chapitre 10

Simulations

Table des matières

Le module `random` de Python fournit un certain nombre de fonctions permettant de générer des nombres dits *pseudo-aléatoires*. Il s'agit de nombres générés par des méthodes déterministes mais se comportant « comme » s'ils étaient vraiment aléatoires. Ils nous permettront ici de simuler des expériences aléatoires.

Nous n'utiliserons que deux des fonctions de ce module :

- ⋆ `randint(a,b)` renvoie un entier tiré au hasard entre a et b inclus (a et b doivent être des entiers) ;
- ⋆ `random()` renvoie un réel tiré « au hasard » entre 0 et 1. *Au hasard* signifie ici que les nombres renvoyés sont uniformément distribués sur $[0,1]$: si $0 \leqslant a \leqslant b \leqslant 1$, la probabilité que le nombre renvoyé soit entre a et b vaut $b-a$.

Vous taperez donc au début de votre fichier `.py`, `from random import randint, random`

I Lancers de dés

Exercice 10.1

1. Écrire une fonction `de(n)` qui simule un lancer d'un dé équilibré à n faces.
2. On considère les deux fonctions suivantes :

```python
def somme_1(n):
  return de(n) + de(n)

def somme_2(n):
  return de(n) * 2
```

Ces deux fonctions sont-elles « équivalentes » (simulent-elles la même expérience aléatoire) ? Pourquoi ?

3. Pour estimer expérimentalement la probabilité d'un événement, on répète un grand nombre de fois la même expérience aléatoire en comptant combien de fois l'événement a été réalisé. On obtient ainsi une *fréquence empirique*, dont la *loi des grands nombres* garantit qu'elle converge [1] vers la probabilité cherchée (quand le nombre de répétitions tend vers $+\infty$).

Ici, on sait très bien que (si le générateur de nombres pseudo-aléatoires est correct) la probabilité d'obtenir chacune des faces vaut $\frac{1}{n}$, mais nous allons vérifier que les fréquences obtenues sont cohérentes.

Écrire une fonction `frequences_de_equilibre(nb_faces, nb_exp)` qui renvoie une liste de longueur `nb_faces` dont le k-ème élément contienne la fréquence d'apparition du la k-ème face d'un dé à `nb_faces` faces lors de `nb_exp` lancers simulés.

```
>>> frequence_de_equilibre(4, 10**4)
 [0.2475, 0.2485, 0.2537, 0.2503]
```

Bien sûr, vos résultats peuvent varier (beaucoup si le nombre d'expériences est faible, peu s'il est important).

Exercice 10.2 *Apparition du premier 6*

Une partie consiste à lancer un dé cubique jusqu'à l'obtention du premier 6.

1. Écrire une fonction qui renvoie le nombre de lancers de dé nécessaires jusqu'à l'apparition du premier 6.
2. On effectue n parties. Écrire une fonction qui calcule le nombre de lancers moyen jusqu'à l'apparition du premier 6.

II Tirage dans une urne

Exercice 10.3

Écrire une fonction `avec_remise(nb_tirees, nb_total)` qui simule un tirage avec remise de `nb_tirees` boules dans une urne contenant `nb_total` boules, numérotées de 0 à `nb_total - 1`. On renverra la liste des numéros obtenus (dans l'ordre dans lequel on les a obtenus).

Exercice 10.4

1. en un certain sens un peu compliqué et sous certaines conditions presque systématiquement vérifiées.

Écrire une fonction `sans_remise(nb_tirees, nb_total)` qui simule un tirage sans remise de `nb_tirees` boules dans une urne contenant `nb_total` boules, numérotées de 0 à `nb_total - 1`. On renverra la liste des numéros obtenus, ou `"impossible"` si l'on demande de tirer plus de boules qu'il n'y en a dans l'urne.

On procédera comme suit, on tiendra à jour une liste `urne` contenant les éléments présents dans l'urne. Et pour ce faire on procèdera par slicing, pour supprimer l'élément d'indice `k` on écrit : `urne=urne[:k]+urne[k+1:]`

Exercice 10.5 *Histoire de boules bleues, vertes et rouges.*

Écrire une fonction `urne(n)` qui simule n tirages avec remise dans une urnes contenant des boules bleues, vertes et rouges avec les proportions respectives suivantes : $\frac{1}{4}$, $\frac{1}{4}$ et $\frac{1}{2}$, et qui renvoie le nombre de boules de chaque couleur obtenues.

(*indic.* Pour simuler le tirage des boules, on pourra faire appel à la fonction `random()` et on rappelle ce qui a été dit en préambule sur cette fonction : si $0 \leqslant a \leqslant b \leqslant 1$, la probabilité que le nombre renvoyé soit entre a et b vaut $b - a$.)

III Méthode Monte-Carlo

Exercice 10.6

1. On choisit au hasard un point M de coordonnées (x, y) dans $[0, 1] \times [0, 1]$. Quelle est la probabilité pour qu'il appartienne au quart de disque de centre O de rayon 1 ?
2. Écrire une fonction `monte_carlo(n)` qui, pour n points choisis au hasard dans $[0, 1] \times [0, 1]$, indique la proportion de points appartenant au quart de disque défini précédemment.
   indic.
   - ⋆ Pour tirer un point au hasard dans le carré $[0, 1] \times [0, 1]$ il suffit de tirer son abscisse au hasard et son ordonnée au hasard.
   - ⋆ Pour savoir si un point appartient au disque ou non il suffit de voir si la distance à l'origine est ou n'est pas inférieure ou égale à 1.
3. Compléter le programme précédent afin qu'il renvoie en plus la liste des abscisses, et la liste de ordonnées des points tirés. *Pour que la question suivante fonctionne les résultats seront retournés dans cet ordre* `xlist,ylist,prop = monte_carlo(n)`
4. Compléter le programme précédent avec le bout de code qui vous est fourni dans le dossier *ad hoc* du réseau. Ce bout de code permet de représenter sur un graphique le nuage de points et le quart de disque.

IV Marche aléatoire dans le plan

Exercice 10.7

Soit p un réel de l'intervalle $]0, 1[$, Soit $B \in \mathbb{N}$. On considère le domaine $\mathscr{D}$ suivant :

$$\mathscr{D} = \{(x, y) \in \mathbb{Z}^2 \;/\; (x, y) \in [-B, B] \times [-B, B]\}$$

L'objet de l'exercice est de simuler le déplacement d'une particule mobile M sur le quadrillage $\mathscr{D}$.
Si la particule se situe en (x_0, y_0) elle se déplace en (x_1, y_1) avec les règles suivantes :

- ⋆ $x_1 = x_0$ et $y_1 = y_0 + 1$ avec la probabilité $p/2$ *la particule va en haut*
- ⋆ $x_1 = x_0$ et $y_1 = y_0 - 1$ avec la probabilité $p/2$ *la particule va en bas*
- ⋆ $x_1 = x_0 + 1$ et $y_1 = y_0$ avec la probabilité $(1 - p)/2$ *la particule va à droite*
- ⋆ $x_1 = x_0 - 1$ et $y_1 = y_0$ avec la probabilité $(1 - p)/2$ *la particule va à gauche*
- ⋆ Si $x_0 \in \{-B, B\}$ ou $y_0 \in \{-B, B\}$, la promenade est terminée. *la particule atteint le bord du domaine*

1. Écrire une fonction `deplacement(x,y,p)` qui considère une particule située en (x, y) et retourne les nouvelles coordonnées après un déplacement élémentaire.
2. Écrire une fonction `finpromenade(x,y,B)` prenant en argument les coordonnées (x, y) de la particule et qui renvoie `True` si la particule a atteint la frontière du domaine et qui rend `False` sinon.
3. Une particule est placée sur le point de coordonnées (x, y) dans l'enceinte.
   Écrire une fonction `longueur(x,y,p,B)` qui simule la promenade aléatoire de cette particule jusqu'à ce qu'elle atteigne l'une des frontières et qui calcule la longueur de cette promenade.
4. Écrire une fonction `echantillon(n,p,B)` qui place au hasard n particules dans l'enceinte et calcule la longueur moyenne de leur promenade.
5. Copier/Coller le bout de code permettant de représenter graphiquement le chemin suivi par une particule.