♦ TP7 – Simulations de variables aléatoires liées à un schéma de Bernoulli

On rappelle que le module `random` comporte la fonction `random()` qui renvoie un nombre flottant au hasard appartenant à l'intervalle $[0\,;1[$ et la fonction `randint(a,b)` qui renvoie un entier aléatoire entre `a` et `b` compris. Dans toute la suite, on suppose que ce module a été importé.

I. — Simulation de variables suivant des lois usuelles

1) Loi de Bernoulli

Question 1. On considère la fonction `simul` suivante :

```
def simul():
  return random()
```

Cette fonction simule une variable aléatoire X. Quelle est le loi de X ?

Solution. X suit une loi uniforme $\mathcal{U}([0\,;1])$.

Question 2. On considère la variable X de la question précédente. Si $p \in]0\,;1[$, que vaut $\mathbf{P}(X \leqslant p)$?

Solution. $\mathbf{P}(X \leqslant p) = \mathbf{P}(X \in [0\,;p]) \displaystyle\int_0^p 1\,\mathrm{d}t = [t]_0^p = p.$

Question 3. Utiliser ce qui précède pour écrire une fonction `bernoulli` prenant en argument un réel `p` appartenant à $]0\,;1[$ et qui simule une variable aléatoire suivant une loi de Bernoulli de paramètre `p`.

Solution.

```
def bernoulli(p):
  if random()<p:
    return 1
  else:
    return 0
```

2) Loi binomiale

Question 4. On souhaite écrire une fonction `bernoulli_liste` qui simule, sous la forme d'une liste, n variables aléatoires indépendantes X_1, X_2, ..., X_n suivant toutes la même loi de Bernoulli de paramètre p.

Parmi les 4 fonctions proposées ci-dessous, certaines conviennent et d'autres non. Identifier lesquelles et expliquer pourquoi il faut éliminer les autres. On vérifiera ensuite ses réponses à l'aide de l'ordinateur.

```
def bernoulli_liste1(n,p):
  L=[bernoulli(p)]*n
  return L
```

```
def bernoulli_liste2(n,p):
  L=[bernoulli(p) for i in range(n)]
  return L
```

```
def bernoulli_liste3(n,p):
  L=[]
  for i in range(n):
    L+=bernoulli(p)
  return L
```

```
def bernoulli_liste4(n,p):
  L=[]
  for i in range(n):
    L.append(bernoulli(p))
  return L
```

Solution. Il faut éliminer la fonction `bernoulli_liste1` car on va créer une liste contenant `n` fois la même valeur (phénomène d'aliasing). Il faut également élimier la fonction `bernoulli_liste1` car la syntaxe `L+=bernoulli(n)` est incorrecte : elle tente de concaténer la liste `L` et l'entier `bernoulli(n)`. Cela aurait été correct si la syntaxe avait été `L+=[bernoulli(n)]`.

Les deux autres fonctions conviennent.

Question 5. On rappelle que si X_1, X_2, ..., X_n sont n variables aléatoires indépendantes suivant toutes la même loi de Bernoulli de paramètre p alors la variable aléatoire $S = X_1 + X_2 + \cdots + X_n$ suit une loi binomiale de paramètres n et p.

En utilisant une des fonctions `bernoulli_liste` convenables de la question précédente, écrire une fonction `binomiale` prenant en argument un entier naturel non nul `n` et un nombre réel `p` appartenant à $]0\,;1[$ et qui simule une variable aléatoire suivant une loi binomiale de paramètres `n` et `p`.

Solution.

```
def binomiale(n,p):
  S=0
  for e in bernoulli_liste(n,p):
    S += e
  return S
```

3) Loi géométrique

Question 6. Rappeler l'interprétation d'une loi géométrique de paramètre p dans le cadre d'un schéma de Bernoulli infini où le succès a une probabilité p.

En déduire une fonction `geometrique` prenant en argument un réel `p` compris entre 0 et 1 et qui simule une variable aléatoire suivant une loi géométrique de paramètre `p`. On utilisera la fonction `bernoulli`.

Solution. La loi géométrique est la loi de la variable égale au rang du premier succès dans le schéma de Bernoulli.

```
def geometrique(p):
  c=1
  while bernoulli(p) == 0:
    c+=1
  return c
```

II. — Estimation de l'espérance

D'après la loi des grands nombres, que nous verrons en fin d'année, lorsqu'on fait la moyenne des valeurs prises par un grand nombre de variables aléatoires indépendantes suivant toutes la même loi alors on obtient un résultat proche de l'espérance commune à ces variables. On va utiliser cela pour estimer les espérances de variables suivant l'une des lois précédentes.

Question 7. Écrire une fonction `moyenne` qui prend en argument une liste `L` de nombres (entiers ou flottants) et qui renvoie la moyenne des éléments de `L`.

Solution.

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

Question 8. Écrire une fonction `estimation_bernoulli` qui prend en argument un entier naturel non nul `N` et un réel `p` compris entre 0 et 1 et qui renvoie une estimation de l'espérance d'une variable suivant une loi de Bernoulli de paramètre p à partir de `N` valeurs d'une telle variable. On utilisera les fonctions `bernoulli_liste` et `moyenne`.

Solution.

```
def estimation_bernoulli(N,p):
  return moyenne(bernoulli_liste(N,p))
```

Question 9. Afficher le résultat renvoyé par `estimation_bernoulli(19000,p)` pour différentes valeurs de `p`. Ceci est-il en accord avec le valeur de l'espérance d'une variable aléatoire suivant une loi de Bernoulli ?

Solution. Voici quelques résultats obtenus :

```
print(estimation_bernoulli(10000,0.8))
print(estimation_bernoulli(10000,0.26))
print(estimation_bernoulli(10000,0.57))
print(estimation_bernoulli(10000,0.98))
```

```
0.8003
0.2263
0.5722
0.983
```

On constate que les valeurs obtenue sont proches du paramètre p donc de l'espérance de la variable.

Question 10. Reprendre, en adaptant, les deux questions précédentes pour estimer l'espérance d'une variable aléatoire suivant une loi binomiale puis pour estimer l'espérance d'une variable aléatoire suivant une loi géométrique.

Solution. Pour la loi binomiale,

```
def estimation_binomiale(N,n,p):
  L=[binomiale(n,p) for i in range(N)]
  return moyenne(L)
```

```
print(estimation_binomiale(10000,20,0.8),20*0.8)
print(estimation_binomiale(10000,100, 0.26),100*0.26)
print(estimation_binomiale(10000,50,0.57),50*0.57)
print(estimation_binomiale(10000,200,0.98),200*0.98)
```

```
16.0153 16.0
25.9384 26.0
28.4911 28.499999999999996
196.0075 196
```

On constate que les résultats obtenus sont proches de np qui est l'espérance de $X \hookrightarrow \mathscr{B}(n,p)$.

Pour la loi géométrique,

```
def estimation_geometrique(N,p):
  L=[geometrique(p) for i in range(N)]
  return moyenne(L)
```

```
print(estimation_geometrique(10000,0.8), 1/0.8)
print(estimation_geometrique(10000,0.26), 1/0.26)
print(estimation_geometrique(10000,0.57), 1/0.57)
print(estimation_geometrique(10000,0.98), 1/0.98)
```

```
1.2581 1.25
3.8178 3.846153846153846
1.7389 1.7543859649122808
1.0206 1.0204081632653061
```

On constate que les résultats obtenus sont proches de $\frac{1}{p}$ qui est l'espérance de $X \hookrightarrow \mathscr{G}(p)$.

III. — Estimation d'un paramètre par intervalle de confiance

On considère un schéma de Bernoulli dont on ne connaît pas le paramètre p. On aimerait obtenir une estimation de p.

Si on considère la variable aléatoire X égale à 1 en cas de succès et 0 en cas d'échec alors $X \hookrightarrow \mathscr{B}(p)$ et $\mathbf{E}(X) = p$. La question 8 donne une façon d'estimer l'espérance de X et donc d'estimer p.

Cependant, dans la pratique, il n'est pas toujours possible de procéder de la sorte pour des raisons de coup financier, de temps disponible ou pour d'autres raisons matérielles. Par exemple, si on veut tester si des boîtes de conserves sont conformes ou pas et qu'il est nécessaire de les ouvrir pour cela, on ne va pas tester toute la production, et même faire un test sur un échantillon de grande taille peut s'avérer extrêmement coûteux.

On va considérer la situation suivante. Une entreprise fabrique des objets. Chaque objet, indépendamment des autres, a une probabilité $p \in [0\,;0{,}1]$ d'être défectueux.

On aimerait estimer la probabilité p à partir d'un échantillon, aussi petit que possible, d'objets prélevés au hasard dans la production.

Question 11. Si on entre l'instruction suivante, à quel intervalle appartient le nombre `p` ?

```
p=random()/10
```

Solution. Par définition, `random()` est un nombre appartenant à $[0\,;1[$ donc `p` appartient $[0\,;0{,}1[$.

Question 12. Écrire une fonction `production` qui prend en argument un entier `N` et qui renvoie une liste `L` de `N` valeurs d'une variable $X \hookrightarrow \mathscr{B}(p)$ où p est un réel inconnu choisi aléatoirement entre 0 et 0,1. On utilisera la fonction `bernoulli_liste`.

Solution.

```
def production(N):
  p=random()/10
  return bernoulli_liste(N,p)
```

Cette liste représente la production de l'usine, la valeur 0 signifiant que l'objet n'a pas de défaut et la valeur 1 signifiant que l'objet possède un défaut.

Question 13. Écrire une fonction `echantillon` qui prend en arguments une liste `L` et un entier `k` inférieur à la longueur de la liste et qui renvoie une liste de `k` éléments choisis aléatoirement à des rangs distincts dans `L`. On pourra utiliser la méthode de listes `.pop()` mais on écrira une fonction sans effet de bord.

Par exemple, si la liste est `L` est `[0,1,1,0,0,0]` et si `k` vaut 4 alors une liste possible est la liste `[1, 0, 0, 0]` obtenue en prenant dans `L` les termes de rangs respectifs 2, 5, 0 et 3.

Solution.

```
def echantillon(L,k):
  M = [e for e in L]
  E = []
  for i in range(k):
    j = randint(0,len(M)-1)
    E.append(M.pop(j))
  return E
```

La fonction `echantillon` permet d'obtenir un échantillon aléatoire de k valeurs de la liste, ce qui représente donc un échantillon de k objets de la production. Notons f la fréquence de 1 dans cet échantillon, ce qui représente la fréquence d'objets défectueux dans l'échantillon.

Un théorème permet d'affirmer que, dans au moins 95% des cas, la probabilité p appartient à l'intervalle $\left[f - \frac{1}{\sqrt{k}} ; f + \frac{1}{\sqrt{k}}\right]$. Cet intervalle est appelé intervalle de confiance de p au seuil de confiance 95%.

Question 14. Écrire une fonction `frequence` qui prend en arguments une liste de nombres `L` et un nombre `x` et qui renvoie la fréquence de `x` dans la liste `L` i.e. le nombre d'occurrences de `x` dans `L` divisé par la longueur de `L`.

Solution.

```
def frequence(L,x):
  c = 0
  for e in L:
    if e == x:
      c += 1
  return c/len(L)
```

Question 15. Soit $m \in \mathbb{N}$. Quelle doit-être la taille k d'un échantillon pour obtenir, grâce à l'intervalle de confiance, un encadrement d'amplitude 10^{-m} de p. (On rappelle que l'amplitude d'un encadrement du type $a \leqslant p \leqslant b$ est le réel $b - a$).

Solution. L'amplitude de l'intervalle de confiance est

$$f + \frac{1}{\sqrt{k}} - \left(f - \frac{1}{\sqrt{k}}\right) = f + \frac{1}{\sqrt{k}} - f + \frac{1}{\sqrt{k}} = \frac{2}{\sqrt{k}}$$

donc cette amplitude est inférieure à 10^{-m} si et seulement si

$$\frac{2}{\sqrt{k}} = 10^{-m} \iff \frac{\sqrt{k}}{2} = 10^{m} \iff \sqrt{k} = 2 \cdot 10^{m} \iff k = 4 \cdot 10^{2m}.$$

Ainsi, pour obtenir une intervalle de confiance d'amplitude 10^{-m}, on doit prendre $k = 4 \cdot 10^{2m}$.

Question 16. Compléter la code de la fonction `int_conf` qui prend en arguments deux entiers naturels `N` et `m`, qui simule, sous forme d'une liste, une production de `N` objets, sélectionne un échantillon dans cette liste de sorte à calculer et renvoyer un encadrement d'amplitude $10^{-\mathtt{m}}$ de la probabilité p.

```
from math import sqrt

def int_conf(N,m):
  L=production(N)
  k=4*10**(2*m)
  M=echantillon(L,k)
  f=frequence(M,1)
  return f-1/sqrt(k) , f+1/sqrt(k)
```

Faire différents tests avec la fonction `int_conf` avec $N = 100\,000$ ou $N = 1\,000\,000$ et $m = 1$ ou $m = 2$ en prenant garde au faite que la taille de l'échantillon doit rester inférieur à la taille de la liste.

Question 17. En utilisant certaines des fonctions précédentes, écrire une fonction `confiance` qui prend en argument une probabilité `p`, une taille de production `N`, une taille d'échantillon `k` et un nombre de simulations `S` et qui renvoie le nombre de points de pourcentage d'intervalles de confiance qui contiennent effectivement la probabilité `p` lors de `S` simulations.

Vérifier, à l'aide de cette fonction, le seuil de confiance de 95% garanti par la théorie.

Solution.

```
from math import sqrt

def confiance(p,N,k,S):
  c = 0
  for i in range(S):
    L=bernoulli_liste(N,p)
    M=echantillon(L,k)
    f=frequence(M,1)
    if f-1/sqrt(k) <= p and p <= f+1/sqrt(k):
      c += 1
  return c/S*100
```

```
print(confiance(0.4, 10000, 500, 1000))
print(confiance(0.2, 10000, 1000, 1000))
print(confiance(0.75, 100000, 2000, 100))
print(confiance(0.1, 10000, 1000, 1000))
```

```
96.6
98.9
99.0
99.8
```

On constate que, dans tous les cas, on est bien au-dessus du seuil des 95%.

IV. — Exercices

Exercice 1. Deux personnes disposent chacune d'une pièce de monnaie qui tombe sur « face » avec une probabilité $p \in]0\,;1[$. Ces deux personnes lancent simultanément leurs pièces et répètent les lancers jusqu'à obtenir « face » pour la première fois en même temps. On note X la variable aléatoire égale au nombre de lancers nécessaires.

1. Écrire une fonction `nb_lancers` qui prend en argument un réel `p` compris entre 0 et 1 et qui simule la variable aléatoire X.
2. Écrire une fonction `estimation_esp` qui prend en arguments un réel `p` strictement compris entre 0 et 1 et un entier N strictement positif et qui permet d'estimer l'espérance de X à partir de `N` simulations de X. On utilisera la fonction `nb_lancers`.
3. En utilisant la fonction précédente avec `N` égal à 10 000, estimer l'espérance de X pour p valant 0,3, 0,5 et 0,9.
4. Montrer que X suit une loi géométrique dont on exprimera le paramètre en fonction de p puis vérifier la pertinence des estimations de la question précédente.

Solution.

1.

```
def nb_lancers(p):
  c = 1
  while random()>p or random()>p:
    c += 1
  return c
```

2.

```
def estimation_esp(p,N):
  S = 0
  for i in range(N):
    S += nb_lancers(p)
  return S/N
```

3.

```
print(estimation_esp(0.3, 10000))
print(estimation_esp(0.5, 10000))
print(estimation_esp(0.9, 10000))
```

```
11.2652
4.0264
1.2461
```

4. L'expérience consiste en un schéma de Bernoulli dont le succès est « le deux personnes obtiennent « face » » et X est la variable aléatoire égale au rang du premier succès donc X suit une loi géométrique. Les lancers des deux personnes étant indépendants, la probabilité de succès est $p \times p = p^2$. Ainsi, $X \hookrightarrow \mathscr{G}(p^2)$. On en déduit que $\mathbf{E}(X) = \dfrac{1}{p^2}$. Ainsi, pour $p = 0{,}3$, $\mathbf{E}(X) = \dfrac{1}{0{,}3^2} \approx 11{,}11$, pour $p = 0{,}5$, $\mathbf{E}(X) = \dfrac{1}{0{,}5^2} = 4$ et, pour $p = 0{,}9$, $\mathbf{E}(X) = \dfrac{1}{0{,}9^2} \approx 1{,}23$. Ainsi, les résultats trouvés à la question précédente sont cohérents.

Exercice 2. On considère une urne qui contient initialement 1 boule blanche et 1 boule noire. On effectue des tirages successifs dans cette urne de la manière suivante :

- si la boule tirée est noire, on s'arrête ;
- si la boule tirée est blanche, on la remet dans l'urne accompagnée d'une autre boule blanche.

On note X la variable aléatoire égale au nombre de tirages effectués i.e. au nombre de tirages nécessaires pour obtenir la boule noire.

1. Écrire une fonction `simul_X` (sans argument) qui simule la variable aléatoire X.

2. Écrire une fonction `loi_X` qui prend en arguments deux entiers `n` et `N` et qui permet d'obtenir une estimation de la probabilité de l'évènement $\{X = \texttt{n}\}$ à partir de `N` simulations de X.

3. En prenant `N` égal à 10 000, estimer les probabilités que X soit égale à 1, à 2, à 3 et à 4.

4. a. Démontrer que, pour tout $n \in \mathbb{N}^*$, $\mathbf{P}(X = n) = \dfrac{1}{n(n+1)}$.

b. Vérifier la pertinence des estimations précédentes.

c. La variable aléatoire X admet-elle une espérance ? Si oui, la calculer.

Solution.

1. On simule le tirage de boules par un tirage d'entiers, en supposant que le boule noire correspond au nombre 1.

```
def simul_X():
  nb_boules = 2
  while randint(1,nb_boules) > 1:
    nb_boules += 1
  return nb_boules - 1
```

2.

```
def loi_X(n,N):
  c = 0
  for i in range(N):
    if simul_X() == n:
      c += 1
  return c/N
```

3.

```
for k in range(1,4):
  print(loi_X(k, 10000))
```

```
0.4968
0.166
0.087
```

4. a. Notons, pour tout $n \in \mathbb{N}^*$, A_k : « Tirer une boule blanche au k-ième tirage ». Alors, pour tout $n \in \mathbb{N}^*$,

$$\{X = n\} = A_1 \cap A_2 \cap \cdots \cap A_{n-1} \cap \overline{A_n}$$

donc, d'après la formule des probabilités composées,

$$\begin{aligned}\mathbf{P}(X = n) &= \mathbf{P}(A_1)\mathbf{P}(A_2 \mid A_1)\mathbf{P}(A_3 \mid A_1 \cap A_2) \cdots \mathbf{P}(A_{n-1} \mid \bigcap_{i=1}^{n-2} A_i)\mathbf{P}(\overline{A_n} \mid \bigcap_{i=1}^{n-1} A_i) \\ &= \frac{1}{2} \times \frac{2}{3} \times \frac{3}{4} \times \cdots \times \frac{n-1}{n} \times \frac{1}{n+1}\end{aligned}$$

Le dénominateur d'un terme se simplifie avec le numérateur du suivant du premier à l'avant-dernier terme donc $\mathbf{P}(X = n) = \frac{1}{n} \times \frac{1}{n+1} = \frac{1}{n(n+1)}$.

b. On en déduit que $\mathbf{P}(X = 1) = \frac{1}{2} = 0{,}5$, $\mathbf{P}(X = 2) = \frac{1}{6} \approx 0{,}167$ et $\mathbf{P}(X = 3) = \frac{1}{12} \approx 0{,}083$. Ces valeurs sont cohérentes avec les estimations de la question **3.**.

c. Comme $n\mathbf{P}(X = n) = \frac{1}{n+1} \underset{n\to+\infty}{\sim} \frac{1}{n}$ et comme $\sum \frac{1}{n}$ diverge, par équivalence, $\sum n\mathbf{P}(X = n)$ diverge. Ainsi, X n'admet pas d'espérance.

Exercice 3. Soit $n \in \mathbb{N}^*$. On considère une urne contenant n boules numérotées de 1 à n. On tire successivement et avec remise deux boules dans l'urne et on note Y le plus grand des deux numéros obtenus.

1. Écrire une fonction `simul_Y` qui prend en argument un entier non nul `n` et qui simule Y.
2. Écrire une fonction `loi_Y` qui prend en arguments deux entiers `n` et `N`, qui simule `N` réalisations de Y et qui renvoie une liste de longueur `n` contenant les fréquences des évènements $\{Y = 1\}$, $\{Y = 2\}$, ..., $\{Y =$ `n`$\}$.
3. Déterminer la loi de Y.
4. Vérifier la pertinence des résultats renvoyés par la fonction `loi_Y` pour différentes valeurs de `n` et de `N`.

Solution.

1.

```
def simul_Y(n):
  a = randint(1,n)
  b = randint(1,n)
  if a > b:
    return a
  else:
    return b
```

2.

```
def loi_Y(n,N):
  L = [0 for i in range(n)]
  for i in range(N):
    L[simul_Y(n)-1] += 1
  for i in range(n):
    L[i] = L[i]/N
  return L
```

3. Notons Z_1 le numéro de la première boule tirée et Z_2 celui de la seconde. Alors, pour tout $k \in [\![0, n]\!]$,

$$\{Y \leqslant k\} = \{Z_1 \leqslant k\} \cap \{Z_2 \leqslant k\}.$$

Comme il y a remise, les tirages sont indépendants dont Z_1 et Z_2 sont indépendantes et ainsi, pour tout $k \in [\![0, n]\!]$,

$$\mathbf{P}(Y \leqslant k) = \mathbf{P}(Z_1 \leqslant k)\mathbf{P}(Z_2 \leqslant k) = \left(\frac{k}{n}\right)^2 = \frac{k^2}{n^2}.$$

On en déduit que, pour tout $k \in [\![1, n]\!]$,

$$\mathbf{P}(Y = k) = \mathbf{P}(Y \leqslant k) - \mathbf{P}(Y \leqslant k-1) = \frac{k^2}{n^2} - \frac{(k-1)^2}{n^2} = \frac{k^2 - (k^2 - 2k + 1)}{n^2}$$

donc $\mathbf{P}(Z \leqslant k) = \dfrac{2k-1}{n^2}$.

4.

```
print(loi_Y(5,10000), [(2*k-1)/5**2 for k in range(1,6)])
print(loi_Y(7,1000), [(2*k-1)/7**2 for k in range(1,8)])
print(loi_Y(10,100000), [(2*k-1)/10**2 for k in range(1,11)])
print(loi_Y(3,1000000), [(2*k-1)/3**2 for k in range(1,4)])
```

```
[0.042, 0.113, 0.2042, 0.2852, 0.3556] [0.04, 0.12, 0.2,
  0.28, 0.36]
[0.028, 0.051, 0.102, 0.144, 0.151, 0.24, 0.284]
  [0.02040816326530612, 0.061224489795918366,
  0.10204081632653061, 0.14285714285714285,
  0.1836734693877551, 0.22448979591836735,
  0.2653061224489796]
[0.01062, 0.02971, 0.04971, 0.07014, 0.09206, 0.10953,
  0.13181, 0.14963, 0.16831, 0.18848] [0.01, 0.03, 0.05,
  0.07, 0.09, 0.11, 0.13, 0.15, 0.17, 0.19]
[0.111139, 0.333525, 0.555336] [0.1111111111111111,
  0.3333333333333333, 0.5555555555555556]
```

On constate que les résultats sont cohérents et qu'ils sont d'autant plus précis que `N` est grand.