Code de partage avec Capytale : a6e6-7631015 **Corrigé**

On utilisera les bibliothèques suivantes :

```
import numpy as np
import numpy.random as rd
import numpy.linalg as al
import matplotlib.pyplot as plt
```

1 Fonctions, suites récurrentes

1. Définir la fonction `f` associée à $f : x \mapsto (x+1)e^{-\frac{x}{2}}$

```
def f(x):
  return (x+1)*np.exp(-x/2)
```

2. Définir la fonction `df` associée à la fonction dérivée de f. On en donnera une forme factorisée.

On trouve $f'(x) = \left[1 + (x+1) \times \left(-\frac{1}{2}\right)\right] e^{-\frac{x}{2}} = (1-x)\,\frac{e^{-\frac{x}{2}}}{2}$

```
def df(x):
  return (1-x)*np.exp(-x/2)/2
```

3. Définir la fonction `d2f` associée à la dérivée seconde de f. On en donnera une forme factorisée.

On trouve $f''(x) = \left[-1 + (1-x) \times \left(-\frac{1}{2}\right)\right] e^{-\frac{x}{2}} = (x-3)\,\frac{e^{-\frac{x}{2}}}{4}$

```
def df(x):
  return (x-3)*np.exp(-x/2)/4
```

4. Tracer la courbe représentative de f' sur $[0,5]$

```
x=np.linspace(0,5,100)
y=f(x)
plt.plot(x,y)
plt.show()
```

5. Justifier que $|f'(x)| \leqslant \max(f'(0), |f'(3)|))$ pour $x \geqslant 0$

$f''(x) \geqslant 0 \Leftrightarrow x \geqslant 3$
donc f' est décroissante sur $[0,3]$ et croissante sur $[3,+\infty[$
de plus $f'(0) = \frac{1}{2}$ et $\forall x \geqslant 1, 1-x \leqslant 0$ donc $f'(x) \leqslant 0$
donc f' a pour maximum $f'(0)$ et pour minimum $f'(3) \leqslant 0$
donc $\forall x \geqslant 0, |f'(x)| \leqslant \max(f'(0), |f'(3)|))$

6. A l'aide d'une dichotomie, donner un encadrement à 10^{-2} près de l'unique solution ℓ de l'équation $f(x) = x$ sur $\mathbb{R}_+$
On commencera par déterminer l'intervalle d'étude.

Avec $g(x) = f(x) - x$, on trouve $g(0) = 1$ et $g(2) = f(2) - 2 = \frac{3}{e} - 2 \leqslant 0$ car $e \geqslant 2$ donc $\frac{3}{e} \leqslant \frac{3}{2}$
donc on va débuter l'algorithme de dichotomie sur l'intervalle $[0,2]$

```
a=0
b=2
while b-a>10**(-3):
  m=(a+b)/2
  if f(m)>m :
    a=m # car f(0)-0>0
  else :
    b=m # et f(2)-2<0
print(m)
```

7. Définir la suite récurrente : $u_0 = 0$ et, pour $n \in \mathbb{N}$: $u_{n+1} = f(u_n)$

   On peut le faire de manière récursive ou avec un boucle `for` :

```
def u(n) : # option récursive
  if n==0 :
    return 0
  else :
    return f(u(n-1))
```

```
def u(n) : # option boucle
  u=0
  for i in range(1,n+1):
    u=f(u)
  return u
```

8. On admet que : pour tout $n \in \mathbb{N}$, $|u_n - \ell| \leqslant 2^{-n+1}$
   A l'aide de cette majoration, déterminer un entier n_0 pour lequel u_n est une valeur approchée de ℓ à 10^{-5} près pour tout entier $n \geqslant n_0$

   Dès que $2^{-n+1} \leqslant 10^{-5}$, on sait que $|u_n - \ell| \leqslant 10^{-5}$ c'est-à-dire que l'écart entre u_n et ℓ est inférieur ou égal à 10^{-5}, autrement dit u_n est une valeur approchée de ℓ à 10^{-5} près. Donc dès qu'un tel n_0 est trouvé, c'est a fortiori vrai pour $n \geqslant n_0$ car 2^{-n+1} est décroissante.

```
n=1
while 2**(n-1)>10**(-5) :
  n=n+1
print(n)
```

   On obtient alors 18. on aurait pu obtenir ce résultat par une résolution algébrique de l'inéquation $2^{-n+1} \leqslant 10^{-5}$

2 Suites récurrentes d'ordre 2

On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 = 0$, $u_1 = \dfrac{1}{16}$ et $\forall n \in \mathbb{N}, \quad u_{n+2} = \frac{3}{4}u_{n+1} + \frac{3}{16}u_n$

1. Définir une fonction `PremiersTermes(n)` qui renvoie la listes des $n+1$ premiers termes de la suite.
   Exemple : `PremiersTermes(4)` doit renvoyer :

```
[0, 0.0625, 0.046875, 0.046875, 0.0439453125]
```

   La construction d'une liste permet de construire chaque nouveau terme à l'aide des deux derniers termes de la liste (on peut utiliser les commandes `L[-1]` et `L[-2]` pour obtenir ces termes).

```
def PremiersTermes(n):
  L=[0,1/16]
  for i in range (2,n+1) :
    L.append(3/4*L[-1]+3/16*L[-2])
  return L
```

2. Vérifier numériquement que : $\displaystyle\sum_{n=0}^{+\infty} u_n = 1$

Il suffit d'additionner les termes de la liste précédente pou n « grand ». On peut donc le vérifier avec la fonction précédente et des « grandes valeurs » de n, par exemple avec la commande `sum(PremiersTermes(100))` ou `sum(PremiersTermes(1000))`

3. On admet qu'il existe une variable aléatoire X admettant un moment d'ordre 2, à valeurs dans $\mathbb{N}$, telle que $\forall n \in \mathbb{N}, \quad P(X = n) = u_n$
Calculer des valeurs approchées de $E(X)$ et $V(X)$ en calculant des sommes partielles.

   La liste `PremiersTermes(100)` nous donne les $P(X = k)$ pour $k \in [\![0, n[\![$, il suffit alors de multiplier chaque terme par k pour obtenir une valeur approchée de l'espérance.

```
L=PremiersTermes(100)
E=[k*L[k] for k in range(1,100)]
sum(E)
```

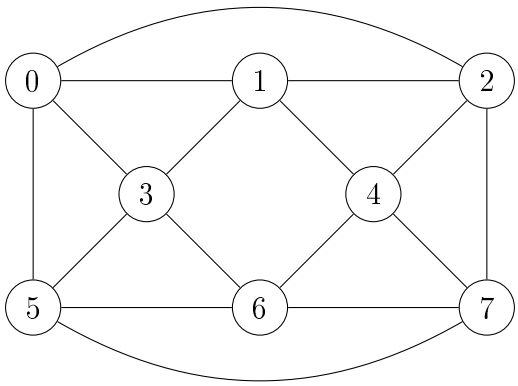
4. Compléter la fonction Python suivante permettant de calculer un terme de la suite u par une fonction récursive :

   Il faut distinguer les deux premiers termes, sinon, on utilise la formule récursive qui définit la suite.

```
def SR(u0, u1, n):
    if n==0 :
        return 0
    elif n==1 :
        return 1/16
    else:
        return 3/4*SR(u0, u1, n-1)+3/16*SR(u0, u1, n)
print(SR(0,1/16,100)) # pour tester
```

3 Graphes

On considère le graphe **G** suivant :



1. Définir la liste d'adjacence `L` de ce graphe.

   Pour rappel, il s'agit de la liste des sommets adjacents à chaque sommet. On se rend compte, en établissant cette liste, que chaque sommet est relié à quatre autres.

```
L=[
    [1, 2, 3, 5],
    [0, 2, 3, 4],
    [0, 1, 4, 7],
    [0, 1, 5, 6],
    [1, 2, 6, 7],
```

```
    [0, 3, 6, 7],
    [3, 4, 5, 7],
    [2, 4, 5, 6]
]
```

2. Définir la matrice d'adjacence `A` de ce graphe.

   On peut la définir *ex nihilo* ou à partir de la liste d'adjacence comme nous l'avions vu au premier T.P.

```
def MatAdj(L):
    n=len(L)
    A=np.zeros([n, n])
    for i in range(n):
        for j in L[i]:
            A[i,  j]=1
    return A
A=MatAdj(L)
print(A)
```

3. Déterminer le nombre de chaînes de longueur 4 allant du sommet 0 au sommet 2

   Il suffit de regarder sur la matrice A^4, le coefficient correspondant à la colonne du sommet 0 (la première) et la ligne du sommet 2 (la troisième) ou l'inverse puisque la matrice est symétrique. La commande `print(al.matrix_power(A, 4)[0, 2])` donne directement le résultat.

4 Probabilités

Une urne contient n boules noires et b boules blanches, avec $n \geqslant 1$ et $b \geqslant 1$
On effectue une succession de tirages dans l'urne avec remise.
On note X_k le rang d'apparition de la $k^{\text{ème}}$ boule blanche pour $k \in \mathbb{N}^*$
Ainsi, X_1 est le rang d'apparition de la première boule blanche, X_2 celui de la deuxième boule blanche, etc.
On pose, pour $i \in \mathbb{N}^*$, T_i la variable aléatoire égale à 1 si la boule tirée au rang i est blanche et égale à 0 sinon.
On pose enfin, $X_0 = 0$ et, pour $k \in \mathbb{N}^*$, $Z_k = X_k - X_{k-1}$

1. Quelle est la loi de T_i pour $i \in \mathbb{N}^*$?
   Ecrire une fonction `Tirage(n, b)` permettant de simuler la loi de T_i, `n` et `b` désignant respectivement le nombre de boules noires et le nombre de boules blanches.

   T_i suit une loi de Bernoulli de paramètre $p = b/(n+b)$

```
def Tirage(n, b):
    return rd.binomial(1, b/(n+b))
```

2. Quelle est la loi de X_1 ? Ecrire une fonction `G(n, b)` permettant de simuler la loi de X_1

   X_1 suit une loi géométrique de paramètre p

```
def G(n, b):
    return rd.geometric(b/(n+b))
```

3. On admet que, pour $k \in \mathbf{N}^*$, Z_k suit la même loi que X_1

   (a) Justifier que $X_k = \sum_{j=1}^{k} Z_j$

   Par télescopage, $\sum_{j=1}^{k} Z_j = \sum_{j=1}^{k} (X_k - X_{k-1}) = X_k - X_0 = X_k$ car $X_0 = 0$

(b) Ecrire une fonction `X(k, n, b)` simulant la loi de X_k

Après l'apparition de la boule blanche $k-1$, la variable aléatoire égale aux nombres de tirages supplémentaires pour obtenir la boule blanche suivante est en fait Z_k puisque ce nombre de tirages est $X_k - X_{k-1}$. Et Z_k suit la même loi que X_1 (« le compteur est remis à zéro » après chaque boule blanche). Donc on va déterminer Z_i pour tout $i \in [\![1, k]\!]$ et on en déduira X_k grâce à la question précédente.

```
def X(k, n, b):
    x=0
    for j in range(k):
        x +=G(n, b) # c'est en fait Zi
    return x # contient à la fin somme des Zi=Xk
```

(c) Donner une valeur approchée de $P(X_2 \leqslant 5)$ dans le cas $n = 3$ et $b = 2$

Il s'agit de répéter, avec la fonction précédente, un grand nombre de fois (10 000 ci-dessous) l'expérience avec les paramètre $n = 2, b = 3$ et $k = 2$, puis de calculer le nombre de fois où le résultat est inférieur ou égal à 5
A noter, la commande `[v<=5 for v in Valeurs]` contient une liste de vrai/faux relativement à la question $\leqslant 5$? Et la commande `sum` appliquée à cette liste donne le nombre de « vrai » ce qui répond à notre question.
On trouve une valeur approchée de $0,66$ pour cette probabilité.

```
Valeurs=[X(2, 3, 2) for i in range(10000)]
extrait=[v<=5 for v in Valeurs]
frequence = np.sum(extrait)/10000
print(frequence)
```