Info

Python : Expériences aléatoires

Bibliothèque : `import numpy.random as rd`

Simulation de lois usuelles : `rd.randint(a,b+1)` pour $\mathcal{U}([\![a,b]\!])$, `rd.binomial(n,p)` pour $\mathcal{B}(n,p)$

Remarque : `rd.randint(a,b+1,m)` ou `rd.binomial(n,p,m)` pour générer un vecteur contenant m valeurs.

Conditions aléatoires : `rd.random()` génère une nombre réel aléatoire uniformément dans le segment $[0,1]$.

Remarque : `rd.random(m)` pour générer un vecteur contenant m valeurs.

Intérêt : Pour tout $p \in [0,1]$, `rd.random() < p` $= \begin{cases} \texttt{True} \text{ avec probabilité } p \\ \texttt{False} \text{ avec probabilité } 1-p \end{cases}$

Exercice 1 (Le classique : loi de Bernoulli, loi binomiale avec `rd.random()`)

Proposer des fonctions qui simulent une réalisation d'une variable aléatoire de loi de Bernoulli/binomiale à l'aide de l'instruction `rd.random()`.

```
def bernoulli(p) :
```

```
def binomiale(n,p) :
```

Exercice 2 (Variable de support $\{1,2,3\}$)

Soient $p, q \in]0,1[$ avec $p+q<1$. On souhaite simuler une réalisation d'une variable aléatoire X de loi suivante :

$$X(\Omega) = \{1,2,3\}, \quad P(X=1) = p, \quad P(X=2) = q, \quad P(X=3) = 1-p-q.$$

Calculer les probabilités conditionnelles suivantes, puis compléter la fonction `alea(p,q)` pour qu'elle simule une réalisation de la variable aléatoire X.

$P_{[X\neq 1]}(X=2) =$ ....

$P_{[X\neq 1]}(X=3) =$ ....

```
def alea(p,q) :
  if .................................. :
    return .......
  elif .................................. :
    return .......
  else :
    return .......
```

Exercice 3 (Tirages avec remise)

Proposer une fonction qui simule k tirages successifs avec remise dans une urne contenant des boules numérotées de 1 à n et qui renvoie un vecteur contenant les k numéros obtenus dans l'ordre.

```
def tirage(p,n) :
```

Exercice 4 (Epreuves de plus en plus difficiles)

On effectue une série de n épreuves de Bernoulli indépendantes. Pour tout $k \in [\![1, n]\!]$ la probabilité de succès à l'épreuve numéro k est $\dfrac{1}{k+1}$.

Proposer une fonction qui simule cette situation et renvoie le nombre de succès obtenus à l'issue des n épreuves.

```
def nombre_succès(n) :
```

Exercice 5 (Premier succès)

On effectue une succession d'épreuves de Bernoulli indépendantes de paramètre $p > 0$ jusqu'à obtenir un succès.

Proposer une fonction qui simule cette situation et renvoie le nombre d'épreuves effectuées.

```
def premier_succès(p) :
```

Exercice 6 (Jeu avec une infinité de niveaux)

Un jeu vidéo est constitué d'une infinité de niveaux, de plus en plus difficiles. Pour tout $k \geqslant 1$, la probabilité qu'un joueur passe le niveau k est $\dfrac{1}{2^k}$. Quand le joueur échoue à un niveau, le jeu s'arrête.

Proposer une fonction qui simule cette situation et renvoie le nombre de niveaux complétés avec succès par le joueur à la fin de la partie.

```
def partie() :
```

Exercice 7 (Tirages jusqu'à ce que...)

On effectue une succession de tirages avec remise dans une urne contenant des boules numérotées de 1 à n, jusqu'à ce que la somme cumulée des numéros des boules obtenues soit supérieure ou égale à n.

Proposer une fonction qui simule cette situation et renvoie le nombre de tirages effectués.

```
def nombre_tirages() :
```

Exercice 8 (Plus difficile : Tirages sans remise)

Soient $1 \leqslant m \leqslant n$. On effectue m tirages sans remise dans une urne contenant des boules numérotés de 1 à n.
Proposer une fonction Python qui renvoie une liste contenant les m numéros obtenus dans l'ordre.
On pourra :

- Se donner une liste `A` représentant les boules présentes dans l'urne (initialement, `A = list(range(1,n+1))`)
- Se donner une liste `B` représentant les boules piochées (initialement, `B = [ ]`)
- A chaque étape, choisir un numéro au hasard présent dans la liste `A` (comment faire cela ?) le retirer de `A` et l'ajouter à `B`.

Quelques instructions sur les listes (hors-programme, mais pratiques) :

- L'instruction `len(L)` renvoie la taille (=nombre d'éléments) de la liste `L`.
- L'instruction `L.append(x)` permet d'ajouter un élément `x` à la fin de la liste `L`.
- L'instruction `L.pop(k)` permet de retirer l'élément d'indice `k` de la liste `L`.