♦ TP11 – Simulations de variables aléatoires par inversion de la fonction de répartition

On rappelle que le module `random` comporte la fonction `random()` qui renvoie un nombre flottant au hasard appartenant à l'intervalle $[0\,;1[$. Dans toute la suite, on suppose que ce module a été importé via la commande : `from random import *`.

I. — Principe de la méthode

Dans le TP9, nous avons vu comment simuler certaines variables aléatoires à partir d'une fonction simulant une variable de Bernoulli. Cela a été possible car dans chaque cas (loi binomiale et loi géométrique), on savait interpréter les variables aléatoires dans le cadre d'un schéma de Bernoulli (nombre de succès et rang du premier succès). Cependant, ce n'est pas toujours le cas. Par exemple, nous n'avons pas d'interprétation simple de la loi de Poisson.

Nous allons voir dans ce TP comment simuler une variable aléatoire uniquement à partir de sa loi en utilisant la fonction de répartition.

1) Un premier exemple

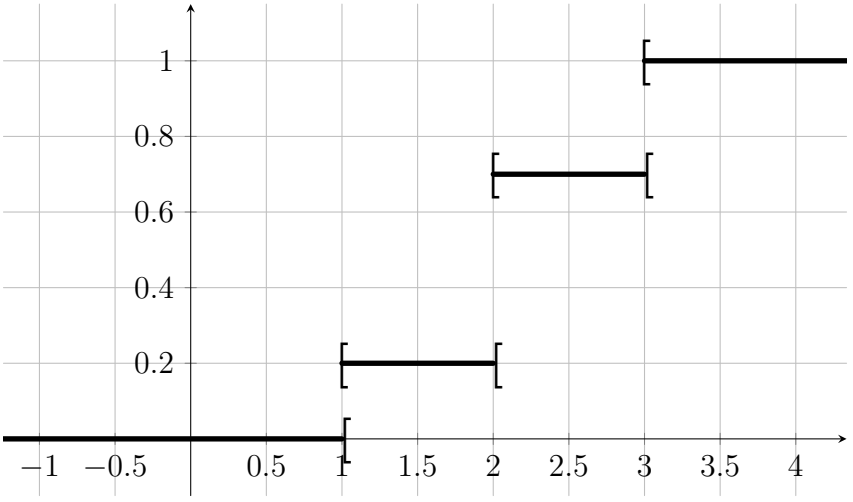
Considérons une variable aléatoire X telle que $X(\Omega) = \{1, 2, 3\}$ et

$$\mathbf{P}(X = 1) = 0{,}2 \qquad \mathbf{P}(X = 2) = 0{,}5 \qquad \mathbf{P}(X = 3) = 0{,}3.$$

La fonction de répartition de X est définie par

$$\forall t \in \mathbb{R},\ F_X(t) = \begin{cases} 0 & \text{si } t < 1 \\ 0{,}2 & \text{si } 1 \leqslant t < 2 \\ 0{,}7 & \text{si } 2 \leqslant t < 3 \\ 1 & \text{si } t \geqslant 3 \end{cases}$$

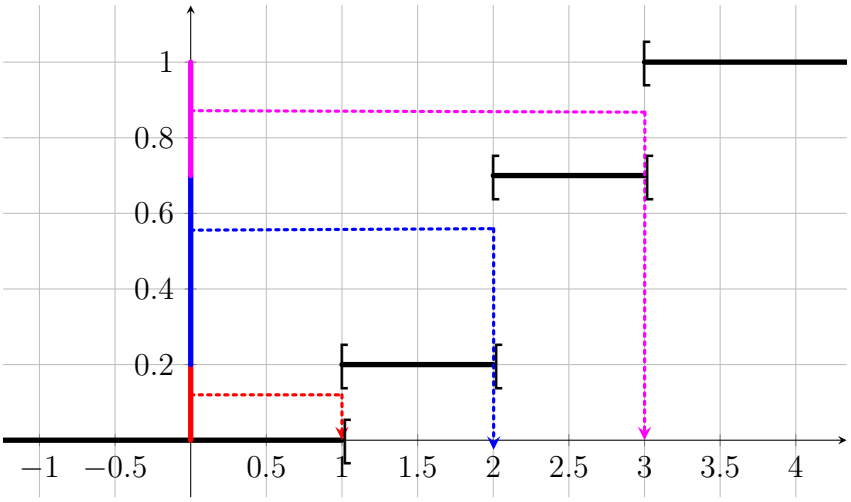
Sa représentation graphique est donc la suivante :



Ainsi, sur l'axe des ordonnées, la courbe de F_X partage l'intervalle $[0\,;1]$ en trois intervalles : $[0\,;0{,}2[$, $[0{,}2\,;0{,}7[$ et $[0{,}7\,;1]$. La longueur des ces intervalles, à savoir 0,2, 0,5 et 0,3 correspondent

exactement aux probabilités $\mathbf{P}(X=1)=0{,}2$, $\mathbf{P}(X=2)=0{,}5$ et $\mathbf{P}(X=3)=0{,}3$ par propriété de la fonction de répartition.

Ainsi, par lecture inverse de la fonction de répartition, on peut associer à chaque nombre de l'intervalle $[0\,;1]$ une des valeurs 1, 2 ou 3.



L'idée pour simuler la variable aléatoire X est de choisir un nombre aléatoire entre 0 et 1 puis, selon que ce nombre appartient à $[0\,;0{,}2[$, $[0{,}2\,;0{,}7[$ ou $[0{,}7\,;1]$, renvoyer la valeur 1, 2 ou 3.

On obtient donc la fonction suivante.

```
def simulX():
  prob = random()
  if prob < 0.2:
    return 1
  elif prob < 0.7:
    return 2
  else:
    return 3
```

2) Un second exemple

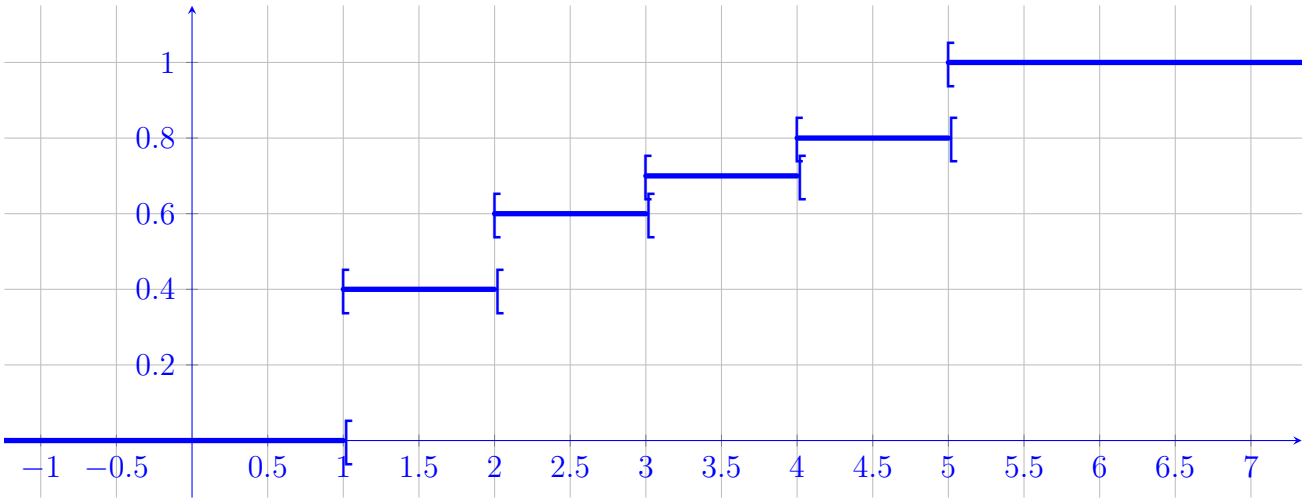
On considère une variable aléatoire Y telle que $Y(\Omega)=\{1,2,3,4,5\}$ et

$$\mathbf{P}(Y=1)=0{,}4 \qquad \mathbf{P}(Y=2)=\mathbf{P}(Y=5)=0{,}2 \qquad \mathbf{P}(Y=3)=\mathbf{P}(Y=4)=0{,}1.$$

Question 1. Tracer la courbe de la fonction de répartition de Y.

Quels sont les 5 intervalles que l'on va considérer dans cette situation ?

Solution.



Les 5 intervalles ici sont $[0\,;0{,}4[$, $[0{,}4\,;0{,}6[$, $[0{,}6\,;0{,}7[$ $[0{,}7\,;0{,}8[$ et $[0{,}8\,;1]$.

Question 2. Écrire une fonction `simulY` qui simule la variable aléatoire Y.

Solution.

```
def simulY():
  prob = random()
  if prob < 0.4:
    return 1
  elif prob < 0.6:
    return 2
  elif prob < 0.7:
    return 3
  elif prob < 0.8:
    return 4
  else:
    return 5
```

Question 3. Calculer (à la main) l'espérance de Y. En utilisant les fonctions implémentées lors du TP9, estimer l'espérance de la variable simulée par la fonction `simulY` et comparer le résultat avec $\mathbf{E}(Y)$.

Solution. L'espérance de Y est

$$\mathbf{E}(Y) = 0{,}4 \times 1 + 0{,}2 \times 2 + 0{,}1 \times 3 + 0{,}1 \times 4 + 0{,}2 \times 5 = 2{,}5$$

On reprend la fonction `moyenne` et on adapte les fonctions `estimation`

```
def moyenne(L):
  S=0
  for e in L:
    S += e
  return S/len(L)

def estimation_Y(N):
  L=[simulY() for i in range(N)]
  return moyenne(L)
```

```
for i in range(5):
  print(estimation_Y(10000))
```

```
2.5035
2.4993
2.5092
2.5281
2.4772
```

On retrouve bien des valeurs proches de 2,5.

3) Principe général pour une variable aléatoire discrète

Le principe général de la méthode d'inversion de la fonction de répartition pour une variable discrète X prenant les valeurs x_0, x_1, x_2, ... est donc de partager l'intervalle $[0\,;1]$ en différents intervalles $[0\,;F_X(x_0)[$, $[F_X(x_0)\,;F_X(x_1)[$, $[F_X(x_1)\,;F_X(x_2)[$, ... puis de choisir un nombre réel aléatoire p entre 0 et 1 et de renvoyer l'unique nombre x_k tel que $p \in [F_X(x_{k-1})\,;F_X(x_k)[$ (avec la convention $F_X(x_{-1}) = 0$).

Ce principe peut s'appliquer aussi bien pour des variables à support fini que pour des variables à support infini. Cependant, lorsque le nombre de valeurs prises par la variable devient grand (ou qu'il est variable), il n'est plus pertinent de programmer comme ci-dessus avec une succession de d'instructions conditionnelles.

Nous allons voir sur divers exemples comment on peut s'y prendre dans ce cas.

II. — Simulation d'une variable suivant une loi binomiale ou une loi géométrique

1) Loi binomiale

Considérons une variable aléatoire X suivant une loi binomiale de paramètres $n \in \mathbb{N}^*$ et $p \in]0\,;1[$.

Pour implémenter la méthode d'inversion de la fonction de répartition F_X, nous allons avoir besoin de connaître les valeurs prises par cette fonction. Pour cela, on peut construire une liste contenant ces valeurs.

Question 4. Écrire une fonction `factorielle` qui prend en argument un entier naturel `n` et qui renvoie la valeur de la factorielle de `n`. On vérifiera que `factorielle(0)` renvoie `1` et que `factorielle(7)` renvoie `5040`.

Solution.

```
def factorielle(n):
  f = 1
  for i in range(n):
    f = (i+1)*f
  return f
```

Question 5. En utilisant la fonction `factorielle`, écrire une fonction `coeff_binom` qui prend en argument un entier naturel `n` et un entier naturel `k` compris entre 0 et `n` et qui renvoie

la valeur du coefficient binomial $\binom{\texttt{n}}{\texttt{k}}$. On vérifiera que `coef_binom(6,3)` renvoie 20 et que `coeff_binom(50,12)` renvoie `121399651100`.

Solution.

```
def coeff_binom(n,k):
  return factorielle(n)//(factorielle(k)*factorielle(n-k))
```

Question 6. Écrire une fonction `loi_binomiale` qui prend en argument un entier naturel `n`, un réel `p` compris entre 0 et 1 et un entier `j` compris entre 0 et `n` et qui renvoie la valeur de $\mathbf{P}(X = \texttt{j})$ où $X \hookrightarrow \mathscr{B}(\texttt{n}, \texttt{p})$. On vérifiera que `loi_binomiale(6,2,0.3)` renvoie (approximativement) `0.324135`.

Solution.

```
def loi_binomiale(n,p,j):
  return coeff_binom(n,j)*p**j*(1-p)**(n-j)
```

Question 7. Soit k un entier compris entre 0 et n. Que vaut $F_X(0)$? Et si $k > 0$, comment peut-on calculer $F_X(k)$ à partir de $F_X(k-1)$?

Solution. Par définition, $F_X(0) = \mathbf{P}(X \leqslant 0) = \mathbf{P}(X = 0) = (1-p)^n$.
Si $k > 0$ alors, comme X est à valeurs entières,

$$F_X(k) = \mathbf{P}(X \leqslant k) = \mathbf{P}(X \leqslant k-1) + \mathbf{P}(X = k) = F_X(k-1) + \mathbf{P}(X = k).$$

Question 8. Écrire une fonction `fonc_rep_binomiale` prenant arguments un entier naturel `n` et un réel `p` compris entre 0 et 1 et qui renvoie la liste des valeurs de la fonction de répartition de X. Ainsi, si la liste renvoyée est `L` alors, pour tout entier `k` compris entre 0 et n, `L[k]` devra être égal à $F_X(k)$. On aura intérêt à utiliser la question précédente. On vérifiera que `fonc_rep_binomiale(5,0.2)` renvoie (approximativement) `[0.32768, 0.73728, 0.94208, 0.99328, 0.99968, 1]`.

Solution.

```
def fonc_rep_binomiale(n,p):
  L = [0 for i in range(n+1)]
  L[0] = (1-p)**n
  for i in range(n):
    L[i+1] = L[i] + loi_binomiale(n,p,i+1)
  return L
```

Question 9. Compléter le script de la fonction suivante de telle sorte qu'elle simule X.

```
def simul_binomiale(n,p):
  L = ...
  prob = random()
  k = ...
  while prob ...:
    k += ...
  return ...
```

Solution.

```
def simul_binomiale(n,p):
  L = fonc_rep_binomiale(n,p)
  prob = random()
  k = 0
    while prob >= L[k]:
      k += 1
  return k
```

Question 10. On souhaite écrire une fonction `freq_binomiale` qui permet de déterminer la fréquence d'un entier j obtenue lorsqu'on effectue N simulation d'une variable $X \hookrightarrow \mathscr{B}(n,p)$. Lequel des deux scripts suivants est préférable ? On expliquera son choix.

```
def freq_binomiale(n,p,j,N):
  c = 0
  for i in range(N):
    if simul_binomiale(n,p) == j:
      c += 1
  return c/N
```

```
def freq_binomiale(n,p,j,N):
  L = fonc_rep_binomiale(n,p)
  c = 0
  for i in range(N):
    prob = random()
    k = 0
    while prob >= L[k]:
      k += 1
    if j == k:
      c += 1
  return c/N
```

Solution. On a intérêt à utiliser le script de droite car il calcule une seule fois la liste contenant les valeurs de la fonction de répartition alors que celui de gauche recalculer tous les termes de cette liste à chaque itération. Si la valeur de n est grande, cela prendre donc beaucoup plus de temps.

Question 11. Comparer, pour quelques valeurs de `n`, `p` et `j` de votre choix, les valeurs renvoyées par `freq_binomiale(n,p,j,10000)` et `loi_binomiale(n,p,j)`.

Solution.

```
print(freq_binomiale(6,0.2,3,10000), loi_binomiale(6,0.2,3))
print(freq_binomiale(20,0.7,10,10000), loi_binomiale(20,0.7,10))
print(freq_binomiale(14,0.1,3,10000), loi_binomiale(14,0.1,3))
print(freq_binomiale(50,0.5,20,10000), loi_binomiale(50,0.5,20))
print(freq_binomiale(10,0.9,8,10000), loi_binomiale(10,0.9,8))
```

```
0.0805 0.08192000000000003
0.0316 0.030817080900085066
0.1095 0.11422705697676006
0.0429 0.041859149252552186
0.1957 0.19371024449999993
```

On constate que les valeurs obtenues sont relativement proches des probabilités.

2) Loi géométrique

Dans le cas particulier d'une loi géométrique, il est possible de calculer explicitement les valeurs de la fonction de répartition, ce que nous allons exploiter.

On suppose ici que X est une variable aléatoire suivant une loi géométrique de paramètre $p \in]0\,;1[$. On note F_X sa fonction de répartition.

Question 12. Montrer que, pour tout $k \in \mathbb{N}$, $F_X(k) = 1-(1-p)^k$.

Solution. Soit $k \in \mathbb{N}$.

$$\begin{aligned} F_X(k) = \mathbf{P}(X \leqslant k) &= \sum_{j=1}^{k}(1-p)^{j-1}p = p\sum_{j=1}^{k}(1-p)^{j-1} \underset{i=j-1}{=} p\sum_{i=0}^{k-1}(1-p)^i \\ &= p\frac{1-(1-p)^k}{1-(1-p)} = p\frac{1-(1-p)^k}{p} \\ &= 1-(1-p)^k \end{aligned}$$

Question 13. En utilisant la question précédente et en adaptant le fonction `simul_binomiale`, écrire une fonction `simul_geo` qui prend en argument un réel `p` compris entre 0 et 1 et qui simule X par la méthode d'inversion de la fonction de répartition.

Solution.

```
def simul_geo(p):
  prob = random()
  k = 1
  while prob >= 1 - (1-p)**k:
    k += 1
  return k
```

Question 14. Écrire une fonction `freq_geo` qui prend en argument un réel `p` strictement compris entre 0 et 1, un entier naturel non nul `j` et un entier naturel `N` et qui renvoie la fréquence de `j` obtenus lorsqu'on effectue `N` simulations d'une variable $X \hookrightarrow \mathscr{G}(\mathtt{p})$ à l'aide de la fonction `simul_geo`.

Solution.

```
def freq_geo(p,j,N):
  c = 0
  for i in range(N):
    if simul_geo(p) == j:
      c += 1
  return c/N
```

Question 15. Comparer, pour quelques valeurs de `p` et `j` de votre choix, les valeurs renvoyées par `freq_geo(p,j,10000)` et la probabilité $\mathbf{P}(X = \mathtt{j})$.

Solution.

```
print(freq_geo(0.2,3,10000), (1-0.2)**(3-1)*0.2)
print(freq_geo(0.7,10,10000), (1-0.7)**(10-1)*0.7)
print(freq_geo(0.1,3,10000), (1-0.1)**(3-1)*0.1)
print(freq_geo(0.5,20,10000), (1-0.5)**(20-1)*0.5)
print(freq_geo(0.9,8,10000), (1-0.9)**(8-1)*0.9)
```

```
0.1307 0.12800000000000003
0.0 1.3778100000000016e-05
0.0839 0.08100000000000002
0.0 9.5367431640625e-07
0.0 8.999999999999987e-08
```

On constate que les résultats sont cohérents même si, lorsque j vient grand, l'évènement $(X = j)$ est très rare et on obtient une fréquence nulle alors que la probabilité n'est pas nulle (mais proche de 0).

Question 16. Soit t un réel compris entre 0 et 1. Démontrer que, pour tout entier naturel $k \geqslant 1$,

$$F_X(k-1) \leqslant t < F_X(k) \iff k \leqslant \frac{\ln(1-t)}{\ln(1-p)} + 1 < k+1$$

Solution. Pour tout $k \in \mathbb{N}^*$,

$$\begin{aligned}
F_X(k-1) \leqslant t < F_X(k) &\iff 1-(1-p)^{k-1} \leqslant t < 1-(1-p)^k \\
&\iff -(1-p)^{k-1} \leqslant t-1 < -(1-p)^k \\
&\iff (1-p)^{k-1} \geqslant 1-t > (1-p)^k \\
&\iff (k-1)\ln(1-p) \geqslant \ln(1-t) > k\ln(1-p) \\
&\underset{\ln(1-p)<0}{\iff} k-1 \leqslant \frac{\ln(1-t)}{\ln(1-p)} < k \\
&\iff k \leqslant \frac{\ln(1-t)}{\ln(1-p)} + 1 < k+1
\end{aligned}$$

Question 17. En utilisant la fonction `int`, déduire de la question précédente une fonction `simul_geo_bis` qui prend en argument un réel `p` strictement compris entre 0 et 1 et qui simule X par la méthode d'inversion de la fonction de répartition sans utiliser de boucle (ni `while` ni `for`). On rappelle qu'en Python, le logarithme népérien s'obtient grâce à la fonction `log` du module `math`.

Solution. La question précédente montre que l'antédécent de t par la fonction F_X est le plus grand entier juste avant $\dfrac{\ln(1-t)}{\ln(1-p)} + 1$ que l'on peut obtenir en Python, pour les nombres positifs, grâce à la fonction `int`.

On obtient ainsi la fonction suivante.

```
from math import log

def simul_geo_bis(p):
  t = random()
  return int(log(1-t)/log(1-p))+1
```

Question 18. Reprendre les questions 14 et 15 en remplaçant la fonction `simul_geo` par la fonction `simul_geo_bis`.

Solution.

```
def freq_geo_bis(p,j,N):
  c = 0
  for i in range(N):
    if simul_geo_bis(p) == j:
      c += 1
  return c/N
```

```
print(freq_geo_bis(0.2,3,10000), (1-0.2)**(3-1)*0.2)
print(freq_geo_bis(0.7,10,10000), (1-0.7)**(10-1)*0.7)
print(freq_geo_bis(0.1,3,10000), (1-0.1)**(3-1)*0.1)
print(freq_geo_bis(0.5,20,10000), (1-0.5)**(20-1)*0.5)
print(freq_geo_bis(0.9,8,10000), (1-0.9)**(8-1)*0.9)
```

```
0.1253 0.12800000000000003
0.0 1.3778100000000016e-05
0.084 0.08100000000000002
0.0 9.5367431640625e-07
0.0 8.999999999999987e-08
```

On aboutit au même constatation que dans la question 15.

III. — Simulation d'une variable suivant une loi de Poisson

Question 19. Écrire une fonction `loi_poisson` qui prend en argument un réel `lamb` et un entier naturel `j` et qui renvoie la valeur de $\mathbf{P}(X = \texttt{j})$ où X suit une loi $\mathscr{P}(\texttt{lamb})$. On rappelle qu'en Python la fonction exponentielle s'obtient gra ?e à la fonction `exp` du modulo `math`. On vérifiera que `loi_Poisson(2,3)` renvoie `0.1804470443154836`.

Solution.

```
def loi_Poisson(lamb,j):
  return exp(-lamb)*lamb**j/factorielle(j)
```

Remarque. En Python, `lambda` est un mot-clé (comme `return` ou `def` ou `for`) qui ne peut pas être utilisé comme nom de variable. Ceci explique pourquoi nous avons appelé le paramètre `lamb` dans la fonction précédente.

Question 20. Soit X une variable aléatoire suivant une loi de Poisson de paramètre λ. Démontrer que, pour tout entier naturel k,

$$\mathtt{P}(X = k+1) = \frac{\lambda}{k+1}\mathbf{P}(X = k).$$

Solution. Soit $k \in \mathbb{N}$. Alors,

$$\mathbf{P}(X = k+1) = \frac{\lambda^{k+1}}{(k+1)!}\mathrm{e}^{-\lambda} = \frac{\lambda \times \lambda^k}{(k+1)k!}\mathrm{e}^{-\lambda} = \frac{\lambda}{k+1} \times \frac{\lambda^k}{k!}\mathrm{e}^{-\lambda} = \frac{\lambda}{k+1}\mathbf{P}(X = k).$$

Question 21. Compléter le script de la fonction suivante afin qu'elle simule une variable aléatoire $X \hookrightarrow \mathscr{P}(\lambda)$ sans utiliser la fonction `factorielle` ni la fonction `loi_Poisson`.

```
from math import exp

def simul_Poisson(lamb):
  prob = random()
  k = ...
  p = ...
  F = ...
  while prob ...:
    p = ...
    F += ...
    k += ...
  return ...
```

Solution.

```
from math import exp

def simul_Poisson(lamb):
  prob = random()
  k = 0
  p = exp(-lamb)
  F = exp(-lamb)
  while prob >= F:
    p = lamb/(k+1)*p
    F += p
    k += 1
  return k
```

Question 22. Écrire une fonction `freq_Poisson` qui prend en argument un réel `lamb` strictement positif, un entier naturel `j` et un entier naturel `N` et qui renvoie la fréquence de `j` obtenus lorsqu'on effectue `N` simulations d'une variable $X \hookrightarrow \mathscr{P}(\texttt{lamb})$ à l'aide de la fonction `simul_Poisson`.

Solution.

```
def freq_Poisson(lamb,j,N):
  c = 0
  for i in range(N):
    if simul_Poisson(lamb) == j:
      c += 1
  return c/N
```

Question 23. Comparer, pour quelques valeurs de `lamb` et `j` de votre choix, les valeurs renvoyées par `freq_Poisson(lamb,j,10000)` et la probabilité $\mathbf{P}(X = \texttt{j})$ où $X \hookrightarrow \mathscr{P}(\texttt{lamb})$.

Solution.

```
print(freq_Poisson(0.2,3,10000), loi_Poisson(0.2,3))
print(freq_Poisson(3.2,10,10000), loi_Poisson(3.2,10))
print(freq_Poisson(5,3,10000), loi_Poisson(5,3))
print(freq_Poisson(2.3,20,10000), loi_Poisson(2.3,20))
print(freq_Poisson(1,8,10000), loi_Poisson(1,8))
```

```
0.0009 0.001091641004103976
0.0011 0.0012647200634353665
0.1419 0.14037389581428056
0.0 7.072204253171058e-13
0.0 9.123994076672677e-06
```

On arrive aux mêmes conclusion que pour la loi géométrique.

IV. — Loi exponentielle

On suppose ici que X suit une loi exponentielle de paramètre $\lambda > 0$. La situation est différente des précédentes puisqu'ici X est une variable aléatoire à densité. Cependant, la méthode d'inversion de la fonction de répartition F_X s'applique tout aussi bien. On va même voir que F_X réalise une bijection de $[0\,;+\infty[$ sur $[0\,;1[$ et on va déterminer sa réciproque.

Question 24. En utilisant uniquement les propriétés du cours sur les fonctions de répartition, justifier que F_X est strictement croissante sur $[0\,;+\infty[$, déterminer $F_X(0)$ et $\lim\limits_{t\to+\infty} F_X(t)$. En déduire que F_X (restreinte à $[0\,;+\infty[$) réalise une bijection de $[0\,;+\infty[$ sur $[0\,;1[$.

Solution. Comme la densité de X est continue sur $[0\,;+\infty[$, la fonction F_X est dérivable sur $[0\,;+\infty[$ et, pour tout $t \geqslant 0$, $F_X'(t) = \lambda \mathrm{e}^{-\lambda t} > 0$ donc F_X est strictement croissante sur $[0\,;+\infty[$. De plus, $F_X(0) = \mathbf{P}(X \leqslant 0) = 0$ car la densité de X est nulle sur $]-\infty\,;0]$ et, par propriété, $\lim\limits_{t\to+\infty} F_X(t) = 1$. Comme, de plus, F_X est continue (car dérivable) sur $[0\,;+\infty[$, on déduit du théorème de la bijection que F_X réalise une bijection de $[0\,;+\infty[$ sur $[0\,;1[$.

Question 25. Calculer explicitement, pour tout $t \geqslant 0$, $F_X(t)$ et en déduire que, pour tout $t \geqslant 0$ et tout $y \in [0\,;1[$,

$$F_X(t) = y \iff t = -\frac{1}{\lambda}\ln(1-y).$$

Solution. Pour tout $t \geqslant 0$,

$$F_X(t) = \mathbf{P}(X \leqslant t) = \int_0^x \lambda \mathrm{e}^{-\lambda t}\,\mathrm{d}t = \left[-\mathrm{e}^{-\lambda x}\right]_0^t = -\mathrm{e}^{-\lambda t} - (-\exp(0)) = 1 - \mathrm{e}^{-\lambda t}.$$

Dès lors, pour tout réel $t \geqslant 0$ et tout réel $y \in [0\,;1[$,

$$F_X(t) = y \iff 1 - \mathrm{e}^{-\lambda t} = y \iff \mathrm{e}^{-\lambda t} = 1 - y \iff -\lambda t = \ln(1-y) \iff t = -\frac{1}{\lambda}\ln(1-y).$$

Question 26. En déduire que si $Y \hookrightarrow \mathscr{U}([0\,;1[)$ alors $Z = -\dfrac{1}{\lambda}\ln(1-Y)$ suit une loi $\mathscr{E}(\lambda)$.

Solution. Déterminons la fonction de répartition de Z. Soit un réel t. Alors, en remarquant que, comme $0 \leqslant Y < 1$, $0 < 1 - Y \leqslant 1$ et donc $Z > 0$, on a $F_Z(t) = 0$ si $t < 0$ et, si $t \geqslant 0$, par croissance F_X,

$$\begin{aligned} F_Z(t) = \mathbf{P}(Z \leqslant t) &= \mathbf{P}\left(F_X\left(-\frac{1}{\lambda}\ln(1-Y)\right) \leqslant F_X(t)\right) \\ &= \mathbf{P}\left(Y \leqslant F_X(t)\right) = \int_0^{F_X(t)} 1\,\mathrm{d}x = F_X(t) \end{aligned}$$

Ainsi, X et Z ont la même fonction de répartition donc elles ont la même loi. Ainsi, $Z \hookrightarrow \mathscr{E}(\lambda)$.

Question 27. En utilisant la question précédente, écrire une fonction `simul_exp` qui prend en argument une réel `lamb` strictement positif et qui simule une variable aléaloire $X \hookrightarrow \mathscr{E}(\texttt{lamb})$.

Solution.

```
def simul_exp(lamb):
  prob = random()
  return -1/lamb*log(1-prob)
```

Question 28. Écrire une fonction `freq_exp` qui prend en argument un réel `lamb` strictement positif, deux réels `a` et `b` (avec `a < b`) posififs et un entier `N` et qui renvoie la fréquence de résultats obtenus compris entre `a` et `b` lorsqu'on effectue `N` simulations d'une variable $X \hookrightarrow \mathscr{P}(\texttt{lamb})$ à l'aide de la fonction `simul_exp`.

Solution.

```
def freq_exp(lamb,a,b,N):
  c = 0
  for i in range(N):
    p = simul_exp(lamb)
    if a <= p and p <= b:
      c += 1
  return c/N
```

Question 29. Soit a et b deux réels et $X \hookrightarrow \mathscr{E}(\lambda)$. Calculer $\mathbf{P}(a \leqslant X \leqslant b)$.

Solution. On peut remarquer que

$$\mathbf{P}(a \leqslant X \leqslant b) = \mathbf{P}(a < X \leqslant b) = F_X(b) - F_X(a) = 1 - \mathrm{e}^{-\lambda b} - (1 - \mathrm{e}^{-\lambda a}) = \mathrm{e}^{-\lambda a} - \mathrm{e}^{-\lambda b}.$$

Question 30. Comparer, pour quelques valeurs de `lamb` et de `a` et `b` de votre choix, les valeurs renvoyées par `freq_exp(lamb,a,b,10000)` et la probabilité $\mathbf{P}(\texttt{a} \leqslant X \leqslant \texttt{b})$ où $X \hookrightarrow \mathscr{P}(\texttt{lamb})$.

Solution.

```
print(freq_exp(0.2,3,5,10000), exp(-0.2*3) - exp(-0.2*5))
print(freq_exp(3.2,10,20,10000), exp(-3.2*10) - exp(-3.2*20))
print(freq_exp(5,0,3,10000), exp(-5*0) - exp(-5*30))
print(freq_exp(2.3,5,20,10000), exp(-2.3*5) - exp(-2.3*20))
print(freq_exp(1,2,8,10000), exp(-1*2) - exp(-1*8))
```

```
0.1832 0.18093219492258406
0.0 1.2664165549094015e-14
1.0 1.0
0.0 1.01300935986307e-05
0.1368 0.13499982060871019
```

Là encore, on arrive aux mêmes conclusions que pour la loi géométrique et la loi de Poisson.

V. — Exercices

Exercice 1 (Loi de Pascal)**.** Soit $r \in \mathbb{N}^*$ et $p \in]0\,;1[$. On dit qu'une variable X suit une loi de Pascal de paramètres r et p si $X(\Omega) = \{k \in \mathbb{N} \mid k \geqslant r\}$ et, pour tout $k \in X(\Omega)$,

$$\mathbf{P}(X = k) = \binom{k-1}{r-1} p^r (1-p)^{k-r}.$$

En utilisant la méthode d'inversion de la fonction de répartition, écrire une fonction `simul_Pascal` qui prend en argument un entier `r` supérieur ou égal à 1 et un réel `p` strictement compris entre 0 et 1 et qui simule une variable suivant une loi de Pascal de paramètres `r` et `p`.

Solution.

```
def simul_Pascal(r,p):
  prob = random()
  k = r
  F = p**r
  while prob >= F:
    k += 1
    F += coeff_binom(k-1,r-1)*p**r*(1-p)**(k-r)
  return k
```

Exercice 2 (Loi de Cauchy)**.** On considère une variable aléatoire X qui suit une loi de Cauchy i.e. une variable aléatoire dont la densité est $f : x \longmapsto \dfrac{1}{\pi(1+x^2)}$ (voir **Chap. 10, Ex. 2, 3.**).

1. Déterminer la fonction de répartition F_X de X.
2. Montrer que F_X réalise une bijection de $\mathbb{R}$ sur $]0\,;1[$ et déterminer F_X^{-1}.
3. En utilisant la question précédente, écrire une fonction `simul_Cauchy` qui simule X.

Solution.

1. Pour tout réel t,

$$F_X(t) = \mathbf{P}(X \leqslant t) = \int_{-\infty}^{t} \frac{1}{\pi(1+x^2)}\,\mathrm{d}x = \frac{1}{\pi}\int_{-\infty}^{t} \frac{1}{1+x^2}\,\mathrm{d}x.$$

Or, si $a < t$,

$$\int_a^t \frac{1}{1+x^2}\,\mathrm{d}x = [\arctan(x)]_a^t = \arctan(t) - \arctan(a) \xrightarrow[a\to\infty]{} \arctan(t) - \left(-\frac{\pi}{2}\right) = \arctan(t) + \frac{\pi}{2}.$$

Ainsi, pour tout $t \in \mathbb{R}$, $F_X(t) = \dfrac{1}{\pi}\left(\arctan(t) + \dfrac{\pi}{2}\right)$.

2. La fonction F_X est continue et strictement croissante sur $\mathbb{R}$ car arctan l'est et $\frac{1}{\pi} > 0$. De plus, $\lim\limits_{t\to-\infty} F_X(t) = 0$ et $\lim\limits_{t\to+\infty} F_X(t) = 1$ donc, par le théorème de la bijection, F_X réalise une bijection de $\mathbb{R}$ sur $]0\,;1[$. De plus, pour tout $y \in]0\,;1[$,

$$F_X(t) = y \iff \frac{1}{\pi}\left(\arctan(t) + \frac{\pi}{2}\right) = y \iff \arctan(t) = \pi y - \frac{\pi}{2} \iff t = \tan\left(\pi y - \frac{\pi}{2}\right)$$

Ainsi, $F_X^{-1} : t \longmapsto \tan\left(\pi t - \frac{\pi}{2}\right)$.

3. En reprenant le même raisonnement que pour la loi exponentielle, on en déduit que si $Y \hookrightarrow \mathscr{U}(]0\,;1[)$ alors $Z = \tan\left(\pi Y - \frac{\pi}{2}\right)$ suit une loi de Cauchy.

On en déduit la fonction suivante.

```
from math import *

def simul_Cauchy():
    prob = random()
    return tan(pi*prob - pi/2)
```

Exercice 3 (Loi de Pareto)**.** On considère une variable aléatoire X qui suit une loi de Paretto de paramètre $a > 0$ i.e. une variable aléatoire dont la densité est

$$f : x \longmapsto \begin{cases} \frac{1}{a}x^{-\frac{a+1}{a}} & \text{si } x \geqslant 1 \\ 0 & \text{sinon} \end{cases}$$

1. Déterminer la fonction de répartition F_X de X sur $[1\,;+\infty[$.

2. Montrer que F_X réalise une bijection de $[1\,;+\infty[$ sur $[0\,;1[$ et déterminer F_X^{-1}.

3. En utilisant la question précédente, écrire une fonction `simul_Pareto` prenant en paramètre un réel `a` strictement positif et qui simule X.

4. En utilisant cette simulation, estimer la valeur de l'expérance de X pour différentes valeurs de $a \in]0\,;1[$ et comparer ces estimations avec la valeur calculée dans l'**Exercice 3** du **Chapitre 10**.

Solution.

1. Pour tout $t \geqslant 1$,

$$F_X(t) = \mathbf{P}(X \leqslant t) = \int_1^t \frac{1}{a}x^{-\frac{a+1}{a}}\,\mathrm{d}x = \left[\frac{1}{a} \times \frac{1}{-\frac{a+1}{a}+1}x^{-\frac{a+1}{a}+1}\right]_1^t = \left[-x^{-\frac{1}{a}}\right]_1^t = 1 - t^{-\frac{1}{a}}$$

2. Pour tout $y \in]0\,;1[$,

$$F_X(t) = y \iff 1 - t^{-\frac{1}{a}} = y \iff t^{-\frac{1}{a}} = 1 - y \iff t = (1-y)^{-a}$$

De plus, comme $0 \leqslant y < 1$, $0 < 1 - y \leqslant 1$ donc, comme $a > 0$, $0 < (1-y)^a \leqslant 1$ et, par inverse, $(1-y)^{-a} \geqslant 1$ donc $(1-y)^{-a} \in [1\,;+\infty[$.

Ainsi, F_X réalise une bijection de $[1\,;+\infty[$ sur $[0\,;1[$ et $F_X^{-1} : t \longmapsto (1-t)^{-a}$.

3. Comme précédemment, on en déduit la fonction suivante :

```
def simul_Pareto(a):
  prob = random()
  return (1-prob)**(-a)
```

4. En utilisant la fonction suivante :

```
def estimation_Pareto(a,N):
  L=[simul_Pareto(a) for i in range(N)]
  return moyenne(L)
```

on obtient les résultats suivants :

```
print(estimation_Pareto(0.5,10000), 1/(1-0.5))
print(estimation_Pareto(0.3,10000), 1/(1-0.33))
print(estimation_Pareto(0.1,10000), 1/(1-0.1))
print(estimation_Pareto(0.01,10000), 1/(1-0.01))
print(estimation_Pareto(0.95,10000), 1/(1-0.95))
print(estimation_Pareto(0.8,10000), 1/(1-0.8))
```

```
1.9927762211699882 2.0
1.446785951774443 1.492537313432836
1.111364027606606 1.1111111111111112
1.010097778708288 1.0101010101010102
8.761876249803898 19.999999999999982
4.583172793468639 5.000000000000001
```

On constate que les résultats sont cohérents sauf dans le cas où le paramètre a est proche de 1.