Correction - TP 9

Simulation de variables aléatoires à densité

Exercice 1

1. En utilisant le premier point de la propriété précédente :

```
def norcentreereduite():
    u = rd.random(12)
    x = np.sum(u)-6
    return x
```

2. En utilisant le deuxième point de la propriété précédente :

```
def normale(m, s):
    x = norcentreereduite()
    y = s*x+m
    return y
```

3. On peut procéder ainsi :

```
def Normale(m, s, N):
    X = np.zeros(N)
    for k in range(N):
        X[k] = normale(m, s)
    return X
```

Exercice 2

1. On a $F_U : x \in \mathbb{R} \mapsto \begin{cases} 0 & \text{si } x \leq 0 \\ x & \text{si } 0 < x < 1 \\ 1 & \text{si } x \geq 1 \end{cases}$.

2. Posons $Y = F^{-1}(U)$. On a $Y(\Omega) = F^{-1}(]0,1[) =]a,b[= X(\Omega)$. Et pour tout $x \in]a,b[$, on a :

$$F_Y(x) = P(Y \leq x) = P(F^{-1}(U) \leq x) = P(U \leq F(x)) \underset{F(x)\in]0,1[}{=} F(x)$$

Les fonctions de répartitions de X et Y sont égales : elles suivent donc la même loi.

Exercice 3

1. Notons F la fonction de répartition de la loi $\mathcal{E}(\lambda)$. On a :

$$F : x \in \mathbb{R} \mapsto \begin{cases} 0 & \text{si } x \leq 0 \\ 1 - e^{-\lambda x} & \text{si } x > 0 \end{cases}.$$

F est continue sur $\mathbb{R}_+^*$, strictement croissante sur cet intervalle (car F est dérivable et $F'(x) = \lambda e^{-\lambda x} > 0$) et on a $\lim\limits_{0^+} F = 0$ et $\lim\limits_{+\infty} F = 1$. Par le théorème de la bijection, F réalise une bijection de $\mathbb{R}_+^*$ dans $]0,1[$.

Déterminons F^{-1} : $]0,1[\to \mathbb{R}_+^*$. Soit pour cela $x \in \mathbb{R}_+^*$ et $y \in]0,1[$. On résout :

$$y = F(x) \Leftrightarrow y = 1 - e^{-\lambda x} \Leftrightarrow e^{-\lambda x} = 1 - y$$
$$\Leftrightarrow x = -\frac{1}{\lambda}\ln(1-y)$$

Ainsi on a $F^{-1} : y \in]0,1[\to -\frac{1}{\lambda}\ln(1-y)$.

2. (a) En utilisant le principe d'inversion :

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

(b) En adaptant la fonction `Normale` ici :

```
def Exponentielle(lbd,N):
    X = np.zeros(N)
    for k in range(N):
        X[k] = exponentielle(lbd)
    return X
```

3. (a) On utilise les lignes de codes suivantes :

```
E = Exponentielle(0.5, 10000)
print(np.mean(E), np.std(E))
```

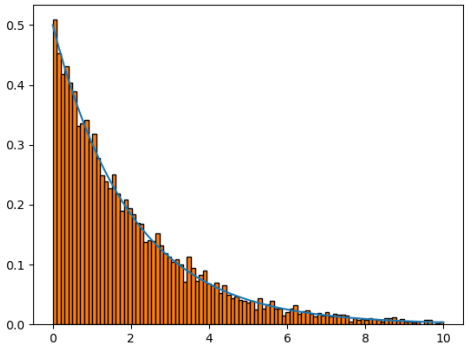
(b) Rappelons que les commandes `np.mean` et `np.std` calculent respectivement la moyenne et l'écart-type d'une série statistique. On obtient ici `2.0344615` et `2.0258905`. C'est bien conforme à ce qu'on attend, puisque l'espérance et la variance d'une loi $\mathcal{E}(\lambda)$ sont égales à $\dfrac{1}{\lambda}$, c'est-à-dire 2 ici.

Exercice 4

On procède ainsi :

```
# Echantillon
N = 10000
lbd = 1/2
x = Exponentielle(lbd, N)

# densité
def f(t):
    y = lbd*np.exp(-lbd*t)
    return y
t = np.linspace(0, 10, 100)
y = f(t)
plt.plot(t, y)

# histogramme des fréquences
c = np.linspace(0, 10, 100)
plt.hist(x, c, density='True', edgecolor='k')

plt.show()
```

On obtient :



On observe graphiquement que les aires des rectangles formant l'histogramme des fréquences sont proches des aires délimitées par la courbe représentative de la densité. On peut donc conclure que `Exponentielle(0.5)` renvoie bien des simulations suivant la loi $\mathcal{E}(0.5)$.

Exercice 5

1. Le vecteur `u` contient 100 éléments en progression arithmétique, de la plus petite valeur de l'échantillon `x` à sa plus grande valeur. À chaque passage dans la boucle `for`, on attribue à la k-ème composante du vecteur `v` la fréquence des éléments de l'échantillon `x` qui sont plus petites que `u[k]`. Cela devrait correspondre approximativement à la probabilité que la variable X soit plus petite que `u[k]`. La commande `plt.plot((u,v)` trace donc bien la fonction de répartition empirique de l'échantillon `x`.

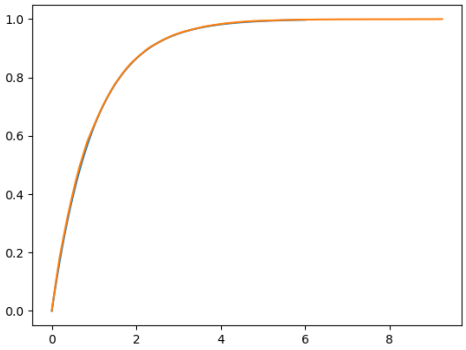
2. On le fait avec la commande suivante :

```
1  #  Echantillon
2  N = 10000
3  lbd = 1
4  x = Exponentielle(lbd, N)
```

3. On utilise le code suivant :

```
5  # Fct de rép théorique
6  def F(x):
7      y = 1-np.exp(-lbd*x)
8      return y
9  u = np.linspace(0, 6, 100)
10 v = F(u)
11 plt.plot(u, v)
12
13 # Fct de rép empirique
14 n = 100
15 u = np.linspace(np.min(x), np.max(x), n)
16 v = np.zeros(n)
17 for k in range(n):
18     v[k] = np.mean(x <= u[k])
19 plt.plot(u, v)
20
21 plt.show()
```

On obtient :



Ces fonctions sont très proches l'une de l'autre, notre simulation semble donc pertinente.

Exercice 6

1. On utilise le code suivant :

```
import scipy.special as sp
p1 = sp.ndtr(0)
p2 = sp.ndtr(1)
p1 = sp.ndtr(1.96)
print(p1, p2, p3)
```

   On obtient respectivement les valeurs `0.5`, `0.8413447` et `0.9750021`.

2. On a :

$$P(X > 10) = 1 - P(X \leq 10) = 1 - P(X - 3 \leq 7) = 1 - P\left(\frac{X-3}{2} \leq \frac{7}{2}\right) = 1 - \Phi(3.5)$$

   À l'aide de la commande :

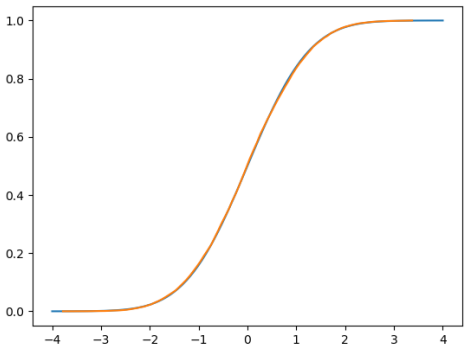
```
p = sp.ndtr(3.5)
print(1-p)
```

   on obtient que cette probabilité vaut `0.0002326290790355401` . On procède de même pour l'autre probabilité.

3. Pour la loi $\mathcal{N}(0,1)$, avec un échantillon de taille $N = 10000$:

```
#  Echantillon
N = 10000
m = 0
s = 1
x = Normale(m, s, N)

# Fct de rép théorique
import scipy.special as sp
u = np.linspace(-4, 4, 100)
v = sp.ndtr(u)
plt.plot(u, v)

# Fct de rép empirique
n = 100
u = np.linspace(np.min(x), np.max(x), n)
v = np.zeros(n)
for k in range(n):
    v[k] = np.mean(x <= u[k])
plt.plot(u, v)

plt.show()
```

   On obtient :

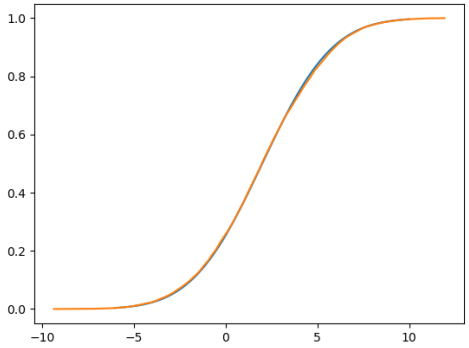


   Ainsi la simulation de la loi $\mathcal{N}(0,1)$ par la fonction `Normale` semble pertinente.

De même pour la loi $\mathcal{N}(2,9)$:

```
# Echantillon
N = 10000
m = 2
s = 3
x = Normale(m, s, N)

# Fct de rép théorique
import scipy.special as sp
u = np.linspace(-6, 10, 100)
v = sp.ndtr((u-m)/s)
plt.plot(u, v)

# Fct de rép empirique
n = 100
u = np.linspace(np.min(x), np.max(x), n)
v = np.zeros(n)
for k in range(n):
    v[k] = np.mean(x <= u[k])
plt.plot(u, v)

plt.show()
```

On obtient :



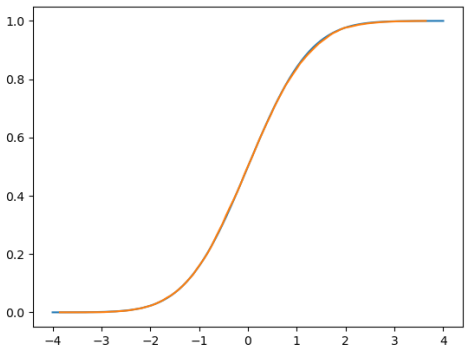
Là aussi, notre simulation semble pertinente.

4. Avec la fonction `rd.normal` :

```
# Echantillon
N = 10000
m = 0
s = 1
x = rd.normal(m, s, N)

# Fct de rép théorique
import scipy.special as sp
u = np.linspace(-4, 4, 100)
v = sp.ndtr(u)
plt.plot(u, v)

# Fct de rép empirique
n = 100
u = np.linspace(np.min(x), np.max(x), n)
v = np.zeros(n)
for k in range(n):
    v[k] = np.mean(x <= u[k])
plt.plot(u, v)
```

```
20
21 plt.show()
```

On obtient :



Les résultats obtenus avec `rd.normal` sont similaires de ceux obtenus avec la fonction `Normale`.

Exercice 7

1. (a) Sur $X(\Omega) =]0;\ln(2)[$, $F(x) = 2(1-e^{-x})$ donc F est dérivable et $F'(x) = 2e^{-x} > 0$.
F est ainsi continue et strictement croissante sur $]0;\ln(2)[$. Elle réalise donc une bijection de $]0;\ln(2)[$ dans $]0;1[$.
Soit $y \in]0;1[$ fixé. Résolvons l'équation $F(x) = y$ d'inconnue $x \in]0;\ln(2)[$:

$$F(x) = y \Leftrightarrow 2(1-e^{-x}) = y \Leftrightarrow e^{-x} = 1 - \frac{y}{2} \Leftrightarrow x = -\ln\left(1-\frac{y}{2}\right).$$

Ainsi, $\forall y \in]0;1[$, $F^{-1}(y) = -\ln\left(1-\frac{y}{2}\right)$.

(b) Voici la fonction demandée :

```
1 def simulX():
2     U = rd.random()
3     X = -np.log(1-U/2)
4     return(X)
```

(c) Voici la fonction demandée :

```
1 def SimulX(N):
2     X = np.zeros(N)
3     for k in range(N):
4         X[k] = simulX()
5     return(X)
```

2. (a) Voici les instructions demandées :

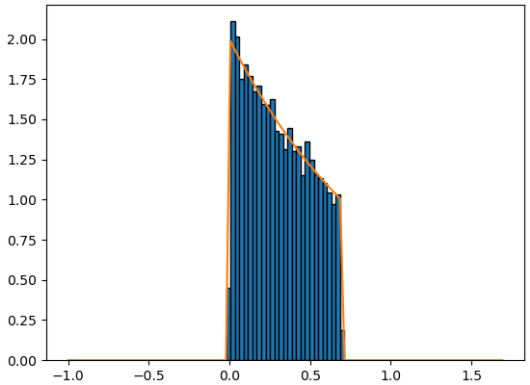
```
1 x = SimulX(10000)
2 c = np.linspace(-1, np.log(2)+1, 100)
3 plt.hist(x, c, density='True', edgecolor='k')
```

(b) On ajoute à la suite des instructions précédentes (après avoir déterminer une densité en dérivant la fonction de répartition de X partout où elle est dérivable) :

```
4 def f(x):
5     if x<0 :
6         y = 0
7     elif x>np.log(2):
8         y = 0
```

```
    else:
        y = 2*np.exp(-x)
    return(y)

t = np.linspace(-1, np.log(2)+1, 100)
y = np.zeros(100)
for k in range(100):
    y[k] = f(t[k])
plt.plot(t, y)

plt.show()
```

On obtient :



On observe graphiquement que les aires des rectangles formant l'histogramme des fréquences sont proches des aires délimitées par la courbe représentative de la densité. On peut donc conclure que `SimulX` renvoie bien des simulations suivant la loi de X.

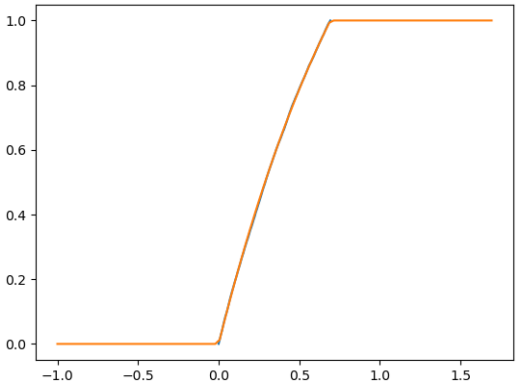
3. (a) Voici les instructions demandées :

```
x = SimulX(10000)
u = np.linspace(np.min(x), np.max(x), 100)
v = np.zeros(100)
for k in range(100):
    v[k] = np.mean(x <= u[k])
plt.plot(u, v)
```

(b) On ajoute à la suite des instructions précédentes :

```
def F(x):
    if x<0 :
        y = 0
    elif x>np.log(2):
        y = 1
    else:
        y = 2*(1-np.exp(-x))
    return(y)

u = np.linspace(-1, np.log(2)+1, 100)
v = np.zeros(100)
for k in range(100):
    v[k] = F(u[k])
plt.plot(u, v)

plt.show()
```

On obtient :



Ces fonctions sont très proches l'une de l'autre (elles se superposent même). On peut donc conclure de nouveau que `SimulX` renvoie bien des simulations suivant la loi de X.

Exercice 8

1. La fonction f_λ est continue sur $\mathbb{R}$ sauf peut-être en λ. Elle est positive en prenant $r \geq 0$. Enfin :

$$\int_{-\infty}^{+\infty} f_\lambda(x)dx = \int_{-\infty}^{\lambda} 0dx + \int_{\lambda}^{+\infty} \frac{r}{x^{k+1}}dx = r \times \int_{\lambda}^{+\infty} \frac{1}{x^{k+1}}dx.$$

Si $A \geq \lambda$,

$$\int_{\lambda}^{A} \frac{1}{x^{k+1}}dx = \left[-\frac{1}{kx^k}\right]_{\lambda}^{A} = \frac{1}{k\lambda^k} - \frac{1}{kA^k} \underset{A\to+\infty}{\longrightarrow} \frac{1}{k\lambda^k}.$$

Ainsi, en posant $r = k\lambda^k$, f_λ est une densité.

2. Comme $X(\Omega) = [\lambda, +\infty[$, $F_X(x) = 0$ si $x < \lambda$.

Si $x \geq \lambda$, on a en utilisant les calculs de la question 1 :

$$F_X(x) = \int_{-\infty}^{x} f_\lambda(t)dt = \int_{-\infty}^{\lambda} 0dt + \int_{\lambda}^{x} \frac{r}{t^{k+1}}dt = 1 - \frac{\lambda^k}{x^k}.$$

3. Sur $X(\Omega) =]\lambda; +\infty[$, $F(x) = 1 - \dfrac{\lambda^k}{x^k}$ donc F est dérivable et $F'(x) = \dfrac{r}{x^{k+1}} > 0$.

F est ainsi continue et strictement croissante sur $]\lambda; +\infty[$. Elle réalise donc une bijection de $]\lambda; +\infty[$ dans $]0; 1[$.

Soit $y \in]0; 1[$ fixé. Résolvons l'équation $F(x) = y$ d'inconnue $x \in]\lambda; +\infty[$:

$$F(x) = y \Leftrightarrow 1 - \frac{\lambda^k}{x^k} = y \Leftrightarrow \frac{\lambda^k}{x^k} = 1 - y \Leftrightarrow x^k = \frac{\lambda^k}{1-y} \Leftrightarrow x = \frac{\lambda}{\sqrt[k]{1-y}}.$$

Ainsi, $\forall y \in]0; 1[$, $F^{-1}(y) = \dfrac{\lambda}{\sqrt[k]{1-y}}$.

On en déduit alors la fonction suivante pour simuler une loi de Pareto de paramètres λ et k avec la méthode d'inversion :

```
def Pareto1(lbd, k):
    U = rd.random()
    X = lbd/(1-U)**(1/k)
    return(X)
```

4. (a) Pour tout i, $X_i(\Omega) =]0, 1]$ donc $(\max(X_1, \ldots, X_k))(\Omega) =]0, 1]$ et donc $Y(\Omega) = [\lambda, +\infty[$. Donc $F_Y(x) = 0$ si $x < \lambda$.

Si $x \geq \lambda$, on a :

$$\begin{aligned}
F_Y(x) &= P(Y \leq x) = P\left(\frac{\lambda}{x} \leq \max(X_1, \ldots, X_k)\right) \\
&= 1 - P\left(\max(X_1, \ldots, X_k) < \frac{\lambda}{x}\right) = 1 - P\left(\bigcap_{i=1}^{k}(X_i < \frac{\lambda}{x})\right) \\
&= 1 - \prod_{i=1}^{k} P(X_i < \frac{\lambda}{x}) \quad \text{(par indépendance des } X_i\text{)} \\
&= 1 - \prod_{i=1}^{k} P(X_i \leq \frac{\lambda}{x}) \quad \text{(car les } X_i \text{ sont à densité)} \\
&= 1 - \prod_{i=1}^{k} \frac{\lambda}{x} \quad \text{(car } X_i \hookrightarrow \mathcal{U}(]0,1]) \text{ et } \frac{\lambda}{x} \in]0,1]) \\
&= 1 - \frac{\lambda^k}{x^k}.
\end{aligned}$$

Donc Y suit une loi de Pareto de paramètres λ et k.

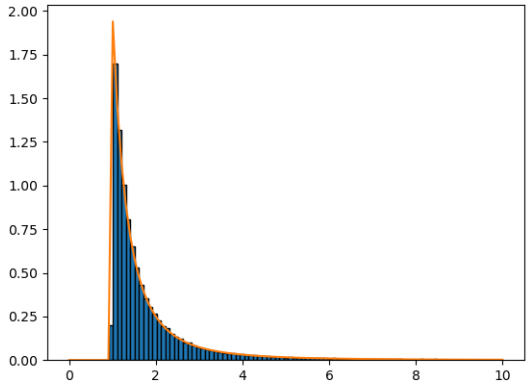
(b) On en déduit le programme suivant :

```
def Pareto2(lbd, k):
    U = rd.random(k)
    Y = lbd/np.max(U)
    return(Y)
```

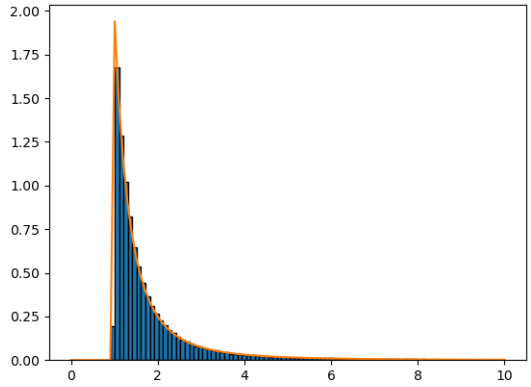
5. Voici le script pour obtenir le temps d'exécution et le graphique histogramme des fréquences/densité théorique pour `Perato1` :

```
tps1 = time.time()
x = np.zeros(100000)
for i in range(100000):
    x[i] = Pareto1(1,2)
tps2 = time.time()
print(tps2-tps1)

c = np.linspace(0, 10, 100)
plt.hist(x, c, density='True', edgecolor='k')

def f(x):
    if x<1:
        y = 0
    else:
        y = 2/x**3
    return(y)

t = np.linspace(0, 10, 100)
y = np.zeros(100)
for i in range(100):
    y[i] = f(t[i])
plt.plot(t, y)
plt.show()
```

On obtient un temps d'exécution de `0.07978` seconde et le graphique suivant :



En faisant de même avec la fonction `Perato2`. On obtient un temps d'exécution de `0.57268` seconde et le graphique suivant :



On constate sur les deux graphiques que les aires des rectangles formant l'histogramme des fréquences sont proches des aires délimitées par la courbe représentative de la densité. On peut donc conclure que les fonctions `Perato1` et `Perato2` renvoie bien des simulations d'une loi de Pareto.

On remarque également que la fonction `Perato1` s'exécute (beaucoup) plus rapidement que `Perato2`. Elle est donc plus performante. On peut expliquer cette différence par le fait qu'on ne fait qu'une simulation d'une loi uniforme sur $[0, 1]$ dans `Perato1` (avec `rd.random()`) et k dans `Perato2` (avec `rd.random(k)`).
