Python 7
Utilisation de la loi faible des grands nombres

L'objectif de cette fiche est l'utilisation de la loi faible des grands nombres dans trois cas classiques :

1. approximation d'histogrammes de lois,
2. valeur approchée d'espérance,
3. calcul approché d'intégrales.

On rappelle les commandes suivantes :

- `rd.random()` renvoie un réel aléatoire de $]0;1[$, autrement dit, renvoie une réalisation d'une variable aléatoire suivant la loi $\mathscr{U}(]0;1[)$
- `rd.random(N)` renvoie N réels aléatoires de $]0;1[$ dans un tableau 1 ligne $\times$ N colonnes, autrement dit, renvoie N réalisations d'une variable aléatoire suivant la loi $\mathscr{U}(]0;1[)$ dans un tableau une ligne $\times$ N colonnes
- `rd.randint(a,b)` renvoie un entier aléatoire de $[\![a;b[\![$
- `rd.randint(a,b,N)` renvoie N entiers aléatoires de $[\![a;b[\![$ dans un tableau 1 ligne $\times$ N colonnes
- `rd.binomial(n,p)` renvoie une réalisation d'une variable aléatoire suivant la loi $\mathscr{B}(n;p)$
- `rd.binomial(n,p,N)` renvoie N réalisations d'une variable aléatoire suivant la loi $\mathscr{B}(n;p)$ dans un tableau 1 ligne $\times$ N colonnes
- `rd.geometric(p)` renvoie une réalisation d'une variable aléatoire suivant la loi $\mathscr{G}(p)$
- `rd.geometric(p,N)` renvoie N réalisations d'une variable aléatoire suivant la loi $\mathscr{G}(p)$ dans un tableau 1 ligne $\times$ N colonnes
- `rd.poisson(lam)` renvoie une réalisation d'une variable aléatoire suivant la loi $\mathscr{P}(\lambda)$, avec $\lambda =$`lam`
- `rd.poisson(lam,N)` renvoie N réalisations d'une variable aléatoire suivant la loi $\mathscr{P}(\lambda)$, avec $\lambda =$`lam`, dans un tableau 1 ligne $\times$ N colonnes
- `rd.exponential(1/lam)` renvoie une réalisation d'une variable aléatoire suivant la loi $\mathscr{E}(\lambda)$, avec $\lambda =$`lam`
- `rd.exponential(1/lam,N)` renvoie N réalisations d'une variable aléatoire suivant la loi $\mathscr{E}(\lambda)$, avec $\lambda =$`lam`, dans un tableau 1 ligne $\times$ N colonnes
- `rd.normal(m,s)` renvoie une réalisation d'une variable aléatoire suivant la loi $\mathscr{N}(m,s^2)$
- `rd.normal(m,s,N)` renvoie N réalisations d'une variable aléatoire suivant la loi $\mathscr{N}(m,s^2)$ dans un tableau 1 ligne $\times$ N colonnes

✘ Attention !
Comme pour la commande `range(a,b)`, la borne de droite est exclue !

✘ Attention !
Le paramètre à saisir est l'espérance de la VA, pas le paramètre mathématique de la loi exponentielle.

Que permet la fonction `mystere` suivante ?

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

def mystere(r):
    u=rd.random()
    x=-1/r*np.log(1-u)
    return x
```

☞ Pour info...
Si $U \hookrightarrow \mathscr{U}(]0;1])$, alors $\frac{-1}{\lambda}\ln(U) \hookrightarrow \mathscr{E}(\lambda)$.
Pourquoi alors choisir la version du cours ? Réponse dans DS5-VB...

I Obtention d'histogrammes et valeurs approchées d'espérances

Nous avons vu en cours que la loi faible des grands nombres permet de justifier le fait que la fréquence d'observation (empirique) d'un évènement sur un grand nombre de répétitions indépendantes d'une même expérience se rapproche de la probabilité (théorique) de cet évènement.
C'est ce résultat qui justifie que l'histogramme des fréquences obtenues sur un grand nombre de réalisations d'une variable aléatoire tend à se rapprocher vers la distribution de probabilité de cette variable aléatoire.

✔ Pour s'entraîner...
Soit X une variable aléatoire à densité telle $X(\Omega)$ est un intervalle I de $\mathbb{R}$ et dont la fonction de répartition, notée F_X, est strictement croissante sur I.
Montrer que F est bijective puis déterminer la loi de la variable aléatoire $U = F_X(X)$.
En déduire un programme `Python` permettant de simuler une réalisation de X en supposant que `reciproqueFX` permet de calculer $F_X^{-1}(x)$.

1. On considère la fonction $f : x \longmapsto \begin{cases} \dfrac{r}{x^{r+1}} & \text{si } x \geq 1 \\ 0 & \text{sinon} \end{cases}$. Démontrer que f est une densité de probabilité.

   Dans la suite, on considère une variable aléatoire X de densité f.

2. Identifier la loi de la variable aléatoire $Y = \ln(X)$.

3. En déduire deux programmes `Python` permettant de simuler 10000 réalisations de X, l'un ne devra pas utiliser la commande `rd.exponential`.

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

def simulX1(r):
```

```
    Y=rd.exponential(1/r,10000)
    X=np.exp(Y)
    return X

def simulX2(r):
    U=rd.random(10000)
    Y=-1/r*np.log(1-U)
    X=np.exp(Y)
    return X
```

4. Écrire alors un programme **Python** permettant d'obtenir l'histogramme de fréquences sur 10000 réalisations de X dans le cas où $r = 2$.

```
import matplotlib.pyplot as plt

Lbornes=range(0,11)
plt.hist(simulX1(2),Lbornes,density=True)
plt.show()
```

5. Pour quelles valeurs de r la variable aléatoire X admet-elle une espérance ? La calculer et écrire une fonction **Python** telle que l'exécution de **esperanceX(r)** renvoie une valeur approchée de $\mathbb{E}(X)$ par la loi faible des grands nombres.

```
def esperanceX(r):
    L=simulX1(r)
    return sum(L)/len(L) #ou np.mean(L)
```

Calcul approché d'intégrales par méthode de Monté-Carlo

L'objectif est d'obtenir une valeur approchée de l'intégrale $\int_0^1 f(t)dt$ *dans le cas d'une fonction* f *continue sur* $]0;1[$ *(ou* $[0;1[$*, ou* $]0;1]$ *ou* $[0;1]$*).*
L'idée est de voir cette intégrale comme l'espérance d'une certaine variable aléatoire... Oui, mais laquelle ?! Pour cela, il faut penser au théorème de transfert...

Oui, mais...

Si on souhaite $\int_0^2 g(t)dt$, comment se ramener à $\int_0^1 f(t)dt$? Ou si on a $\int_0^{+\infty} g(t)dt$? Ou pire, $\int_{-\infty}^{+\infty} g(t)dt$?

1. Justifier la convergence de l'intégrale $\int_0^1 \big(\ln(t)\big)^2 dt$, notée I.

2. Soit U une variable aléatoire suivant la loi uniforme sur $]0;1]$. Exprimer I comme l'espérance d'une certaine variable aléatoire fonction de U.

3. En déduire une fonction **Python** telle que l'exécution de `approx_I(n)` renvoie une valeur approchée de I obtenue par la loi faible des grands nombres sur n réalisations indépendantes de $g(U)$. Autrement dit, `approx_I(n)` renvoie la moyenne empirique de n réalisations indépendantes de $g(U)$.

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

def approx_I(n):
    U=rd.random(n)
    X=np.log(U)**2
    return np.mean(X)

import matplotlib.pyplot as plt

L100=[approx_I(1000) for k in range(50)]
plt.plot(range(50),L100,"r+")
L1000=[approx_I(10000) for k in range(50)]
plt.plot(range(50),L1000,"bo")
L10000=[approx_I(100000) for k in range(50)]
plt.plot(range(50),L10000,"g*")
plt.show()
```

4. Tester ce programme 10 fois de suite avec $n = 100$ puis 10 fois de suite avec $n = 1000$. Que dire ?

5. Écrire un programme **Python** permettant d'obtenir sur un même graphique 3 nuages de points :
   - le premier est le nuage de points de 50 exécutions de `approx_I(100)`, représentés en rouge ;
   - le second est le nuage de points de 50 exécutions de `approx_I(1000)`, représentés en bleu ;
   - le troisième est le nuage de points de 50 exécutions de `approx_I(10000)`, représentés en noir.

```
import matplotlib.pyplot as plt

L100=[approx_I(100) for k in range(50)]
plt.plot(range(50),L100,"r+")
L1000=[approx_I(1000) for k in range(50)]
plt.plot(range(50),L1000,"b+")
L10000=[approx_I(10000) for k in range(50)]
plt.plot(range(50),L10000,"g+")
plt.show()
```