

Nom :

# Mathématiques approfondies

## Cours ECG 2

### Python - partie I

#### Thèmes

1. Révisions
2. Exemples en algèbre linéaire
3. Variables aléatoires discrètes
4. Variables aléatoires à densité
5. Fonctions de deux variables
6. Sujets



Lycée Saint Louis 2024/2025



# Python - semestre I

*La théorie, c'est quand on sait tout et que rien ne fonctionne. La pratique, c'est quand tout fonctionne et qu'on ne sait pas pourquoi. Ici, nous avons réuni théorie et pratique : rien ne fonctionne... et on ne sait pas pourquoi!*

ALBERT EINSTEIN

## 1 Révisions

### 1.1 Boucles for, boucles while

#### Premiers exemples

##### Exercice 1. ✧

Que fait le code suivant?

Editeur

```
x=float(input("Entrez un réel positif"))
n=0
while n<=x-1 :
    n+=1
print(n)
```

##### Exercice 2. ✦ Le grand théorème de Fermat

1. Écrire un programme Python qui calcule le nombre de triplets  $(a, b, c) \in \mathbb{N}^3$  solutions de

$$a^2 + b^2 = c^2 \quad \text{et} \quad 1 \leq a \leq b \leq c \leq 50.$$

2. Même question en modifiant l'exposant 2 par 3, 4, 5... Que remarque-t-on?

##### Exercice 3. ✦ Une preuve à l'aide de Python

L'objectif est de déterminer tous les entiers naturels  $a, b, c$  et  $n$  tels que  $a! + b! + c! = n!$ . On suppose qu'une telle solution existe.

1. Prouver que  $n! \leq 3(n-1)!$ .
2. En déduire que  $n$  est inférieur à 3 et  $a, b, c$  inférieurs à 2.
3. Écrire un programme qui teste toutes les solutions avec  $a, b, c \in \llbracket 0; 2 \rrbracket$  et  $n \in \llbracket 0; 3 \rrbracket$ .
4. Conclure sur l'ensemble des solutions.

##### Exercice 4. ✦✦ Énigme - La bataille de la rivière Baigou

En avril 1400, face aux troupes du général Ming Li Jinglong, combien étaient les hommes de Zhu Di, groupés en 13 carrés identiques et n'en formant plus qu'un seul lorsque leur chef les a rejoints?

##### Exercice 5. ✦✦ Un peu d'ordre!

On s'interdit dans cet exercice d'utiliser les commandes préprogrammées `np.min` et `np.max`.

1. Écrire une fonction `maxi` qui prend comme argument deux nombres réels et affiche le maximum de ces deux nombres.
2. En déduire une fonction `maximum` qui prend en argument une matrice ligne `x` et renvoie le maximum des éléments de `x`.
3. En utilisant seulement la fonction `maximum`, comment obtenir un programme qui calcule le minimum?
4. Donner un programme qui teste si tous les coefficients `x` ne sont pas identiques et dans ce cas renvoie le deuxième maximum.

## Cas particuliers des suites récurrentes, des sommes et produits

### Exercice 6. ♦ Calcul d'un $n$ -ième terme via Python

1. Écrire une fonction qui prend  $n$  en argument et qui renvoie les  $n$ -ièmes termes des suites  $u$  et  $v$  définies par les relations de récurrence :

$$\forall n \in \mathbb{N}, \quad \begin{cases} u_0 = 3 \\ u_{n+1} = \sqrt{u_n^2 + 1} \end{cases} \quad \text{et} \quad \begin{cases} v_0 = 1 \\ v_{n+1} = v_n + 2n + 1. \end{cases}$$

2. Conjecturer et prouver une formule simple pour  $u_n$  et  $v_n$ .

### Exercice 7. ♦ Sinus hyperbolique

*D'après EDHEC 2022*

Écrire une fonction Python qui prend en argument un entier  $n$ , un réel  $x$  non nul et renvoie la somme partielle d'ordre  $n$  de la série  $\sum 1/\text{sh}(kx)$ . Que dire de la convergence ?

La fonction  $\text{sh}$  est la fonction sinus hyperbolique définie sur  $\mathbb{R}$  par  $\text{sh}(x) = (e^x - e^{-x})/2$ .

### Exercice 8. ♦ Variante de la suite de Syracuse

Soit  $a \in \mathbb{N}$ . On définit la suite  $u$  par son premier terme  $u_0 = a$  et

$$\forall n \in \mathbb{N}, \quad u_{n+1} = \begin{cases} u_n/2 & \text{si } u_n \text{ pair} \\ u_n + 5 & \text{sinon.} \end{cases}$$

1. Écrire une fonction qui prend en argument  $u_0 \in \mathbb{N}^*$  et  $n$  et affiche les  $n + 1$  premiers termes de la suite.
2. Tester pour  $a \in \llbracket 1; 6 \rrbracket$ . Que constatez vous ?

### Exercice 9. ♦ Conjecture et limite


On étudie dans cet exercice la suite définie par

$$\forall n \in \mathbb{N}, \quad u_n = \frac{n!}{\sum_{k=0}^n k!}.$$

1. Écrire une fonction `facto` qui prend en argument un entier  $n$  et renvoie la matrice ligne

$$[ 0! \quad 1! \quad 2! \quad 3! \quad \dots \quad (n-1)! \quad n! ].$$

2. En déduire un programme qui prend en argument  $n$  et renvoie  $u_n$ .
3. Tester et conjecturer la limite de la suite  $u$ . Prouver votre conjecture.


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

**Exercice 11.** ♦♦♦ On pose pour tout  $n \in \mathbb{N}$

$$u_n = \int_0^{\frac{\pi}{4}} \tan(t)^n dt.$$

1. Calculer  $u_0, u_1$  et pour tout  $n \in \mathbb{N}$ ,  $u_n + u_{n+2}$ .
2. En déduire un programme qui prend en argument  $n$  et renvoie la matrice ligne


$$[ u_0 \quad u_1 \quad u_2 \quad \dots \quad u_{2n-1} \quad u_{2n} ].$$

**Exercice 12.** ♦  On sait que la suite de terme général  $u_n = \sum_{k=1}^n \frac{1}{k}$  tend vers  $+\infty$ .

Écrire un programme qui prend en argument un réel  $A$  et renvoie le plus petit entier  $n$  tel que  $u_n \geq A$ .

**Exercice 13.** ♦ On définit la suite  $(u_n)_{n \in \mathbb{N}^*}$  par  $u_n = \prod_{k=1}^n \left(1 - \frac{1}{2k}\right)$ .

1. Écrire un programme qui prend en argument  $n$  et renvoie  $u_n$ .
2. On admet que la suite  $u$  converge vers 0.  
Écrire un programme qui renvoie la plus petite valeur  $n$  pour laquelle  $u_n \leq 10^{-3}$ .

**Exercice 14.** ♦  Écrire un programme qui prend en argument une fonction  $f$ , un entier naturel non nul  $n$  et deux réels  $a, b$  (avec  $a < b$ ) et renvoie la somme de Riemann d'ordre  $n$  de  $f$ . C'est-à-dire :

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

Tester ensuite votre programme en approximant  $\pi = \int_0^1 \frac{4 dt}{1+t^2}$ .

### Exercice 15. ♦♦ Calculs des coefficients binomiaux

#### 1. Méthode 1.

- Écrire une fonction `facto` qui prend un entier naturel  $n$  en argument et renvoie  $n!$ .
- En utilisant la formule explicite des coefficients binomiaux, en déduire une fonction `binom1` qui prend en argument deux entiers  $p$  et  $n$  et renvoie  $\binom{n}{p}$ .
- Tester votre programme. Que pouvez vous conclure sur son efficacité?

#### 2. Méthode 2.

- Justifier que pour  $p, n \in \mathbb{N}^*$  avec  $p \leq n$  : 
$$\binom{n}{p} = \prod_{i=1}^p \frac{n-p+i}{i}.$$
- Compléter le programme suivant permettant le calcul de  $\binom{n}{p}$ .

Éditeur

```
if p < n :
    coeff = ...

    for i in range(...):
        coeff = ...

    return coeff
else :
    return ...
```

#### 3. Méthode 3.

Compléter et expliquer le programme suivant qui permet de construire le triangle de Pascal.

Éditeur

```
from numpy import eye # On va utiliser la matrice identité

def binom3(n):
    pascal = eye(n) # On part de la matrice identité

    for i in range(...):
        pascal[i,0] = 1

        for j in range(...):
            pascal[i,j] = ...

    return pascal
```

*Rappels.* La commande `eye(n)` renvoie une matrice avec  $n$  lignes et  $n$  colonnes avec des zéros partout sauf sur la diagonale qui est remplie de 1. De plus, attention au décalage d'indices, `pascal[i, j]` renvoie le coefficient à l'intersection de la  $(i + 1)$ -ème ligne et de la  $(j + 1)$ -ème colonne du tableau.

## 1.2 La bibliothèque matplotlib.pyplot

### Grphe associé à une suite

#### Exercice 16. ♦ Représentation des termes d'une suite - L'effet papillon

On considère la suite définie par la récurrence :

$$u_0 \in ]0; 1[, \quad \forall n \in \mathbb{N}, \quad u_{n+1} = \lambda u_n(1 - u_n).$$

#### 1. Étude théorique pour $\lambda \in ]0; 1]$ .

- Justifier que pour tout  $x \in [0; 1]$ ,  $x(1 - x) \in [0; 1/4]$ . En déduire que pour tout  $n \in \mathbb{N}$ ,  $u_n \in [0; 1]$ .
- Montrer que la suite  $u$  est monotone.
- En déduire la convergence. Préciser la limite.

#### 2. Représentation des termes de la suite.

- Écrire une fonction `suite` qui prend en argument un entier  $n$ ,  $u_0$  et  $\lambda \in \mathbb{R}$  et renvoie la liste des  $n$  premiers termes de la suite  $u$  définie avec le paramètre  $\lambda$  et la condition initiale  $u_0$ .

b) Afficher les premières valeurs sur un graphique à l'aide de la commande `plt.plot(... , ... , 'ro')`.

### 3. Simulation.

a) On choisit  $\lambda = 2$ . Que dire de la convergence de la suite  $u$  pour les conditions initiales :

$$u_0 = 1/4, \quad u_0 = 0.1, \quad u_0 = 0.0001... ?$$

b) Tester pour  $\lambda = 3$  et  $u_0 = 0.1$ . Que dire si on modifie la condition initiale?

c) Même question avec  $\lambda = 4$  et  $u_0 = 0.1$ , puis  $u_0 = 0.1001$  et  $u_0 = 0.100001$ . Conclusion?

### Exercice 17. ♦♦ Suites adjacentes

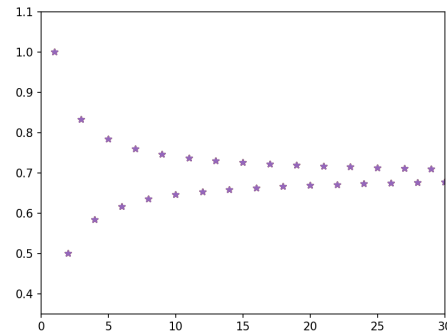
Editeur

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

n=30
x=np.arange(1,n+1); y=np.zeros(n)
eps=1
for i in range(n):
    y[i]=eps/(i+1)
    eps=eps*(-1)
z=np.cumsum(y)

plt.axis([0, 30, 0.35, 1.1])
plt.plot(x,z, '*');
```

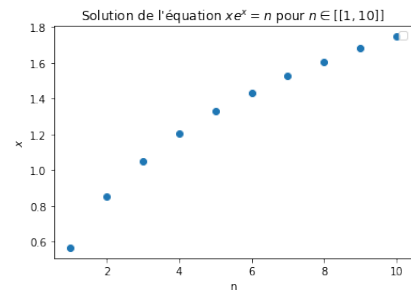
On considère le programme ci-contre et, ci-dessous, le résultat obtenu.



1. Préciser le contenu des variables  $y$  et  $z$  après l'exécution du programme.
2. Ce graphe suggère que deux suites sont adjacentes. Lesquelles?
3. Démontrer cette conjecture.

### Exercice 18. ♦♦ Dichotomie et suite définie de manière implicite

1. Soit  $n \in \mathbb{N}^*$ . Justifier qu'il existe une unique solution à l'équation  $xe^x = n$  d'inconnue  $x \in \mathbb{R}$ . Notons  $u_n$  cette solution.
2. a) Écrire une fonction qui prend en argument  $n$  renvoie une approximation de  $u_n$  à  $10^{-3}$  près.  
b) Modifier la fonction précédente pour retourner une approximation des  $n$  premières valeurs de la suite  $u$ .
3. Voici le résultat pour les 10 premières valeurs. Conjecturer la monotonie et la limite de la suite  $u$ . Prouver votre conjecture.



### Graphe de fonctions

Exercice 19. ✧ Quelles courbes trace le code ci-contre?

Editeur

```
x=np.linspace(-1,2,100)
y=np.exp(x)
plt.plot(x,y), plt.plot(x,x), plt.plot(y,x)
plt.show()
```

### Exercice 20. ♦♦ Polynômes de Lagrange

Soient  $n \in \mathbb{N}^*$  et  $(x_0, x_1, \dots, x_n)$ ,  $n+1$  nombres réels distincts deux à deux. Pour  $k \in \llbracket 0; n \rrbracket$ , on définit le polynôme  $L_k$  par

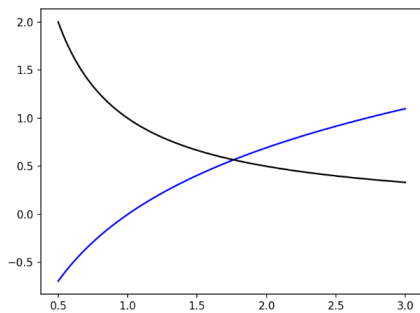
$$\forall x \in \mathbb{R}, \quad L_k(x) = \prod_{\substack{0 \leq j \leq n \\ j \neq k}} \frac{x - x_j}{x_k - x_j}.$$

1. Écrire une fonction **Lagrange(X,k,x)** qui prend en entrée une matrice ligne  $\mathbf{X} = [x_0, x_1, \dots, x_n]$ , un entier  $k \in \llbracket 0; n \rrbracket$  et un réel  $x$ , puis renvoie le nombre  $L_k(x)$ .
2. Tracer sur l'intervalle  $[-3; 3]$  chacune des courbes des polynômes  $L_k$  correspondant à  $\mathbf{X} = [-2, -1, 0, 1, 2]$

### Exercice 21. ♦♦

Expliquer l'intérêt du programme suivant.

Test : `>>> mystere(np.log,0.0001)`



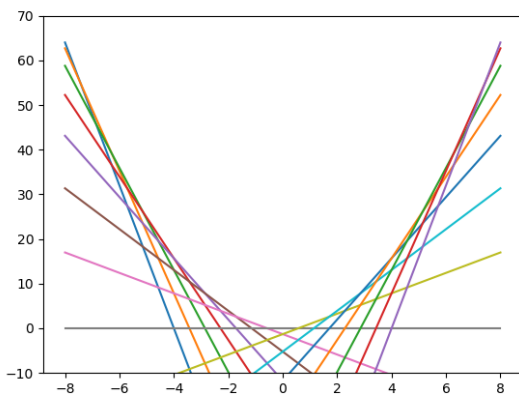
Editeur

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

def mystere(f,e):
    # f une fonction, e est un réel "petit"
    n=200
    x=np.linspace(0.5,3,n)
    y=np.zeros([n,1])
    z=np.zeros([n,1])
    for i in range(n):
        y[i]=f(x[i])
        z[i]=(f(x[i]+e)-f(x[i]))/e
    plt.plot(x,y,'b')
    plt.plot(x,z,'k')
    plt.show()
```

### Exercice 22. ♦♦ Enveloppe convexe

Compléter le code ci-dessous afin qu'il trace les tangentes à la courbe représentative de  $f : x \in \mathbb{R} \mapsto x^2$  aux points d'abscisse  $-8, -7, \dots, 0, 1, 2, \dots, 8$ .



Editeur

```
n=15 # Nombre de tangentes
a=np.linspace(-8,8,n)

for i in range(n) :
    x= ...
    y= ...

    ...

    plt.ylim(-10,70)
    # Limite l'axe des ordonnées à
    # [-10,70]
plt.show() #Affichage
```

**Remarque.** La commande `plt.clf()` pour « Clear Figure » permet de supprimer un graphe précédemment tracé.

## 2 Exemples en algèbre linéaire

### 2.1 Les bibliothèques

#### numpy

La bibliothèque numpy doit d'abord être importée via `import numpy as np`.

Elle permet d'accéder à la plupart des fonctions et constantes mathématiques comme pour la bibliothèque math, on peut par exemple utiliser  $\pi$  avec la commande `np.pi` ou la fonction exponentielle avec `np.exp(.)`.

Elle permet surtout de créer des tableaux de type ndarray, parfait pour représenter des matrices. Pour créer une matrice, il suffit de donner ses différentes lignes séparées par des virgules :

Console

```
>>> B = np.array([[1,2,3],[4,5,6],[7,8,9]])
>>> B
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Pour accéder au coefficient en position  $(i, j)$ , il suffit de taper `B[i, j]` en prenant garde qu'en Python, les indices commencent à 0.

- **Matrices prédéfinies**

Il existe des matrices usuelles prédéfinies dans la bibliothèque numpy.

- `np.ones(n)` crée une matrice ligne de longueur  $n$  remplie 1.
- `np.ones((m,n))` crée une matrice de taille  $(m,n)$  remplie de 1.
- `np.zeros(n)` crée une matrice ligne de longueur  $n$  remplie 0.
- `np.zeros((m,n))` crée une matrice de taille  $(m,n)$  remplie de 0.
- `np.eye(n)` crée la matrice identité de taille  $(n,n)$ .

- **Opérations sur les matrices**

La plupart des opérations sur les tableaux numpy se font terme à terme, par exemple :

<b>Editeur</b>	<pre>A=np.array([[1,2,3],[4,5,6],[7,8,9]]) print(A==2) # On teste si chaque coefficient de la # matrice vaut 2</pre>	<b>Console</b>	<pre>[[False True False]  [False False False]  [False False False]] # Un seul 2, en position (1,2)</pre>
----------------	--	----------------	--



**Attention.** Comme la somme matricielle est une opération terme à terme. Pour effectuer la somme entre deux matrices A, B de même taille, il suffit d'écrire  $A+B$ . Par contre, la commande  $A*B$  renvoie le produit coefficient par coefficient et non le produit matriciel.

- `np.dot(A,B)` effectue le produit matriciel usuel  $AB$ .
- `shape()` renvoie la taille d'un tableau numpy.
- `transpose(A)` renvoie la transposée de la matrice A.

**Exercice 23.** ✧ Comment obtenir avec Python la matrice carrée  $\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$  ?

**Exercice 24.** ✧ Prévoir la réponse de la machine.

Console

```
n=10
A=np.arange(1,n+1)
B=np.ones((n,1))
print(np.dot(A,B))
```

## numpy.linalg

Pour avoir plus de possibilités, on pourra importer la bibliothèque `numpy.linalg` avec la commande : `import numpy.linalg as al`

- `al.inv` pour obtenir l'inverse d'une matrice (s'il il existe);
- `al.matrix_rank` pour le rang;
- `np.linalg.det` pour le déterminant;
- `al.matrix_power` pour calculer les puissances;
- `al.solve` pour résoudre une équation matricielle  $AX = B$  d'inconnue X;
- `al.eig` pour obtenir le spectre de la matrice.

## 2.2 Les exercices

**Exercice 25.** ✧ Résoudre, à l'aide de Python, les deux systèmes linéaires suivants. Que constatez vous?

$$\mathcal{S}_1 : \begin{cases} x_1 + 9x_2 + 10x_3 = -50 \\ 9x_1 + 5x_2 + x_3 = 180 \\ 5x_1 + 10x_2 + 9x_3 = 40 \end{cases} \quad \text{et} \quad \mathcal{S}_2 : \begin{cases} x_1 + 9x_2 + 10x_3 = -50 \\ 9x_1 + 5x_2 + x_3 = 180 \\ 5x_1 + 10x_2 + 9x_3 = 41. \end{cases}$$

**Exercice 26.** ✧✧ Soit  $A = \begin{bmatrix} 1 & -6 & 0 \\ 2 & -6 & 2 \\ 2 & -4 & 3 \end{bmatrix}$ . En s'aidant des instructions suivantes, calculer les puissances de A.



```
>>> import numpy as np
>>> A = np.array([[1, -6, 0], [2, -6, 2], [2, -4, 3]])
>>> P = np.array([[6, 2, -1], [2, 1, 0], [-1, 0, 1]])
>>> P_inv = np.linalg.inv(P) # calcule l'inverse de la matrice P
>>> P_inv
array([[ 1., -2.,  1.],
       [-2.,  5., -2.],
       [ 1., -2.,  2.]])
>>> np.dot(P_inv, np.dot(A, P))
array([[ -1.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       [ 6.66133815e-16, -2.00000000e+00,  0.00000000e+00],
       [ 4.44089210e-16,  0.00000000e+00,  1.00000000e+00]])
```

**Exercice 27.** ✦ Écrire un programme qui compte le nombre de matrices non inversibles parmi les matrices de taille (25,25) :

$$M(a,b) = \begin{bmatrix} a & b & \cdots & \cdots & b \\ b & a & \ddots & \cdots & b \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & a & b \\ b & b & \cdots & b & a \end{bmatrix} \quad \text{avec } -50 \leq a, b \leq 50.$$

**Exercice 28.** ✦ Expliquez l'intérêt de la fonction Python suivante.

```
import numpy as np
import numpy.linalg as al

def bidule(A,x) : # A est une matrice carrée et x, un réel
    [n,p]=np.shape(A)
    if n!=p :
        print('Non, la matrice n est pas carrée ...')
    elif np.linalg.matrix_rank(A-x*np.eye(n))<n :
        print('Oui !')
    else :
        print('Non !')
```

## 3 Variables aléatoires discrètes

### 3.1 Histogrammes, diagrammes en bâtons

• Un histogramme donne une idée graphique de la répartition des valeurs d'un **échantillon**  $E = \{x_1, \dots, x_n\}$ . Pour cela, on découpe l'ensemble des valeurs possibles en un certain nombre d'intervalles. Notons  $p \in \mathbb{N}^*$  le nombre d'intervalles choisis. Ces intervalles doivent être deux à deux disjoints, et leur réunion est égale à (ou contient) l'ensemble des valeurs possibles. Plus précisément, en notant  $a_1, a_2, \dots, a_{p+1} \in \mathbb{R}$ , les bornes de tous ces intervalles avec  $a_1 < a_2 < \dots < a_{p+1}$ , on a

$$E \subset \bigcup_{i=1}^p [a_i; a_{i+1}[.$$

L'intervalle  $[a_i; a_{i+1}[$  est une **classe**. On peut ainsi définir l'**effectif** d'une classe comme le nombre d'éléments de E appartenant à la classe puis sa fréquence  $f_i$  comme le rapport de l'effectif sur le nombre total d'éléments de E.

Ensuite, le graphique est obtenu en traçant, pour chaque  $i \in \llbracket 1, p \rrbracket$ , le rectangle de base  $[a_i, a_{i+1}[$  sur l'axe des abscisses, et

- d'aire égale à  $m_i$  (on parle d'histogramme en effectif);
- d'aire égale à  $f_i$  (on parle d'histogramme en fréquence). Dans ce cas, la somme des aires des rectangles est égale à 1.

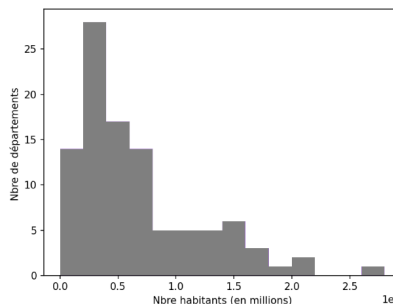
Sous Python, les histogrammes s'obtiennent par la commande `hist` obtenu dans la bibliothèque `matplotlib.pyplot`. Elle prend en argument l'échantillon E, puis on indique les bornes de tous les intervalles  $a_1 < a_2 < \dots < a_{p+1}$  ou simplement le nombre  $p$  d'intervalles souhaité (Python se chargeant de créer alors  $p$  intervalles de même longueur entre la plus petite et la plus grande valeur de l'échantillon).

Exemple. Répartition du nombre d'habitants par département.

Le site de l'I.N.S.E.E (Institut national de la statistique et des études économiques) contient de très nombreuses statistiques en libre accès. On y trouve par exemple la liste du nombre d'habitants par département dont on trace ci-dessous l'histogramme en effectif.

Editeur

```
import matplotlib.pyplot as plt
# E désigne dans la suite la liste
# du nombre d'habitants par département
# Création d'un tableau avec les intervalles de
# longueurs 200 000 (habitants)
inter = np.linspace(0, 2.8*10**6, 15)
plt.hist(E, bins=inter)
plt.xlabel('Nbre habitants (en millions)')
plt.ylabel('Nbre de départements ')
plt.show()
```

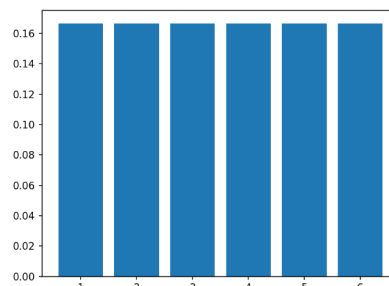


- On peut aussi utiliser la commande `bar` (toujours dans la bibliothèque `matplotlib.pyplot`) pour le tracer de diagramme en bâtons. Voici un exemple qui permet de représenter la loi d'une variable aléatoire uniforme sur  $\mathcal{U}([1;6])$ .

Editeur

```
# La loi de la variable
val=[1,2,3,4,5,6]
loi=[1/6,1/6,1/6,1/6,1/6,1/6]

# Tracé du diagramme en bâtons
# bar(abscisses, ordonnées)
plt.bar(val, loi)
plt.show()
```



**Exercice 29.** ♦ Comment modifier le programme précédent afin qu'il affiche la loi d'une variable aléatoire  $X$ , somme des valeurs de deux dés équilibrés?

$i$	2	3	4	5	6	7	8	9	10	11	12
$P(X=i)$	1/36	1/18	1/12	1/9	5/36	1/6	5/36	1/9	1/12	1/18	1/36

## 3.2 Représentations des lois de probabilité

**Exercice 30.** ♦♦ **Diagramme en bâton d'une loi géométrique**

1. Soient  $p \in ]0, 1[$  et  $X \hookrightarrow \mathcal{G}(p)$ . Écrire un programme qui prend en argument  $p$  et renvoie la plus petite valeur entière  $n_p$  telle que  $P(X \geq n_p) \leq 1\%$ .
2. En déduire un second programme qui prend en argument  $p$  et renvoie le diagramme en bâton sur  $[[0; n_p]]$  associé à la loi géométrique  $\mathcal{G}(p)$ .

**Exercice 31.** ♦♦♦ **Représentation d'une loi de Poisson**

1. Écrire un programme qui prend en argument  $n \in \mathbb{N}^*$ ,  $\lambda \in \mathbb{R}_*^+$  et renvoie la matrice-ligne

$$[p_0 \quad p_1 \quad \dots \quad p_n] \quad \text{où} \quad p_i = \frac{\lambda^i}{i!} e^{-\lambda}.$$

On pourra remarquer que  $p_{i+1} = \lambda p_i / (i+1)$ .

2. Soient  $\lambda \in \mathbb{R}_*^+$  et  $X \hookrightarrow \mathcal{P}(\lambda)$ . Écrire un programme qui prend en argument  $\lambda$  et renvoie la plus petite valeur entière  $n_\lambda$  telle que  $P(X \geq n_\lambda) \leq 1\%$ .
3. En déduire un second programme qui prend en argument  $\lambda$  et renvoie le diagramme en bâtons sur  $[[0; n_\lambda]]$  associé à la loi de Poisson  $\mathcal{P}(\lambda)$ .

**Exercice 32.** ♦♦♦ **Calcul des probabilités d'une loi binomiale**

Soit  $n \in \mathbb{N}$ . Soit  $P$  un polynôme de degré inférieur ou égal à  $n$  défini par

$$P(x) = \sum_{i=0}^n a_i x^i = a_0 + a_1 x + \dots + a_n x^n.$$

En Python, on peut coder le polynôme  $P$  à l'aide d'une matrice-ligne formée de ses coefficients :  $L = [a_0, a_1, \dots, a_n]$ . Dans ce cas, la commande `L[k]` renvoie le coefficient de  $P$  de degré  $k$ .

1. Soient  $p, q \in \mathbb{R}$ . Donner la relation entre les coefficients du polynôme  $P$  et les coefficients du polynôme  $Q_{p,q}$  défini par

$$\forall x \in \mathbb{R}, \quad Q_{p,q}(x) = (q + px)P(x).$$

2. Compléter le programme ci-contre qui prend en argument la matrice  $L$  des coefficients d'un polynôme  $P$ , les réels  $p$  et  $q$  et renvoie la liste des coefficients de  $Q_{p,q}$ .

3. À l'aide de la formule du binôme, développer  $(q + px)^n$ .  
En déduire une fonction qui prend en argument un réel  $p \in [0; 1]$ , un entier naturel  $n$  et renvoie les probabilités associées à la loi  $\mathcal{B}(n; p)$ .

Editeur

```
def Bino(liste,p,q) :
    m = len(liste)
    B = ...
    for i in range(1,m) :
        ...
    return B
```

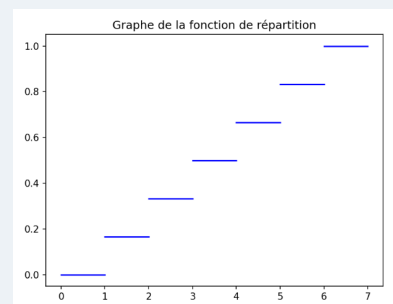
### Complément : tracé d'une fonction de répartition, cas discret

Ci-dessous, un code pour tracer la fonction de répartition d'une variable aléatoire discrète finie à partir de sa loi.

Editeur

```
def FctionRepartition(val,loi) :
    plt.clf()
    n=len(val)
    val=[val[0]-1]+val+[val[-1]+1]
    LoiCumulee=[sum(lois[0:i:1]) for i in range(0, n+1)]
    for i in range(n+1) :
        x=[val[i],val[i+1]]
        y=[LoiCumulee[i],LoiCumulee[i]]
        plt.title("Graphe de la fonction de répartition")
        plt.plot(x,y,'b')
    plt.show()

# On teste avec la loi uniforme sur [1,6]
val=[1,2,3,4,5,6];lois=[1/6,1/6,1/6,1/6,1/6,1/6]
FctionRepartition(val,lois)
```



## 3.3 Simulation des variables aléatoires discrètes

### Premiers exemples avec les lois usuelles

Pour simuler des variables aléatoires, on peut utiliser la bibliothèque `random` que l'on importe par :

Editeur

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

- **rd.random()**

La commande `rd.random()` renvoie un réel choisi au hasard dans l'intervalle  $[0;1[$  suivant une loi de probabilité uniforme continue sur  $[0; 1[$ .

En particulier, pour simuler à l'aide de Python une variable aléatoire qui suit une loi de Bernoulli de paramètre  $p$ , on peut écrire `rd.random() < p` qui renverra `True` avec une probabilité  $p$ . On identifie alors `True` avec 1 et `False` avec 0.

- **rd.randint(debut,fin)**

De la même façon, on peut tirer au hasard (et uniformément) un entier plutôt qu'un réel. Dans ce cas, la commande à utiliser est `rd.randint(début, fin)` qui tire au hasard avec une probabilité uniforme un entier dans l'intervalle  $[\text{début}, \text{fin}[$ .

Noter que l'on peut aussi simuler directement une variable aléatoire  $X \hookrightarrow \mathcal{U}([a; b])$  en tapant :

Console

```
np.floor((b-a)*random()+a)
```

- **rd.geometric(p)**

La fonction `rd.geometric(p)` permet de simuler une variable aléatoire qui suit une loi géométrique de paramètre  $p$ . Pour rappel, si  $X$  renvoie le rang du premier succès dans une infinité d'expériences de Bernoulli mutuellement indépendantes,  $X \hookrightarrow \mathcal{G}(p)$  où  $p$  est la probabilité d'un succès.

- **rd.binomial(n,p)**

De la même façon, on peut simuler une variable aléatoire qui suit une loi binomiale de paramètres  $n, p$  avec la commande `rd.binomial(n,p)`. Pour rappel, le nombre de succès obtenus lors d'une répétition de  $n$  expériences de Bernoulli mutuellement indépendantes de probabilité de succès  $p$  suit une loi binomiale de paramètres  $n, p$ .

- **rd.poisson(lambda)**

Enfin, `rd.poisson(lambda)` simule une variable qui suit une loi de Poisson de paramètre  $\lambda$ .

**Remarque.** Toutes les méthodes précédentes peuvent aussi renvoyer une liste de valeurs tirées selon les différentes lois de probabilité plutôt qu'une seule valeur. Pour faire cela, il suffit de donner comme argument supplémentaire à l'instruction la taille de la liste voulue. Par exemple, `rd.randint(1,100,200)` renvoie un tableau numpy contenant 200 entiers pris aléatoirement entre 1 et 100. De plus, la commande `rd.randint(1,100,[200,10])` renvoie une matrice de taille (200,10).

**Exemple.** Reprenons l'exemple d'un lancer de dé et comparons les probabilités théoriques avec les valeurs d'un échantillon par une simulation informatique. Plus l'échantillon est important, plus l'histogramme a de chance d'être très proche du diagramme en bâtons représentant la loi.

Editeur

```
# La loi de la variable
val=[1,2,3,4,5,6]
loi=[1/6,1/6,1/6,1/6,1/6,1/6]

# Simulation de la variable aléatoire
ech=np.floor(6*np.random.rand(m))+1

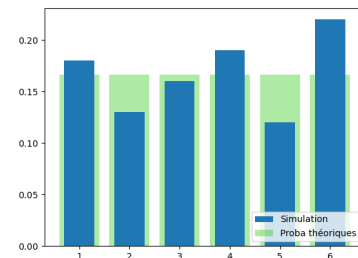
# Tracé du diagramme en bâtons
# bar(abscisses,ordonnées)
plt.bar(val,loi,color=(0.2, 0.8, 0.1, 0.4),label='Proba théoriques')

# Tracé de l'histogramme
# hist(echantillon, bins=classe)
classe=[0.5,1.5,2.5,3.5,4.5,5.5,6.5]
plt.hist(ech,bins=classe,density='true',rwidth=0.6,label='Simulation')

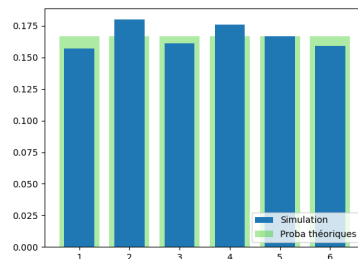
# density='true' pour un histogramme en fréquence

plt.legend(loc='lower right')
plt.show()
```

Échantillon de taille  $m = 100$  :



Échantillon de taille  $m = 1000$  :



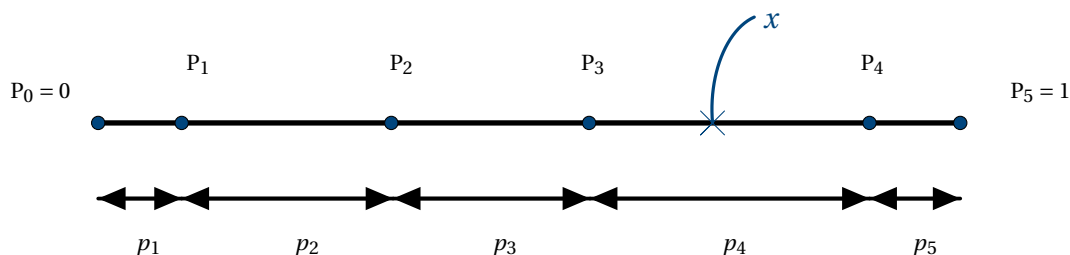
**Exercice 33.** ♦ Comment modifier le programme précédent pour une variable  $X$  donnant la somme de deux dés?

### Simulation pour une loi discrète finie

Soit  $X$  une variable aléatoire finie. Pour simplifier le programme, on suppose  $X(\Omega) = \llbracket 1; n \rrbracket$ . On note

$$\forall i \in \llbracket 1; n \rrbracket, \quad p_i = \mathbf{P}(X = i) \quad \text{et} \quad P_i = \sum_{k=1}^i p_k.$$

On souhaite simuler la variable  $X$  uniquement avec la commande `rand()`. L'idée est la suivante : on simule la loi de  $X$  en tirant un réel  $x$ , au hasard dans  $[0; 1]$  et en renvoyant l'entier  $k$  si  $x$  appartient à l'intervalle  $[P_{k-1}; P_k[$ . La probabilité que l'entier  $k$  soit choisi est alors  $P_k - P_{k-1} = p_k = \mathbf{P}(X = k)$  (avec la convention  $P_0 = 0$ ).



### Exercice 34. ♦♦♦ Simulation des v.a finies

1. Que représente les réels  $P_i$  en termes de probabilités et de  $X$ ?
2. Écrire un programme qui prend en argument la matrice-ligne  $(p_i)_{i \in \llbracket 1; n \rrbracket}$  correspond à la loi de  $X$  et simule  $X$ .
3. Tester votre programme sur la loi uniforme  $\mathcal{U}(\llbracket 1; 10 \rrbracket)$  et vérifier la cohérence du programme à l'aide d'un histogramme.

### Exemple de simulation dans le cas infini dénombrable - la loi de Poisson

On peut toutefois étendre la méthode précédente au cas où  $X$  prend ses valeurs dans  $\mathbb{N}$ . Prenons le cas de  $X \hookrightarrow \mathcal{P}(\lambda)$ . Notons pour  $k \in \mathbb{N}$ ,

$$p_k = \mathbf{P}(X = k) = e^{-\lambda} \frac{\lambda^k}{k!} \quad \text{et} \quad P_k = \sum_{j=0}^k p_j.$$

En particulier, on a

$$P_{k+1} = P_k + \frac{\lambda}{k+1} p_k.$$

### Exercice 35. ♦♦ Simulation d'une loi de Poisson

1. Adapter la méthode précédente pour simuler la variable  $X \hookrightarrow \mathcal{P}(\lambda)$ .
2. Comparer l'histogramme obtenu aux valeurs théoriques (voir exercice 31).

## 3.4 Quelques programmes de référence

### Estimation d'une probabilité

Lorsque le calcul d'une probabilité d'un événement  $A$  est délicat, on peut se contenter d'une approximation. D'après la loi des grands nombres, il suffit de simuler informatiquement « un grand nombre de fois l'expérience » et de regarder la fréquence empirique du nombre de succès.

$$\mathbf{P}(A) \approx F_N = \frac{\text{Nombre de succès}}{\text{Nombre d'essais}}.$$

**Exemple.** Le code permet d'avoir une approximation de la probabilité de faire un double avec deux dés.

Plusieurs essais à comparer à la valeur théorique 1/6.

Editeur

```
# rd.randint(1,7) renvoie  
# un nombre au hasard dans [1,6]  
  
Compteur=0  
for i in range(N):  
    if rd.randint(1,7)==rd.randint(1,7):  
        Compteur+=1  
print(Compteur/N)
```

Console

```
>>> # script executed  
0.161  
>>> # script executed  
0.1695  
>>> # script executed  
0.1765
```

### Exercice 36. ♦ Paradoxe du prince de Toscane

1. Écrire une fonction Python qui prend en argument  $k \in \llbracket 3; 18 \rrbracket$  et renvoie une approximation de la probabilité que la somme de trois dés équilibrés donne  $k$ .
2. Voici deux tests :

Console

```
compte(9)           compte(10)  
>>> 0,1133          >>> 0,1257
```

On constate que la probabilité de faire 10 est légèrement supérieure à celle de 9. Pourtant, comme l'avait remarqué le prince de Toscane au XVII<sup>e</sup> siècle, il y a autant de façons d'écrire 9 et 10 comme sommes de nombres compris entre 1 et 6 :

$$\begin{array}{l} 9 = 1+2+6 \\ = 1+3+5 \\ = 1+4+4 \\ = 2+2+5 \\ = 2+3+4 \\ = 3+3+3 \end{array} \quad \text{et} \quad \begin{array}{l} 10 = 1+3+6 \\ = 1+4+5 \\ = 2+2+6 \\ = 2+3+5 \\ = 2+4+4 \\ = 3+3+4 \end{array}.$$

Pouvez-vous expliquer ce paradoxe?

**Exercice 37.** ♦♦

*d'après oraux HEC 2023*

Soient  $X, Y$  deux variables aléatoires qui suivent une loi uniforme sur  $[[1; n]]$ . Écrire un programme Python qui permet de donner une valeur approchée de

$$\mathbf{P}(E_n) \quad \text{où } E_n \text{ est l'événement "X/Y est un entier."}$$

**Estimation d'une espérance**

Soit  $X$ , une variable aléatoire réelle admettant une espérance.

En utilisant une nouvelle fois la loi faible des grands nombres, on peut approximer  $\mathbf{E}(X)$  en simulant un grand nombre de fois la variable aléatoire  $X$  et en effectuant la moyenne arithmétique de l'échantillon obtenu.

$$\mathbf{E}(X) \approx \frac{x_1 + \dots + x_N}{N} \quad \text{où } N \text{ représente la taille de l'échantillon } (x_1, \dots, x_N).$$

**Exercice 38.** ♦

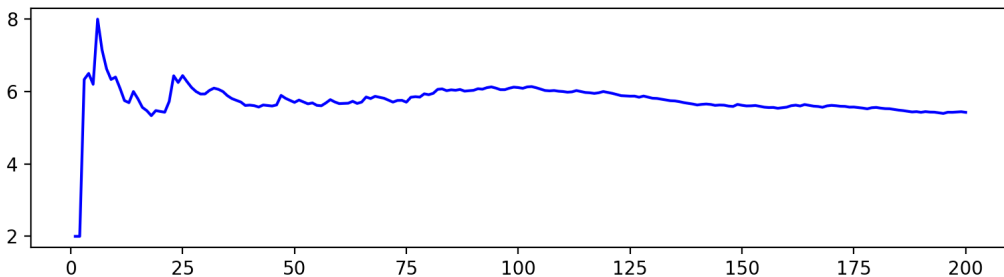
*d'après Ecricome 2021, voie E*

On lance indéfiniment une pièce équilibrée. On s'intéresse au rang du lancer auquel on obtient pour la première fois deux «Pile» consécutifs.

1. Recopier et compléter la fonction Python ci-contre afin qu'elle simule les lancers de la pièce jusqu'à l'obtention de deux «Pile» consécutifs, et qu'elle renvoie le nombre de lancers effectués.
2. Écrire une fonction *moyenne* d'argument  $n$  qui simule  $n$  fois l'expérience ci-dessus et renvoie la moyenne des résultats obtenus.
3. On calcule *moyenne*( $n$ ) pour chaque entier  $n$  de  $[[1, 200]]$ , et on trace les résultats obtenus dans le graphe suivant. Que pouvez-vous conjecturer sur la variable aléatoire  $X$ ?

Editeur

```
def simulX():
    tirs=0
    pile=0
    while pile ... :
        if rd.random() < 1/2 :
            pile ...
        else :
            pile ...
            tirs ...
    return ...
```



**Exercice 39.** ♦♦ **Estimation de l'espérance et de la variance**

Une urne contient trois boules : une rouge, une bleue et une blanche. On tire avec remise dans l'urne jusqu'à l'apparition de la première boule blanche. Soit  $X$ , la variable aléatoire égale au nombre de boules rouges tirées.

Écrire un programme qui permet d'approximer l'espérance et la variance de  $X$ .

### 3.5 Les exercices

**Exercice 40.** ♦ **Moments**

Soit  $X$  une variable aléatoire sur un univers fini. On rappelle que donner la loi de  $X$  signifie donner

$$X(\Omega) = \{x_1, x_2, \dots, x_m\} \quad \text{et} \quad \forall i \in [[1; m]], \quad \mathbf{P}([X = x_i]).$$

Dans ce cas, on définit les listes *val* et *Loi* par :

$$\text{val} = [x_1 \quad x_2 \quad \dots \quad x_m] \quad \text{et} \quad \text{Loi} = [\mathbf{P}([X = x_1]) \quad \mathbf{P}([X = x_2]) \quad \dots \quad \mathbf{P}([X = x_m])]$$

1. Écrire une fonction Python, nommée *moment*, qui prend en argument les deux listes (*val*, *Loi*), un entier  $s$  et renvoie le moment d'ordre  $s$  :  $\mathbf{E}(X^s)$ .
2. En utilisant uniquement la fonction *moment*, écrire une nouvelle fonction qui calcule la variance.

**Exercice 41.** ♦ **Lois usuelles uniquement avec la commande random()**

Une urne contient 5 boules (1 rouge et 4 bleues). On considère l'expérience suivante :

On tire une boule au hasard et on note la couleur. On replace ensuite la boule dans l'urne.

1. Soit  $X$  la variable aléatoire qui renvoie 1 si la boule est rouge et 0 sinon.  
Préciser la loi de  $X$ .  
En utilisant uniquement la commande `random`, écrire un programme qui simule la variable  $X$ .
2. On répète maintenant  $n$  fois l'expérience élémentaire. On suppose les tirages mutuellement indépendants. On note  $Y$  le nombre de boules rouges obtenues.  
Quelle est la loi de  $Y$ ?  
Avec la commande `random`, écrire un programme qui prend en argument  $n$  et simule  $Y$ .
3. On répète maintenant une infinité de fois l'expérience élémentaire. On suppose toujours les tirages mutuellement indépendants. On note  $Z$  le numéro du tirage où on a obtenu la première boule rouge.  
Quelle est la loi de  $Z$ ? Écrire un programme qui simule la variable  $Z$ .
4. Modifier le programme précédent pour simuler la variable  $X_2$  qui donne le numéro du tirage où on obtient la seconde boule rouge.  
Soit  $r \in \mathbb{N}^*$ . Généralisez la question avec  $X_r$ , la variable aléatoire qui renvoie le tirage de la  $r$ -ième boule rouge.

#### Exercice 42. ♦♦ Le problème du collectionneur

Afin de relancer ses ventes, une marque met en ligne une nouvelle série de 20 figurines Schtroumpfs dans ses paquets de céréales. Blaise, père attentionné, souhaite obtenir la collection complète pour son enfant tout en maîtrisant son budget petit-déjeuner. Plutôt que de faire les calculs, Blaise simule le problème pour répondre à la question :

Quelle est la probabilité d'obtenir la collection complète avec 50 paquets ?

Pour simuler l'achat de  $n$  paquets, Blaise propose le pseudo-code :

- Les figurines sont numérotées de 0 à 19.  
On suppose qu'elles ont la même probabilité d'apparition.
- $L$  est une matrice ligne avec 20 coefficients. Chaque coefficient  $L[k]$  peut prendre deux valeurs : 1 si Blaise possède la figurine  $k$ , 0 sinon.  
Au début de la simulation,  $L$  est donc la matrice nulle  $0_{1,20}$ .
- Pour  $i$  variant de 1 à  $n$ , Blaise tire au hasard une figurine. Notons  $j$ , le numéro de la figurine. On affecte 1 au  $j$ -ième coefficient de  $L$ .
- Au bout des  $n$  tirages, on affiche  $L$ .  
Si  $L$  ne contient que des 1, Blaise possède la collection complète.

1. a) Écrire une fonction, qui prend comme argument  $n$  et renvoie la matrice  $L$ .  
*Rappel.* La commande `randint(p, q)` renvoie un entier dans  $[p; q]$  choisi au hasard.  
b) Écrire une fonction qui prend en arguments deux entiers  $n, m$  et simule  $m$  tirages indépendants de  $n$  paquets et renvoie la fréquence empirique de l'événement « La collection est complète ».  
*Indication.* Pour vérifier que la collection est complète, on pourra vérifier que la somme des coefficients de  $L$  vaut 20.
2. a) Donner une réponse à la question de Blaise à l'aide de Python.  
b) Combien de paquets faut-il acheter pour avoir la collection complète avec une probabilité supérieure à 90%?

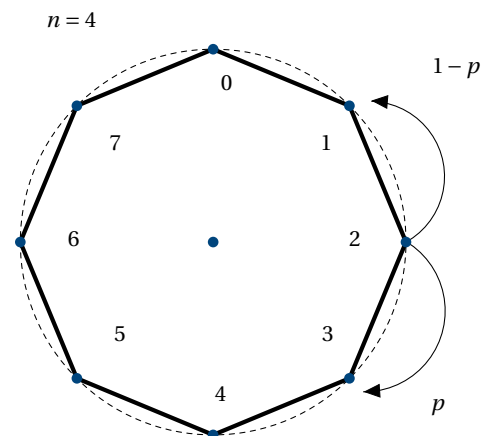
#### Exercice 43. ♦♦ Mobile sur un polygone

On place  $2n$  points numérotés de 0 à  $2n - 1$  sur un cercle. Les points sont répartis uniformément.

Un mobile part initialement de 0 et à chaque étape, il avance d'une unité avec une probabilité  $p$  et recule avec probabilité  $1 - p$ . Soit  $N \in \mathbb{N}^*$ , le nombre d'étapes

On note :

- Pour tout  $k \in \llbracket 0; N \rrbracket$ , on note  $X_k$  la position du mobile à la  $k$ -ième étape.
- $T_p$  la variable aléatoire égale au temps de retour à l'origine. Si le mobile ne retourne pas à l'origine au bout des  $N$  étapes, on pose  $T_p(\omega) = 0$ .




1. Compléter ce programme qui prend en argument  $p$ , simule les 100 premières étapes du mobile et renvoie la position finale du mobile.

```

...
def simulation(p,n) :
    X=0
    for i in range(100) :
        ...
    X=X%2n # renvoie le reste de la division par 2n
    return X

```

2. Modifier le programme pour afficher le temps  $T_p$  de premier retour à l'origine 0.
3. En déduire une approximation de l'espérance de T.
4. Comparer  $E(T_p)$  et  $E(T_{1-p})$ .

**Exercice 44.** ♦♦  Soient X, Y deux variables aléatoires indépendantes suivant des lois géométriques de même paramètre  $p$ . Soit  $f(p)$ , la probabilité que la matrice suivante soit inversible

$$A = \begin{bmatrix} X & Y \\ Y & X \end{bmatrix}.$$

1. Proposer un programme qui prend en argument  $p$  et donne une approximation de  $f(p)$  (notée dans la suite  $\tilde{f}(p)$ ).
2. Tracer le graphe de  $p \mapsto \tilde{f}(p)$  pour  $p \in ]0; 1[$ .
3. Faire le calcul exact de  $f(p)$  et comparer avec les résultats précédents.

**Exercice 45.** ♦♦♦  Deux méthodes pour approximer une somme

1. Justifier la convergence de la série  $\sum \sqrt{1+e^k} \frac{2^k}{k!}$ . Dans la suite, on pose  $S = \sum_{k=0}^{+\infty} \sqrt{1+e^k} \frac{2^k}{k!}$ .
2. En calculant les sommes partielles, donner une approximation de S.
3. En utilisant une variable X suivant une loi de Poisson de paramètre 2, exprimer S sous forme d'une espérance. En déduire une approximation de S.

**Exercice 46.** ♦♦♦ **Résolution et simulation du Monty-Hall**

Dans le film *Las Vegas 21*, Micky Rosa est professeur au M.I.T. Il soumet à ces étudiants, dont Ben, le problème du Monty-Hall.

*Micky Rosa* : Ben, vous participez à un jeu télévisé et on vous donne la possibilité de choisir entre trois portes différentes, d'accord ? Derrière l'une des trois portes, il y a une voiture neuve, et derrière les deux autres, des chèvres. Quelle porte choisiriez-vous, Ben ?



*Ben* : Porte  $n^{\circ}1$  ?

*Micky Rosa* : Ben choisit la porte  $n^{\circ}1$  et là, l'animateur de jeu télé qui, soit dit en passant, sait ce qu'il y a derrière chacune des trois portes, décide d'ouvrir une autre des trois portes ; disons qu'il choisit la  $n^{\circ}3$ , derrière laquelle se trouve une chèvre. Bien. Ben... L'animateur s'approche et dit "Ben, vous voulez garder la porte  $n^{\circ}1$  ou prendre la porte  $n^{\circ}2$  ?". Alors, est-il dans votre intérêt de changer de choix ?

*Ben* : Oui.

*Micky Rosa* : Attendez. Souvenez-vous : l'animateur sait où est la voiture. Alors, comment être sûr qu'il ne joue pas avec vous, qu'il n'essaie pas de vous déstabiliser pour vous faire choisir la chèvre ?

Est-ce que Ben a raison de changer de porte ?

Pour cela, on simule un grand nombre de parties de Monty Hall (avec un choix aléatoire de porte) en distinguant les deux cas : on garde la porte, puis on change de porte.



Lien vers l'extrait du film  
*Game Theory Scene | 21(2008) | Now Playing*



Lien vers un documentaire  
*Voyages au pays des maths | ARTE*



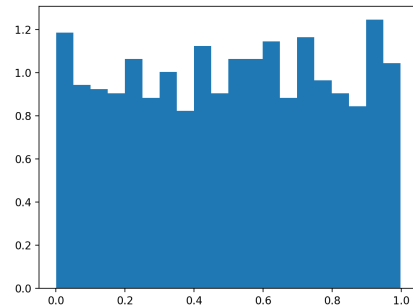
## 4.1 Lien entre histogramme et densité

Commençons par l'exemple d'une simulation d'une loi uniforme.  
Le script Python ci-dessous tire  $m$  réels uniformément dans  $[0; 1]$ , puis affiche l'histogramme.

Editeur

```
m=1000
L= np.random.rand(m)
classes =np.linspace(0,1,10)

plt.hist(L, bins=20, density=True)
# Création de l'histogramme
plt.show()
```



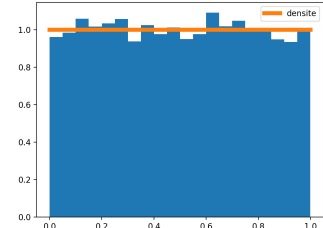
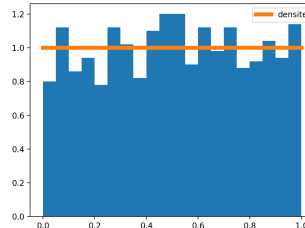
Expliquons les commandes :

- `density=True` : Trace l'histogramme en fréquence.
- `bins=...` : Si on met une valeur entière en argument, on définit le nombre de classes. Sinon, c'est une liste (ou un tableau) qui définit les différents intervalles des classes.
- `rwidth=...` : Largeur relative de la barre par rapport à l'intervalle (facultatif).

Dans cet exemple, il est utile de rajouter la courbe représentative d'une densité et d'augmenter la taille de l'échantillon. On constate alors que **plus la taille de l'échantillon augmente, plus on a de chance que l'histogramme « se rapproche » de la courbe représentative de la densité.**

Editeur

```
x=[0,1]; y=[1,1]
plt.plot(x,y,linewidth=5,label="densite")
plt.legend()
plt.show()
# test avec m=1000, puis m=10000
```



## 4.2 Simulation de variables aléatoires à densité

## Premiers exemples

Comme pour les variables discrètes, la bibliothèque `numpy.random` importée via `rd` permet de simuler les lois à densité usuelles.

- La commande `rd.random()` simule une loi uniforme continue sur  $[0; 1[$ .
- `rd.exponential(a)` renvoie la simulation d'une variable aléatoire de loi exponentielle de paramètre  $1/a$ .
- `rd.normal(mu, sigma)` renvoie la simulation d'une variable aléatoire de loi normale d'espérance  $\mu$  et d'écart-type  $\sigma$ .
- `rd.gamma(a)` renvoie la simulation d'une variable aléatoire de loi gamma de paramètre  $a$ .

**Exercice 47.** ✦ En utilisant uniquement la commande `rd.random()`, expliquer comment simuler une variable aléatoire suivant une loi uniforme continue sur  $[a, b]$ .

## La méthode d'inversion

L'idée repose sur l'énoncé suivant :

- |    |   |
|----|---|
| Si | <ul style="list-style-type: none"> <li>→ <math>U</math> suit la loi uniforme sur <math>]0; 1[</math>.</li> <li>→ <math>X</math> est une variable aléatoire à densité dont la fonction de répartition <math>F</math> est une bijection de <math>]a; b[</math> avec <math>-\infty &lt; a &lt; b &lt; +\infty</math> sur <math>]0; 1[</math>.</li> </ul> |
|----|---|

Alors La variable  $Y = F^{-1}(U)$  suit la même loi que  $X$ .

**Exercice 48.** ✦ 📄 Prouver l'énoncé précédent.

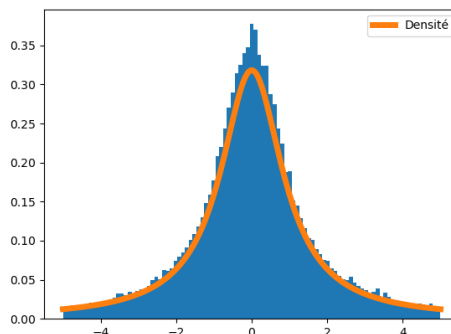
On dit qu'une variable aléatoire  $X$  à densité suit une loi de Cauchy si une densité est donnée par  $f: t \in \mathbb{R} \mapsto 1/(\pi(1+t^2))$ . On a vu que la fonction de répartition est définie sur  $\mathbb{R}$  par  $F(x) = \arctan(x)/\pi + 1/2$ . La fonction  $F$  est strictement croissante, continue de  $\mathbb{R}$  dans  $]0; 1[$ . Le théorème de la bijection s'applique

$$\forall t \in ]0; 1[, \quad F^{-1}(t) = \tan(\pi(t - 1/2)).$$

D'après l'exercice précédent, si  $U$  suit une loi uniforme continue sur  $]0; 1[$ , alors la variable  $F^{-1}(U)$  suit une loi de Cauchy. Ci-dessous, un histogramme d'un échantillon obtenu par simulation par la méthode d'inversion. On vérifie bien que l'histogramme obtenu est très proche d'une densité de la loi de Cauchy.

Editeur

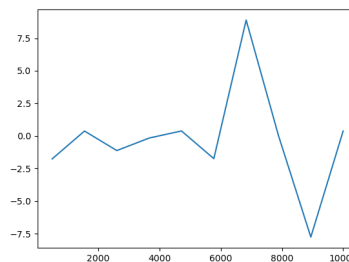
```
U= np.random.rand(50000)
L=np.tan(np.pi*(U-1/2))
inter=np.linspace(-5,5,100)
plt.hist(L, bins=inter, density=True)
# Création de l'histogramme
x=np.linspace(-5,5,200)
y=1/(np.pi*(1+x**2))
plt.plot(x,y,linewidth=5,label="Densité")
plt.legend()
plt.show()
```



**Exercice 49.** ✦✦ 📄 Expliquer l'intérêt du code suivant? Qu'illustre-t on sur la loi de Cauchy?

Editeur

```
Liste=100*np.linspace(5,100,10)
E=[]
for m in Liste :
    U= np.random.rand(int(m))
    L=np.tan(np.pi*(U-1/2))
    e=sum(L)/m
    E.append(e)
plt.plot(Liste,E)
plt.show()
```



**Exercice 50.** ✦ 📄 Adapter la méthode précédente pour simuler une loi exponentielle de paramètre  $\lambda$ .

### Méthode par rejet

Soit  $X$ , une variable à densité et  $f$ , une densité. On suppose que  $f$  est bornée et à support borné. C'est-à-dire, qu'il existe  $K \in \mathbb{R}^+$ ,  $a, b \in \mathbb{R}$  avec  $a < b$  tels que

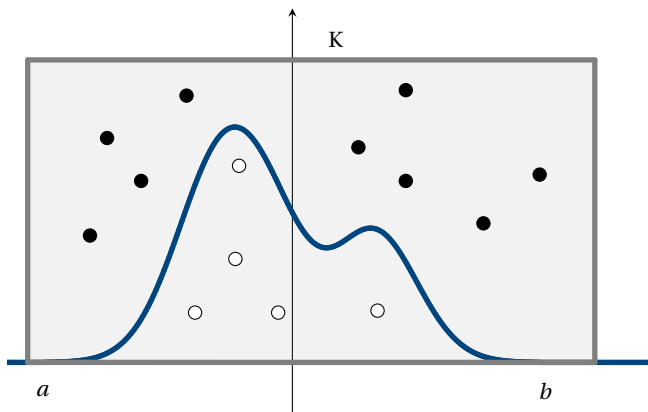
$$\forall x \in \mathbb{R}, \quad 0 \leq f(x) \leq K \quad \text{et} \quad f(x) = 0 \quad \text{si} \quad x \notin [a; b].$$

Rappelons que l'aire comprise entre la courbe, l'axe des abscisses et les droites d'équation  $x = a$ ,  $x = b$  correspond exactement à la probabilité que la variable aléatoire prenne les valeurs comprises entre  $a$  et  $b$ .

$$\text{Aire} = \int_a^b f(t) dt = \mathbf{P}([a \leq X \leq b]).$$

Ainsi, pour simuler la variable  $X$ , on peut tirer un point au hasard sous la courbe représentative de  $f$  de manière uniforme. On prend alors pour réalisation l'abscisse de ce point.

Précisons que pour tirer un point sous la courbe, on tire un point au hasard dans le rectangle  $[a; b] \times [0; K]$  à l'aide de deux lois uniformes. Si ce dernier est sous la courbe, on le garde. Sinon, on recommence.



**Exercice 51.** ✦ Quelle est la loi du nombre d'essais avant de tirer un point qui est bien situé sous la courbe de  $f$ ?

### Exercice 52. ♦♦ Exemple de simulation par la méthode de rejet

Soit  $X$  une variable aléatoire dont une densité  $f$  est définie sur  $\mathbb{R}$  par

$$f(x) = \begin{cases} 6x(1-x) & \text{si } x \in [0;1] \\ 0 & \text{sinon.} \end{cases}$$

- Tracer le graphe de  $f$ . Quelles valeurs choisir pour  $a$ ,  $b$  et  $K$ ?
- Comment tirer un point au hasard dans le rectangle  $[a; b] \times [0; K]$  à l'aide de la commande `rd.random()`?
  - En déduire un programme Python qui simule la variable  $X$ .

## 4.3 Compléments sur les variables à densité

### Exercice 53. ♦♦ Simulation d'une loi géométrique à partir d'une loi exponentielle

Soit  $\lambda \in \mathbb{R}_+^*$ . Soit  $X$  une variable aléatoire sur un espace probabilisé  $(\Omega, \mathcal{A}, \mathbf{P})$  de loi  $\mathcal{E}(\lambda)$ . On pose  $Y = \lfloor X \rfloor + 1$ .

- Montrer  $Y \hookrightarrow \mathcal{G}(1 - e^{-\lambda})$ .
- En déduire un programme qui simule une loi géométrique de paramètre  $p$  à partir de la loi exponentielle.

### Exercice 54. ♦ Simulation d'un loi de Pareto

Soient  $\theta \in \mathbb{R}_+^*$  et la fonction  $f$  définie sur  $\mathbb{R}$  par :

$$f(x) = \begin{cases} \frac{1}{\theta x^{1+1/\theta}} & \text{si } x \geq 1 \\ 0 & \text{si } x < 1. \end{cases}$$

- Montrer que  $f$  peut être considérée comme une densité de probabilité.  
On considère dans la suite une variable aléatoire  $X$  strictement positive de densité  $f$  et on note  $F$  sa fonction de répartition.
- On pose  $Y = \ln(X)$  et on admet que  $Y$  est une variable aléatoire définie sur le même espace probabilisé que  $X$ . On note  $G$  sa fonction de répartition.  
Justifier que  $Y$  suit une loi exponentielle dont on précisera le paramètre.
- On rappelle qu'en Python, la commande `rd.exponential(a)` simule une variable aléatoire suivant la loi exponentielle de paramètre  $1/a$ . Écrire une fonction Python `SimuP` prenant en argument  $\theta$  et permettant de simuler la variable  $X$ .
- Commenter le résultats des lignes suivantes.

Editeur

```
def mystere(theta):
    A=np.zeros(5)
    for p in range(2,7):
        S=0
        for k in range(1,10**p):
            S+=simuP(theta)
        A[p-2]=round(S/10**p,3)
        # round(..,3) arrondit
        # le resultat à 10**(-3)
    return A
```

Console

```
>>> mystere(1/2)
array([2.197, 2.103, 2.011, 1.992, 2.   ])

>>> mystere(1)
array([ 4.892,  8.204, 15.007, 10.577,
        13.309])

>>> mystere(1.2)
array([ 7.318, 12.865, 29.35 , 502.559,
        187.655])
```

### Exercice 55. ♦ Simulation d'une loi de Laplace

Une variable aléatoire à densité  $X$  suit la loi de Laplace si une densité  $f$  est définie sur  $\mathbb{R}$  par  $f(x) = \frac{1}{2}e^{-|x|}$ .

- Soit  $U \hookrightarrow \mathcal{U}([0;1])$ . Donner la loi de  $X = -\ln(U)$ .
- Méthode 1.*
  - Justifier que si  $X$  et  $Y$  sont deux variables aléatoires définies sur un même espace probabilisé qui suivent une loi exponentielle de paramètre 1 alors la différence  $X - Y$  suit une loi de Laplace.
  - En déduire un programme Python qui simule une loi de Laplace en utilisant uniquement la commande `rd.random()`.
  - Tester votre programme en comparant les histogrammes obtenus avec la densité.
- Méthode 2.*  
On considère deux variables aléatoires  $X$  et  $Y$ , définies sur un espace probabilisé  $(\Omega, \mathcal{A}, \mathbf{P})$ , et indépendantes. On suppose que  $X$  suit une loi exponentielle de paramètre 1. On suppose par ailleurs que la loi de  $Y$  est donnée par

$$\mathbf{P}(Y = 1) = \mathbf{P}(Y = -1) = \frac{1}{2}.$$

L'indépendance de  $X$  et  $Y$  se traduit par les égalités suivantes, valables pour tout réel  $x$  :

$$\mathbf{P}(X \leq x \cap Y = 1) = \mathbf{P}(X \leq x)\mathbf{P}(Y = 1) \quad \text{et} \quad \mathbf{P}(X \leq x \cap Y = -1) = \mathbf{P}(X \leq x)\mathbf{P}(Y = -1).$$

On pose  $Z = XY$  et on admet que  $Z$  est, elle-aussi, une variable aléatoire définie sur  $(\Omega, \mathcal{A}, \mathbf{P})$ .

- a) Vérifier que  $Z$  suit une loi de Laplace.
- b) Comment simuler la variable  $Y$ ?
- c) En déduire un nouveau programme qui simule une loi de Laplace.

4. Compléments sur la loi de Laplace.

d'après Essec 2017 E

Soient  $\alpha \in \mathbb{R}$  et  $\beta \in \mathbb{R}^+$ . On dit qu'une variable aléatoire réelle à densité suit une loi de Laplace de paramètre  $(\alpha, \beta)$ , notée  $\mathcal{L}(\alpha, \beta)$ , si elle admet comme densité la fonction  $f$  donnée par :

$$\forall t \in \mathbb{R}, \quad f(t) = \frac{1}{2\beta} \exp\left(-\frac{|t-\alpha|}{\beta}\right)$$

- a) Montrer que si  $U$  suit la loi  $\mathcal{L}(0; 1)$ , alors  $V = \beta U + \alpha$  suit la loi  $\mathcal{L}(\alpha; \beta)$ .
- b) Conclure en donnant un programme qui simule une variable de loi  $\mathcal{L}(\alpha; \beta)$ .

**Exercice 56. ♦♦♦ Simulation d'une loi bêta**

Soient  $\alpha, \beta \in \mathbb{R}_*^+$ . On pose

$$B(\alpha, \beta) = \int_0^1 t^{\alpha-1} (1-t)^{\beta-1} dt.$$

On considère la fonction  $f_{\alpha, \beta}$  définie sur  $\mathbb{R}$  par  $f_{\alpha, \beta}(x) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1} \mathbf{1}_{]0,1[}(x)$ .

1. a) Vérifier que  $f_{\alpha, \beta}$  est une densité de probabilité. On parle alors d'une loi bêta de paramètre  $(\alpha, \beta)$ .  
 b) Écrire un programme qui prend en argument le paramètre  $(\alpha, \beta)$  et renvoie la courbe de  $f_{\alpha, \beta}$  sur  $[0; 1]$ .  
*On pourra utiliser la commande `sp.beta(alpha, beta)` après avoir importé la bibliothèque `scipy.special` via `sp`.*

2. Statistiques d'ordre et loi uniforme

Soient  $n \in \mathbb{N}^*$  et  $X_1, X_2, \dots, X_n$ ,  $n$  variables aléatoires indépendantes et de loi uniforme sur  $[0; 1]$ .

On admet l'existence de variables aléatoires à densité  $Y_1, Y_2, \dots, Y_n$  telles que, pour tout  $\omega$  de  $\Omega$ , les réels  $Y_1(\omega), Y_2(\omega), \dots, Y_n(\omega)$  constituent un réarrangement par ordre croissant des réels  $X_1(\omega), X_2(\omega), \dots, X_n(\omega)$ , de telle sorte que, pour tout  $\omega$  de  $\Omega$  :

$$Y_1(\omega) \leq Y_2(\omega) \leq \dots \leq Y_n(\omega).$$

En particulier,  $Y_1 = \min(X_1, X_2, \dots, X_n)$  et  $Y_n = \max(X_1, X_2, \dots, X_n)$ .

- a) Écrire un programme qui prend en arguments  $n \in \mathbb{N}^*$  et  $k \in \llbracket 1; n \rrbracket$  et renvoie une simulation de  $Y_k$ .  
*Indication. On pourra utiliser la commande `np.sort(A)` qui prend en argument une matrice ligne et renvoie une matrice mais avec les coefficients de  $A$  ordonnés par ordre croissant.*
- b) En déduire un programme qui prend en arguments  $n \in \mathbb{N}^*$  et  $k \in \llbracket 1; n \rrbracket$  et affiche l'histogramme associé à 5000 réalisations de la variable  $Y_k$  ainsi que la courbe représentative de la fonction  $f_{k, n+1-k}$ . Tester et commenter.
3. Comment simuler une loi bêta de paramètres  $(\alpha; \beta)$  par la méthode de rejet?  
*Indication. On pourra s'inspirer de l'exercice 52.*
4. a) Écrire un programme Python qui prend en argument  $\alpha, \beta$  et renvoie  $\bar{x}$  (resp.  $v$ ), la moyenne arithmétique (resp. la variance empirique) de 5000 réalisations d'une loi bêta de paramètre  $(\alpha; \beta)$ .  
 b) *Facultatif.* Donner la valeur exacte de l'espérance et la variance d'une loi bêta de paramètre  $(\alpha, \beta)$ .  
*On pourra admettre que pour tous  $a, b \in \mathbb{R}_*^+$*

$$B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} \quad \text{et} \quad \Gamma(a+1) = a\Gamma(a).$$

5. Modifier le programme précédent en rajoutant le calcul des quantités suivantes. Tester et commenter.

$$\bar{x} \left( \frac{\bar{x}(1-\bar{x})}{v} - 1 \right) \quad \text{et} \quad (1-\bar{x}) \left( \frac{\bar{x}(1-\bar{x})}{v} - 1 \right).$$

**Exercice 57. ♦♦** Pour tout entier naturel  $n \geq 2$ , on pose  $J_n = \int_0^1 \frac{\ln(t)}{1+t^n} dt$ . L'objectif est d'obtenir une valeur approchée de l'intégrale  $J_n$  à l'aide de Python.

1. Soit  $X$  une variable aléatoire suivant la loi exponentielle de paramètre 1. On pose :

$$Y_n = \frac{-X}{1 + e^{-nX}}.$$

Vérifier que  $Y_n$  admet une espérance, et exprimer à l'aide du changement de variable  $u = -\ln(t)$ ,  $\mathbf{E}(Y_n)$  à l'aide de  $J_n$ .

2. On rappelle que dans la bibliothèque Python `numpy.random` importée par `rd` se trouve l'instruction `rd.exponential(a)` qui renvoie une réalisation d'une variable aléatoire suivant la loi exponentielle de paramètre  $1/a$ .  
 Écrire une fonction qui prend en argument  $n$  et permet d'approximer  $J_n$ .

Pour définir et tracer les surfaces représentatives, lignes de niveau, gradient, etc.. On commence par importer :

Editeur

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D # pour pouvoir utiliser la fonction Axes3D
ax=Axes3D(plt.figure())
```

## 5.1 Graphe d'une fonction de deux variables

Dans un premier temps, on définit la fonction. Par exemple, pour la fonction définie sur  $\mathbb{R}^2$  par  $f(x, y) = \sin(x) \cos(y) / (1 + x^2 + y^2)$ , on peut écrire :

Editeur

```
def f(x,y): return np.sin(x)*np.cos(y)/(1+x**2+y**2)
```

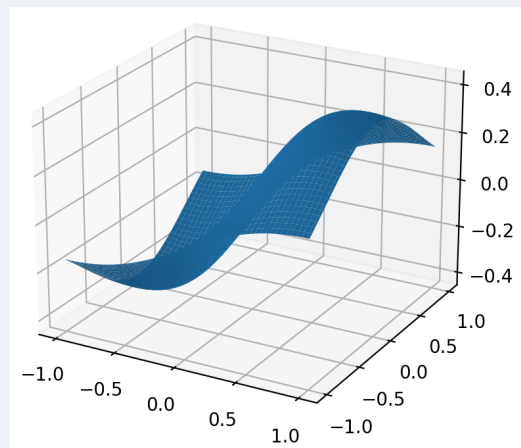
Suivant le même principe que pour le tracé de la courbe représentative d'une fonction d'une variable réelle, le code Python suivant permet de tracer la surface représentative d'une fonction de deux variables.

Editeur

```
x = np.linspace(-1, 1, 100)
# 100 valeurs pour la variable x espacées
# régulièrement entre -1 et 1
y = np.linspace(-1, 1, 100)
# De même pour la variable y

X, Y = np.meshgrid(x, y)
# Tableau contenant les points (xi,yi) où xi
# et yi sont calculés précédemment
Z = f(X,Y)
# Calcul des images pour tous les points (xi,yi)

ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z)
plt.show()
```



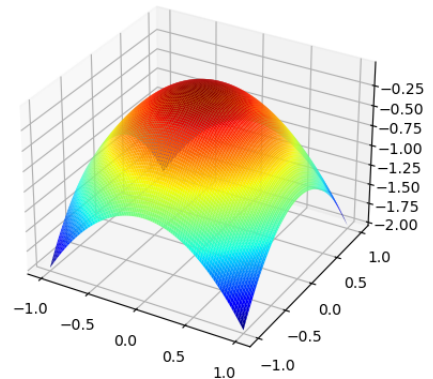
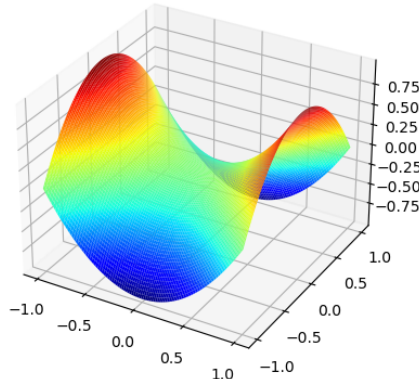
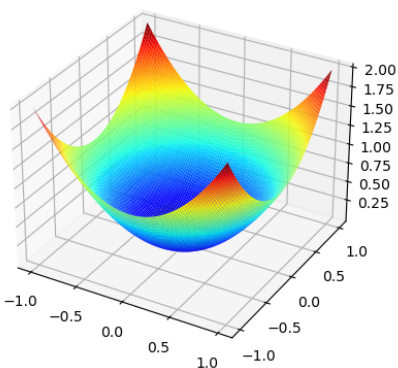
Il existe un grand nombre d'options pour modifier l'affichage. Par exemple, le code suivant permet de faire varier la couleur en fonction de la valeur de la fonction au point considéré.

Editeur

```
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='jet')

# plusieurs couleurs au choix :
# Remplacer 'jet' par 'cool', 'winter', 'spring', 'summer', 'autumn', 'hot', 'prism', ...
```

**Exemples.** Illustrons ce code à l'aide de trois exemples typiques de fonctions polynomiales de degré 2.



```
def f1(x, y):
    return x**2 + y**2
```

```
def f2(x, y):
    return x**2 - y**2
```

```
def f3(x, y):
    return -x**2 - y**2
```

**Exercice 58.** ✧ Tracer la surface représentative de  $f : (x, y) \in \mathbb{R}^2 \mapsto \frac{\cos(x^2 + y^2)}{\sqrt{1 + x^2 + y^2}}$  sur  $[-\pi; \pi]^2$ .

Que peut-on conjecturer sur les extrema locaux? globaux? et le nombre de points où les extrema sont atteints?

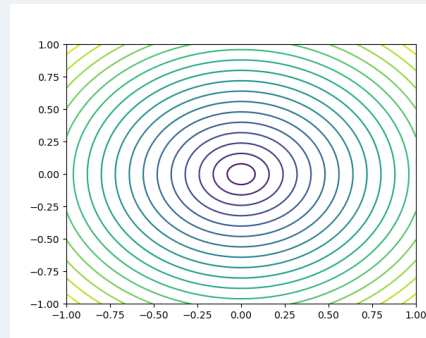
## 5.2 Lignes de niveau

Ci-dessous, un code pour tracer les lignes de niveau d'une fonction de deux variables.

```
def f(x,y):
    return np.sqrt(x**2+y**2)

x=np.linspace(-1,1,200); y=np.linspace(-1,1,200)
X , Y = np.meshgrid(x,y)

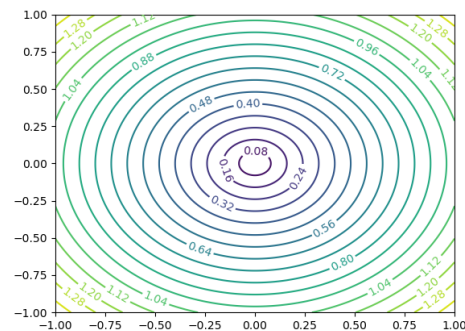
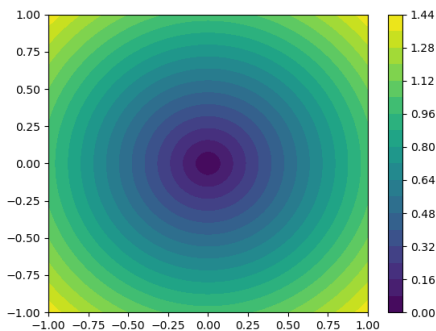
graphe = plt.contour(X,Y,f(X,Y),20)
# 20 pour obtenir 20 lignes de niveau
# ou plt.contour(X,Y,f(X,Y),L)
# avec L, une liste [... , ... , ...] avec
# les valeurs K des lignes de niveaux souhaitées
plt.show()
```



D'autres présentations des lignes de niveaux sont possibles. Par exemple :

```
graphe = plt.contourf(x,y,z,20)
plt.colorbar()
plt.show()
```

```
graphe = plt.contour(x,y,z,20)
plt.clabel(graphe, inline=1, fontsize=10)
plt.show()
```



**Exercice 59.** ✧ Adapter le premier programme pour afficher les lignes de niveau  $K \in [-4; 4]$  de la fonction  $g$  définie sur  $\mathbb{R}^2$  par  $g(x, y) = 4 \sin(x) + \cos(3y)$  sur  $[-\pi; \pi]^2$ . Que dire de la ligne de niveau  $K = 5$ ?

**Exercice 60.** ✦ On étudie la fonction  $f : (x, y) \in \mathbb{R}^2 \mapsto xy$ .

1. Sans utiliser Python, tracer les lignes de niveau pour  $K \in [-2; 2]$ .
2. Vérifier vos calculs en utilisant les codes précédents.

**Exercice 61.** ✦✦ Pour tous réels  $x, y$  tels que  $x^2 + y^2 \geq 1$  et  $x \neq 0$ , on définit le polynôme  $P_{x,y}$  sur  $\mathbb{R}$  par

$$P(t) = x^2 \cdot t^2 + (x^2 + y^2 - 1)^{3/2} \cdot t + y^3.$$

À l'aide de Python, représenter dans  $\mathbb{R}^2$ , l'ensemble des points  $(x, y) \in \mathbb{R}^2$  pour lesquels le polynôme  $P_{x,y}$  a une unique racine.  
Indication. On pourra restreindre l'étude à  $[-2; 2] \times [-1; 2]$ .

• **Les symétries**

Dans le cas des fonctions d'une seule variable, nous avons vu que la parité/imparité, la périodicité permettent de simplifier l'étude ou encore de tester la cohérence d'un résultat. Ces idées s'étendent aux fonctions de plusieurs variables.

Voici quelques conditions de symétries.

- |     |   |   |  |
|-----|---|---|--|
| I   | $\forall (x, y) \in \mathbb{R}^2 \quad f(x, y) = f(-x, -y)$ | II  | $\forall (x, y) \in \mathbb{R}^2 \quad f(x, y) = -f(-x, -y)$ |
| III | $\forall (x, y) \in \mathbb{R}^2 \quad f(x, y) = f(y, x)$   | IV  | $\forall (x, y) \in \mathbb{R}^2 \quad f(x, y) = f(-x, y)$   |
| V   | $\forall (x, y) \in \mathbb{R}^2 \quad f(x, y) = -f(-x, y)$ | VI  | $\forall (x, y) \in \mathbb{R}^2 \quad f(x, y) = f(x, -y)$   |
| VII | $\forall a \in \mathbb{R}^2 \quad f(a) = \varphi(\ a\ )$    | avec $\varphi: \mathbb{R} \rightarrow \mathbb{R}$ . |  |

**Exercice 62.** ♦ Pour chacune des fonctions suivantes, préciser si la fonction vérifie une des symétries parmi I à VII.

$$f: (x, y) \mapsto \frac{\cos(x^2 + y^2)}{4 + x^2 + y^2}, \quad g: (x, y) \mapsto 5xy - x^3y^2, \quad h: (x, y) \in \mathbb{R}^2 \mapsto e^{-(x^2 + y^2)^2 + x + y}$$

$$i(x, y) = -xy \exp(-x^2 - y^2), \quad j(x, y) = -3(x + y)/(1 + x^2 + y^2).$$

1. Pour chacune des fonctions, tracer quelques lignes de niveau sur  $[-2; 2] \times [-2; 2]$ .
2. Comment traduire géométriquement les symétries?  
On pourra rajouter la commande `plt.axis('equal')` pour avoir un repère orthonormé.

### 5.3 Gradient et plan affine tangent au graphe

**Exercice 63.** ♦ Tester et expliquer le fonctionnement du code suivant.

Editeur

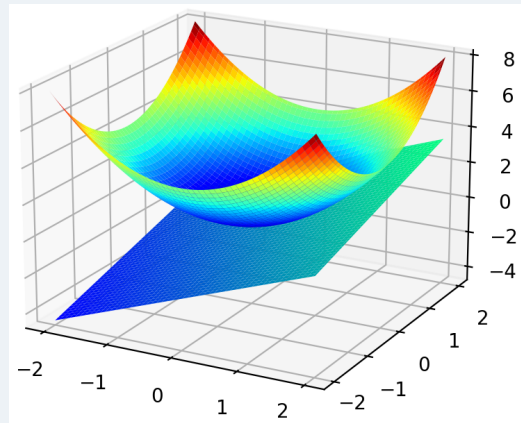
```
import numpy.random as rd
import matplotlib.pyplot as plt

def f(x,y): return x**2+y**2

x = np.linspace(-2, 2, 100)
y = np.linspace(-2, 2, 100)
X, Y = np.meshgrid(x, y)

xa=rd.random()
ya=rd.random()
T=2*xa*(X-xa)+2*ya*(Y-ya)+f(xa, ya)

ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, T, cmap='winter')
ax.plot_surface(X, Y, f(X,Y), cmap='jet')
plt.show()
```



**Exercice 64.** ♦ On définit les fonctions :

$$f: (x, y) \in \mathbb{R}^2 \mapsto x^2 - 3x + xy + y^2 \quad \text{et} \quad g: (x, y) \mapsto x^2 - x - xy - \frac{y^2}{2} + 2y.$$

1. Calculer le ou les points critiques de  $f$ .
2. Tracer la surface représentative de  $f$  ainsi que le plan tangent au point critique. Que peut-on en conjecturer sur la nature de ce point critique?
3. Reprendre les questions précédentes avec  $g$ .

**Exercice 65.** ♦♦ En s'inspirant de l'exercice 21, page 7, proposer une méthode pour afficher la surface représentative de la première dérivée partielle (resp. la deuxième) uniquement en utilisant la fonction  $f$  et sans faire le calcul explicite de la première dérivée.

On pourra tester le code sur  $[-3; 3]^2$  avec la fonction

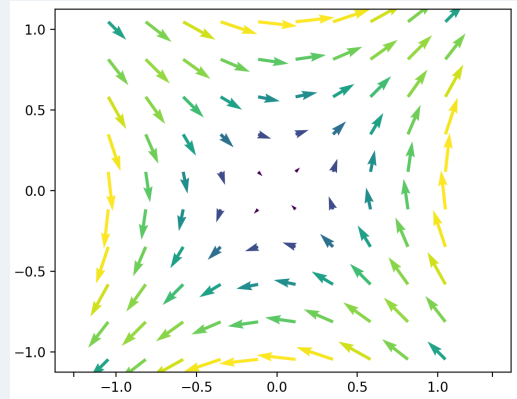
$$(x, y) \in \mathbb{R}^2 \mapsto \sin(x) e^{-x^2 - y^2}.$$

Afin de tracer les vecteurs gradients associés à une fonction de deux variables, on utilise la commande `quiver` dont l'exemple suivant avec la fonction  $f$  définie sur  $\mathbb{R}^2$  par  $f(x, y) = \sin(xy)$  illustre le fonctionnement.

```

x = np.linspace(-np.pi/3, np.pi/3, 10)
y = np.linspace(-np.pi/3, np.pi/3, 10)
X, Y = np.meshgrid(x, y)
Z = np.sin(X*Y)
dx=Y*np.cos(X*Y)
# expression de la première dérivée partielle
dy=X*np.cos(X*Y)
# expression de la seconde
color_array = np.sqrt((dx)**2+(dy)**2)
# Pour que la couleur du vecteur dépende
# de la norme du gradient
fig, ax = plt.subplots(figsize=(7,7))
ax.quiver(X,Y,dx,dy,color_array)
plt.axis('equal')
# pour avoir un repère orthonormé
plt.show()

```



**Exercice 66.** ♦ Pour chacune des fonctions suivantes, tracer des lignes de niveau et des gradients. Quelle propriété du gradient illustre-t-on?

$$\rightarrow f(x, y) = 4 \sin(x) + 3 \sin(y) \text{ sur } [-3; 3]^2.$$

$$\rightarrow g(x, y) = x^2 + y \text{ sur } [-3; 3]^2.$$

$$\rightarrow h(x, y) = -xye^{-x^2-y^2} \text{ sur } [0; 2]^2.$$

$$\rightarrow i(x, y) = -\frac{3y}{1+x^2+y^2} \text{ sur } [-5, 5] \times [0; 10].$$

## 6

## Compléments

### 6.1 Sujet : la loi de Benford

**Exercice 67.** ♦♦ La loi de Benford, énoncée pour de nombreux types de listes de données statistiques, le 1er chiffre le plus fréquent est 1 pour près du tiers des observations. Puis le 2 est lui-même plus fréquent que 3 ... alors que la probabilité d'avoir un 9 comme premier chiffre est inférieure à 5 %.

Cette propriété empirique, mise en évidence par Simon Newcomb en 1881, puis à nouveau par Frank Benford 57 ans plus tard, s'étend à de très nombreux types de listes (nombres dans un journal, longueurs des fleuves en mètres, prix dans les supermarchés, etc.). De plus, ils ont précisé que le premier chiffre suivait une certaine loi, dite de Benford. Une variable aléatoire  $X$  suit la loi de Benford si :

$$X(\Omega) = \llbracket 1; 9 \rrbracket \quad \text{et} \quad \forall k \in \llbracket 1; 9 \rrbracket, \quad \mathbf{P}(X = k) = p_k = \frac{\ln(k+1) - \ln(k)}{\ln(10)}.$$

1. En reprenant le code page 10, afficher le diagramme en bâton d'une loi de Benford?
2. Prévoir la réponse de la machine à cette succession de commandes :

Console

```

num=2022
NUM=str(num)
print(int(NUM[0]))

```

3. En déduire un programme qui prend en argument une matrice ligne  $\mathbf{L}$  de nombres et renvoie l'histogramme de la répartition de la première décimale.
4. Télécharger les deux listes de comptes des usines (celle Twix gauches et celle des Twix droits) sur le site de la classe. Une des deux usines a falsifié ses comptes. Laquelle?

### 6.2 Sujet : le biais d'alternance

**Exercice 68.**

• **Exemple 1 : deux piles consécutifs**

On se place dans le cas d'une infinité de lancers, mutuellement indépendants, d'une pièce équilibrée.

Pour tout  $n \in \mathbb{N}^*$ , on pose l'événement  $G_n$  « une succession d'au moins 2 piles se produit entre le premier et le  $n$ -ième lancer ». On admet que pour tout  $n \in \mathbb{N}^*$ ,

$$\mathbf{P}(G_{n+2}) = \frac{1}{2} \mathbf{P}(G_{n+1}) + \frac{1}{4} \mathbf{P}(G_n) + \frac{1}{4} \quad \text{et} \quad \mathbf{P}(G_1) = 0, \quad \mathbf{P}(G_2) = \frac{1}{4}.$$



On propose deux méthodes pour approximer  $\mathbf{P}(G_n)$ .

1. *Méthode 1.*

Écrire un programme qui prend en argument  $n$  et renvoie la valeur de  $\mathbf{P}(G_n)$  pour  $n \geq 1$  en utilisant une boucle for.

2. *Méthode 2.* Pour tout  $n \in \mathbb{N}^*$ , on pose :

$$U_n = \begin{bmatrix} \mathbf{P}(G_n) - 1 \\ \mathbf{P}(G_{n+1}) - 1 \end{bmatrix}.$$

Trouver une matrice  $C$  telle que pour tout  $n \in \mathbb{N}^*$ ,  $U_{n+1} = CU_n$ . Compléter ensuite la fonction suivante qui prend en argument  $n$  et donne une valeur approchée de  $\mathbf{P}(G_n)$ .

Editeur

```
def proba(n) :
    U=np.array( ....
    C=np.array( ....
    for i in range( ....
        U= ....
    u= ....
    return u+1
```

• **Exemple 2.**

3. Compléter le programme suivant qui simule une série de 100 lancers mutuellement indépendants d'une pièce équilibrée et renvoie 1 si une suite d'au moins 5 piles apparaît et 0 sinon.

*On pourra identifier 0 pour Face et 1 pour Pile.*

Editeur

```
import numpy.random as rd
def simulation() :
    E=list(rd.randint(0,2,100))
    Reponse=0
    for i in .... :
        if E[i:i+5]== .... :
            Reponse= ....
    return Reponse
```

4. En déduire un programme qui donne une approximation de la probabilité d'une succession d'au moins 5 piles dans 100 lancers.

*On rappelle que la fréquence empirique d'apparition donne une approximation de la probabilité.*

• **Cas général**

5. Modifier les programmes précédents pour écrire un programme `approxliste` qui prend en entrée une liste de {0;1} (correspondant à une liste donnée de Pile et Face) et renvoie une approximation de la probabilité que la liste soit dans les 100 lancers.

Quelques résultats :

```
>> approx2([0,1,0,1,0,1])           >> approx2([1,1,1,1,1])           >> approx2([1,1,1,1])
une approximation 0.706                une approximation 0.806                une approximation 0.973
```

• **Application**

6. Voici trois listes, dont une seule a été réalisée en lançant 100 fois une pièce, les deux autres ont été inventées par deux étudiants d'EC. Saurez-vous retrouver la vraie liste?

```
Lun=[1,0,1,0,0,1,0,0,1,1,1,0,1,0,1,1,1,0,0,1,1,1,1,0,0,0,1,0,1,1,1,0,0,1,1,
0,1,0,1,1,0,0,1,1,0,0,1,0,0,1,1,0,1,0,1,0,0,0,0,1,0,1,1,1,0,1,1,0,0,1,0,
1,0,1,0,1,0,0,1,1,1,0,1,0,0,1,0,0,1,1,1,0,0,1,0,0]
```

```
Ldeux=[1,1,0,0,1,0,1,0,1,1,1,0,1,0,0,1,0,1,0,1,0,0,1,1,0,1,0,1,0,0,1,0,0,1,1,
0,1,1,1,0,0,1,0,1,0,1,0,0,1,0,1,1,0,1,0,0,1,1,0,0,0,0,1,0,1,1,1,0,1,0,0,0,1,1,
1,1,0,1,0,1,1,0,1,1,1,0,0,1,0,1,0,1,1,1,0,0,0,1]
```

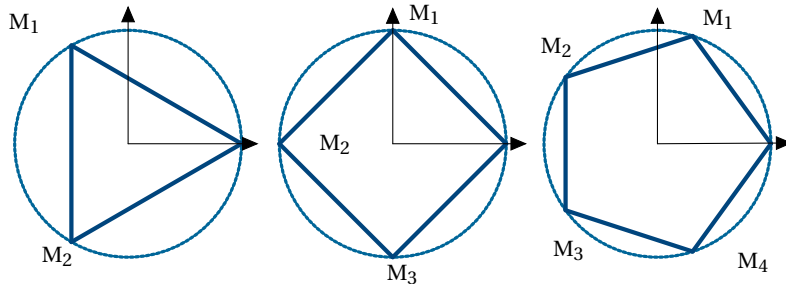
Ltrois=[0,1,0,1,0,0,1,0,1,0,0,1,0,0,1,0,1,0,1,0,0,0,0,1,0,0,1,1,1,0,1,1,1,0,1,0,1,1,0,0,1,1,1,0,0,1,0,1,1,1,0,0,1,0,1,1,1,0,1,0,1,1,1,0,1,1,1,0,0,1,1,1,0,0,1,1,1,0,0,1,1,1,0,0,1,0,1,1,0,1,1,1,1,0,0]

### 6.3 Sujet : les polygones réguliers

**Exercice 69.** ♦♦ Soit  $n \in \mathbb{N} \setminus \{0; 1\}$ . Pour tout  $k \in \mathbb{N}$ , on considère le point

$$M_k : \left( \cos\left(\frac{2\pi k}{n}\right), \sin\left(\frac{2\pi k}{n}\right) \right).$$

On montre que les points  $M_0, M_1, M_2, \dots, M_{n-1}, M_n = M_0$  constituent les sommets d'un polygone régulier à  $n$  côtés inscrit dans le cercle unité. Pour  $n = 3, n = 4, n = 5$ , on a un triangle équilatéral, un carré et un pentagone.



1. Écrire un programme qui prend en argument  $n$  et renvoie les matrices lignes

$$X = \left[ \cos\left(\frac{2\pi}{n} \cdot 0\right), \cos\left(\frac{2\pi}{n} \cdot 1\right), \cos\left(\frac{2\pi}{n} \cdot 2\right) \dots \cos\left(\frac{2\pi}{n} \cdot n\right) \right] \quad \text{et} \quad Y = \left[ \sin\left(\frac{2\pi}{n} \cdot 0\right), \sin\left(\frac{2\pi}{n} \cdot 1\right), \sin\left(\frac{2\pi}{n} \cdot 2\right) \dots \sin\left(\frac{2\pi}{n} \cdot n\right) \right].$$

2. En déduire un programme qui prend en argument un entier  $n$  et trace un polygone régulier à  $n$  côtés inscrit dans le cercle unité. Tester le programme en traçant un triangle, un pentagone, un hexagone et finalement un chiliogone.

3. *Visualisation des tables de multiplication*

a) Donner un programme qui prend en entrée un couple d'entiers  $(n, a)$  et qui, sur un même graphique, commence par tracer le polygone régulier à  $n$  cotés, puis, à l'intérieur de ce polygone, trace pour tout  $k \in \llbracket 0; n \rrbracket$  le segment entre  $M_k$  et  $M_{ak}$ . *Indication.* Pour tracer un segment entre les points  $A(x_A, y_A)$  et  $B(x_B, y_B)$ , il suffit de taper

```
ab=[xA , xB] ; ord=[yA , yB]
plt.plot(ab , ord)
```

b) Observer les figures obtenues pour les couples  $(n, a)$  avec  $(300, 2), (300, 3), (300, 6), (84, 55), (324, 28), (405, 28)$  et  $(993, 399)$ .

4. *Périmètre*

a) Écrire une fonction qui prend en argument deux points  $A(x_A, y_A)$  et  $B(x_B, y_B)$  de  $\mathbb{R}^2$  et renvoie la distance entre  $A$  et  $B$ . *Pour rappel,*  $AB^2 = (x_B - x_A)^2 + (y_B - y_A)^2$ .

b) Calculer ou donner avec python une approximation du périmètre du polygone régulier à  $n$  côtés inscrit dans le cercle unité. Que dire lorsque  $n \rightarrow +\infty$ ?

c) Écrire un programme qui prend en argument  $n$  et renvoie un polygone inscrit dans le cercle unité dont les  $n$  points sont choisis au hasard.

*Indication.* On pourra utiliser la commande `np.sort(A)` qui, à partir de la matrice ligne  $A$ , renvoie une matrice ligne dont les coefficients sont les coefficients de  $A$  ordonnés dans l'ordre croissant.

d) Adapter le programme pour afficher en plus le périmètre du polygone. Vérifier que le périmètre obtenu est systématiquement plus grand que le périmètre du polygone régulier à  $n$  côtés.

### 6.4 Sujet : Matrices de Hankel et de Hilbert

**Exercice 70.** ♦♦ Soit  $n \in \mathbb{N}^*$ . À tout vecteur  $x = (x_k)_{k \in \llbracket 0; 2n \rrbracket}$ , on définit la matrice de Hankel de taille  $(n+1) \times (n+1)$  par

$$X = \begin{bmatrix} x_0 & x_1 & x_2 & \dots & x_n \\ x_1 & x_2 & x_3 & \dots & x_{n+1} \\ x_2 & x_3 & x_4 & \dots & x_{n+2} \\ \vdots & \vdots & \vdots & & \vdots \\ x_n & x_{n+1} & x_{n+2} & \dots & x_{2n} \end{bmatrix}.$$

**1. Construction d'une matrice de Hankel et lien avec les suites récurrentes linéaires.**

- a) Écrire un programme qui prend en argument  $x \in \mathbb{R}^{2n+1}$  et renvoie la matrice de Hankel associée.
- b) On définit la suite  $(x_n)_{n \in \mathbb{N}}$  par la récurrence

$$x_0 = -1, \quad x_1 = 3 \quad \text{et, pour tout } n \in \mathbb{N} \quad x_{n+2} = 3x_{n+1} - 2x_n.$$

- i) Écrire un programme qui prend en argument un entier naturel  $n$  et renvoie la matrice ligne

$$[x_0 \quad x_1 \quad \dots \quad x_n].$$

- ii) Afficher les rangs des matrices de Hankel associées à  $(x_i)_{i \in \llbracket 0, 2n \rrbracket}$  pour  $n$  variant de 1 à 7. Que constatez-vous? Expliquez et généralisez.

On considère maintenant  $H_n$ , la matrice de Hilbert de taille  $n \times n$  définie par :

$$H_n = \left[ \left( \frac{1}{i+j-1} \right) \right]_{(i,j) \in \llbracket 1, n \rrbracket^2} = \begin{bmatrix} 1 & \frac{1}{2} & \dots & \dots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \dots & \dots & \frac{1}{n+1} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \dots & \dots & \frac{1}{2n-1} \end{bmatrix}.$$

L'objectif de la suite est de vérifier numériquement quelques propriétés des matrices de Hilbert.

On admet que la matrice  $H_n$  est inversible. Pour  $(i, j) \in \llbracket 1, n \rrbracket^2$ , on note  $h_{i,j}^{(-1,n)}$  le coefficient en position  $(i, j)$  de la matrice  $H_n^{-1}$  et on désigne par  $s_n$  la somme des coefficients de la matrice  $H_n^{-1}$ , c'est-à-dire :

$$s_n = \sum_{(i,j) \in \llbracket 1, n \rrbracket^2} h_{i,j}^{(-1,n)}.$$

**2. Construction d'une matrice de Hilbert.**

En utilisant le programme de la question 1.(a), déduire un programme qui prend en argument  $n$  et renvoie la matrice  $H_n$ .

**3. Propriétés arithmétiques de l'inverse.**

- a) Vérifier que pour des valeurs de  $n$  petites, les coefficients de  $H_n^{-1}$  sont des entiers.
- b) Conjecturer une expression très simple de  $h_{1,1}^{(-1,n)}$ .
- c) Écrire un programme qui prend en argument  $n$  et renvoie  $s_n$ . Tester le programme, que peut-on conjecturer sur  $s_n$ ?

## 6.5 Exercices supplémentaires

**Exercice 71.** ♦ Écrire un programme qui prend en argument une matrice ligne  $x \in \mathcal{M}_{1,n}(\mathbb{R})$  et un entier  $p$  et renvoie

$$S_p(x) = \sqrt[p]{\sum_{i=1}^n x_i^p}.$$

Tirer au hasard une matrice  $x \in \mathcal{M}_{1,4}(\mathbb{R})$  et tracer  $(S_p(x))_{p \in \llbracket 1; 20 \rrbracket}$ . Que constatez-vous?

**Exercice 72.** ♦ Soit  $n \in \mathbb{N}$ , on pose :

$$u_n = \sin\left(2\pi(2 + \sqrt{3})^n\right).$$

1.
  - a) Tracer le graphe de  $x \in [0; \pi] \mapsto \sin(x)$  et  $x \in [0; \pi] \mapsto x$ . Quelle courbe est au-dessus?
  - b) Calculer  $(2 + \sqrt{3})^n + (2 - \sqrt{3})^n$  pour  $n \in \llbracket 0; 20 \rrbracket$ . Que remarque-t-on?
  - c) Écrire un programme qui calcule  $u_n$ . Que conjecturer sur la convergence de la limite  $u$ ?
2. À l'aide de ces trois informations, prouver que la suite  $u$  est bien convergente et préciser la limite.

**Exercice 73.** ♦♦ On définit une suite  $(u_n)_{n \geq 1}$  par  $u_1 = 1$ , et pour tout  $n \geq 1$  :

$$u_{n+1} = \frac{1 + u_1^2 + u_2^2 + \dots + u_n^2}{n}.$$

1. Étudier la monotonie de cette suite et donner sa limite.
2. Construire une fonction Python nommée  $u$  prenant en argument  $n$  qui renvoie  $u_n$ .
3. On considère le programme informatique suivant :

```

i=0
p=1
while u(i)>=p:
    i=i+1
    p=p*2

```

Ce programme affiche 7 lorsqu'on l'exécute. Déterminer la nature de la série de terme général  $\frac{u_n}{2^n}$ .

4. On considère le programme informatique suivant :

```

n=0
S=1
a=1
while u(n) != int(u(n)):
    n=n+1

```

On suppose que ce programme puisse «tourner». Il affiche 43 lors de son exécution. Que peut-on en déduire?

**Exercice 74.** ♦ On dit qu'une matrice  $A = (a_{i,j}) \in \mathcal{M}_n(\mathbb{R})$  est à diagonale dominante si :

$$\forall i \in \llbracket 1; n \rrbracket, \quad |a_{i,i}| \geq \sum_{\substack{j=1 \\ j \neq i}}^n |a_{i,j}|.$$

Écrire une fonction Python **DiagoD** qui prend en argument une matrice A et retourne "vrai" si A est à diagonale dominante et "faux" sinon.