E.

d) Pour $P\in E$, on pose $f(P)=2XP'+(X^2-4)P''$. Montrer que f est un endomorphisme symétrique de E. Déterminer la matrice de f dans la base canonique de E et en déduire les valeurs propres de f. Donner une base de vecteurs propres de f.

VII. Quelques exercices posés à l'oral de Centrale

Exercice 11. *Centrale 21*

Soient les matrices de $\mathcal{M}_{n+1}(\mathbb{R})$: $M_n = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ n & 0 & 2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & n \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix}$, $A_n = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & 2 & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ \vdots & & & \ddots & n \\ 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix}$, $B_n = \begin{pmatrix} 0 & \cdots & \cdots & \cdots & 0 \\ n & 0 & & & \vdots \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix}$

a) Créer une fonction Python $M(n)$ qui renvoie M_n.

b) Utiliser Python pour trouver les valeurs propres de M_i avec $2 \leq i \leq 4$. Conjecturer celles de M_n.

On note g et f les endomorphismes de $\mathbb{R}_n[X]$ canoniquement associés respectivement à B_n et M_n.

c) Calculer $g(P)$ pour $P \in \mathbb{R}_n[X]$.

d) Trouver des réels α et β tels que : $\forall t \in \mathbb{R} \setminus \{-1, 1\}$, $\frac{nt-\lambda}{t^2-1} = \frac{\alpha}{1-t} + \frac{\beta}{1+t}$.

e) Résoudre, sur $]-1, 1[$, l'équation différentielle : $(nt - \lambda)y(t) + (1 - t^2)y'(t) = 0$.

f) Donner une base de vecteurs propres de f.

g) Trouver une matrice U_n, à coefficients entiers, inversible, telle que $U_n^{-1} M_n U_n$ soit diagonale.

h) Programmer la matrice U_n et vérifier le résultat de la question précédente avec Python.

Exercice 12. *Centrale 18*

On pose, pour $n \in \mathbb{N}$, $u_n = \left[\binom{n}{0} \times \binom{n}{1} \times \cdots \times \binom{n}{n}\right]^{1/(n+1)}$. On pose, pour $n \in \mathbb{N}^*$, $v_n = \sqrt[n]{u_n}$.

a) Écrire un programme renvoyant u_n.

b) Vérifier numériquement que (v_n) converge.

c) Montrer que $\prod_{p=0}^{n} p! = \prod_{p=1}^{n} p^{n+1-p}$. En déduire $\prod_{p=0}^{n} \binom{n}{p} = \prod_{p=1}^{n} p^{2p-n-1}$.

d) Déterminer la limite de (v_n).

Exercice 13. *Centrale 2018*

On définit les fonctions f et g, sous condition de convergence, par :

$$f : x \mapsto \int_0^1 \frac{t^{x-1}}{1+t}\,\mathrm{d}t \quad , \quad g : x \mapsto \int_0^{+\infty} \frac{t^{x-1}}{1+t}\,\mathrm{d}t$$

a) Déterminer les ensembles de définition de ces deux fonctions et tracer leurs courbes représentatives.

b) i) Montrer que pour tout $x > 0$, on a : $f(x) = \sum_{k=0}^{+\infty} \frac{(-1)^k}{x+k}$

ii) En déduire une procédure de calcul d'une valeur de $f(x)$ à ε près.

La fonction aura comme paramètres x et ε. On pourra utiliser l'instruction $\mathtt{int}(np.floor(y))$ qui donne la partie entière de y. Tester cette fonction pour quelques valeurs de votre choix. Que donne cette fonction pour $x = 1/2$ et $\varepsilon = 10^{-4}$? Quelle est la valeur exacte ? Justifier la réponse.

c) Trouver une relation simple entre $f(x)$, $f(x)$ et $f(1-x)$ pour $x \in]0, 1[$. Vérifier cette relation graphiquement.

d) Montrer que f et g sont de classe $\mathcal{C}^1$ sur leurs ensembles de définition. En déduire les variations de ces fonctions.

Exercice 14. *Centrale 18,21*

Pour $n \in \mathbb{N}$, on pose $C_n = \binom{2n}{n}$ et $H_n = (C_{i+j-2})_{1 \leqslant i,j \leqslant n+1} \in \mathcal{M}_{n+1}(\mathbb{R})$.

a) Montrer que, pour tout $n \in \mathbb{N}$, la matrice H_n est diagonalisable.

b) Pour $n \in \{1, 2, 3\}$, calculer le déterminant de H_n et vérifier que les valeurs propres de H_n sont strictement positives.

c) Écrire un programme permettant d'étudier H_n.

d) Soit $A \in \mathcal{M}_{n+1}(\mathbb{R})$. Montrer que A est symétrique à valeurs propres strictement positives si et seulement s'il existe $P \in \mathrm{GL}_{n+1}(\mathbb{R})$ telle que $A = P^\top P$.

e) Pour $q \in \mathbb{N}$, on pose $P_q = (X+1)^{2q}$. Exprimer le coefficient de X^{i+j} dans $P_i P_j$ en fonction de C_{i+j}.

f) Démontrer le résultat général attendu.

Exercice 15. *Centrale 21*

Soit $n \in \mathbb{N}^*$. On pose $E_n = \mathbb{R}_n[X]$.

a) On pose $\varphi_n(P) = 4XP' - P''$ pour tout $P \in E_n$. Montrer que φ_n est un endomorphisme de E_n et déterminer sa matrice A_n dans la base canonique.

b) Montrer que φ_n est diagonalisable. Montrer que φ_n possède un unique vecteur propre S_n qui est unitaire de degré n et indiquer la valeur propre associée.

c) Écrire des fonctions A(n) et S(n) donnant A_n et S_n.

On définit la suite (H_n) de $\mathbb{R}[X]$ par : $H_0 = 1$, $H_1 = X$ et, pour $n \geq 2$, $H_n = XH_{n-1} - \frac{n-1}{4}H_{n-2}$.
Un code H(n) donnant H_n est fourni.

d) Tracer H_n et S_n sur $[-2, 2]$ pour $1 \leq n \leq 6$. Que peut-on conjecturer ?

e) Conjecturer le nombre de racines réelles de S_n.

f) Calculer $\varphi_{n-1}(S_n')$. Prouver les conjectures des questions d) et e)

Exercice 16. *Centrale 18*

On pose, pour $n \in \mathbb{N}$, $I_n = \displaystyle\int_0^1 \frac{\mathrm{d}t}{(1+t)^n\sqrt{1-t}}$, $J_n = \displaystyle\int_0^{1/2} \frac{\mathrm{d}t}{(1+t)^n\sqrt{1-t}}$.

a) Montrer que (I_n) et (J_n) sont bien définies.

b) Représenter $(I_n)_{0 \leqslant n \leqslant 50}$.

c) Montrer que (I_n) est monotone et convergente. Déterminer sa limite.

d) Conjecturer α tel que $(n^\alpha I_n)$ converge vers une limite non nulle.

e) Montrer que, pour ce α, $n^\alpha(I_n - J_n) \to 0$.

f) Pour $x \geqslant 0$, montrer que $\ln(1+x) \geqslant \dfrac{x}{1+x}$.

g) En déduire un équivalent de I_n.

Exercice 17. *Centrale 18*

Soient $\alpha \in \mathbb{R}$ et $f_\alpha : x \mapsto \displaystyle\int_0^{+\infty} \frac{t^\alpha e^{-tx^2}}{1+t^2}\,\mathrm{d}t$.

a) Déterminer l'ensemble A des valeurs de α pour lesquelles f_α est définie sur $\mathbb{R}$.

b) Tracer sur l'intervalle J les graphes de f_α pour une dizaine de valeurs de α dans les cas suivants :
(i) $J = [-10, 10]$ et $\alpha \in A$; (ii) $J = [-1, 1]$ et $\alpha \in A \cap \mathbb{R}^+$; (iii) $J = [-1, 1]$ et $\alpha \in A \cap \mathbb{R}^-$.
Conjecturer la limite en $+\infty$ de f_α, la continuité, la dérivabilité de f_α en 0.

c) Comparer $f_\alpha(0)$ et $f_{-\alpha}(0)$. En utilisant $f_\alpha + f_{-\alpha}$, montrer que $f_\alpha(0) \geqslant \pi/2$.

d) Soit $\alpha \in A$. Montrer que f_α est continue sur $\mathbb{R}$ et de classe $\mathcal{C}^1$ sur $\mathbb{R}^*$. Déterminer la limite de f_α en $+\infty$.

e) Simplifier $f_{1/2}(x) - f_{1/2}(0)$ en posant $u = tx^2$. L'application $f_{1/2}$ est-elle dérivable en 0 ?

Exercice 18. *Problème du collectionneur, Centrale 18*

Soit $r \in \mathbb{N}^*$. Soient $(\Omega, \mathcal{T}, \mathbb{P})$ un espace probabilisé, $(X_k)_{k \geqslant 1}$ une suite de variables aléatoires mutuellement indépendantes suivant une loi uniforme sur $\{1, \dots r\}$, $Z_1, \dots, Z_r$ des variables aléatoires indépendantes telles que Z_i suive la loi géométrique de paramètre $\dfrac{r-i+1}{r}$.
On pose $S_r = Z_1 + \cdots + Z_r$ et $T_r = \inf\{n \in \mathbb{N}^*,\ \mathrm{Card}\{X_1, \dots, X_n\} = r\}$.

a) Exprimer $E(S_r)$ et $V(S_r)$ en fonction de $H_r = \displaystyle\sum_{k=1}^{r} \frac{1}{k}$.

b) Représenter graphiquement $(H_r / \ln r)$ pour $r \in \{2, \dots, 50\}$. Conjecture ? La démontrer.

c) Comparer sur des simulations les lois de S_r et T_r. Conjecture ? La démontrer.

Exercice 19. *Somme d'un nombre aléatoire de variables aléatoires, Centrale 18*

Soient $(X_k)_k$ une suite de variables aléatoires réelles discrètes suivant la même loi et mutuellement indépendantes et N une variable aléatoire à valeurs dans $\mathbb{N}$ indépendante des précédentes. On pose $Y = \displaystyle\sum_{k=1}^{N} X_k$.

a) i) Définir une fonction permettant de calculer Y pour $X_1 \hookrightarrow \mathscr{B}(50, \frac{1}{50})$ et $N \hookrightarrow \mathscr{P}(\frac{1}{12})$.
ii) Réaliser dix fois de suite une série de 1000 expériences et donner la moyenne et l'écart-type de Y.

b) On se replace dans le cas général. Exprimer G_Y en fonction de G_{X_1} et G_N. En déduire $E(Y)$.
Déterminer $E(Y)$ pour $N \hookrightarrow \mathscr{P}(m)$ et $X_1 \hookrightarrow \mathscr{B}(n, p)$.

c) Dans une entreprise de 50 employés, soit Y la variable aléatoire indiquant le nombre de blessés dans une période τ, N indiquant le nombre d'accidents dans cette période et X_1 le nombre de blessés par accident. On suppose que $X_1 \hookrightarrow \mathscr{B}(50, \frac{1}{50})$ et $N \hookrightarrow \mathscr{P}(\frac{1}{12})$.
i) Quel est le nombre moyens de blessés durant la période τ ?
ii) Quelle est la probabilité qu'il ait au moins un blessé ?
iii) Déterminer la variance de Y.

VIII. Indications et éléments de réponse

Exercice 1 :

```
def u(n):
    x,y=1,1
    for k in range(1,n+1):
        x,y=y,y+x/k
    return x
X=[n for n in range(1,100)]
Y=[ u(n+1)/u(n) for n in X]
plt.grid()
plt.plot(X,Y)
plt.show()  # u(n+1)/u(n) semble tendre vers 1
```

Posons $\delta_n = \dfrac{u_{n+1}}{u_n} - 1$. *On a facilement* $0 \leqslant \delta_n = \dfrac{1}{n(\delta_{n-1}+1)} \leqslant \dfrac{1}{n}$ *et on conclut par th de convergence par encadrement.*

On peut aussi poser $x_n = \dfrac{u_{n+1}}{u_n}$, *de sorte que pour tout* $n \in \mathbb{N}^*$, $x_n = 1 + \dfrac{1}{nx_{n-1}}$.

Il vient $x_n \geqslant 1$, *puis* $1 \leqslant x_n \leqslant 1 + \dfrac{1}{n}$ *pour tout* $n \in \mathbb{N}^*$ *et on conclut par encadrement.*

Exercice 2 :

```
def M(n):
    A=np.zeros((n+1,n+1))
    for i in range(n+1):
        for j in range(n+1):
            if j==i+1: A[i,j]=j
            elif i==j+1: A[i,j]=n+1-i
    return A
```

Exercice 3 :

```
def tensoriel(A,B):
    L0=np.concatenate((A[0,0]*B,A[0,1]*B),axis=1)
    L1=np.concatenate((A[1,0]*B,A[1,1]*B),axis=1)
    return np.concatenate((L0,L1),axis=0)
```

Exercice 4 :

```
def M(n):
    A=np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            if i!=j: A[i,j]=j+1
    return A
for k in range(2,10):
    print(alg.eigvals(A(k)))
```

d) On montre que A_n *a n valeurs propres distinctes.*

Exercice 5 :

a) $f_n : x \mapsto n(x+x^3)$ est continue et strictement croissante sur $\mathbb{R}$, vaut 0 en 0 et $2n > 1$ en $x = 1$, donc elle s'annule une et une seule fois sur $\mathbb{R}$ et ce zéro est dans $[0,1]$.

b) On définit une fonction `x` qui prend n en entrée et retourne x_n.

```
def x(n):
    def f(t):
        return n*(t+t**3)-1
    return resol.fsolve(f,0)
N=[n for n in range(1,31)]
X=[x(n) for n in range(1,31)]
plt.plot(N,X)
plt.show()
```

La suite (x_n) semble tendre vers 0, ce que l'on prouve en utilisant la méthode habituelle ou bien en remarquant que $f_n(1/n) \geqslant 1$ donc $0 \leqslant x_n \leqslant 1/n$.

c) On calcule nx_n pour quelques valeurs de n :

```
>>>[n*x(n)[0] for n in range(20,30)]
[0.9975185646221213, 0.9977477134978423, 0.9979465859530401, 0.9981202808394914, 0.998272868836279
```

Il semble que $nx_n \to 1$, donc que $a = 1$.

On déterminer b en calculant $n^2(x_n - a/n)$ pour de grandes valeurs de n

```
>>> [n**2*(x(n)[0]-1/n) for n in range(40,50)]
[-0.02495324185284309, -0.024346819100153898, -0.02376912300366442, -0.02321816285295746, -0.02269
```

Donc b est vraisemblablement nul. On détermine ensuite c en calculant $n^3(x_n - a/n - b/n^2)$ pour de grandes valeurs de n

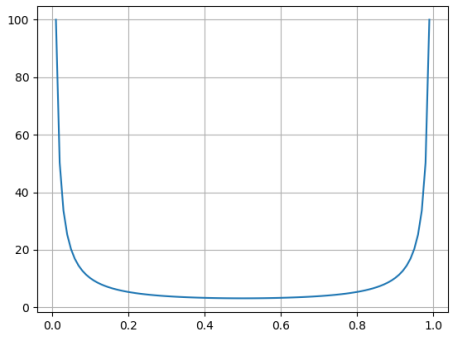
```
>>> [n**3*(x(n)[0]-1/n) for n in range(40,50)]
# [-0.9981296741137236, -0.9982195831063098, -0.9983031661539056, -0.9983810026771707, -0.99845360
```

d) $x_n = \dfrac{1}{n} - \dfrac{1}{n^3} + o\left(\dfrac{1}{n^3}\right)$

Exercice 6 :

On commence par définir la fonction g :

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



Autre manière possible de définir g :

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

```
X=np.linspace(0.01,0.99,100)
Y=[g(x) for x in X]
plt.plot(X,Y)
plt.grid()
plt.show()
```

On prouve que $g(x) = g(1-x)$ avec le changement de variable $u = 1/t$

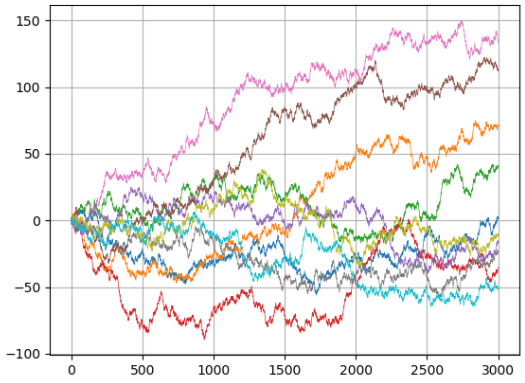
Exercice 7 :

```
def B(n,x):
    s=1
```

```
    u=1
    for k in range(1,n+1):
        u=-x**2*u/(4*k**2)
        s=s+u
    return s
T=np.linspace(0.01,20,100)
Y=[B(100,t) for t in T]
plt.plot(T,Y)
def J(x):
  return 1/np.pi*integr.quad(lambda t:np.cos(x*np.sin(t)),0,np.pi)[0]
les_z=[J(x) for x in T]
plt.plot(T,les_z)
plt.show()
```

Exercice 8 :

```
def marche(n):
    s=[0]*n
    for i in range(1,n):
        s[i]=s[i-1]+2*rd.randint(0,2)-1
    return(s)
for i in range(10):
    S=marche(3000)
    N=[n for n in range(3000)]
    plt.plot(N,S,linewidth=0.5)
plt.grid()
plt.show()
```



Exercice 9 :

```
def bern(p):#on définit la loi de Bernoulli
    return rd.binomial(1,p)
def YZ(p): #calcule Y et Z
    x=bern(p) ;  y=1
    while x==bern(p):
        y=y+1
    z=1
    while x==1-bern(p):
        z=z+1
    return [y,z]
def esperances(p,N):
    SommeY=0 ;  SommeZ=0
    for i in range(N):
        SommeY=SommeY+YZ(p)[0]
        SommeZ=SommeZ+YZ(p)[1]
    return [SommeY/N,SommeZ/N]
# >>> esperances(0.5,1000)          # >>> esperances(0.1,1000)
# [2.079, 2.005]                   # [8.793, 1.968]
```

Exercice 10 :

a)

```
from numpy import *
from numpy.polynomial import Polynomial
import scipy.integrate as integr
def Phi(P,Q):
    def f(t):
        return P(t)*Q(t)
    return integr.quad(f,-2,2)[0]
def Norme(P):
    return sqrt(Phi(P,P))
Un=Polynomial([1]) #on définit les polynômes 1 et X
X=Polynomial([0,1])
```

b) observer qu'une fonction impaire a une intégrale nulle sur $[-2,2]$.

c) On code l'algorithme de Gram-Schmidt (L est une liste de polynômes, BON stocke son orthonormalisée pour le produit scalaire Φ.)

```
def orthonormalise(L):
    BON=[L[0]/Norme(L[0])]
    for i in range(1,len(L)):
        s=L[i]
        for j in range(i):
            s=s-Phi(L[i],BON[j])*BON[j]/Norme(BON[j])**2
        BON.append(s/Norme(s))
    return BON
BON=orthonormalise([Un,X,X**2,X**3,X**4])
for k in range(5):
    print(BON[k].coef)
#on vérifie que la base obtenue est bien orthonormée :
ps=array([[Phi(BON[i],BON[j]) for j in range(4)] for i in range(4)])
```

d) La matrice de f dans la base $(X^k)_{0\leqslant k\leqslant 4}$ est triangulaire supérieure, donc son spectre se lit sur sa diagonale. Pour montrer que f est symétrique, le plus simple est d'observer que la matrice de f dans BON est symétrique (elle est diagonale !)

```
def f(P):
    return 2*X*P.deriv()+(X**2-4)*P.deriv(2)
for k in range(5):
    print(f(BON[k]).coef, "et", (f(BON[k])//BON[k]).coef)
```

Exercice 11 :

a) b)

```
import numpy as np
import numpy.linalg as alg
def M(n):
    A=np.zeros((n+1,n+1))
    for i in range(n+1):
        for j in range(n+1):
            if j==i+1: A[i,j]=j
            elif i==j+1: A[i,j]=n+1-i
    return A
```

```
for n in range(2,5):
    print(M(n),alg.eigvals(M(n)))
```

On conjecture que M_n a $n+1$ valeurs propres distinctes, les $n-2k$, avec k dans $[\![0,n]\!]$. ; en particulier M_n est DZ

c) $g(P) = nXP - X^2P'$ et $f(P) = g(P) + P'$.

f) Pour $k \in [\![0,n]\!]$, $(X-1)^k(X+1)^{n-k}$ est un vecteur propre de f, associé à la valeur propre $n-2k$

g) Prendre pour U_n la matrice de passage de la base canonique de $\mathbb{R}_n[X]$ à la base $\left((X-1)^k(X+1)^{n-k}\right)_{0\leqslant k\leqslant n}$

h)

```
from numpy.polynomial import Polynomial
P=Polynomial([-1,1])
Q=Polynomial([1,1])
def U(n):
    return np.transpose(np.array([(P**k*Q**(n-k)).coef for k in range(n+1)]))
for n in range(2,5):
    print(alg.inv(U(n)).dot(M(n)).dot(U(n)))
```

Exercice 12 :

On commence par définir une fonction qui calcule $\binom{n}{k}$ *via la formule* $\binom{n}{k} = \dfrac{n(n-1)\cdots(n-k+1)}{k!} = \prod_{i=0}^{k-1} \dfrac{n-i}{k-i}$

```
def binomial(n,k):
    x=1
    for i in range(k):
        x=x*(n-i)/(k-i)
    return x
```

Cette fonction retourne un flottant ; si on veut un résultat entier, écrire `return int(x)` *à la dernière ligne ou* `x=(x*(n-i))//(k-i)` *(division dans* $\mathbb{N}$*) dans l'avant- dernière.*

```
def V(n):
    p=1
    for k in range(n):
        p=p*binomial(n,k)
    return p**(1/(n*(n+1)))
```

Attention : quand on essaie de calculer $V(n)$ *pour* $n > 30$*, on obtient un message d'erreur car les calculs intermédiaires portent sur des flottants trop grands. On peut réparer cela :*

```
def V2(n):
    p=1
    for k in range(n):
        p=p*binomial(n,k)**(1/(n*(n+1)))
    return p
```

On aurait pu se montrer plus astucieux en remarquant que $\binom{n}{k+1} = \dfrac{n-k}{k+1}\binom{n}{k}$.

```
def VV(n):
    p = 1 ; c = 1
    for k in range(n) :
        c =(n-k)/(k+1)**(1/(n*(n+1))*c
        p = p * c
    return p
```

On montre que $v_n \xrightarrow[n\to\infty]{} \sqrt{e}$

Exercice 13 : .

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.integrate as integr
def f(x):
    def h(t):
        return t**(x-1)/(1+t)
    return integr.quad(h,0,1)[0]
def g(x):
    def h(t):
        return t**(x-1)/(1+t)
    return integr.quad(h,0,np.inf)[0]
X=np.linspace(0.01,2,100)
Y=[f(x) for x in X]
plt.plot(X,Y)
plt.show()
X=np.linspace(0.01,0.9,100)
Y=[g(x) for x in X]
plt.plot(X,Y)
plt.show()
```

```
def F(x,eps):
    N=int(np.floor(1/eps))+1
    s=0
    for k in range(N):
        s+=(-1)**k/(x+k)
    return(s)

# In [1]: F(1/2,10**(-4))
# Out[1]: 1.5708463217952673
#
# In [2]: f(1/2)
# Out[2]: 1.5707963267949292

X=np.linspace(0.01,0.99,100)
Y=[f(x)+f(1-x) for x in X]
plt.plot(X,Y)
plt.show()
```

On montre que $f(x) + f(1-x) = g(x)$ avec le changement de variable $u = 1/t$.

Exercice 14 :

On commence par définir une fonction qui calcule $\binom{n}{k}$ via la formule $\binom{n}{k} = \dfrac{n(n-1)\cdots(n-k+1)}{k!} = \displaystyle\prod_{i=0}^{k-1} \frac{n-i}{k-i}$

```
def binomial(n,k):
    x=1
    for i in range(k):
        x=x*(n-i)/(k-i)
    return x
```

Cette fonction retourne un flottant ; si on veut un résultat entier, écrire **`return int(x)`** *à la dernière ligne.*

On peut aussi faire : `from scipy.special import binom`

```
def H(n):
    A=np.zeros((n+1,n+1))
    for i in range(n+1):
        for j in range(n+1):
            A[i,j]=binomial(2*(i+j),i+j)
    return A
for k in range(2,10):
    print(alg.det(H(k)),alg.eigvals(H(k)))
```

Exercice 15 :

A_n est triangulaire supérieure et ses coeffcients diagonaux sont tous distincts. Ainsi A_n possède $n+1$ valeurs proprees distinctes (les $4k, 0 \leqslant k \leqslant n$) donc est diagonalisable et ses espaces propres sont des droites.

```
def Phi(P):
    return 4*Polynomial([0,1])*P.deriv()-P.deriv(2)

X=Polynomial([0,1])

def A(n):
    A=np.zeros((n+1,n+1))
    for j in range(n+1):
        r=len((Phi(X**j)).coef)
        for i in range(r):
            A[i,j]=(Phi(X**j)).coef[i]
    return A

def S(n):
    D,P=alg.eig(A(n))
    return Polynomial(P[:,n])/P[n,n]

for n in range(2,5):
    print(S(n),Phi(S(n))/(4*n))
```

Un code donnant H_n est par exemple :

```
def H(n):
    if n==0: return Polynomial([1])
    elif n==1: return Polynomial([0,1])
    else:return X*H(n-1)-(n-1)/4*H(n-2)
```

On dessine les graphes de S_n et H_n pour de petites valeurs de n :

```
lesx=np.linspace(-2,2,100)
lesy=[S(5)(x) for x in lesx]
lesz=[H(5)(x) for x in lesx]
plt.axis([-2,2,-3,3])
plt.grid()
plt.plot(lesx,lesy)
plt.plot(lesx,lesz)
plt.show()
```

Tout porte à croire que $H_n = S_n$ pour tout n et que S_n admet n racines réelles distinctes.

Pour prouver cela, commencer par observer que $\varphi_{n-1}(S_n') = 4(n-1)S_n'$ et que donc $S'n = nS_{n-1}$.

Exercice 16 :

```
def I(n):
    def g(t):
        return 1/(1+t)**n/np.sqrt(1-t)
    L=integr.quad(g,0,1)
    return L[0]
print([np.log(I(n))/np.log(n)for n in range(25,50)])
```

Exercice 17 :

Le code ci-dessous permet de tracer les graphes des f_α sur $[-1,1]$ pour $\alpha \in \{-0.8,-0.6,-0.4,-0.2,0,0.2,0.4,0.6,0.8\}$.

```
import numpy as np
import scipy.integrate as integr
import matplotlib.pyplot as plt
def f(x,a):
    def h(t):
        return t**a/(1+t**2)*np.exp(-t*x**2)
    return integr.quad(h,0,np.inf)[0]
for a in range(1,10):
    X=np.linspace(-1,1,100)
    Y=[f(x,-1+a/5) for x in X]
    plt.plot(X,Y)
    plt.grid()
    plt.show()
```

Exercice 18 :

```
import numpy.random as rd
def H(n):
    somme=0
    for k in range(1,n+1):
        somme=somme+1/k
    return somme


def simulT(r):
    valeurs_diff=[] #stocke la liste des valeurs différentes déjà obtenues
    nb_diff=0       # nombre de valeurs différentes déjà obtenues
    nb_essais=0     # nombre d'essais

    while nb_diff<r:
        nb_essais=nb_essais+1
        v=rd.randint(1,r+1)          #on tente une nouvelle valeur
        if v not in valeurs_diff:    # v est une nouvelle valeur
            valeurs_diff.append(v)
            nb_diff=nb_diff+1
    return nb_essais
def S(r):
    s=0
    for i in range(1,r+1):
        s=s+rd.geometric((r-i+1)/r)
    return s

for k in range(20):
    print(simulT(10),S(10)) #on essaie de comparer les lois de S et T

def EspT(r,N):#moyenne de T sur N essais
    s=0
    for i in range(N):
        s=s+simulT(r)
```

```
    return s/N

def EspS(r,N):#moyenne de S sur N essais
    s=0
    for i in range(N):
        s=s+S(r)
    return s/N

>>> EspT(100,10**4),EspS(100,10**4),100*H(100)
(517.2425, 518.3053, 518.737751763962)
```

On cherche à compléter un album Panini qui contient r images. Lorsqu'on achète une boite de céréales, on obtient une image (les images sont réparties aléatoirement et uniformément dans les boites de céréales). T_r donne le nombre de boites de céréales qu'il faut acheter pour obtenir la collection complète. Pour compléter un album Panini qui contient 100 images, il faut acheter en moyenne 518 images.

Exercice 19 :

```
def Y():
    n = rd.poisson(1/12)
    if n == 0:
        return(0)
    else:
        s = 0
        for i in range(n):
            s=s+rd.binomial(50,1/50)
    return(s)
```

On approche l'espérance par la moyenne et l'écart-type par la racine carrée de la moyenne des carres moins le carre de la moyenne.

```
for i in range(10):
    m1 = 0
    m2 = 0
    for k in range(1000):
        y = Y()
        m1 += y
        m2 += y**2
    Esp=m1/1000
    Ecart=np.sqrt(m2/1000- (m1/1000)**2)
    print([Esp, Ecart])
```

b) On obtient (question classique) $G_Y = G_N \circ G_X$. D'où $\mathrm{E}(Y) = \mathrm{E}(N)\mathrm{E}(X) = mnp$.

c) ii) On calcule la probabilité de l'évènement contraire : $P(Y = 0) = G_Y(0) = G_N(G_X(0)) \approx 0.95$.

c) iii) Expliciter G_Y et utiliser $V(Y) = G_Y''(1) + G_Y'(1) - G_Y'(1)^2$