Corrigé

Code de partage avec Capytale : 376d-1541121

Préambule

Quelques précisions pour apprivoiser les commandes `random` permettant de modéliser les lois usuelles. Il faut dans un premier temps importer la librairie `random` que l'on renomme `rd`

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

Lois finies :
- `rd.randint(a,b)` simule une v.a.r. suivant la loi $\mathscr{U}([\![a, b-1]\!])$ (⚠ le b est exclu avec cette librairie)
- `rd.binomial(1,p)` simule une v.a.r. suivant la loi $\mathscr{B}(p)$
- `rd.binomial(n,p)`simule une v.a.r. suivant la loi $\mathscr{B}(n, p)$

Lois infinies :
- `rd.geometric(p)` simule une v.a.r. suivant la loi $\mathscr{G}(p)$
- `rd.poisson(lambda)` simule une v.a.r. suivant la loi $\mathscr{P}(\lambda)$

Les 5 types commandes ci-dessus renvoient un nombre entier.

Répétition d'expériences aléatoires
Les commandes ci-dessous renvoient des tableaux de nombres entiers.
- `rd.loi(parametre1,parametre2,1)` : permet de simuler une v.a.r. suivant la loi indiquée avec son(ses) paramètre(s). On obtient un tableau de nombres entiers de taille 1×1 (i.e. contenant un seul nombre).
- `rd.loi(parametre1,parametre2,N)` : permet de simuler une v.a.r. suivant la loi indiquée avec son(ses) paramètre(s). On obtient un tableau de nombres entiers de taille $1 \times N$
- `rd.loi(parametre1,parametre2,[m,N])` : permet de simuler une v.a.r. suivant toutes la loi indiquée avec son(ses) paramètre(s). On obtient un tableau de nombres entiers de taille $m \times N$

Exercices

Exercice 1

1. Créer un vecteur x contenant 10 simulations de la loi $\mathscr{B}(15; 0,8)$. Refaire et comparer.

   Après avoir importé la bibliothèque `numpy.random`, on utilise `x=rd.binomial(15,0.8,10)`, on créée une variable `x` pour stocker les valeurs de nos simulations afin de pouvoir les analyser plus facilement.
   Comme il s'agit d'une loi binomiale, cela correspond à un nombre de succès, ici après 15 itérations de la même épreuve de Bernoulli (probabilité de succès de $0,8$). Le résultat de chaque simulation sera donc un nombre compris entre 0 et 15 mais comme il y a de l'aléa, si on relance la commande, le résultat est différent. Comme la probabilité de succès est proche de 1, il est peu probable de n'obtenir aucun succès au cours de 15 épreuves (et même $1, 2, 3\ldots$ succès).

2. Même question avec une loi de Poisson de paramètre 1. Combien de valeurs ont été prises dans votre simulation ?

   La commande est cette fois `rd.poisson(1,10)`, j'obtiens les valeurs $\{0, 1, 3, 4, 5\}$ lors de ma simulation (mais cela peut être différent pour vous).

Exercice 2 - loi binomiale

1. Quelle expérience aléatoire permet de simuler une loi $\mathscr{B}(30; 0,6)$

   Il s'agit de la répétition (30 itérations) d'une épreuve de Bernoulli ayant une probabilité de succès de $0,6$

2. Méthode algorithmique : à l'aide d'une boucle `for`, écrire un programme qui simule « à la main » cette loi.

   On peut simuler l'épreuve de Bernoulli à l'aide de la fonction suivante (le résultat sera 1 avec une probabilité de $0,6$ et 0 avec une probabilité de $0,4$.

```
def epreuve():
  a=rand()
  if a<0.6 :
    return 1
  else :
    return 0
```

   On utilise ensuite la fonction et une boucle `for` pour répéter 30 fois l'expérience. On met en place un compteur pour compter le nombre de succès (le résultat de la fonction `epreuve` vaut 1 en cas de succès).

```
c=0
for i in range(1,31):
  c=c+epreuve()
print(c)
```

3. Quelle est la commande prédéfinie de Python qui permet de faire la même chose ?

   `x=rd.binomial(30,0.6)`

Exercice 3 - loi géométrique

1. Quelle expérience aléatoire permet de simuler une loi $\mathscr{G}(0,1)$?

   Il s'agit de la répétition d'une épreuve de Bernoulli jusqu'à l'obtention du premier succès, par exemple lancer une pièce jusqu'à l'obtention du premier « pile » (où la probabilité de faire « pile » vaudrait ici $0,1$).

2. A l'aide d'une boucle `while`, écrire un programme qui permet de simuler « à la main » une loi $\mathscr{G}(0,1)$

   On modélise l'épreuve de Bernoulli par la même fonction qu'à l'exercice précédent (en changeant simplement la probabilité de succès qui vaut $0,1$ ici).

```
def epreuve():
  a=rand()
  if a<0.6 :
    return 1
  else :
    return 0
```

   La boucle `while` est l'outil adapté pour ce genre de situation : « jusqu'à l'obtention du premier succès » . On met en place un compteur pour compter le nombre d'épreuves nécessaire pour obtenir le premier succès.

```
c=0
while epreuve()==0:
  c=c+1

print(c)
```

3. Quelle est la commande prédéfinie de Python qui permet de faire la même chose ?

   `rd.geometric(0.1)`

Exercice 4 (inspiré d'Edhec E 2015)

Un joueur réalise des lancers indépendants d'une pièce truquée donnant « pile » avec la probabilité p. On note N la variable aléatoire égale au rang d'apparition du premier « pile ». Si N prend la valeur n, le joueur place n boules numérotées de 1 à n dans une urne, puis il extrait une boule au hasard de cette urne. On dit que ce joueur a gagné si le numéro porté par la boule tirée est impair. On appelle X la variable aléatoire égale au numéro de la boule extraite.

1. Montrer que avec $m \in \mathbb{N}$, m est pair si et seulement si $2\left\lfloor \frac{m}{2} \right\rfloor = m$

   si m est pair alors $\exists p \in \mathbb{N}, m = 2p$ donc $\frac{m}{2} = p \in \mathbb{N}$ donc $\left\lfloor \frac{m}{2} \right\rfloor = p$ et donc $2\left\lfloor \frac{m}{2} \right\rfloor = 2p = m$

2. Compléter le programme suivant afin qu'il simule l'expérience ci-dessus.

   Dans un premier temps, on simule la série de lancer comme vu à l'exercice **3.**, puis une fois N obtenu on simule le tirage d'un nombre aléatoire entre 1 et N comme à l'exercice **2.**

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

p=float(input("entrer p"))
N=1
while rd.random()>=p :
  N=N+1
X=rd.randint(1,N+1) # le N+1 est exclu ici
if 2*np.floor(X/2)==X :
  print("perdu")
else :
  print("gagné")
```

3. Raccourcir ce programme en utilisant une des fonctions `rd.random`

   Dans le programme précédent, on simule une loi géométrique avec les lignes (lancer d'une pièce jusqu'à l'obtention du premier pile) :

```
N=1
while rd.random()>=p :
  N=N+1
```

   ce que l'on peut remplacer par `N=rd.geometric(p)`