TP2

Rappels de 1ère année : simulation de variables aléatoires discrètes

Les commandes intégrées

Python propose avec les bibliothèques `random` et `numpy.random` des commandes permettant générer des conditions ou des variables aléatoires.

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

Le générateur `rd.random()`

La commande `rd.random()` permet de créer une condition aléatoire dans un programme
On remarquera en particulier que la condition (`rd.random()<p`) se produit avec probabilité p.

REMARQUE **2.1.**
C'est avec la commande `rd.random()` que nous pouvons réaliser de nombreuses simulations d'expériences ou de variables aléatoires. Bien sûr il existe des commandes toutes prêtes concernant les lois usuelles :

EXERCICE 2.1 (Tirages successifs avec remise)**.**
On dispose d'une urne contenant 2 boules noires et une boule blanche. On effectue des tirages successifs avec remise d'une boule dans cette urne. On note X_i la variable aléatoire qui vaut 1 si on pioche la boule blanche et 0 sinon.
On note Y_n la variable égale au nombre de fois où l'on a tiré la boule blanche au cours des n premiers tirages.

1. Compléter la fonction suivante permettant de simuler la variable X_i :

```
1 def tirage():
2     X=0
3     if rd.random()< .... :
4       X=1
5     return X
```

2. Ecrire une fonction permettant de simuler la variable Y_n.

EXERCICE 2.2 (Tirages successifs avec ajout)**.**
Soit $c \in \mathbb{N}^*$.
On dispose d'une urne contenant 2 boules noires et 1 boule blanche. On effectue des tirages successifs avec remise d'une boule dans cette urne et on ajoute c boules de la même couleur que celle que l'on vient de tirer.
On note Y_n la variable égale au nombre de fois où l'on a tiré une boule blanche au cours des n premiers tirages.

1. Ecrire une fonction `tirage(a,b)` qui simule la variable X valant 1 si on tire une boule blanche dans une urne ayant a boules noires et b boules blanches et 0 sinon.

2. Compléter la fonction Python permettant de simuler la variable Y_n :

```
1  def simule_Y(c,n):
2    a = ...
3    b = ...
4    Y = ...
5    for _ in range(n):
6      if tirage(a,b) == ........ :
7        Y=Y+1
8        b=b+c
9      else:
10       a =......
11     return Y
```

3. Ecrire une fonction permettant de simuler la variable Y_n.

Les lois usuelles discrètes : commandes intégrées à Python

• `nr.randint(a,b,N)` renvoie N simulations de variables indépendantes suivant la loi $\mathscr{U}[\![a;b-1]\!]$.
• `nr.binomial(n,p,N)` renvoie N simulations de variables indépendantes suivant la loi $\mathscr{B}(n,p)$.
• `nr.geometric(p,N)` renvoie N simulations de variables indépendantes suivant la loi $\mathscr{G}(p)$.
• `nr.poisson(mu,N)` renvoie N simulations de variables indépendantes suivant la loi $\mathscr{P}(\mu)$.

A noter que, dans chaque commande, on peut remplacer l'argument `N` par `(r,s)`, ce qui permet de manière plus générale de renvoyer un tableau de taille $r \times s$ de simulations de variables indépendantes de même loi.

Exercice 2.3.
En utilisant les commandes intégrées :

1. Ecrire une fonction Python d'en-tête `def jeu_pile_ou_face(N):` qui prend en argument un entier N et renvoie une liste de taille N de PILE ou FACE avec une pièce équilibrée. *On codera PILE par 1 et FACE par 0.*

   Tester votre fonction.

2. Soit X une variable aléatoire suivant une loi de Poisson de paramètre 20.

   Si $(X = n)$ on place n boules numérotées de 1 à n dans une urne, on pioche une boule dans cette urne et on note Y le numéro de la boule tirée.

   Ecrire une fonction Python d'en-tête `def simule_Y(N):` qui prend en argument un entier naturel N et renvoie N simulations de la variable Y.

   Tester votre fonction.

Les fonctions exigibles à l'écrit

Loi de Bernoulli

La fonction suivante propose une simulation la loi de Bernoulli de paramètre p lorsque l'utilisateur entre la valeur de $p \in [0,1]$. *Elle est à retenir car exigible à l'écrit.*

```
1  def bernoulli(p):
2      X=0
3      if rd.random()<p:
4        X=1
5      return X
```

Exercice 2.4.

1. Taper cette fonction dans un nouveau fichier, la tester pour différentes valeurs de p.

2. Compléter la fonction suivante qui prend en argument un réel $p \in [0;1]$ et un entier naturel N et renvoie en sortie une estimation de la fréquence du nombre de 1 sur N simulations de la loi de Bernoulli de paramètre p lorsque l'utilisateur entre la valeur de $p \in [0,1]$ (*en utilisant la fonction précédente*).

```
def frequence(p,N):
    X=0
    S= ......
    for k in ..........:
      S =..........
    return ................
```

3. Taper ce script complété à la suite du script précédent. Le tester pour différentes valeurs de p. Ces résultats vous semblent il cohérents ?

4. Proposer à la suite de vos scripts, un nouveau script qui affiche sous forme de graphique l'évolution de la fréquence du nombre de 1 pour N simulations de la loi de Bernoulli de paramètre p lorsque l'utilisateur entre la valeur p lorsque $N = 1..2000$. Tester pour différentes valeurs de p puis commentez les graphiques obtenus.

Loi binomiale

On rappelle qu'on a vu en cours le résultat bien pratique suivant :

$$X \hookrightarrow \mathscr{B}(n,p) \quad \Longleftrightarrow \quad X = \sum_{i=1}^{n} X_i \quad \text{où les } (X_1,...,X_n) \text{ sont indépendants et suivent la loi } \mathscr{B}(p).$$

Autrement dit, la loi binomiale $B(n,p)$ peut se voir comme la somme de n lois de Bernoulli indépendantes de paramètre p.

EXERCICE 2.5.

1. Compléter la fonction Python suivante qui prend en argument un entier n et un réel $p \in [0;1]$ et qui renvoie une simulation d'une variable aléatoire suivant la loi binomiale $\mathscr{B}(n,p)$.

```
def binomiale(n,p):
  X=0
  for _ in range(.......):
    if ....................  :
      X=..........
  return X
```

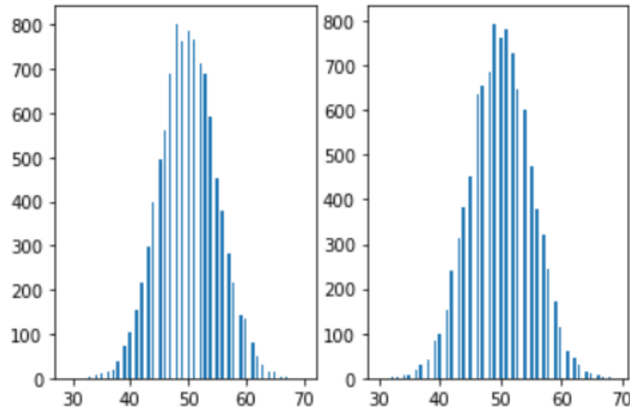
Cette fonction est à retenir car exigible à l'écrit.

2. Tester votre fonction dans la console pour différentes valeurs de n et de p. Les résultats vous semblent-ils cohérents ?

3. La fonction suivante a pour but de comparer les deux modèles de la loi binomiale : le modèle `nr.binomial(n,p,N)` intégré à Python et le modèle "fait main" `binomiale(n,p)` définit plus haut.

```
import matplotlib.pyplot as plt
def comparaison_modeles_binomiales(n,p,N):
  X=list(nr.binomial(n,p,N))
  Y=[]
  for _ in range(N):
    Y.append(binomiale(n,p))
  f, (ax1, ax2) = plt.subplots(1, 2, sharey=False)
  ax1.hist(X,bins='auto')
  ax2.hist(Y,bins='auto')
```

Pour $(n,p,N) = (100, 0.5, 10000)$ on obtient le graphique suivant :



(a) Que représentent chacun de ces deux graphiques ?

(b) Commentez.

Loi géométrique

La loi géométrique est la loi du premier succès dans une répétition d'épreuves de Bernoulli de paramètre p indépendante.

EXERCICE 2.6.

1. Compléter la fonction `Python` d'en-tête `def geometric(n,p):` qui prend en argument un réel $p \in [0;1]$ et qui renvoie une simulation d'une variable aléatoire suivant la loi géométrique $\mathscr{G}(p)$.

```
def geometric(p):
  X=.........
  while ...............  :
    X=.........
  return ...........
```

Cette fonction est à retenir car exigible à l'écrit.

2. Rappeler l'espérance d'une variable suivant la loi géométrique. Tester plusieurs fois votre fonction dans la console pour une valeur particulière de p. Les résultats vous semblent-ils cohérents ?
3. En prenant exemple sur l'exercice précédent, écrire une fonction permettant de comparer les deux modèles de la loi géométrique : le modèle `nr.geometric(p,N)` intégré à `Python` et le modèle "fait main" `geometric(p)` définit plus haut.

Approximation de la de Poisson par des lois binomiales

On peut montrer le résultat suivant :

PROPRIÉTÉ 2.1 (Approximation de la loi de Poisson par des lois binomiales)

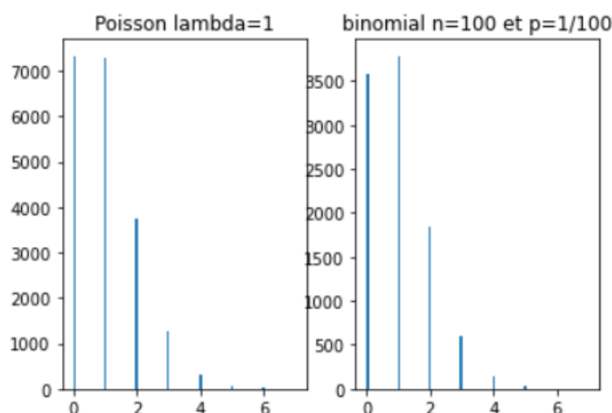
Soit $(X_n)_{n\in\mathbb{N}}$ une suite de variables aléatoires indépendantes telles que :

- $\forall n \in \mathbb{N}, \quad X_n \hookrightarrow \mathscr{B}\left(n, p_n\right)$ avec $p_n \in [0;1]$
- $p_n \sim \frac{\lambda}{n}$ avec $\lambda > 0$.

Alors dans ces conditions : $\boxed{\forall k \in \mathbb{N}, \quad \lim_{n\longrightarrow+\infty} P(X_n = k) = e^{-\lambda}\frac{\lambda^k}{k!}.}$

Autrement dit, dans ces conditions, lorsque $n \longrightarrow +\infty$, les lois binomiales $\mathscr{B}(n, p_n)$ s'approchent de la loi de Poisson $\mathscr{P}(\lambda)$.

EXERCICE 2.7.
En utilisant le résultat précédent, proposer une fonction Python permettant d'obtenir le graphique suivant :



Extrait de concours : déplacements d'une puce sur un axe

Une puce se déplace sur un axe gradué. À l'instant 0, la puce se trouve sur le point d'abscisse 0.
À partir de l'instant 0, la puce effectue à chaque instant, un saut vers la droite selon le protocole suivant :

- elle effectue un saut d'une unité vers la droite avec la probabilité $\frac{1}{2}$;
- elle effectue un saut de deux unités vers la droite avec la probabilité $\frac{1}{4}$;
- elle effectue un saut d'une unités vers la gauche avec la probabilité $\frac{1}{4}$.

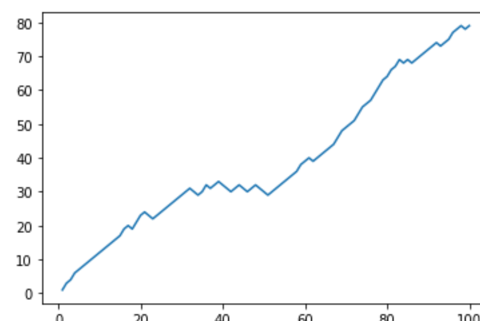
Les différents sauts sont supposés indépendants.

1. Rappeller la commande Python permettant de simuler une variable aléatoire suivant la loi discrète uniforme sur $[\![1,4]\!]$.
2. Compléter la fonction suivante prenant en entrée un entier n et renvoyant en sortie une simulation des n premiers déplacements de la puce.

```
def sauts(n):
  L=[0. for k in range(100)]
  for k in range(100):
    t =...............
    if t<=......  :
      L[k]  =1
    if t==......  :
      L[k]  =2
    if t==......
      L[k] =-1
  return L
```

3. On ajoute à la fonction Python de la question les commandes suivantes et on obtient la sortie graphique suivante :

```
A = sauts(100)
x = [k for k in range(1,101)]
y = np.cumsum(A)
plt.plot(x,y)
plt.show()
```



Interpréter le graphique.