TRAVAUX PRATIQUES n° 15
Variables aléatoires

Dans ce TP, on simule des phénomènes aléatoires avec Python, puis on construit des lois usuelles (Bernoulli, binomiale) et enfin une méthode générale pour simuler une variable aléatoire discrète quelconque.

Dans le module `random`, se trouvent deux fonctions indispensables : `random` et `randint`.

Écrivez dans le shell les lignes suivantes :

```
>>> from random import random, randint
>>> random()
>>> random()
>>> randint(1,6)
>>> randint(10,20)
```

L'appel de la fonction `random()` renvoie un nombre aléatoire U réel compris entre 0 et 1 : ce nombre aléatoire suit la loi uniforme sur l'intervalle $[0,1]$. Elle a la propriété que pour tout intervalle $I \subset [0,1]$, $P(U \in I) = \text{longueur}(I)$.

L'appel de la fonction `randint(n,m)` renvoie un nombre entier aléatoire entre `n` et `m` **compris**, c'est-à-dire un nombre qui suit la loi uniforme sur $[\![n,m]\!]$.

Exercice 1 (*Loi uniforme maison*) :

1) Écrire, à partir **uniquement** de la fonction `random`, votre propre fonction `monrandint(n, m)` qui renvoie un entier aléatoire uniforme dans $[\![n,m]\!]$ (bornes incluses).

   Pour cela, on découpera l'intervalle $[0,1]$ *en* $m-n+1$ *intervalles de même longueur.*

   Rappel : la fonction partie entière se nomme `floor` *et se trouve dans le module* `math`.
2) Écrire un script simulant 100000 variables aléatoires de loi uniforme sur $[\![1,6]\!]$ grâce à votre fonction `monrandint` et calculant le nombre de 6 apparus. Le résultat est-il conforme à vos attentes (environ $100000/6$) ?

Exercice 2 (*Bernoulli et binomiale*) :

1) Écrire une fonction `bernoulli(p)` prenant un paramètre `p`$\in [0,1]$ et simulant une variable aléatoire de loi de Bernoulli de paramètre `p` (renvoie `0` ou `1`).
2) Écrire une fonction `binomiale(n, p)` prenant deux paramètres `n` et `p` et simulant une variable aléatoire de loi binomiale de paramètres `n` et `p` (somme de `n` Bernoulli indépendantes de paramètre `p`).
3) Pour chacune des lois précédentes, écrire une fonction simulant `k` variables aléatoires de cette loi et renvoyant la liste des résultats (`bernoulli_liste(p, k)` et `binomiale_liste(n, p, k)`).
4) Écrire une fonction `effectifs_binomiale(n, p, k)` simulant `k` variables de loi binomiale de paramètres `n` et `p` et renvoyant une liste `L` de taille `n+1` telle que pour tout `i`, `L[i]` est le nombre de fois où `i` est apparu dans la série.

   Par exemple, si `L = [3,0,2]`, cela signifie qu'on a obtenu 3 fois le nombre 0, 0 fois le nombre 1 et 2 fois le nombre 2 (ici $k=5$ et $n=2$).

5) Décommenter puis exécuter le script suivant (après avoir défini `n`, `p`, `k` et calculé `L`). Interpréter le graphique.

```
n, p, k = 10, 0.3, 10000
L = effectifs_binomiale(n, p, k)

import matplotlib.pyplot as plt
x = [i for i in range(n+1)]
plt.bar(x, L)
plt.show()
```

Exercice 3 (*Simulation d'une loi discrète quelconque*) :

1) Écrire une fonction `cumsum(L)` prenant en argument une liste `L` et renvoyant une liste `Lp` telle que pour tout `k`, `Lp[k] = L[0]+L[1]+...+L[k]`.

   Par exemple, `cumsum([0.2, 0.5, 0.3])` renvoie `[0.2, 0.7, 1.0]`.
2) Écrire une fonction `indice(L, x)` prenant en argument une liste `L` de probabilités (sommant à 1) et un réel `x`$\in [0,1[$, et renvoyant le plus petit indice `k` tel que

$$L[0]+\cdots+L[k-1] \leqslant x < L[0]+\cdots+L[k]$$

   (avec la convention que la somme vide vaut 0, donc si `x`$<$`L[0]`, on renvoie `0`).

   Par exemple, si `L = [0.2, 0.5, 0.3]` et `x = 0.4`, alors `indice(L, x)` renvoie `1`.
3) Écrire une fonction `simule(valeurs, probas)` qui prend en argument deux listes de même longueur `valeurs` et `probas` et qui renvoie `valeurs[i]`, où `i = indice(probas, random())`.
4) Expliquer pourquoi la fonction `simule` sert à simuler une variable aléatoire d'univers image `valeurs` et de loi `probas`.
5) **Application** : Un joueur mise 10 euros sur un cheval dans une course. Si le cheval joué gagne la course, le joueur *reçoit* 100 euros (soit un gain net de 90 euros) ; s'il finit 2e, il reçoit 50 euros (gain net 40) ; s'il finit 3e, il reçoit 20 euros (gain net 10) ; sinon il ne récupère rien (gain net -10). La course comporte 20 chevaux et on suppose que tous les ordres d'arrivée sont équiprobables.

   a) Écrire une fonction `gain()` qui simule le **gain net** du joueur.
   b) Écrire une fonction `gains(k)` qui simule `k` fois le gain net du joueur et renvoie la liste des résultats.
   c) Calculer informatiquement une valeur approchée de l'espérance du gain net. Le jeu est-il favorable au joueur ? Comparer le résultat obtenu à l'espérance théorique.
   d) Calculer informatiquement une valeur approchée de la variance du gain net.

Exercice 4 (*Boîtes d'allumettes*) :

On dispose de deux boîtes d'allumettes : la boîte « pile » et la boîte « face », et d'une pièce équilibrée. Au départ, les deux boîtes contiennent 20 allumettes chacune. On tire à pile ou face : si on obtient pile, on enlève une allumette de la boîte « pile » ; si on obtient face, on enlève une allumette de la boîte « face ».

On s'arrête lorsqu'une boîte est vide et on note X le nombre d'allumettes dans la boîte non vide. Ainsi X est une variable aléatoire d'univers image $[\![1,20]\!]$.

1) Écrire une fonction `simule_X()` qui ne prend aucun argument et qui simule une valeur de X.
2) Écrire une fonction `simuleMFois(m)` qui prend en argument un entier `m`, qui simule `m` réalisations de X et qui renvoie une liste `L` de taille 20 telle que pour tout `i`$\in [\![0,19]\!]$, `L[i]` est le nombre d'expériences pour lesquelles le résultat a été `X = i+1`.
3) Réaliser 2000 expériences avec la fonction précédente. En utilisant uniquement les fonctions de base de Python, estimer la moyenne empirique de X et son écart-type.
4) En utilisant la fonction `bar` de `matplotlib.pyplot`, dessiner un diagramme en bâtons des valeurs de `L`.