Chapitre 4 : Simulation de variables aléatoires discrètes

Dans tout ce chapitre, on utilisera la bibliothèque numpy.random de la manière suivante :

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

1. Utilisation de la fonction rd.random

1.1 Généralités

Python contient un générateur de nombres aléatoires. Pour obtenir un nombre aléatoire, on utilise la fonction random().

A retenir : **rd.random()** renvoie un réel aléatoire entre 0 et 1.
On peut considérer que **rd.random()** $\longrightarrow \mathcal{U}([0;1])$

Rappel :
Si $X \longrightarrow \mathcal{U}([0;1[)$, alors pour tout $p \in [0;1]$, $P(X < p) = p$ et $P(X \geq p) = 1 - p$.

A retenir :

Si $p \in [0;1]$, (rd.random()<p) est un événement de probabilité p

Simuler la loi d'une V.A.R. X, c'est écrire un programme contenant une variable x dont les valeurs sont aléatoires, et qui suit la loi de X.

Exemple :
Une urne contient deux boules bleues et une boule verte. On tire au hasard une boule dans cette urne. On définit une variable boule, qui contient 'B' si on tire une boule bleue et 'V' si on tire une boule verte. Simuler l'expérience aléatoire.

```
if rd.random()<2/3:
   boule='B'
else:
   boule='V'
```

Rappel : **Compteur :** $\begin{cases} \textbf{Initialisation : i = 0} \\ \textbf{A chaque résultat positif : i = i + 1} \end{cases}$

Remarques :
_ On peut considérer que des appels successifs à la fonction random donnent des résultats **indépendants.** Si vous voulez utiliser un résultat, stockez-le immédiatement dans une autre variable (un nouvel appel donnerait une valeur différente).
_ si n est un entier, **rd.random(n)** renvoie un vecteur contenant n valeurs qui suivent toutes la loi U([0;1]), et que l'on peut considérer comme indépendantes.
_ si n et p sont des entiers, **rd.random([n,p])** renvoie une matrice de taille $n \times p$, dont tous les coefficients sont des variables aléatoires indépendantes qui suivent la loi $\mathcal{U}([0,1])$

1.2 Loi de Bernoulli

Soit $p \in]0;1[$ et $X \longrightarrow \mathcal{B}(p)$. (c'est-à-dire : $\begin{cases} X(\Omega) = \{0,1\} \\ P(X = 0) = 1 - p \\ P(X = 1) = p \end{cases}$

Si $\mathbf{X \longrightarrow \mathcal{B}(p)}$, on peut simuler X par :

```
if rd.random() < p :
    x = 1
else :
    x = 0
```

1.4 Loi binomiale

Rappel : On considère une épreuve de Bernoulli de succès de probabilité p. On répète n fois cette épreuve de manière indépendante. Soit X le nombre de succès. Alors $X \longrightarrow \mathcal{B}(n,p)$.

Si $\mathbf{X \longrightarrow \mathcal{B}(n,p)}$, on peut simuler X par :

```
x = 0
for i in range(n):
 if rd.random()<p :
        x = x + 1
```

Remarque : Si un tableau L contient vrai (True) ou faux (False), sum(L) est égal au nombre de "True" contenus dans ce tableau (comme si True = 1 et False = 0).

Exemple : Simulation de la loi B(n,p) (2ème version) :

```
y=rd.random(n)
z=y<p
x=sum(z)
```

1.5 Loi géométrique

Rappel : On considère une épreuve de succès de probabilité p.
On répète cette épreuve de manière indépendante jusqu'à obtenir un succès.
On note X le nombre d'essais nécessaires. Alors $X \longrightarrow \mathcal{G}(p)$.

Si $X \longrightarrow \mathcal{G}(p)$:
```
x = 1
while rd.random()>p :
    x = x + 1
```

Exemple : On considère l'urne précédente. On tire une boule en la remettant, jusqu'à obtenir une boule bleue. Soit X le nombre de tirages nécessaires.

Donc $X \longrightarrow \mathcal{G}(2/3)$

Simulation de la loi de X :

```
x=1
while rd.random()<1/3:
   x=x+1
print(x)
```

2. Simulations directes des lois usuelles

Pour simuler directement les lois usuelles, on peut également utiliser les fonctions de la bibliothèque numpy.random :
Dans tous les exemples, n sont des entiers naturels,

Loi uniforme $\mathcal{U}(\{n_1, \ldots, n_2\})$: **x = rd.randint(n1, n2+1)** *(entiers à partir de n_1 et < n_2 + 1)*

Loi binomiale $\mathcal{B}(N, p)$: **x = rd.binomial(N,p)**

Loi géométrique $\mathcal{G}(p)$: **x = rd.geometric(p)**

Loi de Poisson $\mathcal{P}(\lambda)$: **x = rd.poisson(lambda)**

Remarques :
_ La loi de Bernoulli est une loi binomiale avec N = 1.
_ si n est un entier naturel, rd.randint(n1, n2+1, n), rd.binomial(N,p,n), rd.geometric(p,n), rd.poisson(lambda, n) renvoient un tableau à n coefficients, dont tous les coefficients suivent la même loi, et sont indépendants.
_ De même, en remplaçant n par [n,p], on obtient une matrice de taille n × p ayant les mêmes propriétés.

Exemple :
Simulation de $X \longrightarrow \mathcal{U}([[1;10]])$: x=rd.randint(1,11)

Conclusion : Simulation des lois discrètes usuelles

Avec **import numpy.random as rd**

Loi	**avec rd.random()**	**directement**
$\mathcal{U}(\{n_1, \ldots, n_2\})$		x = rd.randint(n1,n2+1)

$\mathcal{B}(p)$	if rd.random()<p :
	  x = 1
	else :
	  x = 0
		x = rd.binomial(1,p)

$\mathcal{B}(n,p)$	x= 0
	for i in range(n) :
	  if rd.random()<p :
	  x = x + 1
		x = rd.binomial(n, p)

$\mathcal{G}(p)$	x = 1 while rd.random()>p : x = x + 1	x = rd.geometric(p)
$\mathcal{P}(\lambda)$		x = rd.poisson(lambda)