TP de Python numéro 13 : Simulations de phénomènes aléatoires

*Semaine du **19 juin 2025**.*

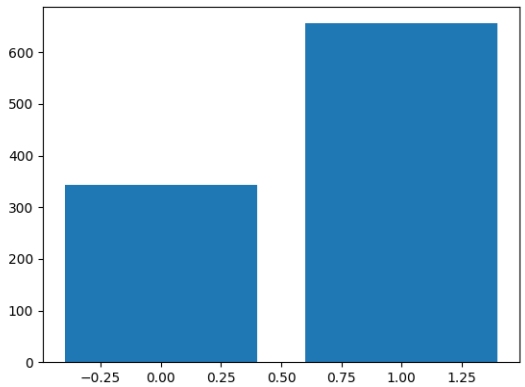
L'objectif de ce TP est de découvrir des outils informatiques permettant de simuler des expériences aléatoires. On utilisera pour cela le module `numpy.random` de Python.

Ces simulations informatiques présentent un avantage majeur : en permettant d'effectuer un grand nombre de simulations d'une expérience aléatoire, on peut utiliser les outils des statistiques pour appréhender une expérience aléatoire et comprendre le comportement de variables aléatoires.

Par exemple, le code suivant simule 1000 lancers d'une pièce à Pile ou face dont la probabilité de faire Pile est 2/3, et affiche le diagramme en bâton, la moyenne empirique et l'espérance empirique qu'on obtient pour cette simulation (On encore "Pile" par 1 et "Face" par 0)

```
import numpy.random as rd
import numpy as np
import matplotlib.pyplot as plt

#Simule un lancer de pièce dont la probabilité de faire "Pile" (noté 1) est donnée en
    argument (p)
def Lancer(p):
    if rd.random()<p:
        return(1)
    return(0)

# 1000 simulations d'un lancer d'une pièce tombant sur "Pile" avec probabilité 2/3
Simu=[Lancer(2/3) for x in range(1000)]

#Affichage des résultats (diagramme en baton)
M,E=np.unique(Simu, return_counts=True)
plt.bar(M,E)
plt.show()

# Moyenne empirique et écart-type empirique
print("Moyenne empirique : ", np.mean(Simu))
print("Variance empirique : ", np.var(Simu))
```

```
>>> (executing cell "" (line 1 of "<tmp 1>"))
Moyenne empirique :  0.656
Variance empirique :  0.22566399999999995
```



On utilise la commande `rd.random()` du module `numpy.random`, qui renvoie *un nombre pseudo-aléatoire* selon *la loi uniforme sur* $[0,1]$.

I. Simulations et nombres pseudo-aléatoires

1. Simulation d'expérience aléatoire en informatique

Commençons par du vocabulaire.

- ***Simuler une expérience aléatoire*** A, c'est réaliser une réaliser une expérience aléatoire B permettant d'imiter le comportement de l'expérience aléatoire A pour tous les aspects de A auxquels on s'intéresse. Par exemple, on peut simuler l'expérience aléatoire A consistant à «lancer une pièce équilibrée à Pile ou Face et à noter le côté obtenu» en réalisant l'expérience aléatoire B consistant à «lancer un dé équilibré à 6 faces» et à noter Pile si le dé tombe sur 1,2 ou 3, et à noter Face sinon. Les probabilités d'apparition des résultats seront inchangées.

  Vous pouvez penser à la chose suivante. Un observateur est face à porte fermée d'une salle close. Lorsqu'il frappe à la porte, une trappe s'ouvre avec un papier sur lequel l'un des mots "Pile" ou "Face" est écrit. On lui a dit qu'à l'intérieur de la salle, une personne (nommée l'expérimentatrice) lance une pièce équilibrée à Pile ou Face et lui écrit sur un papier le résultat (expérience A). A la place, l'expérimentatrice réalise l'expérience B et note sur le papier Pile ou Face selon les termes décrits plus hauts. L'observateur ne pourra jamais savoir qu'on lui a menti, tout se passe exactement comme si l'expérience A était réalisée. En revanche, si l'expérimentatrice lance plutôt une Pièce déséquilibrée (par exemple, Pile avec probabilité 2/3), l'observateur pourra se rendre compte au bout d'un grand nombre de résultats que les fréquences obtenues ne correspondent pas à une pièce équilibrée et, sans jamais en être sûr, son doute quand à la véracité du processus A décrit ne sera que croissant au fur et à mesure qu'il collecte des papiers.

- L'idée de ce TP est de faire simuler des expérience aléatoires à l'ordinateur. Les capacités de calcul de l'ordinateur permettent de réaliser en un instant un grand nombre de simulations d'une même expérience aléatoire.

- Pour pouvoir simuler une expériences aléatoire, l'ordinateur doit avant tout avoir accès à... ne serait-ce qu'une source d'aléa. L'ordinateur ne peut pas lancer de dés ou de pièces, c'est une machine déterministe qui ne résulte que de comportements déterministes de circuits électriques! A la place, on fait produire à l'ordinateur des nombres selon des processus très complexes, qui donnent une illusion de hasard. On parle de ***nombres pseudo-aléatoires***. Ces processus complexes ne sont pas notre objet d'étude, on admettra que l'ordinateur a accès à des générateurs de nombres pseudo-aléatoires, qu'on considérera comme aléatoires.

2. Le module `numpy.random`

Les générateurs de nombres pseudo-aléatoires qu'on utilisera proviennent du module `numpy.random` de `numpy` (d'où le préfixe). Ce module est traditionnellement importé avec l'alias `rd` via la commande :

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

Pour commencer, nous apprendrons à utiliser deux commandes, sur lesquelles toutes nos simulations reposeront dans un premier temps.

a) La commande `rd.randint(a,b)`

La première commande permettant de générer des nombres pseudo-aléatoires est très simple.

La commande `rd.randint`

Soient `a` et `b` deux entiers (type int) tels que $\texttt{a} \leq \texttt{b} - 1$.
Alors, la commande `rd.randint(a,b)` renvoie un nombre entier pseudo-aléatoire choisi uniformément dans l'intervalle $[\![\texttt{a}, \texttt{b} - 1]\!]$.
De plus, les différents appels de cette commande sont considérés indépendants.

Remarque. Par exemple, si X est une variable aléatoire suivant la loi uniforme sur $[\![1, 10]\!]$, alors la commande

`rd.randint(1, 11)`

simule une réalisation de X.

Cette commande simule donc des situations d'équiprobabilités.

Exercice 1. Une urne contient 10 boules indiscernables numérotées de 1 à 10.

1. Écrire le code d'une fonction Python d'entête `def SimuleTirage()` sans entrée et renvoyant en sortie le numéro obtenu via la simulation d'une réalisation de cette expérience aléatoire par l'ordinateur.
2. Écrire une fonction Python d'entête `def MaxTirages(n):` prenant en entrée un entier naturel `n` et simulant `n` tirages avec remise dans cette urne, puis renvoyant en sortie le plus grand des numéros tirés.
3. On note X la variable aléatoire donnant le plus grand numéro tiré lors de 5 tirages successifs avec remise d'une boule dans cette urne. Écrire un code Python permettant, à l'aide d'un grand nombre (10000) de réalisations de X, d'afficher une estimation de la loi de probabilité de X. On utilisera la commande `plt.bar`.

b) La commande `rd.random` et la loi uniforme sur $[0,1]$

Exercice 2. Importer le module `numpy.random` via la commande ci-dessus. Puis :

1. Dans l'interpréteur, taper et exécuter une dizaine de fois la commande `rd.random()`. Rappel : si votre curseur est dans l'interpréteur, la touche "flèche haut" du clavier réécrit la dernière chose écrite. Que remarquez-vous ?
2. Que pourrait bien renvoyer la commande `print(rd.random==rd.random)` ?
3. Vérifiez votre réponse au point ci-dessus. Interprétation ?

Pour l'exercice suivant, on rappelle que si `L` est une série statistique brute et si `C` est une liste donnant les extrémités d'un regroupement en classe, alors `plt.hist(L,C,density=True)` engendre l'histogramme normalisé de la série statistique `L` selon le regroupement en classe décrit pas `C`

Exercice 3. Lisez le code suivant et comprenez le graphique qu'il produit. Copiez et exécutez-le. Que remarquez-vous ? Qu'en déduire sur `rd.random()` ?

```
import numpy as np
import numpy.random as rd
import matplotlib.pyplot as plt

L=[rd.random() for x in range(10000)]
C=np.linspace(0,1,10)
plt.hist(L,C,density=True)
plt.show()
```

La commande `rd.random()`

La commande`rd.random()` renvoie un nombre de type `float` pseudo-aléatoire élément de $[0,1]$ selon les modalités suivantes :

- Les différents appels de cette commande sont considérés indépendants.
- Les appels de cette fonction sont considérés comme suivant une loi dite *uniforme* sur l'ensemble $[0,1]$, comme expliqué ci-dessous.

Qu'est-ce que la loi uniforme sur $[0,1]$? Nous le verrons lors du dernier chapitre de l'année. Il s'agit de ce qui se rapproche le plus de la situation d'équiprobabilité, mais sur l'ensemble infini $[0,1]$.

Définition simplifiée : on dit qu'une variable aléatoire X définie sur un espace probabilisé $(\Omega, \mathcal{A}, \mathbb{P})$ suit la loi uniforme sur $[0,1]$ si :

$$X(\Omega) = [0,1] \text{ et } \forall (a,b) \in [0,1], a \leq b \implies \mathbb{P}(a \leq X \leq b) = b - a.$$

Autrement dit, une telle variable aléatoire prends ses valeurs dans l'ensemble $[0,1]$, et tombe "à chaque endroit avec la même probabilité" : elle tombe dans un sous-intervalle de $[0,1]$ avec une probabilité proportionnelle à sa longueur. En un sens, cela fait que chaque valeur $[0,1]$ a la même chance d'être prise (la formalisation de ce fait est plus subtile que ça). Exemples :

- La probabilité qu'elle tombe dans $[0,1/2]$ est $1/2$ (cet intervalle occupe la moitié de l'intervalle total).
- La probabilité qu'elle tombe dans $[1/4,3/2]$ est $1/2$ (cet intervalle occupe la moitié de l'intervalle total : $3/4 - 1/2 = 1/2$).

- La probabilité qu'elle tombe dans $[0, 1/3]$ est $1/3$.
- La probabilité qu'elle tombe dans $[\frac{1}{\pi}, \frac{1}{\pi} + \frac{1}{10}]$ est $1/10$ (ce raisonnement est valide tant que l'intervalle considéré est inclus dans $[0, 1]$: $\frac{1}{\pi} \simeq 0,32$).

On peut dire que la probabilité uniforme sur $[0, 1]$ est l'analogue continu de l'équiprobabilité discrète.

En particulier, soit X une variable aléatoire suivant la loi uniforme sur $[0, 1]$. Alors, pour tout $p \in [0, 1]$, $[X < p]$ est un événement de probabilité p. Vous comprendrez plus tard pourquoi cet événement a la même probabilité que $[X \leq p]$, pour lequel le raisonnement décrit plus haut s'applique : $[X \leq p]$ est l'événement $[0 \leq X \leq p]$, de probabilité $p - 0 = p$.

Ce fait est utilisé dans quasiment tous les codes Python qu'on écrira, sous la forme suivante.

Idée capitale pour toutes les simulations avec `rd.random()`

Soit `p` un élément de $[0, 1]$.
Alors, la commande `rd.random()<p` renvoie `True` avec probabilité `p`, et `False` avec probabilité `1-p`.
On peut penser à l'événement "`rd.random()<p` renvoie `True`" comme à un événement «générique» de probabilité `p`.

Exercice 4. On lance 6 fois une pièce déséquilibrée à Pile ou Face, dont la probabilité de faire Pile est $1/3$. On note X la variable aléatoire donnant le nombre de Piles obtenus

1. Écrire le code d'une fonction Python d'entête `def SimulExp():` sans entrée, simulant cette expérience aléatoire et renvoyant en sortie le 6-uplet des résultats obtenus, en notant 1 pour Pile et 0 pour Face.
2. Écrire le code d'une fonction `def SimuleX():` simulant un tirage de X (renvoyant en sortie la valeur obtenue pour X).
3. En déduire un graphique donnant une estimation de la loi $\mathcal{B}(6, 1/3)$, et, à l'aide de 10000 tirages de X, donnant l'estimation de $E(X)$ obtenue par moyenne empirique.

Remarque. Le vocabulaire de l'estimation sera précisé en seconde année.

c) Vecteurs aléatoires

Une dernière chose sur ces commandes : `rd.random()` et `rd.randint` peuvent être utilisés avec des arguments optionnels, pour effectuer d'un coup plusieurs appels indépendants de ces commandes.

Vecteurs aléatoires

Soit `N` un entier naturel, non nul(type `int`).
- Pour tous entiers `a` et `b` convenables, la commande `rd.randint(a,b,N)` renvoie un vecteur (type `array`) constitué de `N` appels indépendants de `rd.randint(a,b)`.
- La commande `rd.random(N)` renvoie un vecteur constitué de `N` appels indépendants de `rd.randint()`.

Exemple 5. Testez ces commandes pour quelques valeurs de `a,b,N`.

II. Premiers exercices

Exercice 6. On considère l'expérience aléatoire qui consiste à lancer un pièce équilibrée dont les côtés sont numérotés 0 et 1 et à noter en résultat le numéro du côté obtenu.

1. Proposer un code Python permettant de simuler une réalisation de cette expérience aléatoire.
2. Générer un échantillon de 10000 réalisations de cette expérience aléatoire.
3. Représenter graphiquement une approximation de la loi de probabilité.

Exercice 7. On considère l'expérience aléatoire qui consiste à lancer deux dés équilibrés à six faces numérotées de 1 à 6 et à noter en résultat la distance entre les chiffres obtenus.

1. Proposer un code Python permettant de simuler une réalisation de cette expérience aléatoire.

2. Générer un échantillon de 10000 réalisations de cette expérience aléatoire.
3. Représenter graphiquement une approximation de la loi de probabilité.

Exercice 8. On considère l'expérience aléatoire qui consiste à lancer deux dés équilibrés à six faces numérotées de 1 à 6 et à noter en résultat le plus grand des chiffres obtenus.

1. Proposer un code Python permettant de simuler une réalisation de cette expérience aléatoire.
2. Générer un échantillon de 10000 réalisations de cette expérience aléatoire.
3. Représenter graphiquement une approximation de la loi de probabilité.
4. Déterminer la moyenne statistique de l'échantillon et comparer cette valeur à la moyenne théorique.

Exercice 9. On considère l'expérience aléatoire qui consiste à lancer six fois successivement un dé équilibré à six faces numérotées de 1 à 6 et à noter en résultat le nombre de chiffres 6 obtenus.

1. Proposer un code Python permettant de simuler une réalisation de cette expérience aléatoire.

   Indication : on peut convertir un vecteur numpy nommé A en liste avec la commande list(A).
2. Générer un échantillon de 10000 réalisations de cette expérience aléatoire.
3. Représenter graphiquement une approximation de la loi de probabilité.
4. Déterminer la moyenne statistique de l'échantillon et comparer cette valeur à la moyenne théorique.

Exercice 10. On considère l'expérience aléatoire qui consiste à lancer six fois successivement un dé équilibré à six faces numérotées de 1 à 6 et à noter en résultat le rang d'apparition du premier 6 obtenu si l'on a obtenu au moins une fois le chiffre 6 et 0 si l'on n'a jamais obtenu le chiffre 6 .

1. Proposer un code Python permettant de simuler une réalisation de cette expérience aléatoire.
2. Générer un échantillon de 10000 réalisations de cette expérience aléatoire.
3. Représenter graphiquement une approximation de la loi de probabilité.
4. Déterminer la moyenne statistique de l'échantillon et comparer cette valeur à la moyenne théorique.

Exercice 11. On considère l'expérience aléatoire qui consiste à effectuer 6 lancers indépendants d'une pièce équilibrée dont les côtés sont numérotés 0 et 1 et à noter en résultat le nombre de 0 précédés d'un 0 obtenus.

Par exemple, si les lancers de dés donnent la suite ($0, 0, 1, 0, 0, 0$), alors le nombre noté en résultat vaut 3 .

1. Proposer un code Python permettant de simuler une réalisation de cette expérience aléatoire.
2. Générer un échantillon de 10000 réalisations de cette expérience aléatoire.
3. Représenter graphiquement une approximation de la loi de probabilité.

Exercice 12. Dans cet exercice, on s'intéresse au problème connu sous le nom de "pari du chevalier de Méré". On considère les deux jeux de hasard suivant :

- Jeu 1 : Le joueur répète quatre fois de suite un lancer d'un dé équilibré à six faces numérotées de 1 à 6 et est déclaré gagnant s'il obtient au moins une fois le chiffre 6 sur ses six tentatives.
- Jeu 2 : Le joueur répète vingt-quatre fois de suite un lancer de deux dés équilibrés à six faces numérotées de 1 à 6 et est déclaré gagnant s'il obtient au moins une fois un double 6 sur ses vingt-quatre tentatives.

1. Écrire un code Python permettant de simuler une partie du Jeu 1 (afficher 1 si le joueur gagne et 0 sinon).
2. Écrire un code Python permettant de simuler une partie du Jeu 2 (afficher 1 si le joueur gagne et 0 sinon).
3. Proposer un protocole de simulations permettant de déterminer quel jeu est le plus favorable au joueur.
4. Déterminer ensuite la probabilité théorique que le joueur gagne pour chacun des deux jeux.