TP N°3 : SIMULATIONS

I Simulation du hasard en Python

Le module `random` propose diverses fonctions permettant de générer des nombres (pseudo-aléatoires) qui suivent des distributions mathématiques. Il y a plusieurs façons d'appeler les fonctions du module `random` mais le plus conseillé est d'écrire `import random as rd` et d'appeler les fonctions à l'aide de `rd.nomdelafonction` pour éviter les éventuels conflits de modules.
On peut considérer que plusieurs appels d'une fonction de ce module donnent des résultats indépendants. Si vous tapez dans le shell `import random` puis `help(random)`, vous aurez toutes les fonctions du module.

Voici quelques exemples extraits du formulaire de l'agro à connaître par coeur :

- la fonction `random()` génère aléatoirement un nombre réel de $[0, 1[$.
- la fonction `randint(a,b)` génère un nombre entier aléatoire dans $[\![a, b]\!]$, c'est à dire une variable aléatoire qui suit une loi uniforme sur $[\![a, b]\!]$.
- la fonction `choice(L)` choisit aléatoirement un élément de la liste `L`.

Simuler un événement de probabilité p

On suppose que le module random est importé de la manière suivante :
`import random as rd`
L'instruction `rd.random( ) < p` ou `rd.random() <= p` permet de simuler informatiquement un événement de probabilité p.

II Exercices de simulations

- **Création de listes en utilisant de l'aléatoire**

1 Ecrire une fonction `infsup(L)` qui renvoie lorsqu'on choisit un élément aléatoire de la liste `L` une liste triée de la manière suivante :
Si $L = [3, 7, 55, 8, 6, 12, 8]$ et le nombre aléatoire vaut 8, la fonction doit renvoyer la liste :

$$[3, 7, 6, 8, 8, 55, 12].$$

2 ♥ Ecrire une fonction `sansremise` de paramètre une liste L et un entier k qui renvoie une liste de k valeurs prises au hasard dans la liste L, sans répétition possible.

✓ **Simulations d'événements aléatoires**

3 Soit $n \in \mathbb{N}^*$. On dispose de n urnes $U_1,\ U_2, \dots, U_n$ avec $(n \in \mathbb{N}^*)$. Pour $1 \leqslant k \leqslant n$, l'urne n°$k$ contient k^2 boules blanches et $n^2 - k^2$ boules noires. On choisit une urne au hasard, puis on tire une boule de cette urne.
Ecrire une fonction Python `simul(n)` qui permet de simuler l'expérience en renvoyant si la boule tirée est noire ou blanche.

✓ ♥ **Estimation d'une probabilité**

On souhaite estimer une probabilité de réalisation d'un événement A lors d'une expérience que l'on sait simuler. Il suffit pour cela de simuler un grand nombre de fois l'expérience et de déterminer la fréquence de réalisation de A (en mathématique, cela correspond à la loi des grands nombres).

4 Deux joueurs lancent tour à tour un dé. Le premier qui obtient un 6 a gagné. On suppose que le joueur A commence.

1) Ecrire une fonction `simul()` permettant de simuler un lancer de dé.
2) Ecrire une fonction `Agagne()` permettant de renvoyer 1 si le joueur A gagne et 0 sinon.
3) Ecrire une fontion `estimA(nbsim)` permettant de calculer la fréquence de réalisation de l'événement "A gagne" sur $nbsim$ parties. Qu'a -t-on estimé ?
4) Faire de même pour le joueur B.

5 **Autour des dérangements** (Cet exercice est à rapprocher de l'exercice n°7 du TD n°3.)
On appelle dérangement toute permutation f de $[\![1, n]\!]$ n'ayant aucun point fixe, c'est à dire que

$$\forall k \in [\![1, n]\!],\ f(k) \neq k.$$

1) Ecrire une fonction `permut(n)` prenant un entier n en argument et renvoyant une permutation d'une liste $1, \cdots, n$.

2) Ecrire une fonction `derangement(L)` prenant en argument une permutation $1, \cdots, n$ qui renvoie `True` si la liste est un dérangement et `False` sinon.

3) Dans cette question, on choisit $n = 50$. Estimer la probabilité qu'une permutation soit un dérangement.

✓ Simulation d'une variable aléatoire, Estimation d'une espérance

• Soit X une variable aléatoire définie sur un espace probabilisé $(\Omega, \mathcal{T}, P)$ lui-même associé à une expérience aléatoire. Si on répète l'expérience un grand nombre de fois, on peut soupçonner que la moyenne des réalisations de$X$ va se stabiliser autour d'une valeur particulière. On dit dans ce cas, on a estimé l'espérance de X. Cela repose aussi sur la loi faible des grands nombres.

6 ♥Une puce se déplace sur l'axe des abscisses en partant de l'origine. A chaque seconde, elle saute d'une unité vers la droite avec la probabilité p ($p \in]0, 1[$) ou vers la gauche avec la probabilité $1 - p$. X_n la position de la puce après n secondes.

1) Ecrire une fonction Python `position(n,p)` de paramètres p et n qui renvoie la position de la puce après n secondes, c'est à dire une simulation de X_n.

2) Ecrire une fonction Python `estimesp (nbsim,n)` qui permet de renvoyer une estimation de l'espérance de X_n, c'est à dire que l'on calcule la moyenne des réalisations sur `nbsim` simulations de X_n.

✓ Loi et histogramme

7 ♥Une urne contient 2 boules blanches et 3 boules noires. On tire les boules une à une au hasard jusqu'à ce qu'il ne reste que des boules d'une seule couleur dans l'urne. Soit Z le nombre de tirages nécessaires.

1) Déterminer $Z(\Omega)$.

2) Ecrire une fonction Python `simulZ()` permettant de renvoyer une simulation de la variable aléatoire Z.

3) Ecrire une fonction `estimproba(nbsim)` qui permet de renvoyer une estimation de $P(Z = 2)$.

4) Ecrire une fonction `estimloi(nbsim)` qui renvoit une liste $[p_0, p_1, p_2]$ où p_i est une estimation de $P(Z = i + 2)$.

5) Ecrire une fonction `estimesp(nbsim)` qui permet de calculer la moyenne de `nbsim` simulations de la variable aléatoire Z. Qu' a t-on estimé ?

6) Tracer un diagramme en bâtons permettant d'obtenir l'histogramme de la loi de Z pour `nbsim` simulations. On pourra la fonction `estimloi(nbsim)` précédente.

   💡 *On pourra utiliser l'instruction du module `matplotlib.pyplot` (`as plt`) `plt.bar(U,Y)` avec U tableau correspond à $Z(\Omega)$ et Y le tableau d'estimation des $(P(Z = k))_{k \in U}$*

7) Déterminer la loi de Z et son espérance par le calcul.

8 **Simulation de la loi de Pascal et estimation de son espérance**

Une bactérie a une probabilité $p = \dfrac{3}{8}$ d'être touchée par un laser. Elle meurt lorsqu'elle est touchée 3 fois. On appelle X la variable aléatoire égale à la durée de vie de la bactérie.

1) Ecrire une fonction `SimulPas(p)` qui permet d'effectuer une réalisation de X. Testez votre fonction pour $p = 3/8$

2) Ecrire une fonction `LST(p,nbsim)` qui simule $nbsim$ réalisations de X et qui renvoie la liste de ces n réalisations.

3) Tracer l'histogramme de la distribution empirique obtenu à partir de $nbsim$ simulations de X.

   💡 *On pourra utiliser l'instruction du module `matplotlib.pyplot` (`as plt`) `plt.hist` permettant de tracer un histogramme.*
   On pourra consulter : http://python-simple.com/python-matplotlib/histogram.php ou taper `help(plt.hist)` dans la console.

4) Ecrire une fonction `moyenne(p,n)` qui renvoie la moyenne de ces réalisations. Testez votre fonction avec $p = 3/8$ et $n = 1000$. Qu'est-ce que cette moyenne permet-elle d'estimer ? Quelle conjecture pouvez vous faire quant à la variable X ?

III Simulation d'une variable aléatoire discrète de loi connue

Dans le cours, on a simulé des variables aléatoires qui suivent des lois de Bernoulli, binomiale, géométrique grâce à leur situation type. Mais ici, on va voir une nouvelle méthode plus générale qui repose sur la loi plutôt que sur la situation type.

Objectif : Soit $n \in \mathbb{N}^*$ et X une variable aléatoire finie avec $X(\omega) = \{x_i, i \in [\![0, n]\!]\}$ et $\forall i \in [\![0, n]\!], P(X = x_i) = p_i$. On suppose que les x_i sont rangés dans l'ordre croissant.
Voici une démarche pour simuler X qui utilise la fonction de répartition de X définie par :

$$F_X(x) = \begin{cases} 0 & \text{si } \ x < x_0 \\ p_0 & \text{si } \ x_0 \leqslant x < x_1 \\ p_0 + p_1 & \text{si } x_1 \leqslant x < x_2 \\ \vdots & \\ p_0 + p_1 + \cdots + p_i & \text{si } x_i \leqslant x < x_{i+1} \\ \vdots & \\ 1 & \text{si } x \geqslant x_n \end{cases}$$

Simulation de X à l'aide de sa fonction de répartition

- On tire un nombre au hasard entre 0 et 1, noté a.
- On calcule le plus petit entier i tel que

$$P(X \leqslant x_i) = p_0 + p_1 + \cdots + p_i > a$$

- On renvoie x_i.

9 Ecrire une fonction Python `simulX(Lunivers , Lloi)` qui renvoie une simulation de la variable aléatoire X qui suit la loi suivante :

$$X(\Omega) = \{x_i, i \in [\![0, n]\!]\} \quad \text{et} \quad \forall i \in [\![0, n]\!], P(X = x_i) = p_i.$$

`Lunivers` est la liste correspondant à $X(\Omega)$ rangée dans l'ordre croissant `Lunivers =[x0,...,xn]` et `Lloi` correspond à la loi de X, `Lloi = [p0,...,pn]`.

10 Adapter la démarche précédente pour écrire une fonction `poisson(lam)` permettant de simuler une variable aléatoire qui suit une loi de Poisson de paramètre `lam`