Python Corrigé du td 12 - Révisions

Exercice 1
Ecricome 2024
Soit $n \in \mathbb{N}^*$. On considère $X = (x_0, ..., x_n) \in \mathbb{R}^{n+1}$. Ecrire une fonction nommée `prodX`, prenant en entrée X, un entier i et un entier k (on suppose $i \leq k \leq n$) et qui renvoie le produit $\prod_{\substack{j=0 \\ j \neq i}}^{k} (x_i - x_j)$.

Corrigé :

```
def prodX(X,i,k):
    P=1
    for j in range(0,k+1):
        if j!=i:
            P=P*(X[i]-X[j])
    return P
```

Exercice 2
EML 2023
Soit pour tout $n \in \mathbb{N}^*$, $S_n = \sum_{k=1}^{n} \frac{1}{k}$. On a montré (de façon classique à savoir dérouler quasi par coeur) que $S_n \sim_{n\to+\infty} \ln(n)$.

1. On considère la fonction Python suivante :

   ```
   def rang(a):
      k=1
      s=1
      while s<a:
         k=k+1
         s=s+1/k
      return k
   ```

   Expliquer ce que produit l'appel rang(50).
   Corrigé :
   Le programme renvoie le premier entier k pour lequel $\sum_{i=1}^{k} \frac{1}{i} \geq a$.

2. Le code suivant

   ```
   np.exp(49)
   ```

   renvoie : `1.90734657e+21`
   Expliquer rapidement ce que cela laisse penser si l'on fait l'appel `rang(50)`.

   Corrigé :
   comme $S_n \sim_{n\to+\infty} \ln(n)$, on a $S_n \geq 50$ à peu près si $\ln(n) \geq 50$ donc $n \geq e^{50} > \exp(49) \simeq 1.9 \times 10^{21}$.
   Le premier entier n pour lequel $S_n \geq 50$ sera très grand !
   (On atteindra sans doute les limites de calcul de notre ordinateur...)

Exercice 3
Edhec 2015 ECS

On considère deux suites réelles $(I_n)_{n\in\mathbb{N}}$ et $(J_n)_{n\in\mathbb{N}}$ vérifiant :

$$I_n + I_{n+1} = \frac{1}{n+2}, \quad J_n + J_{n+1} = I_{n+1}, \quad J_0 = \frac{1}{2}, \quad I_1 = \ln(2)$$

Compléter le programme Python ci-dessous afin qu'il calcule les valeurs de I_n et J_n.


```
n=int(input("Entrer n supérieur ou égal à 2 :"))
I=np.log(2)
J=1/2
J=I-J #car on doit calculer J_1 !
for k in range(2,n+1):
    I=1/(k+1)-I #attention au décalage !
    J=I-J
print(I,J)
```

Exercice 4
Ecricome 2015 ECS

On définit pour tout entier naturel n, les suites $(a_n)_{n\in\mathbb{N}}$ et $(b_n)_{n\in\mathbb{N}}$ par : $a_0 = \sqrt{\frac{3}{2}}$, $b_0 = \frac{1}{2}$ et

$$a_{n+1} = \sqrt{\frac{1-b_n}{2}} \ (*) \quad \text{et} \quad b_{n+1} = \sqrt{\frac{1+b_n}{2}} \ (**)$$

On admet que pour tout $n \in \mathbb{N}$,

$$9 \times 2^n . \frac{a_n}{2+b_n} < \pi < 2^n . (2a_n + \frac{a_n}{b_n})$$

On admet enfin que $a_n \sim_{n\to+\infty} \frac{\pi}{3.2^n}$ et $b_n \sim_{n\to+\infty} 1$.

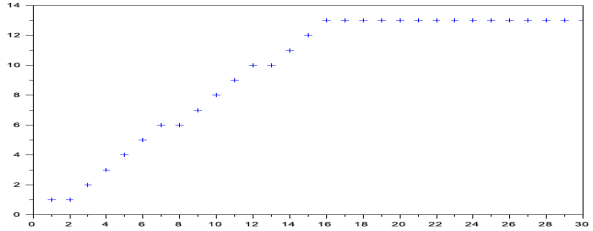
On montre par des équivalences simples que a_n et b_n tendent vers π quand n tend vers $+\infty$.

Compléter la fonction Python suivante afin qu'elle retourne, à l'aide des relations $(*)$ et $(**)$ et des résultats admis, une approximation de π à e près, ainsi que le nombre d'itérations nécessaire.

```
def h(e):
   k=0
   a=\sqrt{3}{2}
   b=1/2
   while 2**k*(2*a+a/b)-9*2**k*a/(2+b)>e:
        a=np.sqrt((1-b)/2)
        b=np.sqrt((1+b)/2)
        k=k+1
   x=a      #ou x=b
   return x,k
```

On souhaite étudier l'évolution du nombre d'itérations nécessaires en fonction de la précision souhaitée. Ecrire une fonction Python qui prend comme paramètre d'entrée un paramètre p et qui retourne un vecteur de taille p qui contient les nombres d'itérations nécessaires pour les précisions 10^{-k}, pour $k \in \{1, 2, \cdots, p\}$.

On utilise la fonction précédente avec $p = 30$ et on représente graphiquement les valeurs obtenues. On obtient le graphe suivant :



Le nombre d'itérations pour une précision est à peu près linéaire : il faut environ k itérations pour une précision de 10^k. On voit que l'on atteint ensuite les limites de calcul de Python !

Exercice 5
Edhec 2015 ECE
Un joueur réalise une suite de lancers indépendants d'une pièce. Cette pièce donne Pile avec la probabilité p $(0 < p < 1)$ et Face avec la probabilité $q = 1 - p$.
On note N la variable aléatoire égale au rang d'apparition du premier Pile.
Si N prend la valeur n, le joueur place n boules numérotées de 1 à n dans une urne, puis il extrait une boule au hasard de cette urne. On dit que le joueur a gagné si le numéro porté par la boule tirée est impair et on note A l'événement "le joueur gagne". On appelle X la variable aléatoire égale au numéro porté par la boule extraite de l'urne.

1. Si m est un entier naturel, que renvoie la commande $2 * np.floor(m/2)$?
   Corrigé :
   Cette commande renvoie m si m est un entier pair, et $m-1$ si m est un entier impair.

2. Compléter les commandes Python suivantes pour qu'elles simulent N et X et qu'elles renvoient l'un des deux messages "le joueur a gagné" ou "le joueur a perdu".
   Corrigé :

```
p=float(input(p"="))
N=rd.geometric(p)
X=rd.randint(1,N+1)
if 2*np.floor(m/2)==m:
   print("Perdu")
else:
   print("Gagné")
```

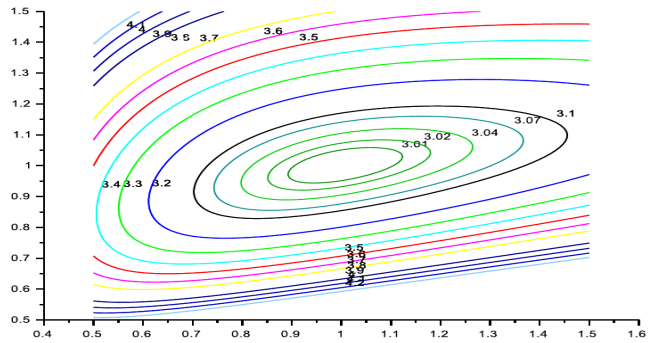
Exercice 6
Edhec 2024
On considère une suite $(J_n)_{n\in\mathbb{N}^*}$ vérifiant : $J_1 = \frac{\pi}{2}$ et pour tout $n \in \mathbb{N}^*$, $J_n = 2n(J_n - J_{n+1})$. Compléter le programme suivant pour qu'il renvoie la valeur de J_n.
Corrigé :

```
def suiteJ(n):
   J=np.pi/2
   for k in range(2,n+1):
      J=(2*k-3)/(2*k-2)*J #attention au décalage !
   return J
```

Exercice 7
Ecricome 2019 ECE
On considère la fonction f définie sur l'ouvert $]0;+\infty[\times]0;+\infty[$ par :

$$\forall (x,y) \in (\mathbb{R}_+^*)^2, \quad f(x,y) = \frac{x}{y^2} + y^2 + \frac{1}{x}$$

1. On utilise Python pour tracer les lignes de niveau de la fonction f. On obtient le graphe suivant :



On conjecture que la fonction f admet un minimum global en $(1,1)$ qui a pour valeur 3.

2. Pour tout entier non nul, on note h_n la fonction définie sur $\mathbb{R}_+^*$ par :

$$\forall x > 0, \quad h_n(x) = f(x^n, 1) = x^n + 1 + \frac{1}{x^n}$$

On admet que l'équation $h_n(x) = 4$ admet exactement deux solutions, notées u_n et v_n et vérifiant : $0 < u_n < 1 < v_n$.

(a)
```
def h(n,x):
   return x**n+1+1/x**n
```

(b) Pour tout $x > 1$,

$$h_n'(x) = n.x^{n-1} - n.\frac{1}{x^{n+1}} = n.(x^{n-1} - \frac{1}{x^{n+1}}) > 0$$

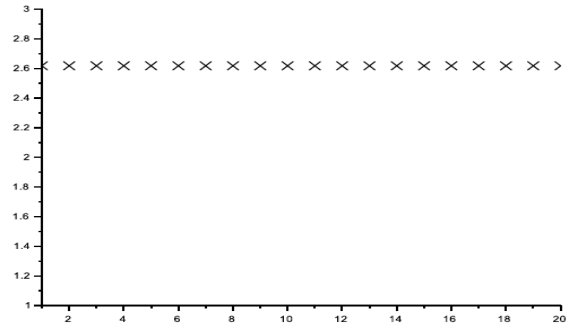
car $x > 1$. La fonction h_n est donc croissante sur $]1;+\infty[$.

```
def v(n):
   a=1
   b=3
   while (b-a)>10**(-5):
        c=(a+b)/2
        if h(n,c)<4 :
            a=c
        else :
            b=c
return c # ou b, ou a
```

(c) A la suite de la fonction v, on écrit le code suivant :

```
X=np.arange(1,21,1)
Y=np.zeros(1,20)
for k in range(1,21):
    Y[k]=v[k]**k
plt.plot(X,Y)
```

A l'exécution du programme, on obtient la sortie graphique suivante :



On peut conjecturer que l'on a environ $v_k^k \simeq 2,6$, donc $v_k \simeq 2,6^{1/k} = e^{\frac{1}{k}\ln(2,6)}$

Exercice 8
HEC 2017 ECS

On rappelle que si pour tout $n \in \mathbb{N}^*$, $Z_n \hookrightarrow \mathcal{B}(n,p)$, en notant $\overline{Z_n} = \frac{Z_n}{n}$, la suite $(\overline{Z_n})_{n\geq 1}$ converge en probabilités vers p. Si f est une fonction continue, on a alors $(f(\overline{Z_n}))_{n\geq 1}$ qui converge vers $f(p)$. On considère le programme suivant :

```
def f(x):
   if x==0 :
       y=0
   else :
       y=-x*np.log(x)
   return y
p=0.4
n=100;   N=1000;   S=0
for k in range(1,N+1):
    S=S+f(rd.binomial(n,p)/n)
print(S/N)
```

Ce programme affiche

$$\frac{1}{N}S = \frac{1}{N}.\sum_{k=1}^{N} -(X_k/n).\ln(X_k/n)$$

où X_1, ..., X_N sont des variables indépendantes suivant toutes la loi $\mathcal{B}(n,p)$. Il s'agit d'une moyenne empirique, qui converge en loi vers $E(-(X_1/n).\ln(X_1/n)$ d'après la loi faible des grands nombres. Puis $-(X_1/n).\ln(X_1/n$ converge vers $-p\ln(p)$. Donc le programme devrait afficher une valeur approchée de $-p\ln(p)$.
Nous avons utilisé la loi faible des grands nombres, et il s'agit d'une méthode de Monte Carlo pour trouver une valeur approchée de $-p\ln(p)$.

Exercice 9
Extrait Ecricome 2017
On considère la suite $(S_n)_{n\in\mathbb{N}}$ définie par :

$$\forall n \in \mathbb{N},\quad S_n = \sum_{k=0}^{n} \frac{(-1)^k}{(k+1)^{k+1}}$$

On montre que la suite (S_n) converge vers une limite I, et que l'on a :

$$\forall n \in \mathbb{N},\quad |I - S_n| \leq \frac{1}{e^{n+1}.(n+1)!}$$

Ecrire une fonction Python d'en-tête `def estimation(eps)` qui prend comme paramètre d'entrée un réel ϵ strictement positif, et qui produit en sortie une valeur approchée de I à ϵ près.
Corrigé :

```
def estimation(eps):
   k=0
   S=1 #c'est S_0
   a=1/np.exp(1)
   while a>eps:
       k=k+1
       S=S+(-1)**k/(k+1)**(k+1)
       a=(1/np.exp(1))*(1/(k+1))*a
   return S
```


Exercice 10
Extrait Ecricome 2017
Pour tout entier n strictement positif, on considère l'expérience suivante : on dispose de n urnes initialement vides, numérotées de 0 à $n-1$ et on dispose d'un grand stock de boules que l'on dépose une à une dans ces urnes. Pour chaque boule, on choisit au hasard, de façon équiprobable, l'urne dans laquelle la boule est déposée.
On note X_n le rang du premier tirage pour lequel une des urnes contiendra deux boules. Compléter la fonction Python suivante pour qu'elle simule une réalisation de la variable aléatoire X_n.
Corrigé :
Il n'y a pas grand chose à compléter !!

```
def SimulX(n):
    urnes=np.zeros(n)
    X=1
    choix=int(np.floor(rd.random()*n))
    while np.max(urnes)<=1:
        urnes[choix]=urnes[choix]+1
        choix=int(np.floor(rd.random()*n))
        X=X+1
    return X,urnes
```

Exercice 11
On définit deux suites (x_k) et (y_k) par : $x_0 = 1$, $y_0 = 0$ et pour tout $k \in \mathbb{N}$,

$$\begin{cases} x_{k+1} = 3x_k + 4y_k \\ y_{k+1} = 2x_k + 3y_k \end{cases}$$

Pour tout $k \geq 1$, on pose $r_k = \dfrac{x_k}{y_k}$ et on montre que

$$\forall k \in \mathbb{N}^*,\quad |r_k - \sqrt{2}| < \frac{1}{2y_k^2}$$

Ecrire le code d'une fonction Python `approx2(n)` qui prend en entrée un entier n et qui renvoie un terme de la suite (r_k) qui soit une approximation de $\sqrt{2}$ à 10^{-n} près.
Corrigé

```
def approx2(n):
    x=3 #x_1
    y=2 #y_1
    while 1/(2*y**2)>10**(-n):
        xnew=3*x+4*y
        ynew=2*x+3*y
        x=xnew
        y=ynew
    return x/y
```

Exercice 12
Soit $N \in \mathbb{N}^*$. Le code Python ci-dessous simule une variable aléatoire T_N. Décrire une expérience aléatoire correspondant à cette simulation.

```
def T(N):
   X1,X2,X3=N+1,N+1,N+1
   t=0
   while X1*X2*X3>0:
      x=rd.random()
      if x<1/3:
          X1=X1-1
      elif x<2/3:
          X2=X2-1
      else:
          X3=X3-1
      t=t+1
   return t
```

Expérience aléatoire : on considère trois urnes U_1, U_2, U_3 remplies chacune de $N+1$ boules. A chaque étape, on choisit de façon équiprobable l'une des trois urnes et on prélève une boule dans cette urne. On procède ainsi jusqu'à ce que l'une des trois urnes soit vide et on note T la variable aléatoire égale au nombre de tirages effectués.

Exercice 13
Soient b, c, d des constantes réelles et considérons l'équation $f(x)=0$ avec $f(x)=x^3+bx^2+cx+d$ pour $x\in\mathbb{R}$.

1. $\lim_{x\to+\infty} f(x)=+\infty$, et $\lim_{x\to+\infty} f(x)=-\infty$. Par conséquent, il existe un entier m (assez grand) tel que $f(m)>0$ et $f(m)<0$. On a alors bien $f(m)f(-m)<0$

2. Compléter la fonction ci-dessous pour qu'elle renvoie le plus petit entier naturel m vérifiant $f(m)f(-m)<0$.

```
def trouver-chg-signe(b,c,d):
   def pol3(x):
      return x**3+b*x**2+c*x+d
   m=1
   while pol3(m)*pol3(-m)>0:
      m=m+1
  return m
```

3. On utilise une méthode de dichotomie pour trouver une solution de l'équation $f(x)=0$ dans $[-m,m]$: on considère les suites (a_n) et (b_n) définies comme suit : $a_0=-m$, $b_0=m$, et pour tout $n\geq 0$,

$$(a_{n+1},b_{n+1})=\begin{cases}(a_n,c_n) \text{ si } f(c_n)f(a_n)\leq 0,\\ (c_n,b_n) \text{ sinon}\end{cases}$$

où l'on a posé $c_n=(a_n+b_n)/2$. Si on se donne un réel $\epsilon>0$, on considèrera que la solution est atteinte à ϵ près si $|a_n-b_n|<\epsilon$, et dans ce cas on peut choisir c_n comme solution. Compléter la fonction suivante pour qu'elle renvoie une solution à ϵ près de l'équation $f(x)=0$.
Cette fonction peut faire appel à la fonction `trouver-chg-signe`

```
def madichotomie(b,c,d,epsilon):
   def pol3(x):
      return x**3+b*x**2+c*x+d
   m=trouver-chg-signe(b,c,d)
   a=-m
   b=m
   while np.abs(a-b)>epsilon:
      c=(a+b)/2
      if f(a)*f(c)<0:
           b=c
      else:
           a=c
      res=c
  return res
```


Exercice 14
On considère une expérience consistant à lancer une pièce équilibrée n fois de manière consécutive. L'expérience en question est simulée par le programme Python ci-dessous pour la valeur $n=10$ (et répétée ici 1000 fois).

```
Nexp=1000
n=10
u=rd.randint(1,3,[n,Nexp])
test=0
for k in range(0,Nexp):
    ok=1
    for i in range(1,n):
       if u[i-1,k]==1 and u[i,k]==1:
          ok=0
    test=test+ok
Pn=test/Nexp
print(Pn)
```

On considère par exemple que Pile est codé par 1 et Face par 2. L'événement E_n est alors : "au cours de la succession de n lancers, il n'y a jamais deux Pile de suite".
P_n est alors la fréquence d'apparition de cet événement.

On remarque que pour $n\geq 3$,

$$E_n=(E_n\cap P_n)\cup(E_n)\cap F_n)=(E_{n-2}\cap F_{n-1}\cap P_n)\cup(E_{n-1}\cap F_n)$$

d'où en passant aux probabilités :

$$p_n=\frac{1}{4}p_{n-2}+\frac{1}{2}p_{n-1}$$

et on obtient ainsi une suite récurrente linéaire d'ordre 2.
Pour tout n,

$$p_{n-1}=\frac{1}{4}p_{n-3}+\frac{1}{2}p_{n-2}\geq\frac{1}{2}p_{n-2}$$

donc $p_{n-2}\leq 2.p_{n-1}$. Par suite,

$$p_n=\frac{1}{4}p_{n-2}+\frac{1}{2}p_{n-1}\leq\frac{1}{4}.2.p_{n-1}+\frac{1}{2}p_{n-1}\leq p_{n-1}$$

La suite est décroissante et minorée par 0 donc convergente. De plus, en passant à la limite dans la relation qui la définit :

$$l=\frac{1}{4}l+\frac{1}{2}l\Leftrightarrow l=0$$