.

Thème 2 : Simulations de Lois

Avant-Propos : Conformément au programme officiel, ce thème du programme aborde des simulations de lois *toutes* attendues.

Lois usuelles étudiées

Dans cette thématique, nous proposons sur plusieurs séances, d'étudier comparativement les résultats de simulations de lois usuelles par appels de commandes Python et au regard de la répartition théoriques. Un exemple concret de situation sera ensuite abordé.
Les lois étudiées sont :

- Les lois uniformes disctètes $\mathcal{U}[\![a;b]\!]$, uniforme continue $\mathcal{U}[0;1[$ et plus généralement $\mathcal{U}[a;b[$
- La loi binomiale $\mathcal{B}(n;p)$ ainsi que les lois géométrique $\mathcal{G}(p)$ et de Poisson $\mathcal{P}(\lambda)$
- La loi exponentielle $\mathcal{E}(\lambda)$
- La loi normale $\mathcal{N}(0;1)$ centrée-réduite et sa famille $\mathcal{N}(\mu;\sigma^2)$

Tous les TP de ce thème utiliseront les bibliothèques d'importation suivantes :

```
from math import*
import numpy as np
import numpy.random as rd
import matplotlib.pyplot as plt
```

Séance I : Les lois uniformes

Voici un programme permettant de générer 100 simulations de variables aléatoires de loi uniforme sur $[0;1[$ et de tracer un histogramme de répartition des résultats :

```
N=100
Lunif = np.zeros([N])
for k in range(N):
    Lunif[k] = rd.random()
Gu=plt.hist(Lunif, range = (0, 1), bins = 10, color = 'green', edgecolor = 'red')
plt.show()
```

1. Adaptez ce script pour générer une liste de 1000 tirages de loi $\mathcal{U}[0;1[$ et afficher l'histogramme associé
2. Comment modifier la ligne `Lunif[k] = rd.random()` pour obtenir des valeurs de type `float` entre -2 (inclus) et 10 (exclu) ?
   Générer ainsi un histogramme associé.
3. Modifier le script proposé pour en faire une fonction `SimUnif`(a,b,N) qui génère une liste de N tirages de loi uniforme sur $\mathcal{U}[a;b[$
4. Quelle devrait être l'aspect théorique de l'histogramme si N est arbitrairement grand ?
5. On note $p=\dfrac{1}{\sqrt{\pi}}$. Générer une liste de 10 000 tirages de loi $\mathcal{U}[-p;p[$ au moyen de `SimUnif`(a,b,N)
   A l'aide de Python, produisez un histogramme de répartition des valeurs obtenus *adapté* et comparer les valeurs de moyenne et variance empiriques avec les valeurs d'espérance et variance théoriques. (on pourra préférer comparer les écart-types)
6. <u>En version discrète :</u>
   Adapter le script qui précède pour simuler des lois <u>discrètes</u> entières entre a et b :
   (a) En utilisant la commande `rd.randint(a,b)`

(b) En complétant le script suivant :

```
def ........(a,b,N):
    Lunab = np.zeros([N])
    for ..................:
        Lunab[k] = (b+1-a)*rd.random()+a
        Lunab[k] = ........................
....
```

7. Une application : Dans un jeu de rôle, on peut générer les caractéristiques d'un personnage en lançant trois dés à 6 faces équilibrés et en multipliant le résultat par 5. On notera alors $C = 3D6 \times 5$ la caractéristique C obtenue.
   Pour tester une caractéristique C, on lance un dé à 100 faces ($D100$) et on considère que le test est réussi si $D100 \leq C$.

   Générer le profil d'un personnage à 10 caractéristiques $C_1 \dots C_{10}$ puis tester chacune de ces caractéristiques. Votre programme devra renvoyer la fréquence $f \in [0; 1[$ de tests réussis.
   Encore plus loin ! Générer 1000 profils puis créer un histogramme des fréquences de réussites des tests de caractéristiques.

Séances II et III : Autour des schémas de Bernoulli

Voici un programme permettant de générer 100 simulations de variables aléatoires de loi binomiale $\mathcal{B}(n; p)$ et de tracer un histogramme de répartition des résultats en fonction des paramètres d'entrée choisi :

```
# Loi binomiale
def Binom(n,p,N):
    X = np.zeros([N])
    for k in range(N):
        X[k] = rd.binomial(n,p)
    Gu = plt.hist(X, range = (0, n), bins = n)
    return(Gu)
```

On propose un exemple d'appel à la console :

```
>>> Gu=Binom(21,0.33,1000)
>>> plt.show()
```

1. Quelles sont les valeurs de paramètres utilisés pour la loi $\mathcal{B}(n; p)$ simulée dans l'exemple d'appel à la console ?
2. Quelle devrait être l'aspect théorique de l'histogramme si N est arbitrairement grand ?
   On pourra le construire à l'aide de Python en fonction des choix de n et p
3. L'objectif de cette question est de simuler la loi binomiale sans l'appel de la commande `rd.binomial`
   (a) Compléter le script suivant pour qu'il génére la simulation d'une variable de Bernoulli X de paramètre p après avoir vérifié que p est un paramètre convenable :

   ```
   p=float(input("paramètre de Bernoulli"))
   X=0
   if p ..................
           if rd.random()<p:
                   X=1
   ...
   ```

   (b) A l'aide du script qui précède, générer n variables X_1 , X_2, … , X_n de Bernoulli $\mathcal{B}(p)$.
   On testera la fréquence d'obtention de 1 sur $n = 1000$ avec $p = 0.25$.

(c) Si l'on suppose les appels de `rd.random()` indépendants les uns des autres, quelle est la loi de $S_n = \sum_{k=1}^{n} X_k$? En déduire un script de génération de loi $\mathcal{B}(n;p)$ sous forme de fonction `SchBin(n,p)`

(d) Comparer les histogrammes de réalisations de `Binom`$(21, 0.33, 1000)$ avec 1000 appels de `SchBin`$(21, 0.33)$

4. En adaptant le programme initialement proposé à la loi géométrique, on obtient :

```
def geom(p,N):
    X = np.zeros([N])
    for k in range(N):
        X[k] = rd.geometric(p)
    Gu = plt.hist(X, range = (0, floor(2/p)), bins = floor(2/p))
    return(Gu)
```

On propose un exemple d'appel à la console :

```
>>> Gu=geom(0.33,1000)
>>> plt.show()
```

(a) Tester la fonction `geom` pour simuler 10 000 appels de loi géométrique de paramètre $p = \frac{1}{38}$.

(b) En vous inspirant de la démarche effectuée pour la loi binomiale et à l'aide d'une boucle `while`, rédiger un script permettant la simulation d'une loi $\mathcal{G}(p)$ qui s'appuie sur la définition par schéma de Bernoulli.

(c) Comparer les histogrammes de retour avec le cas $N = 10\ 000$ et $p = \frac{1}{38}$ (probabilité de réussite d'une mise sur un numéro unique à la roulette répétitif jusqu'à obtention d'une réussite).

(d) *Application :* Un joueur parie 2 euros sur le numéro 13 et joue répétitivement jusqu'à réussite qui procure un gain de $36 \times 2 = 72$ euros.

Proposer une simulation pour ce joueur qui renvoie la valeur totale perdue ou gagnée en jouant ainsi.

5. **Le mot de la faim** - *parce qu'il est tard :*

Mais que fait ce script ? On pourra le tester :

```
def fish(N,chips):
    Salt = np.zeros([N])
    for k in range(N):
        Salt[k] = rd.poisson(chips)
    Gout = plt.hist(Salt, range = (0, chips*2), bins = 2*chips)
    return(Gout)
```

```
>>> Gout=fish(5,1000)
>>> plt.show()
```

Séance IV : Lois continues

Les lois abordées dans cette section seront vues en cours au semestre 4 du point de vue théorique (VAR à densité : une application des intégrales généralisées) Une première approche empirique et pratique est donc proposée ici.

```
def expo(a,N):
    X = np.zeros([N])
    for k in range(N):
        X[k] = rd.exponential(a)
    Gu = plt.hist(X, range = (0, a**2), bins = 10*ceil(a)**2)
    return(Gu)
```

On propose un exemple d'appel à la console :

```
>>> Gu=expo(5,1000)
>>> plt.show()
```

1. Pour $a > 0$, on considère la fonction f_a définie sur $\mathbb{R}$ par :

$$f_a(x) = \begin{cases} ae^{-ax} & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases}$$

   (a) Encoder la fonction f_a sous forme de script Python. La tester sur quelques valeurs de x et de a.
   (b) A l'aide de `np.linspace` construire une représentation graphique de $f_{0.2}$ sur $[0; 25]$ avec un par de 0.1
   (c) Confronter cette dernière courbe avec l'histogramme généré par le code fourni initialement.
   Quelle conjecture peut-on alors faire ?
   (d) Modifier le code proposé pour que l'histogramme affiche des barres dix fois plus fines avec $N = 10\,000$ puis reprendre la confrontation précédante.

2. Adapter ce script proposé initialement pour qu'il générère une liste X de N valeurs générées par la commande `rd.exponential(a)` :

```
def ...........(a,N):
    X = np.zeros([N])
    for k in range(N):
        X[k] = rd.exponential(a)
    return(...........)
```

3. A l'aide de ce script et de la commande `np.mean`, remplir le tableau de valeurs suivant :

$\overline{X}$	$N = 200$	$N = 500$	$N = 1000$	$N = 5\,000$	$N = 10\,000$
$a = 1$					
$a = 2$					
$a = 3.5$					
$a = 0.25$					

4. Quelle conjecture pourrait-on émettre reliant a et $\overline{X}$ à la vue de ce tableau ? Quel serait le rôle de N ?
5. On fixe $a = 1$. Proposer une représentation graphique d'un nuage de points du type $(\overline{X}; N)$ permettant de vérifier cotre conjecture.
6. Reprendre l'approche qui précède avec le script :

```
def gauss(N):
    X = np.zeros([N])
    for k in range(N):
        X[k] = rd.normal()
    return(X)
Laplace = plt.hist(gauss(1000), range = (-3, 3), bins = 100)
return(Laplace)
```

en confrontant avec la fonction φ définie sur $\mathbb{R}$ par $\varphi(x) = \dfrac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$