TP de Python numéro 12 : Simulations de phénomènes aléatoires

*Semaine du **jeudi 21 mai 2026**.*

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 :

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

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.

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 la 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). À 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 quant 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 expériences 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 ! À 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. Construire une liste contenant 15 nombres entiers tirés au hasard entre -2 et 8 puis l'afficher.

Exercice 2.

1. Écrire une fonction qui simule le lancer de deux dés équilibrés à six faces, et qui renvoie 0 si la somme des deux faces obtenues est différente de 7, et 1 sinon.
2. Écrire une fonction qui prend en entrée un argument `N`, qui réalisera `N` lancers de deux dés, et renverra la fréquence d'apparition du résultat 7.
3. Que devient cette fréquence sur 100 lancers ? 1000 lancers ? ...
4. Quelle estimation de la probabilité d'appartition du 7 peut-on proposer ?

Exercice 3.

1. Écrire une fonction qui permet de tirer aléatoirement 0 ou 1, jusqu'à ce que la somme des nombres tirés valent 8, et qui retourne le nombre X de tirages qui ont été faits.
2. Écrire une fonction prenant en argument `N` le nombre d'expérience effectuée et qui renvoie la moyenne du nombre de tirages X. Tester cette fonction pour $N = 10$, $N = 100$ et $N = 1000$.

Exercice 4. 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`.

Diagramme en bâtons en Python

Notons `M` la liste des issues et `E` la liste donnant, dans le même ordre, le nombre de réalisations de ces issues.
Alors :

- la commande `plt.bar(M,E)` génère le diagramme en bâton représentant les résultats de cette expérience aléatoire
- la commande `plt.show()` demande à Python d'afficher ce graphique.

Ainsi, pour afficher le diagramme en bâtons associé à une expérience aléatoire, on écrira généralement le code suivant.

```
plt.bar(M,E)
plt.show()
```

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

Exercice 5. 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 6. 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(0,1,11)
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 prend 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 de $[0,1]$ 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 $\left[\frac{1}{\pi}, \frac{1}{\pi} + \frac{1}{10}\right]$ 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 7. 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).

Exercice 8. Écrire une fonction qui simule le lancer d'un dé à quatre faces, tel que la face 1 apparaît avec probabilité $\frac{1}{4}$, la face 2 avec probabilité $\frac{1}{2}$, et les faces 3 et 4 avec probabilité $\frac{1}{8}$.

Exercice 9. Sans utiliser la fonction `randint`, et à l'aide de la fonction `floor`, écrire une fonction qui simule le lancer d'un dé équilibré à 6 faces.
Indication : soit $x \in [0,1[$ un réel. À quel intervalle appartient le réel $ax+b$ avec a, b deux entiers naturels ?

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

II. Premiers exercices

Exercice 11. On considère l'expérience aléatoire qui consiste à lancer une 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 12. 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 13. 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 14. 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 15. 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 16. 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 17. Suite de l'exercice 7

1. Donner une estimation de la loi $\mathcal{B}(6, 1/3)$ en utilisant un diagramme à bâtons (puis en traçant le graphe de la fonction de répartition de cette estimation).
2. À l'aide de 10000 tirages de X, donnant l'estimation de $E(X)$ obtenue par moyenne empirique.

Exercice 18. 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.

III. Simulation des lois discrètes usuelles

1. Bibliothèque numpy.random et simulation des lois discrètes usuelles

On dispose de diverses commandes pour simuler des variable aléatoires.

Les lois discrètes usuelles

- La commande `rd.randint(n)` simule une réalisation de la loi uniforme sur $[\![0, n-1]\!]$ avec $n \in \mathbb{N}^*$.
- La commande `rd.randint(a,b)` simule une réalisation de la loi uniforme sur $[\![a, b-1]\!]$ avec $a < b$.
- La commande `rd.binomial(n,p)` simule une réalisation de la loi binomiale $\mathcal{B}(n, p)$.
- La commande `rd.geometric(p)` simule une réalisation de la loi géométrique $\mathcal{G}(p)$.
- La commande `rd.poisson(lambda)` simule une réalisation de la loi géométrique $\mathcal{P}(\lambda)$.

Remarque. On peut obtenir r simulations d'une loi usuelle sous la forme d'un vecteur de taille r ou $r \times s$ simulation sous la forme d'une matrice de $\mathcal{M}_{r,s}(\mathbb{R})$ en ajoutant à ces commandes l'argument `r` ou `[r,s]` respectivement. Par exemple :

- `rd.geometric(p,r)` renvoie un vecteur contenant r simulations de la loi $\mathcal{G}(p)$;
- `rd.poisson(lambda,[r,s])` renvoie une matrice de $\mathcal{M}_{r,s}(\mathbb{R})$ contenant $r \times s$ simulations de la loi $\mathcal{P}(\lambda)$.

Toutefois, le but de cette partie est de réaliser des simulations **à partir uniquement de la fonction `random`.**

Exercice 19. Soient n et m deux entiers tels que $n < m$.

1. Écrire une fonction `uniforme(n,m)` simulant la loi $\mathcal{U}([\![n, m-1]\!])$.
2. Écrire une fonction `Uniforme(n,m,N)` afin qu'elle revoie un vecteur contenant N réalisations indépendantes de la loi $\mathcal{U}([\![n, m-1]\!])$.

Exercice 20. Soit $p \in [0, 1]$.

1. Écrire une fonction `bernoulli(p)` simulant la loi $\mathcal{B}(p)$.
2. Écrire une fonction `Bernoulli(p,N)` afin qu'elle revoie un vecteur contenant N réalisations indépendantes de la loi $\mathcal{B}(p)$.

Exercice 21. Soit $p \in [0, 1]$ et $n \in \mathbb{N}^*$.

1. Écrire une fonction `binomiale(n,p)` simulant la loi $\mathcal{B}(n, p)$.
2. Écrire une fonction `Binomiale(n,p,N)` afin qu'elle revoie un vecteur contenant N réalisations indépendantes de la loi $\mathcal{B}(n, p)$.

Exercice 22. Soit $p \in]0, 1]$.

1. Écrire une fonction `geometrique(p)` simulant la loi $\mathcal{G}(p)$.
2. Écrire une fonction `Geometrique(p,N)` afin qu'elle revoie un vecteur contenant N réalisations indépendantes de la loi $\mathcal{G}(p)$.
3. Simuler un échantillon de taille $N = 10000$ de la loi $\mathcal{G}(0, 2)$, puis vérifier que la valeur moyenne de cet échantillon est cohérente avec ce qu'on attend.

2. Comparaison diagramme en bâtons des fréquences / probabilités théoriques

Soit X une variable aléatoire discrète. Supposons qu'on dispose d'une fonction `Loi` permettant de simuler la loi de X. Pour juger la pertinence des simulations, on peut utiliser des représentations graphiques. Pour cela, on procèdera comme suit :

- on crée un échantillon de taille N c'est-à-dire une liste contenant N réalisations de la fonction `Loi`
- on compare graphiquement les fréquences empiriques obtenues avec les probabilités théoriques pour vérifier la pertinence de la simulation.

Dans le cas de simulations de variables aléatoires discrètes, on va comparer plus particulièrement :

- le diagramme en bâtons des fréquences de notre échantillon de taille N
- le diagramme en bâtons des probabilités théoriques $P(X = k)$ pour $k \in X(\Omega)$

Remarque. Pour réaliser deux graphiques dans une même fenêtre et ainsi pouvoir mieux les comparer, on peut utiliser l'instruction `plt.subplot(1,2,k)` avant chaque instruction de tracé de graphique, qui découpe la fenêtre graphique en 1 ligne et 2 colonnes, k indiquant le numéro de la colonne souhaitée pour chaque graphique

Exercice 23. Tester la simulation des lois des exercices 19 à 22

3. Complément : Méthode d'inversion discrète

La méthode d'inversion discrète est une méthode générale pour simuler la loi d'une variable aléatoire discrète X partie d'une variable aléatoire U suivant la loi uniforme sur $[0,1]$. Elle consiste à "inverser" la fonction de répartition de X

Exemple 24. On souhaite simuler une variable aléatoire X ayant pour ensemble image $X(\Omega) = \{1,2,3\}$ et pour probabilités ponctuelles :

$$P(X=1) = 0.5, \quad P(X=2) = 0.3 \quad \text{et} \quad P(X=3) = 0.2$$

On va découper pour cela l'intervalle $[0,1]$ en trois intervalles de longueurs 0.5,0.3 et 0.2.
Pour simuler la variable aléatoire X, on tire au hasard un nombre $t \in [0,1]$ avec la fonction `rd.random()` et on retourne :

- $x = 1$ si $t \in [0, 0.5]$, ce qui se produit avec la probabilité 0.5 ;
- $x = 2$ si $t \in]0.5, 0.8]$, ce qui se produit avec la probabilité 0.3 ;
- $x = 3$ si $t \in]0.8, 1]$, ce qui se produit avec la probabilité 0.2.

Exercice 25. On souhaite simuler une loi uniforme $\mathcal{U}([\![1,6]\!])$.

1. Déterminer le découpage correspondant de l'intervalle $[0,1]$ pour cette loi.
2. Écrire une fonction `unif()` simulant la loi $\mathcal{U}([\![1,6]\!])$ à l'aide de la méthode d'inversion.

Cas général. Soit à présent X une variable aléatoire discrète qui prend les valeurs $x_1 < x_2 < \ldots x_k < \ldots$. Rappelons que sa fonction de répartition est en escalier.

Exercice 26. Tracer la fonction de répartition de la géométrique de paramètre $\frac{1}{2}$ Pour simuler la variable aléatoire discrète X, on procèdera comme suit :

(i) on choisit le paramètre t aléatoirement dans $[0,1]$ à l'aide de la fonction `rd.random()` ;

(ii) on déterminer l'intervalle $I_k =]F_X(x_{k-1}), F_X(x_k)]$ tel que $t \in I_k$ (si $k = 1, I_1 = [0, F_X(x_1)]$) ;

(iii) on retourne x_k

Remarque. On admet que ce programme retourne x_k avec une probabilité $P(X = x_k)$

Exercice 27. On souhaite simuler une loi de Poisson de paramètre $\lambda > 0$ par la méthode d'inversion.

1. Déterminer la fonction de répartition de cette loi de Poisson (en conservant le symbole Σ)
2. On tire au hasard un nombre $t \in [0,1]$. Dans quel intervalle I_k le nombre t doit se trouver pour qu'on retourne k ?
3. En utilisant la méthode d'inversion, écrire une fonction `poisson(lambda)` simulant la loi $\mathcal{P}(lambda)$.
4. Écrire une fonction `Poisson(lambda,N)` afin qu'elle revoie un vecteur contenant N réalisations indépendantes de la loi $\mathcal{P}(lambda)$.
5. À l'aide d'un diagramme en bâtons, juger de la pertinence de cette simulation pour $\lambda = 5$. On admet pour cela que les cas où la variable prend une valeur supérieure à 10 sont négligeables.

Exercice 28. Écrire une fonction qui prend en paramètre un nombre $p \in]0,1[$, puis à l'aide de la méthode d'inversion, simule une loi géométrique de paramètre p.

IV. Simulation de variables aléatoires à densité

1. Bibliothèque numpy.random et simulation des lois à densité usuelles

On dispose de diverses commandes pour simuler des variable aléatoires.

Les lois à densité usuelles

- La commande `rd.random()` simule une réalisation de la loi uniforme sur $[0,1]$.
- La commande `rd.exponential(1/a)` simule une réalisation de la loi exponentielle $\mathcal{E}(a)$ de paramètre $a>0$.
- La commande `rd.normal(m,sigma)` simule une réalisation de la loi normale $\mathcal{N}(m,\sigma^2)$ de paramètres $m\in\mathbb{R}$ et $\sigma>0$.

Remarque. On peut obtenir r simulations d'une loi usuelle sous la forme d'un vecteur de taille r ou $r\times s$ simulation sous la forme d'une matrice de $\mathcal{M}_{r,s}(\mathbb{R})$ en ajoutant à ces commandes l'argument `r` ou `[r,s]` respectivement. Par exemple :

- `rd.exponential(1/a,r)` renvoie un vecteur contenant r simulations de la loi $\mathcal{E}(a)$;
- `rd.normal(m,sigma,[r,s])` renvoie une matrice de $\mathcal{M}_{r,s}(\mathbb{R})$ contenant $r\times s$ simulations de la loi $\mathcal{N}(m,\sigma^2)$.

Remarque. Attention à certains paramètres :

- le paramètre de la loi exponentielle choisi par Python est l'inverse de celui du cours.
- le second paramètre de la loi normal est σ et non σ^2.
- par défaut, `rd.normal()` simule la loi normale centrée réduite.

Toutefois, le but de cette partie est de réaliser des simulations **à partir uniquement de la fonction `random`.**

2. Méthode d'inversion

Théorème 29. *On suppose que X est une variable aléatoire à densité de densité f telle que*

$$X(\Omega)=\{t\in\mathbb{R}\,|\,f(t)>0\}=]a,b[\ \text{ où }\ -\infty\leq a<b\leq+\infty$$

Alors :

- *la fonction de répartition F de X réalise une bijection de $]a,b[$ sur $]0,1[$;*
- *si U suit la loi $\mathcal{U}(]0,1[)$, $F^{-1}(U)$ suit la même loi que X.*

Méthode : Simulation à l'aide de la méthode d'inversion

Soit X une variable aléatoire à densité, F sa fonction de répartition. Supposons qu'on dispose d'une expression explicite de F^{-1}. Pour simuler la variable X, on procèdera comme suit :

1. on choisit un paramètre t de manière aléatoire dans $]0,1[$ à l'aide de la fonction `rd.random()` ;
2. on retourne $F^{-1}(t)$.

Exercice 30.

1. Rappeler l'expression de la fonction de répartition d'une loi exponentielle, et montrer qu'elle réalise une bijection de $\mathbb{R}_+^*$ sur $]0,1[$. Déterminer sa bijection réciproque.
2. (a) Écrire une fonction `exponentielle(lambda)` simulant une loi $\mathcal{E}(\lambda)$. à partir de la fonction `rd.random()`.
   (b) Écrire une fonction `Exponentielle(lambda,N)` donnant un échantillon de taille N de la loi $\mathcal{E}(\lambda)$.
3. (a) Créer un vecteur de taille 10000 contenant 10000 simulations d'une variable aléatoire suivant la loi $\mathcal{E}(1/2)$.
   (b) En utilisant les commandes `np.mean` et `np.std` de la librairie `numpy`, vérifier que la moyenne et l'écart-type empiriques (c'est-à-dire de ce vecteur) sont bien conformes à ce qu'on attend.