Feuille de TP n° 0

Révisions de première année (corrections)

I Commandes et structures de bases

Exercice 1 (Renversement d'une liste). Écrire une fonction qui prend en argument une liste et qui renvoie la liste dans laquelle les éléments sont listés dans l'autre sens (du dernier de la liste d'entrée au premier de la liste d'entrée).

Correction : On remarque simplement que, lorsque k varie de 1 à n, alors $n-k$ varie de $n-1$ à 0, c'est-à-dire parcourt les indices d'une liste à n éléments dans le sens inverse. D'où la fonction :

```
def RenverseListe(L):
    n=len(L)
    return [L[n-k] for k in range(1,n+1)]
```

Exercice 2 (Somme, produit, sommes cumulées, moyenne, variance, écart type).

1) Écrire une fonction qui prend en argument une liste de réels et renvoie la somme des éléments (sans utiliser `np.sum`).
2) Écrire une fonction qui prend en argument une liste de réels et renvoie le produit des éléments (sans utiliser `np.prod`).
3) Écrire une fonction qui prend en argument une liste de réels et renvoie la liste des sommes cumulées des éléments (sans utiliser `np.cumsum`).
4) Écrire une fonction qui prend en argument une liste de réels et renvoie la moyenne des éléments (sans utiliser `np.mean`).
5) Écrire une fonction qui prend en argument une liste de réels et renvoie la variance des éléments (sans utiliser `np.var`).
6) Écrire une fonction qui prend en argument une liste de réels et renvoie l'écart type des éléments (sans utiliser `np.std`).

Correction :

1) Pour la somme :

```
def Somme(L):
    S=0
    for x in L:
        S=S+x
    return S
```

2) Pour le produit :

```
def Produit(L):
    S=0
    for x in L:
        S=S*x
    return S
```

3) Pour le vecteur des sommes cumulées :

```
def SommeCumulee(L):
    Res=[]
    S=0
    for x in L:
        S=S+x
        Res.append(S)
    return Res
```

4) Pour la moyenne :

```
def Moyenne(L):
    S=0
    for x in L:
        S=S+x
    return S/len(L)
```

ou encore (sans utiliser la commande `len`) :

```
def Moyenne(L):
    S=0
    n=0
    for x in L:
        S=S+x
        n=n+1
    return S/n
```

5) Pour la variance :

```
def Variance(L):
    S=0
    m=Moyenne(L)
    for x in L:
        S=S+(x-m)**2
    return S/len(L)
```

ou encore, en utilisant la formule de Koenig-Huygens :

```
def Variance(L):
    S1=0
    S2=0
    n=0
    for x in L:
        S1=S1+x
        S2=S2+x**2#On utilise Koenig Huygens
        n=n+1
    return S2/n-(S/n)**2
```

6) Pour l'écart-type :

```
def EcartType(L):
    return (Variance(L))**(1/2)
```

ou encore :

```
def EcartType(L):
    S1=0
    S2=0
    n=0
    for x in L:
        S1=S1+x
        S2=S2+x**2#On utilise Koenig Huygens
        n=n+1
    return (S2/n-(S/n)**2)**(1/2)
```

Exercice 3 (Recherche du nombre d'occurrences). Écrire une fonction qui prend en argument une liste et une variable et qui renvoie le nombre d'occurrences de la variable dans la liste.

Correction : On parcourt la liste et, si on trouve l'élément, on ajoute 1 à un compteur :

```
def Nb_Occurences(L,x):
    c=0
    for a in L:
        if a==x:
            c=c+1
    return c
```

Exercice 4 (Recherche du max et du min).

1) Écrire une fonction qui prend en argument une liste de réels et qui renvoie le maximum de la liste (sans utiliser `np.max`), ainsi que le plus petit indice où se trouve le maximum.

   Indications : le premier élément de la liste est le maximum provisoire. Ensuite on parcourt la liste de gauche à droite et, dès qu'on trouve un élément strictement plus grand que le maximum provisoire, celui-ci devient le nouveau maximum provisoire.

2) Écrire une fonction qui prend en argument une liste de réels et qui renvoie le minimum de la liste (sans utiliser `np.min`), ainsi que le plus petit indice où se trouve le minimum.

3) Tester ces fonctions avec la liste `L=[rd.binomial(15,0.3) for k in range(20)]`.

4) Modifier ces fonctions pour qu'elles renvoient la liste de tous les indices où se trouvent le maximum et le minimum respectivement. Les tester avec la liste de la question précédente.

Correction :

1)
```
def Trouve_max(L):
    imax=0#Au début de l'exploration, le max est en indice 0
    n=len(L)
    for i in range(1,n):
        if L[i]>L[imax]:
            imax=i
    return [L[imax],imax]
```

2)
```
def Trouve_min(L):
    imin=0#Au début de l'exploration, le min est en indice 0
    n=len(L)
    for i in range(1,n):
        if L[i]<L[imin]:
            imin=i
    return [L[imin],imin]
```

3)
```
L=[rd.binomial(15,0.3) for k in range(20)]
print(L)
print(Trouve_max(L))
print(Trouve_min(L))
```

4)
```
def Trouve_max2(L):
    imax=0#Au début de l'exploration, le max est en indice 0
    n=len(L)
    for i in range(1,n):
        if L[i]>L[imax]:
            imax=i
    Pos=[imax]
    for k in range(imax+1,n):
        if L[k]==L[imax]:
            Pos.append(k)
    return [L[imax],Pos]

print(Trouve_max2(L))
```

```
14
15 def Trouve_min2(L):
16     imin=0#Au début de l'exploration, le min est en indice 0
17     n=len(L)
18     for i in range(1,n):
19         if L[i]<L[imin]:
20             imin=i
21     Pos=[imin]
22     for k in range(imin+1,n):
23         if L[k]==L[imin]:
24             Pos.append(k)
25     return [L[imin],Pos]
26
27 print(Trouve_min2(L))
```

En testant avec la liste `[3, 3, 3, 5, 6, 6, 7, 5, 4, 4, 4, 5, 3, 5, 5, 7, 5, 3, 7, 3]`, j'ai obtenu :

```
[7, 6]
[3, 0]
[7, [6, 15, 18]]
[3, [0, 1, 2, 12, 17, 19]]
```

II Suites, sommes, produits

Exercice 5 (Classique : factorielle). Écrire une fonction qui prend en argument un entier naturel et renvoie sa factorielle.

Correction :

```
1 import numpy as np
2 def Factorielle(n):
3     f=0
4     for k in range(1,n+1):
5         f=f*k
6     return f
```

ou encore :

```
1 import numpy as np
2 def Factorielle(n):
3     return np.prod([k in range(1,n+1)])
```

Exercice 6 (Classique : coefficients binomiaux). Écrire une fonction qui prend en argument deux entiers naturels n et p et qui renvoie $\binom{n}{p}$ (avec la convention que $\binom{n}{p} = 0$ si $p > n$).

Cette fonction ne devra pas utiliser de fonction factorielle.

Correction : On utilise le fait que

$$\binom{n}{p} = \frac{n(n-1)(n-2)\dots(n-p+1)}{p!} = \frac{n(n-1)(n-2)\dots(n-p+1)}{p(p-1)(p-2)\dots(p-p+1)} = \prod_{k=0}^{p-1} \frac{n-k}{p-k}.$$

D'où la fonction :

```
1 def CoeffBinom(n,p):
2     if p>n:
3         return 0
4     else:
5         c=0
6         for k in range(p):
7             c=c*(n-k)/(p-k)
8         return c
```

Exercice 7. On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 = 1$ et

$$\forall n \in \mathbb{N}, \qquad u_{n+1} = \frac{(n-2)u_n + 2}{u_n + n}.$$

1) Construire une fonction en Python qui prend en argument un entier naturel N et qui renvoie u_N.

2) Construire une fonction en Python qui prend en argument un entier naturel N et qui représente graphiquement $u_0, u_1, \ldots, u_N$. La tester pour $N = 50$. Que peut-on conjecturer ? Que peut-on superposer à la courbe pour améliorer la conjecture ?

Correction :

1)
```
def Suite(N):
    u=1
    for n in range(N):
        u=((n-2)*u+2)/(u+n)
    return u
```

2)
```
import matplotlib.pyplot as plt
def RepSuite(N):
    u=1
    L=[1]
    for n in range(N):
        u=((n-2)*u+2)/(u+n)
        L.append(u)
    plt.plot(range(N+1),L)
    plt.show()

RepSuite(50)
```

On conjecture que la suite converge. La valeur `Suite(50)` est une approximation de la limite. On peut superposer à la courbe la droite d'ordonnée l'approximation de la limite :

```
import matplotlib.pyplot as plt
def RepSuite2(N):
    u=1
    L=[1]
    for n in range(N):
        u=((n-2)*u+2)/(u+n)
        L.append(u)
    plt.plot(range(N+1),L)
    lim=L[-1]
    plt.plot([0,N+1],[lim,lim])
    plt.show()

RepSuite2(50)
```

Exercice 8. Soient a et b deux réels. On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 = a$, $u_1 = b$ et

$$\forall n \in \mathbb{N}, \qquad u_{n+2} = \frac{u_{n+1}}{2} + e^{-u_n}.$$

1) Recopier et compléter la fonction Python ci-dessous pour qu'elle prenne en argument les réels a et n et un entier naturel N et renvoie u_N.

```
def Suite(a,b,N):
    u = ............
    v = ............
    for n in range(N):
        aux = ............
        u=v
        v = ..............
    return  ............
```

2) Comment peut-on remplacer les lignes 5,6,7 par une seule ligne ?

3) Construire une fonction en Python qui prend en argument les réels a et n et un entier naturel N et qui représente graphiquement $u_0, u_1, \dots, u_N$. La tester pour $N = 50$ et pour plusieurs valeurs de a et b. Que peut-on conjecturer ?

Correction :

1)
```
import numpy as np
def Suite(a,b,N):
    u=a
    v=b
    for n in range(N):
        aux=v/2+np.exp(-u)
        u=v
        v=aux
    return u
```

2)
```
import numpy as np
def Suite(a,b,N):
    u=a
    v=b
    for n in range(N):
        u,v=v,v/2+np.exp(-u)
    return u
```

3)
```
def RepSuite(a,b,N):
    u=a
    v=b
    L=[u]
    for n in range(N):
        u,v=v,v/2+np.exp(-u)
        L.append(u)
    plt.plot(range(N+1),L)
    plt.show()
```

Testons-la pour plusieurs valeurs :

```
for a in [-5,0,2]:
    for b in [-10,-1,4]:
        plt.figure()#Ouvre une nouvelle fenêtre
        RepSuite(a,b,50)
```

Dans tous les cas, on conjecture qu'il y a convergence vers un réel proche de 0,8526.

La difficulté mathématique est de montrer que la suite converge. Mais, si convergence il y a lieu, la limite ℓ vérifie $\ell = \frac{\ell}{2} + e^{-\ell}$ ce qui est équivalent à $\ell = 2e^{-\ell}$. Le théorème de la bijection (appliqué à $f : x \longmapsto x - 2e^{-x}$) montre que cette équation admet bien une unique solution qui vaut environ 0,8526).

Exercice 9 (Nombres de Bell). On pose $B_0 = 1$ et

$$\forall n \in \mathbb{N}, \qquad B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k.$$

1) Recopier et compléter le programme ci-dessous pour qu'il prenne en entrée un entier naturel N non nul et pour qu'il renvoie la liste des entiers $B_0, \dots, B_N$.

```
import ......................
def Bell(N):
    B=np.zeros(N+1)
    B[0]=1
    Facto=[]
    for n in range(N):
        #Utilisation de la formule de Pascal pour former la
        #liste des (p parmi n) pour tout p de 0 à n
        Temp=Facto+[1]
        for p in range(n-1):
            Temp[p+1]=.............................
        Facto=Temp
        #Calcul de B_(n+1)
        ..............................
        ..............................
        ..............................
        B[n+1]=S
    return B
```

2) On appelle partition d'un ensemble E, toute famille de parties non vides deux à deux disjoints dont la réunion est E tout entier. Montrer que, pour tout $n \in \mathbb{N}^*$, B_n est le nombre de partitions de $[\![1\,;n]\!]$.
On raisonnera par récurrence. Pour tout $n \in \mathbb{N}$ et $k \in [\![0\,;n]\!]$, on pourra introduire E_k l'ensemble des partitions de $[\![1\,;n+1]\!]$ pour lesquelles la partie contenant $n+1$ est de cardinal $k+1$.

Correction :

1) Rappelons la formule de Pascal : pour tout $p \in [\![1\,;n]\!]$, $\binom{n+1}{p} = \binom{n}{p} + \binom{n}{p-1}$.

```
1  import numpy as np
2  def Bell(N):
3      B=np.zeros(N+1)
4      B[0]=1
5      Facto=[]
6      for n in range(N):
7          #Utilisation de la formule de Pascal pour former la
8          #liste des (p parmi n) pour tout p de 0 à n
9          Temp=Facto+[1]
10         for p in range(n-1):
11             Temp[p+1]=Facto[p]+Facto[p+1]
12         Facto=Temp
13         #Calcul de B_n
14         S=0
15         for k in range(n+1):
16             S=S+Facto[k]*B[k]
17         B[n+1]=S
18     return B
```

Expliquons plus en détail :

- Si $N = 0$, la boucle démarrant à la ligne 6 n'a pas lieu et on renvoie bien la liste contenant seulement B_0.
- Si $N = 1$, la boucle démarrant à la ligne 6 consiste à faire prendre à n la valeur 0 seulement. Dans cette unique étape, `Facto` devient `[1]` (puisque la boucle des lignes 10 et 11 n'a pas lieu), on calcule

$$B_1 = \sum_{k=0}^{0} \binom{0}{k} B_k = 1$$

et on la stocke bien dans la variable `B[0+1]`.

- Si $N \geqslant 2$, à l'étape n de la boucle démarrant à la ligne 6, on calcule B_n. Il s'agit bien de la somme

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$$

implémentée dans la variable `S` calculée aux lignes 14,15,16 (le `range(n+1)` parcourt bien les indices de `0` à `n`) et on doit bien la stocker dans la variable `B[n+1]`. Justifions que, à l'étape n de cette boucle `Facto` contient bien $\binom{n}{0}, \binom{n}{1}, \dots, \binom{n}{n}$. Pour cela examinons les lignes 9,10,11 :

— Lorsque $n = 0$, la boucle démarrant à la ligne 10 n'a pas lieu et `Facto` devient `[1]`.

— Lorsque n prend la valeur 2, la boucle démarrant à la ligne 10 n'a pas lieu de nouveau et `Facto` devient `[1,1]`.

— Lorsque n prend la valeur 2, `Temp` devient `[1,1,1]`. Ensuite la boucle démarrant à la ligne 10 a lieu une fois (avec p valant 0). Pour obtenir les coefficients voulus, il faut donc que la coordonnée d'indice 1=0+1 devienne 2, la somme des coordonnées 0 et 1 de `Facto` (et non pas de `Temp`, on verra pourquoi ensuite) en vertu de la formule de Pascal. Enfin `Facto` devient bien `[1,2,1]`.

— Lorsque n prend la valeur 3, `Temp` devient `[1,2,1,1]`. Ensuite la boucle démarrant à la ligne 10 a lieu deux fois (avec p valant 0 puis 1). Pour obtenir les coefficients voulus, il faut donc que la coordonnée d'indice 1=0+1 devienne 3, la somme des coordonnées 0 et 1 de `Facto` puis que la coordonnée d'indice 2=1+1 devienne 3, la somme des coordonnées 1 et 2 de `Facto` (et non pas de `Temp`, puisqu'on a vient de modifier sa coordonnée d'indice 1 donc ce n'est plus la bonne).

— A l'étape n, cela revient à ajouter un `1` à la fin de la liste du `Facto` de rang $n-1$ et modifie les coordonnées d'indices 1 à $n-1$ en la somme des deux précédents... c'est bien la formule de Pascal.

2) Raisonnons par récurrence.

- Si $n = 1$, il n'y a qu'une seule partition de $[\![1\,;n]\!] = \{1\}$. Il s'agit de $(\{1\})$. On a bien $B_1 = 1$.
- Soit $n \in \mathbb{N}^*$. Supposons que B_n et le nombre de partitions de $[\![1\,;n]\!]$. Pour tout $k \in [\![0\,;n]\!]$, introduisons E_k l'ensemble des partitions de $[\![1\,;n+1]\!]$ pour lesquelles la partie contenant $n+1$ est de cardinal $k+1$. Pour construire une telle partition, il suffit de choisir les k autre entiers de la partie contenant $n+1$ parmi les n entiers possibles (il y a $\binom{n}{k}$ possibilités). Il reste alors $n-k$ éléments à partitionner. Or (quitte à les renuméroter, on peut supposer qu'il s'agit de $1, \dots, n-k$), par hypothèse de récurrence, il y a B_{n-k} façons de les partitionner. Finalement $\mathrm{card}(E_k) = \binom{n}{k} B_{n-k}$.

  L'ensemble des partitions de $[\![1\,;n+1]\!]$ est bien sûr $\bigcup_{k=0}^{n} E_k$ (puisqu'une telle partition contient bien une partie contenant $n+1$ et que celle-ci a pour cardinal $1, 2, 3, \dots$ ou bien $n+1$) et il s'agit d'ensemble deux à deux disjoints (la partie contenant $n+1$ ne peut pas contenir à la fois $i+1$ et $j+1$ éléments pour tous i et j distincts). Le nombre de partitions de $[\![1\,;n+1]\!]$ est donc

$$\sum_{k=0}^{n} \mathrm{card}(E_k) = \sum_{k=0}^{n} \binom{n}{k} B_{n-k} = \sum_{j=0}^{n} \binom{n}{n-j} B_j = \sum_{j=0}^{n} \binom{n}{j} B_j = B_{n+1}$$

  (on a fait le changement d'indice $j = n-k$).
- Par récurrence, pour tout $n \in \mathbb{N}^*$, B_n est la nombre de partitions de $[\![1\,;n]\!]$.

III Analyse numérique

1) Dichotomie

Exercice 10 (D'après EDHEC 2016). Soit $f : x \in \mathbb{R}_+^* \longmapsto \dfrac{e^{-x}}{x}$.

1) Étudier les variations de la fonction $g : x \longmapsto e^{-x} - x^2$ sur $\mathbb{R}_+$.
2) Montrer que l'équation $f(x) = x$, d'inconnue $x \in \mathbb{R}_+^*$, admet une unique solution que l'on notera α.
3) Montrer que $e^{-1} < \alpha < 1$.
4) Compléter le script Python suivant qui affiche une valeur approchée de α à 10^{-3} près :

```
import ..................
def g(x):
    return ..............
a=...........
b=...........
while ...................
    c=(a+b)/2
    if g(c)>0:
        ...........
    else:
        ...........
print('Une valeur approchée de alpha est '+str(.....))
```

Correction :

1) La fonction g est dérivable sur $\mathbb{R}_+$ par somme de fonctions qui le sont. Pour tout $x \in \mathbb{R}_+$, $g'(x) = -e^{-x} - 2x < 0$. Ainsi g est strictement décroissante sur $\mathbb{R}_+$.
2) Pour tout $x \in \mathbb{R}_+^*$, $f(x) = x$ si et seulement si $e^{-x} = x^2$ si et seulement si $g(x) = 0$. La fonction g étant continue et strictement décroissante sur $\mathbb{R}_+^*$, elle réalise une bijection de $\mathbb{R}_+^*$ dans $f(\mathbb{R}_+^*) = \left]\lim\limits_{+\infty} g\,;\lim\limits_{0^+} g\right[=]-\infty\,;1[$. Puisque $0 \in]-\infty\,;1[$, il existe donc un unique $\alpha \in \mathbb{R}_+^*$ tel que $g(\alpha) = 0$, qui est équivalent à $f(\alpha) = \alpha$.
3) On a $g(e^{-1}) = e^{-e^{-1}} - e^{-2} > 0$ (car $e^{-1} < 2$ donc $-e^{-1} > -2$ et pas stricte croissante de l'exponentielle). On a $g(1) = e^{-1} - 1 < 0$. Ainsi $g(1) < g(\alpha) < g(e^{-1})$. Comme g est strictement décroissante, on obtient $e^{-1} < \alpha < 1$.
4) Comme g est strictement décroissante, lorsque $g(c) > 0$, c'est que le zéro est « à droite » (on initialise a).

```
import numpy as np
def g(x):
    return np.exp(-x)-x**2
a=np.exp(-1)
b=1
while b-a>10**(-3):
    c=(a+b)/2
    if g(c)>0:
        a=c
    else:
        b=c
print('Une valeur approchée de alpha est '+str(a))
```

On trouve environ 0,70 (on trouve une plus grande approximation mais la méthode ne garantit que les 2 premiers chiffres après la virgule).

Exercice 11 (D'après HEC 2020). Soit $f : t \in]0\,;1[\longmapsto \dfrac{t}{1-t}e^{-1/t}$.

1) Montrer que f est prolongeable en une fonction continue sur $[0\,;1[$.
2) Montrer que f, ainsi prolongée, est bijective de $[0\,;1[$ sur un intervalle I à préciser.
3) On cherche à représenter graphiquement f^{-1} sur I à l'aide de Python.
   a) Première méthode : avec un algorithme de dichotomie. Recopier et compléter le script Python suivant pour qu'il affiche une représentation graphique de f^{-1} sur $[0\,;f(0,95)]$.

```
import ....................
import ....................
#Implémentation de f
def f(t):
    if .........
        return .........
    else:
        return .........

#Implémentation de la réciproque de f
def Inv_f(x):
    a=0
    b=0.95
    while ..............
        c=(a+b)/2
        if f(c)>x:
            ..............
        else:
            ..............
    return ..............

#Tracé de la fonction
X=.............................
Y=.............................
plt.plot(X,Y)
plt.show()
```

b) Deuxième méthode : en se rappelant que la courbe de f^{-1} s'obtient facilement via une transformation géométrique de la courbe de f.

4) Conjecturer le domaine sur lequel l'application f^{-1} est dérivable. Prouver cette conjecture.

Correction :

1) La fonction f est continue sur $]0\,;1[$ en tant que produit de 3 fonctions qui le sont ($t\longmapsto t$, $t\longmapsto e^{-1/t}$ et $t\longmapsto \dfrac{1}{1-t}$).

Ensuite on a $-\dfrac{1}{t}\xrightarrow[t\to 0^+]{}-\infty$ donc $e^{-1/t}\xrightarrow[t\to 0^+]{}0$. Par produit $f(t)\xrightarrow[t\to 0^+]{}0$. Ainsi on prolonge f par continuité en 0 en posant $f(0)=0$.

2) Pour les mêmes raisons, f est dérivable sur $]0\,;1[$. Pour tout $t\in[0\,;1[$,

$$f'(t)=\frac{e^{-1/t}}{1-t}+\frac{t\times(1/t^2)e^{-1/t}}{1-t}+te^{-1/t}\times\frac{1}{(1-t)^2}=\frac{e^{-1/t}}{(1-t)^2}\left(1-t-(1-t)/t+t\right)=\frac{e^{-1/t}}{t(1-t)^2}>0.$$

La fonction f est donc strictement croissante sur $\mathbb{R}_+^*$ donc sur $\mathbb{R}_+$ si bien que le théorème de la bijection entraîne qu'elle réalise une bijection de $[0\,;1[$ sur $I=f([0\,;1[)=\left[f(0)\,;\lim_{1^-}f\right[=[0\,;+\infty[$.

3) a) Comme f est strictement croissante, lorsque $f(c)>x$, c'est que l'antécédent de x est « à gauche » (on initialise b).

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

#Implémentation de f
def f(t):
    if t>0:
        return t*np.exp(-1/t)/(1-t)
    else:
        return 0

#Implémentation de la réciproque de f
def Inv_f(x):
    a=0
```

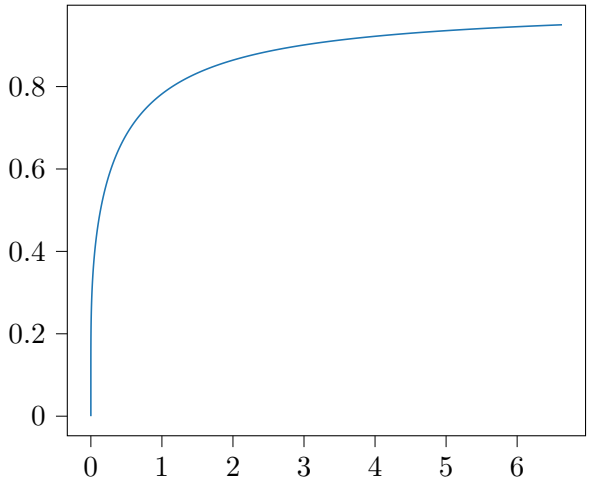
```
    b=0.95
    while b-a>10**(-3):
        c=(a+b)/2
        if f(c)>x:
            b=c
        else:
            a=c
    return a

#Tracé de la fonction
X=np.linspace(0,f(0.95),1000)
Y=[Inv_f(x) for x in X]
plt.plot(X,Y)
```

b) Il suffit d'opérer sur la courbe de f une symétrie axiale par rapport à l'axe d'équation $y = x$. Cela revient à échanger les abscisses et les ordonnées :

```
X=np.linspace(0,0.95,1000)
Y=[f(x) for x in X]
plt.plot(Y,X)
plt.show()
```

4) On obtient :



On conjecture que f^{-1} est de classe C^1 sur $\mathbb{R}_+^*$ (et pas en 0). Prouvons-le : f est de classe C^1 sur $]0\,;1[$, $f'(t) = \dfrac{e^{-1/t}}{t(1-t)^2} \neq 0$ donc f^{-1} est de classe C^1 sur $f(]0\,;1[) = \mathbb{R}_+^*$. Ensuite

$$\frac{f(t)-f(0)}{t-0} = \frac{e^{-1/t}}{1-t} \xrightarrow[t\to 0^+]{} 0.$$

Ainsi f est dérivable en 0 et $f'(0) = 0$ si bien que f^{-1} n'est pas dérivable en 0.

2) Méthode des rectangles

Exercice 12 (« Vraie » vitesse de convergence). Soit $f : x \longmapsto \dfrac{4}{1+x^2}$. Pour tout $n \in \mathbb{N}^*$, notons $S_n(f)$ la somme de Riemann à gauche associée à f sur $[a\,;b]$.

1) Calculer $I = \displaystyle\int_0^1 f(x)\,dx$.
2) Écrire une fonction qui prend en argument un entier naturel non nul n et qui renvoie $S_n(f)$.
3) Soit $\varepsilon > 0$. Déterminer $n \in \mathbb{N}$ tel que $S_n(f)$ soit une approximation de I à ε près.
   On utilisera le fait que f est C^1 puis le fait que f est monotone. Laquelle donne le plus petit n ?
4) Écrire une fonction qui prend en argument ε et qui renvoie le plus petit n tel que $S_n(f)$ soit une approximation de I à ε près (pourquoi le programme s'arrête-t-il au fait ?). Comparer avec les valeurs trouvées à la question

précédente.

On utilisera une approximation de I fournie par l'ordinateur (on sait alors que, si n est une des valeurs obtenues à la question précédente, $S_n(f)$ est une approximation de I à ε près et le but de cette question est de déterminer si l'approximation était déjà bonne pour un plus petit n).

5) a) Justifier que

$$u_n(f) = 2S_{2n}(f) - S_n(f) \xrightarrow[n\to+\infty]{} I \qquad \text{et} \qquad v_n(f) = \frac{8S_{4n}(f) - 6S_{2n}(f) + S_n(f)}{3} \xrightarrow[n\to+\infty]{} I.$$

b) Écrire un programme qui prend en argument ε et qui renvoie le plus petit n tel que $u_n(f)$ soit une approximation de I à ε près. Comparer avec les valeurs trouvées précédemment.

c) Écrire un programme qui prend en argument ε et qui renvoie le plus petit n tel que $v_n(f)$ soit une approximation de I à ε près. Comparer avec les valeurs trouvées. précédemment.

Correction :

1) On a

$$I = [4\operatorname{Arctan}(x)]_0^1 = 4\operatorname{Arctan}(1) - 4\operatorname{Arctan}(0) = \pi.$$

2) Pour tout $n \in \mathbb{N}^*$,

$$S_n(f) = \frac{1}{n}\sum_{k=0}^{n-1} f\left(\frac{k}{n}\right) = \frac{4}{n}\sum_{k=0}^{n-1} \frac{1}{1+(k/n)^2}.$$

D'où la fonction :

```
def Snf(n):
    S=0
    for k in range(n):
        S=S+1/(1+(k/n)**2)
    return S*4/n
```

3) • Si on utilise le fait que f est de classe C^1 sur $[0\,;1]$, cela revient à trouver n tel que

$$\frac{4}{n}\sup_{x\in[0\,;1]}\left|\frac{x}{(1+x^2)^2}\right| = \frac{(1-0)^2}{2n}\sup_{x\in[0\,;1]}\left|-\frac{8x}{(1+x^2)^2}\right| \leqslant \varepsilon.$$

Posons $g : x \longmapsto \dfrac{x}{(1+x^2)^2}$. Il s'agit d'une fonction dérivable sur $[0\,;1]$. Pour tout $x \in [0\,;1]$, $g'(x) = \dfrac{(1+x^2)^2 - x\times 4x(1+x^2)}{(1+x^2)^4} = \dfrac{1-3x^2}{(1+x^2)^3}$. La fonction g est donc maximale lorsque $x = 1/\sqrt{3}$. On a $g\left(\dfrac{1}{\sqrt{3}}\right) = \dfrac{3*\sqrt{3}}{16}$. Ainsi on prend $n = \left\lfloor \dfrac{3\sqrt{3}}{4\varepsilon} \right\rfloor + 1$.

• Si on utilise le fait que f est décroissante sur $[0\,;1]$, cela revient à trouver n tel que

$$\frac{2}{n} = \frac{\big|(1-0)\big(f(1)-f(0)\big)\big|}{n} \leqslant \varepsilon.$$

On prend donc $n = \left\lfloor \dfrac{2}{\varepsilon} \right\rfloor + 1$.

Puisque $2 > \dfrac{3\sqrt{3}}{4}$ (en effet $64 > 27$), la première méthode donne une plus petite valeur de n.

4)

```
import numpy as np
def Verif(eps):
    n=1
    while np.abs(Snf(n)-np.pi)>eps:
          n=n+1
    return n
```

Le programme s'arrête car on sait que $S_n(f) \xrightarrow[n\to+\infty]{} I$ donc, il existe un rang n à partir duquel $S_n(f)$ est proche de I à ε près.

En testant pour $\varepsilon = 10^{-3}$, la fonction renvoie 1001 tandis que $\left\lfloor \frac{3\sqrt{3}}{4\varepsilon} \right\rfloor + 1 = 1300$.

En testant pour $\varepsilon = 10^{-4}$, la fonction renvoie 10001 tandis que $\left\lfloor \frac{3\sqrt{3}}{4\varepsilon} \right\rfloor + 1 = 12991$.

La valeur de n obtenue avec l'approximation est donc du même ordre que la valeur de n réellement nécessaire pour obtenir l'approximation.

5) a) On a $S_{2n} \xrightarrow[n\to+\infty]{} I$ et $S_{4n}(f) \xrightarrow[n\to+\infty]{} I$ donc

$$u_n(f) \xrightarrow[n\to+\infty]{} 2I - I = I \qquad \text{et} \qquad v_n(f) \xrightarrow[n\to+\infty]{} \frac{8I - 6I + I}{3} = I.$$

b)
```
import numpy as np
def Verif2(eps):
    n=1
    while np.abs(2*Snf(2*n)-Snf(n)-np.pi)>eps:
        n=n+1
    return n
```

Cette fois on trouve 10 et 29 pour $\varepsilon = 10^{-3}$ et 10^{-4} respectivement.

c)
```
def Verif3(eps):
    n=1
    while np.abs((8*Snf(4*n)-6*Snf(2*n)+Snf(n))/3-np.pi)>eps:
        n=n+1
    return n
```

Cette fois on trouve 2 pour $\varepsilon = 10^{-3}$ comme pour 10^{-4} respectivement.

Exercice 13. Le but de cet exercice est de démontrer que, en gardant les notations de l'exercice précédent, les convergences de $(u_n(f))_{n\geqslant 1}$ et $(v_n(f))_{n\geqslant 1}$ vers I sont en effet bien plus rapides que celle de $(S_n(f))_{n\geqslant 1}$. On fait pour cela l'hypothèse que f est une fonction quelconque de classe C^4 sur un segment $[a\,;b]$ avec $a < b$. Pour tout $k \in [\![1\,;4]\!]$, on note $M_k = \sup_{[a\,;b]} \left|f^{(k)}\right|$. On se donne $n \in \mathbb{N}^*$ et, pour tout $i \in [\![0\,;n]\!]$, on pose $x_i = a + i\frac{b-a}{n}$.

1) a) Pour tous $i \in [\![0\,;n-1]\!]$ et $t \in [x_i\,;x_{i+1}]$, justifier que, $|f(t) - f(x_i)| \leqslant M_1(t - x_i)$.

b) En déduire que $|I - S_n(f)| \leqslant \frac{(b-a)^2}{2n} M_1$.

2) Soit $[\alpha\,;\beta]$ un segment inclus dans $[a\,;b]$. On introduit :

$$q : x \longmapsto \int_\alpha^x f(t)\,dt - (x-\alpha)f(\alpha) - \frac{(x-\alpha)(f(x)-f(\alpha)}{2}.$$

a) Calculer les deux premières dérivées de p et q.

b) A l'aide d'inégalité de Taylor-Lagrange, montrer que

$$\left|\int_\alpha^\beta f(t)\,dt - (\beta-\alpha)f(\alpha) - \frac{(\beta-\alpha)(f(\beta)-f(\alpha))}{2}\right| \leqslant \frac{(\beta-\alpha)^3}{4} M_2.$$

c) En déduire que

$$\left|I - S_n(f) - \frac{(b-a)(f(b)-f(a))}{2n}\right| \leqslant \frac{(b-a)^3}{4n^2} M_2.$$

3) Soit $[\alpha\,;\beta]$ un segment inclus dans $[a\,;b]$. On introduit de plus :

$$r : x \longmapsto q(x) + \frac{(x-\alpha)^2(f'(x)-f'(\alpha))}{12}.$$

En procédant de la même manière qu'à la question précédente, établir que

$$\left| I - S_n(f) - \frac{(b-a)(f(b)-f(a))}{2n} + \frac{(b-a)^2(f'(b)-f'(a))}{12n^2} \right| \leqslant \frac{(b-a)^5}{72n^4} M_4.$$

4) A l'aide des résultats précédents, déterminer le développement asymptotique en $\dfrac{1}{n}$ à l'ordre 3 de $S_n(f)$ lorsque n tend vers $+\infty$. Cela consiste à trouver des réels A, B, C et D tels que

$$S_n(f) \underset{+\infty}{=} A + \frac{B}{n} + \frac{C}{n^2} + \frac{D}{n^3} + \mathrm{o}\left(\frac{1}{n^3}\right).$$

5) En déduire les développements asymptotiques en $\dfrac{1}{n}$ à l'ordre 3 de $u_n(f)$ et $v_n(f)$ lorsque n tend vers $+\infty$. Conclure.

Correction :

1) a) Soit $i \in [\![0\,;n-1]\!]$. Soit $t \in [x_i\,;x_{i+1}]$. On applique l'inégalité des accroissements finis à la fonction f qui est dérivable sur $]t\,;x_i[$ et dont la dérivée f' est bornée par M_1 : on obtient

$$|f(t) - f(x_i)| \leqslant M_1\,(t - x_i).$$

b) Classique : on transforme I en une somme à l'aide de la relation de Chasles :

$$I = \int_a^b f(t)\,dt = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(t)\,dt.$$

On a aussi

$$S_n(f) = \frac{b-a}{n} \sum_{i=0}^{n-1} f(x_i) = \sum_{i=1}^{n} \int_{x_i}^{x_{i+1}} f(x_i)\,dt.$$

(on intègre une constante sur un segment d'amplitude $\frac{b-a}{n}$). On a donc, par inégalité triangulaire

$$|I - S_n(f)| \leqslant \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} |f(t) - f(x_i)|\,dt.$$

On utilise la question précédente :

$$|I - S_n(f)| \leqslant M_1 \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} (t - x_{i-1})\,dt = M_1 \sum_{i=0}^{n-1} \frac{(x_i - x_{i-1})^2}{2} = M_1 \sum_{i=0}^{n-1} \frac{(b-a)^2}{2n^2}$$

et donc $|I - S_n(f)| \leqslant \dfrac{(b-a)^2}{2n} M_1$.

2) a) Les fonctions p et q sont bien deux fois dérivables puisque f est dérivable. On a

$$q' : x \longmapsto f(x) - f(\alpha) - \frac{f(x) - f(\alpha) + (x-\alpha)f'(x)}{2} = \frac{f(x) - f(\alpha) - (x-\alpha)f'(x)}{2}$$

$$q'' : x \longmapsto \frac{f'(x) - f'(x) - (x-\alpha)f''(x)}{2} = \frac{(x-\alpha)f''(x)}{2}.$$

b) On remarque que

$$\int_\alpha^\beta f(t)\,dt - (\beta-\alpha)f(\alpha) - \frac{(\beta-\alpha)(f(\beta)-f(\alpha)}{2} = q(\beta).$$

On a aussi $q(\alpha) = q'(\alpha) = 0$. L'inégalité de Taylor-Lagrange (il faudrait que q soit de classe C^∞ pour l'appliquer dans le respect du programme mais en fait C^2 suffit) entraîne que

$$|q(\beta)| = |q(\beta) - q(\alpha) - (\beta-\alpha)q'(\alpha)| \leqslant \frac{(\beta-\alpha)^2}{2} \max_{[\alpha,\beta]} |q''|.$$

Pour tout $x \in [\alpha, \beta]$, $|q''(x)| = \left|\dfrac{(x-\alpha)f''(x)}{2}\right| \leqslant \dfrac{\beta-\alpha}{4} M_2$. On en déduit que

$$\left|\int_\alpha^\beta f(t)\,dt - (\beta-\alpha)f(\alpha) - \frac{(\beta-\alpha)(f(\beta)-f(\alpha))}{2}\right| \leqslant \frac{(\beta-\alpha)^3}{4} M_2.$$

c) Pour tout $i \in [\![0\,;n-1]\!]$, appliquons cela avec $\alpha = x_i$ et $\beta = x_{i+1}$:

$$\left|\int_{x_i}^{x_{i+1}} f(t)\,dt - (x_{i+1}-x_i)f(x_i) - \frac{(x_{i+1}-x_i)(f(x_{i+1})-f(x_i))}{2}\right| \leqslant \frac{(x_{i+1}-x_i)^3}{4} M_2$$

donc

$$\left|\int_{x_i}^{x_{i+1}} f(t)\,dt - \frac{b-a}{n}f(x_i) - \frac{(b-a)(f(x_{i+1})-f(x_i))}{2n}\right| \leqslant \frac{(b-a)^3}{4n^3} M_2.$$

On somme et on utilise l'inégalité triangulaire :

$$\left|\sum_{i=0}^{n-1}\int_{x_i}^{x_{i+1}} f(t)\,dt - \frac{b-a}{n}\sum_{i=0}^{n-1} f(x_i) - \frac{(b-a)}{2n}\sum_{i=0}^{n-1}(f(x_{i+1})-f(x_i))\right| \leqslant \sum_{i=0}^{n-1}\frac{(b-a)^3}{4n^3} M_2.$$

On reconnaît une somme télescopique :

$$\left|I - S_n(f) - \frac{(b-a)}{2n}(f(b)-f(a))\right| \leqslant \sum_{i=0}^{n-1}\frac{(b-a)^3}{4n^3} M_2 = \frac{(b-a)^3}{4n^2} M_2.$$

3) Notons $s : x \longmapsto \dfrac{(x-\alpha)^2(f'(x)-f'(\alpha))}{12}$. Les fonctions q, s et $r$sont de classe C^3 car f est de classe C^4. On a

$$r' : x \longmapsto \frac{2(x-\alpha)(f'(x)-f'(\alpha)) + (x-\alpha)^2 f''(x)}{12}$$

$$r'' : x \longmapsto \frac{2(f'(x)-f'(\alpha)) + 4(x-\alpha)f''(x) + (x-\alpha)^2 f^{(3)}(x)}{12}$$

$$r^{(3)} : x \longmapsto \frac{6f''(x) + 6(x-\alpha)f^{(3)}(x) + (x-\alpha)^2 f^{(4)}(x)}{12}.$$

On a aussi

$$q^{(3)} : x \longmapsto \frac{f''(x) + (x-\alpha)f^{(3)}(x)}{2}.$$

Ainsi $r(\alpha) = r'(\alpha) = r''(\alpha) = 0$ et

$$r^{(3)} : x \longmapsto \frac{6f''(x) + 6(x-\alpha)f^{(3)}(x) - 6f''(x) - 6(x-\alpha)f^{(3)}(x) - (x-\alpha)^2 f^{(4)}(x)}{12} = \frac{(x-\alpha)^2 f^{(4)}(x)}{12}.$$

Ainsi $r^{(3)}$ est bornée par $\dfrac{(\beta-\alpha)^2}{12} M_4$. L'inégalité de Taylor-Lagrange (il faudrait que r soit de classe C^∞ pour l'appliquer dans le respect du programme mais en fait C^2 suffit) entraîne que

$$|r(\beta)| = |r(\beta) - r(\alpha) - (\beta-\alpha)r'(\alpha) - \frac{(\beta-\alpha)^2}{2} r''(\alpha)| \leqslant \frac{(\beta-\alpha)^3}{6} \max_{[\alpha,\beta]} |r^{(3)}| \leqslant \frac{(\beta-\alpha)^5}{72} M_4.$$

Pour tout $i \in [\![0\,;n-1]\!]$, on applique cela avec $\alpha = x_i$ et $\beta = x_{i+1}$. On somme, on utilise l'inégalité triangulaire et on conclut que

$$\left|I - S_n(f) - \frac{(b-a)(f(b)-f(a))}{2n} + \frac{(b-a)^2(f'(b)-f'(a))}{12n^2}\right| \leqslant \frac{(b-a)^5}{72n^4} M_4.$$

4) Puisque $\frac{(b-a)^5}{72n^4}M_4 = \mathrm{o}\left(\frac{1}{n^3}\right)$,

$$S_n(f) - I + \frac{(b-a)(f(b)-f(a))}{2n} - \frac{(b-a)^2(f'(b)-f'(a))}{12n^2} = \mathrm{o}\left(\frac{1}{n^3}\right).$$

On a bien

$$S_n(f) \underset{+\infty}{=} A + \frac{B}{n} + \frac{C}{n^2} + \frac{D}{n^3} + \mathrm{o}\left(\frac{1}{n^3}\right)$$

avec $A = I$, $B = -\frac{(b-a)(f(b)-f(a))}{2}$, $C = \frac{(b-a)^2(f'(b)-f'(a))}{12}$ et $D = 0$.

5) On a

$$S_{2n}(f) = A + \frac{B}{2n} + \frac{C}{4n^2} + \mathrm{o}\left(\frac{1}{n^3}\right)$$

donc

$$2S_{2n}(f) = 2A + \frac{B}{n} + \frac{C}{2n^2} + \mathrm{o}\left(\frac{1}{n^3}\right)$$

donc

$$2S_{2n}(f) - S_n(f) = A + \frac{C}{2n^2} + \mathrm{o}\left(\frac{1}{n^3}\right).$$

et donc $u_n(f) - I = \frac{C}{2n^2} + \mathrm{o}\left(\frac{1}{n^3}\right)$. On a aussi

$$8S_{4n}(f) = 8A + \frac{2B}{n} + \frac{C}{2n^2} + \mathrm{o}\left(\frac{1}{n^3}\right)$$

$$6S_{2n}(f) = 6A + \frac{3B}{n} + \frac{3C}{2n^2} + \mathrm{o}\left(\frac{1}{n^3}\right)$$

donc

$$8S_{4n}(f) - 6S_{2n}(f) + S_n(f) = 3A + \mathrm{o}\left(\frac{1}{n^3}\right).$$

et donc $v_n(f) - I = \mathrm{o}\left(\frac{1}{n^3}\right)$. On retrouve bien que $u_n(f)$ et $v_n(f)$ sont des approximations bien plus précises (d'ordre $1/n^2$ et d'ordre $1/n^3$ respectivement) que $S_n(f)$ (d'ordre $1/n$).

Exercice 14 (Implémentation de Φ). Notons $\varphi : t \longmapsto \frac{e^{-t^2/2}}{\sqrt{2\pi}}$ la densité d'une variable aléatoire de loi $\mathcal{N}(0,1)$ et Φ sa fonction de répartition.

1) Justifier que φ est de classe C^1 sur $\mathbb{R}$ et que φ' est bornée par $\frac{1}{\sqrt{2\pi e}}$ sur $\mathbb{R}$.

2) Montrer que, pour tout $x \in \mathbb{R}$, $\Phi(x) = \frac{1}{2} + \int_0^x \varphi(t)\,dt$.

3) Écrire une fonction, appelée `Phi`, qui prend en argument un réel x et qui renvoie une valeur approchée de $\Phi(x)$ à 10^{-3} près à l'aide de la méthode des rectangles.

Correction :

1) La fonction φ est de classe C^1 sur $\mathbb{R}$ par composition de fonctions qui le sont. Pour tout $t \in \mathbb{R}$, $\varphi'(t) = \frac{-te^{-t^2/2}}{\sqrt{2\pi}}$.

La fonction φ' est elle-même dérivable et, pour tout $t \in \mathbb{R}$, $\varphi''(t) = \frac{-e^{-t^2/2} + (-t)^2 e^{-t^2/2}}{\sqrt{2\pi}} = (t^2-1)\frac{e^{-t^2/2}}{\sqrt{2\pi}}$ est du signe de $t^2 - 1$. Ainsi φ' est strictement croissante sur $]-\infty\,;-1]$ et $[1\,;+\infty[$ et strictement décroissant sur $[-1\,;1]$. On a $\sqrt{u}e^{-u/2} \xrightarrow[u\to+\infty]{} 0$ par croissances comparées et $|t| \xrightarrow[t\to\pm\infty]{} +\infty$ donc, par composition, $f'(t) \xrightarrow[t\to\pm\infty]{} 0$. On en déduit que f' admet un max en -1 et un min en 1. Ceux-ci valent respectivement $\frac{e^{-1/2}}{\sqrt{2\pi}} = \frac{1}{\sqrt{2\pi e}}$ et $\frac{-e^{-1/2}}{\sqrt{2\pi}}$. Par conséquent φ' est bien bornée par $\frac{1}{\sqrt{2\pi e}}$.

2) Pour tout $x \in \mathbb{R}$, $\Phi(x) = \displaystyle\int_{-\infty}^{0} \varphi(t)\,dt + \int_{0}^{x} \varphi(t)\,dt$. Comme φ est paire sur $\mathbb{R}$, on a

$$\int_{-\infty}^{0} \varphi(t)\,dt = \frac{1}{2}\int_{-\infty}^{+\infty} \varphi(t)\,dt = \frac{1}{2} \times 1 = \frac{1}{2}.$$

D'où l'égalité.

3) Vu la question 1 et l'approximation dans le cas des fonctions C^1, pour tout $x \in \mathbb{R}$, on approche $\Phi(x)$ par $\dfrac{1}{2} + S_n(x)$ avec n tel que

$$\frac{(x-0)^2}{2n} \max_{[0\,;x]} |\varphi'| \leqslant 10^{-3}.$$

Vu la question 2, il suffit de prendre n tel que

$$\frac{x^2}{2n}\frac{1}{\sqrt{2\pi e}} \leqslant 10^{-3}.$$

c'est-à-dire

$$n = \left\lfloor \frac{500x^2}{\sqrt{2\pi e}} \right\rfloor + 1$$

D'où le programme en Python :

```
import numpy as np
import matplotlib.pyplot as plt
def Phi(x):
    def phi(t):
        return np.exp(-t**2/2)/np.sqrt(2*np.pi)
    n=int(500*x**2/np.sqrt(2*np.pi*np.e))+1
    S=0
    for k in range(1,n+1):
        S=S+phi(x*k/n)
    S=1/2+x*S/n
    return S
```

On notera que cela fait beaucoup d'itérations lorsque x devient grand (par exemple, si $x = 10$, cela fait déjà 12099 itérations. Néanmoins, lorsque l'on connait bien la fonction Φ, on sait bien qu'elle est quasiment nulle sur $\mathbb{R}\setminus[-5\,;5]$ donc ce n'est pas bien grave : on n'utilisera jamais cette fonction pour de grandes valeurs de n... du moins à cette précision ci.

IV Matrices

Exercice 15. Écrire une fonction en Python qui prend en argument une matrice A, un vecteur colonne B ayant autant de ligne que A a de colonnes, deux entiers naturels i, j (indices possibles pour les lignes de A) et deux réels α et β et qui fait l'opération élémentaire $L_i \leftarrow \alpha L_i + \beta L_j$ sur les matrices A et B.
Cette fonction ne renverra rien si $\alpha \neq 0$ mais renverra le message « le pivot est nul » si α est nul.

Correction :

```
def Operation(A,B,i,j,alpha,beta):
    if alpha==0:
        print('Le pivot est nul.')
    else:
        A[i,:]=alpha*A[i,:]+beta*A[j,:]
        B[i]=alpha*B[i]+beta*B[j]
```

Exercice 16 (Bon courage sans Python). Résoudre le système suivant, d'inconnues x, y, z réelles, à l'aide

de la méthode du pivot de Gauss implémentée avec Python.

$$\begin{cases} -45x & - & 11y & + & 22z & = & -1 \\ -32x & - & 16y & + & 20z & = & -4 \\ -13x & + & 24y & - & 10z & = & 5 \end{cases}$$

On utilisera l'exercice précédent et on vérifiera ensuite avec la commande `al.solve`.

Correction : On se contente de faire exactement les opérations que l'on ferait à la main pour mettre le système sous forme triangulaire puis sous forme diagonale puis résoudre finalement en divisant par les termes diagonaux respectifs. On utilise bien sûr l'exercice précédent.

```
import numpy as np
import numpy.linalg as al
#On implémente A et B
A=np.array([[-45,-11,22],[-32,-16,20],[-13,24,-10]])
B=np.array([[-1],[-4],[5]])
#On résout avec Python :
al.solve(A,B)
#On résout avec un pivot de Gauss :
    #D'abord on le met sous forme triangulaire
Operation(A,B,1,0,A[0,0],-A[1,0])
Operation(A,B,2,0,A[0,0],-A[2,0])
Operation(A,B,2,1,A[1,1],-A[2,1])
    #Le système a-t-il une solution ? Oui ssi les 3 coeffs
     diagonaux sont non nuls
print(A[0,0]*A[1,1]*A[2,2]==0)
    #Puis sous forme diagonale :
Operation(A,B,1,2,A[2,2],-A[1,2])
Operation(A,B,0,2,A[2,2],-A[0,2])
Operation(A,B,0,1,A[1,1],-A[0,1])
    #On termine en divisant par les coefficients diagonaux
     respectifs :
B[0]=B[0]/A[0,0]
B[1]=B[1]/A[1,1]
B[2]=B[2]/A[2,2]
#On affiche B (qui a été modifié en la solution du système) :
print(B)
```

On trouve que $(1,2,3)$ est l'unique solution.

Exercice 17 (Codage matriciel d'un endomorphisme défini à l'aide de polynômes).

1) Soit $n \in \mathbb{N}$. En Python, on représentera un polynôme P de degré inférieur ou égal à n par le vecteur ligne (de type `numpy` et non une liste) égal aux coordonnées de P dans la base canonique de $\mathbb{R}_n[X]$.

   Par exemple :

   — *Si* $n=3$, `P=np.array([0,1,2,-1])` *implémente le polynôme* $P = X + 2X^2 - X^3$.

   — *Si* $n=4$, `P=np.array([1,0,0,-1,0])` *implémente le polynôme* $P = 1 - X^3$.

   — *Si* $n=5$, *alors on implémente* $P = 7 - X + \sqrt{2}X^3 - 9X^4$ *par* `P=np.array([7,-1,0,2**(1/2),-9,0])`.

   Ainsi si P et Q sont deux polynômes de $\mathbb{R}_n[X]$ implémentés par `P` et `Q` respectivement et si a est un réel implémenté par `a`, alors `a*P+Q` implémente $aP+Q$. C'est moins immédiat pour les autres opérations (ne serait-ce que par défaut de stabilité) et c'est que nous allons voir.

   a) Recopier et compléter la fonction ci-dessous afin qu'elle prenne en argument un polynôme implémenté en liste ou vecteur ligne et renvoie sa dérivée sous forme de vecteur ligne.

```
import ......................
def Deriv(P):
    n=..........
    Q=np.zeros(n)
    for i in range(n-1):
        Q[i]=.............
    return Q
```

b) Recopier et compléter la fonction ci-dessous afin qu'elle prenne en argument un entier naturel non nul k et un polynôme P implémenté en vecteur ligne et renvoie X^kP implémenté en Python (par un vecteur ayant k coordonnée de plus, n'est-ce pas ?).

```
def MultX(P,k):
    n=.............
    Q=np.zeros(n+k)#On travaille dans R_{n+k}[X]
    for i in range(....,....):
        Q[i]=.............
    return Q
```

c) Écrire une fonction en Python qui prend en argument deux entiers naturels n et k avec $k \leqslant n$ et qui implémente le polynôme X^k de $\mathbb{R}_n[X]$ en vecteur ligne.

2) Soit $n \in \mathbb{N}^*$. On considère l'application $f_n : P \in \mathbb{R}_n[X] \longmapsto nXP + (1-X^2)P'$.

a) Montrer que f_n est un endomorphisme de $\mathbb{R}_n[X]$.

b) A l'aide des fonctions Python précédentes, construire une fonction en Python qui prend en argument un entier naturel non nul n et un polynôme P de $\mathbb{R}_n[X]$ implémenté en vecteur ligne et renvoie $f_n(P)$.
Attention avant de sommer les trois polynômes de $f_n(P)$ car les fonctions définies précédemment vont fournir des polynômes de $\mathbb{R}_{n+1}[X]$, $\mathbb{R}_n[X]$ et $\mathbb{R}_{n+2}[X]$ respectivement (il faudra donc enlever les coordonnées superflues).

c) Recopier et compléter la fonction ci-dessous afin qu'elle prenne en argument un entier naturel non nul n et qu'elle renvoie la matrice de f_n dans la base canonique de $\mathbb{R}_n[X]$.

```
def Mat_f(n):
    A=.....................#Matrice carrée nulle d'ordre n+1
    #On va construire la transposée de la matrice
    for k in range(n+1):
        A[k,:]=.............
    return ................#La transposée de A
```

d) Pour $n \in [\![1\,;10]\!]$, déterminer les valeurs propres et vecteurs propres de f_n à l'aide de Python et de la commande `al.eig`. Que peut-on conjecturer quant à la diagonalisabilité de f ?

e) Montrer que, pour tous $n \in \mathbb{N}$ et $k \in [\![0\,;n]\!]$, $(X-1)^k(X+1)^{n-k}$ est un vecteur propre de f_n. Prouver alors la conjecture de la question précédente.

Correction :

1) a) Si $P = \sum_{i=0}^{n} a_i X^i$, alors $P' = \sum_{i=0}^{n-1} (i+1)a_{i+1}X^i$.

```
import numpy as np
def Deriv(P):
    n=len(P)
    Q=np.zeros(n)
    for i in range(n-1):
        Q[i]=(i+1)*P[i+1]
    return Q
```

b) Si $P=\sum_{i=0}^{n} a_i X^i$, alors $X^k P=\sum_{i=0}^{n} a_i X^{i+k}=\sum_{i=k}^{n+k} a_{i-k} X^i$.

```
def MultX(P,k):
    n=len(P)
    Q=np.zeros(n+k)#On travaille dans R_{n+k}[X]
    for i in range(k,n+k):
        Q[i]=P[i-k]
    return Q
```

c)

```
def Monome(n,k):
    P=np.zeros(n+1)
    P[k]=1
    return P
```

2) a) Soient $P\in\mathbb{R}_n[X]$, $Q\in\mathbb{R}_n[X]$ et $\alpha\in\mathbb{R}$. On a

$$\begin{aligned} f_n(\alpha P+Q) &= nX(\alpha P+Q)+(1-X^2)(\alpha P+Q)'' \\ &= \alpha(nXP+(1-X^2)P'')+(nXQ+(1-X^2)Q'')=\alpha f_n(P)+f_n(Q). \end{aligned}$$

Ainsi f_n est linéaire. Ensuite :

- Si $P\in\mathbb{R}_{n-1}[X]$, $\deg(nXP)\leqslant 1+n-1=n$, $\deg((1-X^2)P')\leqslant 2+n-2=n$ donc $\deg(f_n(P))\leqslant n$.
- Si P est de degré n et de coefficient dominant λ, $\deg(nXP)=n+1$, $\deg((1-X^2)P')=n+1$ mais le coefficient dominant de nXP est $n\lambda$, tandis que celui de $(1-X^2)P'$ est $-n\lambda$. Ainsi $\deg(f_n(P))\leqslant n$.

Ainsi f_n est un endomorphisme de $\mathbb{R}_n[X]$.

b)

```
def f(n,P):
    if len(P)!=n+1:
        print('erreur')
    else:
        Q1=n*MultX(P,1)
        Q2=Deriv(P)
        Q3=-MultX(Deriv(P),2)
        return Q1[:n+1]+Q2[:n+1]+Q3[:n+1]
```

c)

```
def Mat_f(n):
    A=np.zeros([n+1,n+1])
    for k in range(n+1):
        A[k,:]=f(n,Monome(n,k))
    return np.transpose(A)#La transposée de A
```

d)

```
import numpy.linalg as al
for n in range(1,8):
    print(al.eig(Mat_f(n))[0])
#Le 0 sert à n'avoir que les valeurs propres (sinon on aussi
    des vecteurs propres associés).
```

On conjecture que, pour tout $n\in\mathbb{N}^*$, les valeurs propres de f_n sont les $n-2k$ avec $k\in[\![0\,;n]\!]$. Puisqu'il y en a $n+1$ et qu'elles sont distinctes, f_n est donc diagonalisable.

e) Soient $n\in\mathbb{N}$ et $k\in[\![0\,;n]\!]$. On a

$$\begin{aligned} \big((X-1)^k(X+1)^{n-k}\big)' &= k(X-1)^{k-1}(X+1)^{n-k}+(n-k)(X-1)^k(X+1)^{n-k-1} \\ &= \big(k(X+1)+(n-k)(X-1)\big)(X-1)^{k-1}(X+1)^{n-k-1} \\ &= \big(nX-n+2k\big)(X-1)^{k-1}(X+1)^{n-k-1}. \end{aligned}$$

Comme $1-X^2=-(X-1)(X+1)$, on obtient

$$\begin{aligned} f_n\big((X-1)^k(X+1)^{n-k}\big) &= nX(X-1)^k(X+1)^{n-k}+(1-X^2)\big(nX-n+2k\big)(X-1)^{k-1}(X+1)^{n-k-1} \\ &= nX(X-1)^k(X+1)^{n-k}-\big(nX-n+2k\big)(X-1)^k(X+1)^{n-k} \\ &= (n-2k)(X-1)^k(X+1)^{n-k}. \end{aligned}$$

Cela montre bien que $(X-1)^k(X+1)^{n-k}$ est un vecteur propre associé à la valeur propre $n-2k$.

V Probabilités

Exercice 18. Une urne contient 1 boule jaune, 2 boules rouges et 4 boules bleues.

1) On effectue le tirage d'une boule dans l'urne. Écrire un programme en Python qui simule cette expérience et qui affiche à l'écran la couleur de la boule tirée.

   Pour simplifier la simulation, on pourra supposer que la boule jaune est numérotée 1, que les boules rouges sont numérotées 2 et 3 et que les boules bleues sont numérotées 4,5,6 et 7.

2) Soit n un entier naturel non nul. On effectue n tirages d'une boule dans l'urne, avec remise de la boule tirée avant chaque tirage. On note X la variable aléatoire égale au nombre de boules rouges obtenues au cours de cette expérience.

   a) Écrire une fonction qui prend en entrée n, qui simule l'expérience aléatoire décrite et qu'elle renvoie une réalisation de X.

   Dans un premier temps, on pourra utiliser une boucle `for`. *Puis on pourra simplifier en utilisant la commande* `np.sum(A<x)` *, où* `A` *est un vecteur et* `x` *un réel. En effet, lorsqu'on effectue des opérations algébriques sur* `True` *et* `False`, *Python les assimile respectivement à 1 et 0.*

   b) Déterminer la loi de X. Préciser son espérance et sa variance.

3) On considère désormais l'expérience suivante : on tire une boule dans l'urne avec remise jusqu'à obtenir une boule jaune. On note N la variable aléatoire égale au nombre de tirages effectuée et Y la variable aléatoire égale au nombre de boules bleues obtenues au cours de cette expérience.

   a) Compléter la fonction Python suivante afin qu'elle simule cette expérience et qu'elle renvoie des réalisations de N et Y.

```
1  import .....................
2  def simul():
3      T=rd.random()
4      N=1; Y=0
5      if T>3/7:
6          .....................
7      while ..................
8              T=rd.random()
9              ..................
10             if T>3/7:
11                 ..............
12     return N,Y
```

   b) Déterminer la loi de N. Préciser son espérance et sa variance.

   c) Déterminer la loi du couple (N, Y).

   d) Calculer la probabilité $\mathbb{P}(Y = 0)$.

4) Expliquer ce que fait le script Python ci-dessous. Plus précisément que contient la variable `S` ?

```
import numpy as np
import matplotlib.pyplot as plt
Nexp=10000
L=[]
for k in range(Nexp):
    [N,Y]=simul()
    L.append(Y)
M=np.cumsum(L)/np.array(range(1,Nexp+1))
plt.plot(range(1,Nexp+1),M)
plt.show()
```

Éxécuter-le plusieurs fois. Que peut-on conjecturer ? Sur quel résultat théorique s'appuie-t-on pour faire cette conjecture ?

5) Montrer la conjecture à l'aide de la formule de transfert.

Correction :

1)
```
import numpy.random as rd
N=rd.randint(1,8)
if N==1:
   return 'Jaune'
elif N<=3:
   return 'Rouge'
else:
   return 'Bleue'
```

2) a)
```
import numpy.random as rd
def SimulX(n):
    X=0
    for k in range(n):
        N=rd.randint(1,8)
        if 2<=N<=3:
            X=X+1
    return N
```

Version alternative :
```
import numpy as np
import numpy.random as rd
def SimulX(n):
    U=rd.randint(1,8,n)
    return np.sum((2<=U)*(U<=3))
```

b) La variable aléatoire X compte le nombre de succès (obtenir rouge) dans n réalisations indépendantes d'une épreuve de Bernoulli (obtenir rouge ou non). Ainsi X suit une loi $\mathcal{B}(n, 2/7)$. Son espérance est $\frac{2n}{7}$ et sa variance $\frac{2n}{7}\left(1-\frac{2}{7}\right) = \frac{10n}{49}$.

3) a)
```
import numpy.random as rd
def simul():
    T=rd.random()
    N=1; Y=0
    if T>3/7:
       Y=Y+1
    while T>1/7:
          T=rd.random()
          N=N+1
          if T>3/7:
             Y=Y+1
    return N,Y
```

Ici `T>3/7` renvoie True avec la probabilité $4/7$, ce qui est la probabilité d'obtenir une boule bleue.

b) La variable aléatoire N compte le nombre d'expériences indépendantes nécessaires pour obtenir le premier succès (tirer une boule jaune). Ainsi N suit une loi géométrique de paramètre $\frac{1}{1+2+4} = \frac{1}{7}$. On a $\mathbb{E}(N) = 7$ et $\mathbb{V}(N) = \frac{1-1/7}{(1/7)^2} = 42$.

c) On a $N(\Omega) = \mathbb{N}^*$ et $Y(\Omega) = \mathbb{N}$. Soit $(n, y) \in \mathbb{N}^* \times \mathbb{N}$. Déjà le nombre de boules bleues tirés est forcément inférieur strictement au nombre de tirages de l'expérience. Ainsi, si $y \geqslant n$, alors $\mathbb{P}([N = n] \cap [Y = y]) = 0$. Supposons que $y < n$. L'événement $[N = n] \cap [Y = y]$ est réalisé si on a effectué n tirages avec 1 boule jaune (au dernier), y boules bleues et $n - 1 - y$ doubles rouges. Il y a $\binom{n-1}{y}$ façons de placer les tirages où l'on a obtenu du bleu. Par incompatibilité et indépendance, on a donc

$$\mathbb{P}([N = n] \cap [Y = y]) = \binom{n-1}{y} \left(\frac{4}{7}\right)^y \left(\frac{2}{7}\right)^{n-1-y} \left(\frac{1}{7}\right).$$

d) La formule des probabilités totales appliquée au s.c.e associé à N entraîne que

$$\mathbb{P}(Y = 0) = \sum_{n=1}^{+\infty} \mathbb{P}([N = n] \cap [Y = 0]) = \sum_{n=1}^{+\infty} \binom{n-1}{0} \left(\frac{4}{7}\right)^0 \left(\frac{2}{7}\right)^{n-1} \left(\frac{1}{7}\right) = \frac{1}{7} \sum_{j=0}^{+\infty} \left(\frac{2}{7}\right)^j = \frac{1}{7} \frac{1}{1-2/7} = \frac{1}{5}.$$

4) Le script réalise 10000 fois l'expérience et stocke dans la liste L les 10000 réalisations de Y obtenues. La commande `np.cumsum(L)` contient le vecteur des sommes cumulées : si $y_1, \ldots, y_{1000}$ sont les 100000 réalisations obtenues, il s'agit du vecteur contenant

$$y_1, y_1 + y_2, y_1 + y_2 + y_3, y_1 + y_2 + y_3 + y_4, \ldots, y_1 + y_2 + \cdots + y_{10000}$$

Ainsi M contient

$$\frac{y_1}{2}, \frac{y_1 + y_2}{2}, \frac{y_1 + y_2 + y_3}{3}, \frac{y_1 + y_2 + y_3 + y_4}{4}, \ldots, \frac{y_1 + y_2 + \cdots + y_{10000}}{10000}.$$

On trace donc l'évolution de cette quantité qui, selon la loi faible des grands nombres, doit converger vers l'espérance de Y. On exécutant le script on conjecture une convergence vers 4. Ainsi, on conjecture que $\mathbb{E}(Y) = 4$.

5) Sous réserve d'existence, on a

$$\mathbb{E}(Y) = \sum_{n=1}^{+\infty} \sum_{y=0}^{+\infty} y\mathbb{P}([N = n] \cap [Y = y]) = \sum_{n=1}^{+\infty} \sum_{y=0}^{n-1} y \binom{n-1}{y} \left(\frac{4}{7}\right)^y \left(\frac{2}{7}\right)^{n-1-y} \left(\frac{1}{7}\right).$$

La formule du chef entraîne que

$$\mathbb{E}(Y) = \frac{1}{7} \sum_{n=1}^{+\infty} \sum_{y=1}^{n-1} (n-1) \binom{n-2}{y-1} \left(\frac{4}{7}\right)^y \left(\frac{2}{7}\right)^{n-1-y}$$

(le $0^{\text{ième}}$ terme étant nul). Le changement d'indice $j = y - 1$ entraîne que

$$\mathbb{E}(Y) = \frac{1}{7} \sum_{n=1}^{+\infty} (n-1) \sum_{y=1}^{n-2} \binom{n-2}{j} \left(\frac{4}{7}\right)^{j+1} \left(\frac{2}{7}\right)^{n-2-j} = \frac{4}{49} \sum_{n=1}^{+\infty} (n-1) \sum_{y=1}^{n-2} \binom{n-2}{j} \left(\frac{4}{7}\right)^j \left(\frac{2}{7}\right)^{n-2-j}.$$

La formule du binôme de Newton entraîne que

$$\mathbb{E}(Y) = \frac{4}{49} \sum_{n=1}^{+\infty} (n-1) \left(\frac{4}{7} + \frac{2}{7}\right)^{n-2} = \frac{4}{49} \sum_{n=1}^{+\infty} (n-1) \left(\frac{6}{7}\right)^{n-2}.$$

Faisons le changement d'indice $k = n - 1$:

$$\mathbb{E}(Y) = \frac{4}{49} \sum_{k=0}^{+\infty} k \left(\frac{6}{7}\right)^{k-1} = \frac{4}{49} \times \frac{1}{(1-6/7)^2} = \frac{4}{49} \times \frac{49}{1} = 4.$$

A posteriori la somme double convergeait bien absolument puisque tout est positif et qu'il s'agissait de sommes finies et sommes géométriques convergentes.

Exercice 19 (D'après ECRICOME 2019).

Une urne contient initialement une boule blanche et une boule noire. On effectue une succession de tirages d'une boule dans cette urne. Après chaque tirage, on remet la boule tirée dans l'urne, et on rajoute dans l'urne une boule de couleur opposée à celle qui vient d'être tirée. On suppose que cette expérience est modélisée par un espace probabilisé $(\Omega, \mathcal{A}, \mathbb{P})$. Pour tout $k \in \mathbb{N}$, on note X_k le nombre de boules blanches présentes dans l'urne juste avant le $(k+1)^{\text{ième}}$ tirage. En particulier, on a $X_0 = 1$. On admet que pour tout entier k, X_k est une variable aléatoire de $(\Omega, \mathcal{A}, \mathbb{P})$.

1) Soit $k \in \mathbb{N}$.

   a) Déterminer $X_k(\Omega)$.

   b) Pour tous $i \in X_{k+1}(\Omega)$ et $j \in X_k(\Omega)$, déterminer $\mathbb{P}_{[X_k=j]}(X_{k+1} = i)$.
   On distinguera différents cas selon les valeurs relatives de i et j.

   c) En déduire que

$$\forall i \in [\![2\,;k+1]\!], \qquad \mathbb{P}(X_{k+1} = i) = \frac{i}{k+2}\mathbb{P}(X_k = i) + \frac{3+k-i}{k+2}\mathbb{P}(X_k = i-1).$$

   d) Montrer que $\mathbb{P}(X_k = 1) = \dfrac{1}{(k+1)!}$.

   e) Déterminer la valeur de $\mathbb{P}(X_k = k+1)$.

2) Que renvoie la fonction Python suivante pour un entier k non nul implémenté en Python par `k` ?

```
import numpy.random as rd
def mystere(k):
    n=1
    b=1
    for i in range(k):
        r=rd.randint(1,n+b+1)
        if r>n:
            n=n+1
        else:
            b=b+1
    return b
```

3) Écrire une fonction Python d'en-tête appelée `loi_exp` qui prend en argument deux entiers strictement positifs k et N, qui effectue N simulations de k tirages successifs dans l'urne et qui retourne un vecteur `LE` qui contient une estimation de la loi de X_k (c'est-à-dire que pour chaque $i \in [\![1\,;k+1]\!]$, `LE[`$i-1$`]` contient la fréquence d'apparition de l'événement $[X_k = i]$ au cours des N simulations). On justifiera cette approche en citant un théorème du cours.

   On pourra utiliser la fonction `mystere`.

4) Recopier et compléter la fonction suivante afin qu'elle prenne en entrée un entier strictement positif n et qu'elle retourne un vecteur qui contient la loi théorème de X_n.

```
import numpy as np
def loi_theo(n):
    M=np.zeros([n+1,n+1])
    M[0,0]=1
    M[1,0]=1/2
    M[1,1]=1/2
    for k in range(1,n):
        M[k+1,0]=.....
        for i in range(2,k+2):
            M[k+1,i-1]=.....
        M[k+1,k+1]=.....
    return .....
```

5) Un étudiant nous propose comme loi de X_5 le résultat suivant :

k	1	2	3	4	5	6
$\mathbb{P}(X_5 = k)$	0.001368	0.079365	0.419434	0.418999	0.079454	0.00138

A-t-il utilisé `loi_exp` ou bien `loi_theo` ?

6) Pour différentes valeurs de n, superposer le diagramme à barres de la loi théorique et le digramme à barres de $N = 10000$ réalisations de l'expérience.

Si `X` *et* `Y` *sont des vecteurs de même taille, la commande* `plt.bar(X,Y)` *construit un diagramme à barre dont les abscisses des barres sont les coordonnées de* `X` *et la hauteur des barres les coordonnées de* `Y`*. On peut ajouter une option de couleur comme pour* `plt.plot(X,Y)` *et une option pour changer la largeur des barres (essayez* `width=0.4` *pour l'un des deux diagramme).*

Correction :

1) a) Au minimum X_k vaut 1, si on ne tire que la boule blanche k fois de suite (on a jamais rajouté de boule blanche). Au maximum X_k vaut $k+1$ si on ne tire que la boule noire k fois de suite (on a ajouté k boules noires supplémentaires à celle déjà présente). Les valeurs intermédiaires sont atteintes (pour tout $i \in [\![1\,;k+1]\!]$ X_k prend la valeur i si, par exemple, on a tiré $i-1$ fois la boule noire lors des k premiers tirages) donc $X_k(\Omega) = [\![1\,;k+1]\!]$.

b) Soient $i \in \mathbb{N}^*$ et $j \in X_k(\Omega)$. Déjà, d'un tirage à l'autre, le nombre de boules blanches ne peut augmenter que de 1 au plus. Ainsi, si $j \notin \{i; i-1\}$, $\mathbb{P}_{[X_k=j]}(X_{k+1} = i) = 0$. Ensuite :

- Si $j = i$, sachant que $[X_k = j]$, X_{k+1} vaut i si et seulement si on a pioché une boule blanche au $(k+1)^{\text{ième}}$ tirage. A ce tirage, il y a $k+2$ boules au total (les 2 du départ et les k que l'on a rajoutées) et i boules blanches donc $\mathbb{P}_{[X_k=j]}(X_{k+1} = i) = \dfrac{i}{k+2}$.
- Si $j = i-1$, sachant que $[X_k = j]$, X_{k+1} vaut i si et seulement si on a pioché une boule noire au $(k+1)^{\text{ième}}$ tirage. A ce tirage, il y a $k+2$ boules au total (les 2 du départ et les k que l'on a rajoutées) et $k+2-(i-1) = 3+k-i$ boules blanches donc $\mathbb{P}_{[X_k=j]}(X_{k+1} = i) = \dfrac{3+k-i}{k+2}$.

c) Soit $i \in [\![2\,;k+1]\!]$. On applique la formule des probabilités totales avec le système complet d'événements associé à X_k :

$$\mathbb{P}(X_{k+1} = i) = \sum_{j=1}^{k+1} \mathbb{P}_{[X_k=j]}(X_{k+1} = i)\mathbb{P}(X_k = j).$$

Dans cette somme, tous les termes sont nuls sauf ceux d'indices $j = i$ et $j = i-1$ compte tenu de la question précédente. On a donc

$$\begin{aligned}\mathbb{P}(X_{k+1} = i) &= \mathbb{P}_{[X_k=i}(X_{k+1} = i)\mathbb{P}(X_k = i) + \mathbb{P}_{[X_k=i-1]}(X_{k+1} = i)\mathbb{P}(X_k = i-1) \\ &= \frac{i}{k+2}\mathbb{P}(X_k = i) + \frac{3+k-i}{k+1}\mathbb{P}(X_k = i-1).\end{aligned}$$

d) L'événement $[X_k = 1]$ est réalisé si, lors des k premiers tirages, on a tiré que des boucles blanches. Pour tout $j \in \mathbb{N}^*$, notons B_j l'événement « tirer une blanche au $j^{\text{ième}}$ tirage ». On a donc $[X_k = 1] = \bigcap_{j=1}^{k} B_k$.

La formule des probabilités composées entraîne alors que

$$\mathbb{P}(X_k = 1) = \mathbb{P}(B_1)\mathbb{P}_{B_1}(B_2)\mathbb{P}(B_1 \cap B_2)(B_3)\cdots\mathbb{P}(B_1 \cap B_2 \cap \cdots \cap B_{k-1})(B_k) = \frac{1}{2}\times\frac{1}{3}\times\frac{1}{4}\times\cdots\times\frac{1}{k+1}$$

(en effet, pour tout $j \in \mathbb{N}^*$, sachant que l'on a pioché que des blanches avant le $j^{\text{ième}}$ tirage, il y a $2+j-1 = j+1$ boules dont une seule blanche dans l'urne). On a bien $\mathbb{P}(X_k = 1) = \dfrac{1}{(k+1)!}$.

e) L'événement $[X_k = k+1]$ est réalisé si, lors des k premiers tirages, on a tiré que des boucles noires. On a donc $[X_k = k+1] = \bigcap_{j=1}^{k} \overline{B_k}$. Le même calcul donne que $\mathbb{P}(X_k = k+1) = \dfrac{1}{(k+1)!}$.

2) Initialement n et b valent 1. Il s'agit du nombre de boules noires et blanches respectivement à l'origine. Ensuite il y a une boucle à k itérations représentant les k premières expériences successives. A l'étape i de la boucle :
   - On tire un nombre r entre 1 et $n+b$ ($n+b$ est le nombre total de boules).
   - Si $r > n$ (ce qui arrive avec probabilité $\dfrac{b}{n+b}$, la probabilité de tirer une blanche), alors on incrémente n pour signifier que l'on a tiré une blanche et donc mis une nouvelle noire dans l'urne.
   - Sinon c'est le contraire : on incrémente b car on a tiré une noire.

   Enfin on renvoie la valeur de b donc le nombre de boules blanches au bout des k tirages. Cette fonction simule donc X_k.

3) La loi des grands nombres nous garantit que, lorsqu'on répète un grand nombre de fois l'expérience indépendamment, la proportion de fois où $[X_k = i]$ est réalisé est une approximation de $\mathbb{P}([X_k = i])$.

```
def loi_exp(k,N):
    LE=np.zeros(k+1)
    for i in range(N):
        b=mystere(k)
        LE[b-1]+=1
    return LE/N
```

4)

```
import numpy as np
def loi_theo(n):
    M=np.zeros([n+1,n+1])
    M[0,0]=1
    M[1,0]=1/2
    M[1,1]=1/2
    for k in range(1,n):
        M[k+1,0]=1/np.prod([i for i in range(1,k+3)])
        for i in range(2,k+2):
            M[k+1,i-1]=(i*M[k,i-1]+(3+k-i)*M[k,i-2])/(k+2)
        M[k+1,k+1]=M[k+1,0]
    return M[n,:]
```

5) L'absence de symétrie dans les valeurs extrêmes obtenues suggère qu'il a obtenu ces résultats avec les simulations (avec `loi_exp`).

Exercice 20 (D'après les oraux ESCP 1999 et 2001). Une urne contient n boules numérotées de 1 à n. On prélève au hasard ces n boules une par une et sans remise. A la suite de l'expérience, on note, pour tout $i \in [\![1\,;n]\!]$, u_i le numéro de la boule obtenue au cours du $i^{\text{ième}}$ tirage. Pour tout $i \in [\![2\,;n]\!]$, on dit qu'il y a un record en i si l'on a $u_i > \max\{u_1, \dots, u_{i-1}\}$. D'autre part, on convient qu'il y a systématiquement un record à l'instant 1.

1) Calculer, pour tout $i \in [\![1\,;n]\!]$, la probabilité r_i qu'il y ait un record à l'instant i.
   On pourra commencer par calculer, pour tout $k \in [\![i\,;n]\!]$, la probabilité qu'il y ait un record au $i^{\text{ième}}$ tirage en tirant la boule numérotée k lors de ce tirage. On pourra utiliser, en la redémontrant, la formule de Pascal généralisée :
   $$\forall n \in \mathbb{N}^*, \qquad \forall p \in [\![1\,;n]\!], \qquad \sum_{j=p}^{n} \binom{j-1}{p-1} = \binom{n}{p}.$$

2) Calculer les probabilités que, durant la totalité des tirages, on assiste exactement à :
   a) un seul record.
   b) n records.
   c) deux records.

3) On considère la fonction Python ci-dessous :

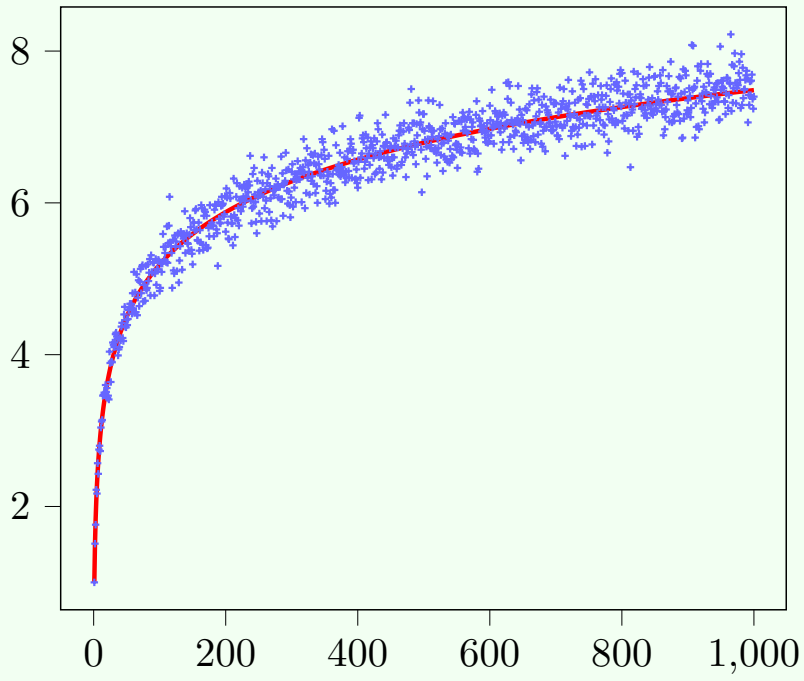
```python
import numpy.random as rd
def Tirages(n):
    L=list(range(n))
    T=[]
    for k in range(n):
        i=rd.randint(n-k)
        x=L.pop(i)#Renvoie et enlève L[i]
        T.append(x)
    S=1
    m=T[0]
    for k in range(1,n):
        if T[k]>m:
            m=T[k]
            S=S+1
    return T,S
```

Expliquer ce que représentent les variables `T,m,S` ? Après exécution de cette fonction pour un n quelconque, qu'est-il affiché à l'écran ?

4) Après exécution de la fonction suivante,

```python
import matplotlib.pyplot as plt
import numpy as np
N=100
n=1000
E=[1]
for k in range(2,n+1):
    S=0
    for i in range(N):
        S=S+Tirages(k)[1]
    E.append(S/N)
plt.plot(range(1,n+1),E,'+')
H=np.cumsum([1/i for i in range(1,n+1)])
plt.plot(range(1,n+1),H)
plt.show()
```

on obtient le graphique suivant :



Interpréter le graphique puis justifier vos observations.

On pourra introduire, pour tout $i \in [\![2\,;n]\!]$, la variable aléatoire X_i qui prend la valeur 1 s'il y a un record à l'instant i et 0 sinon.

Correction :

1) Pour tout $i \in [\![1\,;n]\!]$, notons R_i l'événement « obtenir un record à l'instant i ». Pour tout $k \in [\![1\,;n]\!]$, notons $B_{i,k}$ l'événement « tirer la boule k au $i^{\text{ième}}$ tirage ».

- L'événement $R_i \cap B_{i,k}$ est réalisé si et seulement si les $i-1$ premières boules sont inférieures ou égale à $k-1$ et si la $i^{\text{ième}}$ boule vaut k. Il y a $(k-1)\times(k-2)\times\cdots(k-1-(i-1)+1)\times 1 = \dfrac{(k-1)!}{((k-1)-(i-1))!} = (i-1)!\binom{k-1}{i-1}$ tirages sans remise de k boules menant à cette configuration et $n\times(n-1)\times\cdots\times(n-i+1) = \dfrac{n!}{(n-i)!} = i!\binom{n}{i}$ tirages successifs de i boules sans remise. Ainsi, par équiprobabilité des tirages,

$$\mathbb{P}(R_i \cap B_{i,k}) = \frac{(i-1)!\binom{k-1}{i-1}}{i!\binom{n}{i}} = \frac{1}{i} \times \frac{\binom{k-1}{i-1}}{\binom{n}{i}}.$$

- Remarquons que, pour obtenir un record à l'instant i, les $i-1$ boules précédents doivent être inférieures au numéro de la $i^{\text{ième}}$ boule. Ainsi, dans le pire des cas, le numéro de cette boule dans être i. On en déduit que R_i est l'union disjointe des $R_i \cap B_{i,k}$, $k \in [\![i\,;n]\!]$.

Par additivité,

$$r_i = \mathbb{P}(R_i) = \sum_{k=i}^{n} \mathbb{P}(R_i \cap B_{i,k}) = \frac{1}{i}\sum_{k=i}^{n} \frac{\binom{k-1}{i-1}}{\binom{n}{i}}.$$

Le calcul de cette somme est HP mais classique. On utilise la formule de Pascal :

$$\sum_{k=i}^{n}\binom{k-1}{i-1} = \sum_{k=i}^{n}\left(\binom{k}{i} - \binom{k}{i-1}\right) = \binom{n}{i} - \binom{i-1}{i} = \binom{n}{i}.$$

On en déduit que $r_i = \dfrac{1}{i}$.

Autre méthode totalement combinatoire : *pour construire une configuration avec un record en position i, on peut choisir les i premiers éléments parmi les n (il y a $\binom{n}{i}$ possibilités) puis placer le plus grand d'entre eux en position i (il y a une seule possibilité) puis ranger les $i-1$ autres dans les $i-1$ premières position (il y a $(i-1)!$ possibilités). Enfin on range les $n-i$ derniers éléments (il y a $(n-i)!$ possibilités). Il y a donc*

$$\binom{n}{i} \times 1 \times (i-1)!(n-i)! = \frac{n!}{i}$$

façons de construire une configuration avec un record à l'instant i. Puisqu'il y a $n!$ tirages successifs sans remise et que les tirages sont équiprobables, on en déduit que $\mathbb{P}(R_i) = \dfrac{1}{i}$.

2) a) Il y a un seul record si et seulement si on obtient, dès le premier tirage, la boule numérotée n (afin qu'il y ait un record à cet instant et plus jamais ensuite). Cela arrive avec probabilité $\dfrac{1}{n}$.

b) Il y a n records si et seulement si, à chaque tirage, il y a un record. Cela arrive si et seulement si on tire les boules dans l'ordre strictement croissant de leurs numéros. Cela arrive avec probabilité $\dfrac{1}{n!}$ puisqu'il n'y a qu'un seul ordre strictement croissant parmi les $n!$ tirages de n boules sans remise.

c) Il y a exactement deux records si et seulement si il y a un seul autre record que le record ayant lieu au premier tirage. Forcément :

- au premier tirage, ce n'est pas la boule numérotée n sinon il n'y aurait pas de deuxième records.
- le second record a lieu en tirant la boule numérotée n sinon, au moment où l'on tire cette boule, il y aurait un troisième tirage.

Pour construire une configuration avec 2 records, on choisit (de façon disjointe) le tirage k où l'on tire la boule numérotée n, puis on choisit $k-1$ boules parmi les $n-1$ autres boules (il y a $\binom{n-1}{k-1}$ possibilités), puis on place la plus grande d'entre elle en première position (une seule possibilité), et on range les $k-2$

autres (il y a $(k-2)!$ possibilités). Enfin on range les $n-k$ derniers éléments (il y a $(n-k2)!$ possibilités). Il y a donc

$$\sum_{k=2}^{n} \binom{n-1}{k-1} \times 1 \times (k-2)! \times (n-k)! = \sum_{k=2}^{n} \frac{(n-1)!}{(k-1)!(n-k)!}(k-2)!(n-k)! = \sum_{k=2}^{n} \frac{(n-1)!}{(k-1)}$$

configurations avec 2 records. La probabilité qu'il y a ait deux records est donc

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

3) La fonction prend n en entrée. Avant la première boucle L contient la liste des entiers de 0 à $n-1$ (représentant les numéros des boules, décalés de 1). Ensuite il y a une boucle for et à chaque étape on retire un numéro de la liste L pour l'ajouter à la liste T. Plus précisément, à l'étape k, on pioche une coordonnée entre 0 et n-k-1 (il n'y a plus que $n-k$ coordonnées dans L), on retire le numéro se trouvant à cette coordonnée et on l'ajoute dans T. Ainsi à l'issue de cette boucle, on a vidé la liste L et T contient la liste des numéros dans l'ordre du tirage.

   Ensuite on initialise S à 1 (signifiant qu'il y a un premier record à au premier tirage). Ensuite, on parcourt la liste T de la deuxième coordonnée (celle d'indice 1) à la dernière et, si on obtient un numéro plus grand que le dernier plus grand en date, on incrémente S pour signifier que l'on a un record de plus.

   A la fin T contient la liste des numéros tirés dans l'ordre du tirage, m contient le numéro du dernier record (c'est forcément n) et S le nombre de records.

4) Cette fonction construit une liste E dont la première coordonnée (celle d'indice 0) est 1. Pour tout $k \in [\![2\,;n]\!]$, on ajoute à la liste E en $k^{\text{ième}}$ position (celle d'indice $k-1$ donc) une quantité S. Celle dernière est le nombre moyen de records lorsque l'on réalise $N = 1000$ fois l'expérience avec k boules dans l'urne. Ensuite on représente (avec la commande de la ligne 11) ces espérances pour k variance de 1 à $n = 100$ avec des $+$ pour marqueurs.

   Les commandes des lignes 12 et 13 représente les points de coordonnées $(1, H_1)$, $(2, H_2)$, ..., (n, H_n) où, pour tout $k \in [\![1\,;n]\!]$, $H_k = \sum_{i=1}^{k} \frac{1}{i}$.

   Pour tout $i \in [\![1\,;n]\!]$, `E[k-1]` contient une approximation de l'espérance du nombre de records lorsque $n = k$. C'est en effet une conséquence de la loi faible des grands nombres. On conjecture que cette espérance vaut $\sum_{i=1}^{k} \frac{1}{i}$. Montrons cela : Si S_k désigne le nombre de records en k tirages, alors $S_k = \sum_{i=1}^{k} X_i$. Ainsi

$$\mathbb{E}(S_k) = \sum_{i=1}^{k} \mathbb{E}(X_i) = \sum_{i=1}^{k} r_i = \sum_{i=1}^{k} \frac{1}{i}.$$