UNIVERSITE CADI AYYAD

Ecole Nationale des Sciences

Appliquées Safi

Projet de simulation en Python

Module : Modélisation stochastique

Projet 1 : Simulation de variables aléatoires discrètes

Filière : Génie Informatique et Intelligence Artificielle (G.I.I.A.)

Première Année G.I.I.A.

EL HASSAN LAKHEL

Réalisée par :Chebbab Aya

Houssam Meryam

Année Universitaire : 2021-2022

1. Générateur pseudo-aléatoire en Python : la fonction random

(S) Évaluation du random.random() :

On obtient un réel de [0, 1]. Les résultats obtenus sont différents d'un poste à l'autre, ce qui tend à faire penser à un comportement aléatoire

```
import random
print(random.random())
```

```
In [2]: runfile('C:/Users/hp/.spyder-py3/temp.py', wdir='C:/Users/hp/.spyder-py3')
0.6515241686507439
```

(S) Évaluation de la commande random.seed(0) :

On obtient les résultats suivants :

```
import random
random.seed(0)
print(random.random())
print(random.random())
print(random.random())
print(random.random())
print(random.random())
```

```
0.8444218515250481
0.7579544029403025
0.420571580830845
0.25891675029296335
0.5112747213686085
```

Les résultats obtenus sont les mêmes sur tous les postes

(S) Explication des résultats obtenus :

La deuxième série de résultat illustre la notion de générateur pseudo aléatoire. En partant tous de la même graine, la suite de nombre générée est la même pour tout le monde.

Le premier résultat (lorsqu'il y a différence entre les postes) démontre que la graine initiale n'est pas la même sur chaque poste. L'explication est fournie par la documentation Python :

current system time is also used to initialize the generator when the module is first imported

2. Généralités sur la simulation d'une variable aléatoire

(T) Démonstration de la loi faible des grands nombres :

Rappelons tout d'abord l'inégalité de Bienaymé-Tchebychev

Soit Y une v.a.r discrète admettant une variance $V(Y)$

- soit X une v.a.r admettant un moment d'ordre 2.
- Soit (X_n) une suite de v.a.r indépendantes et de même loi que X
- Notons alors $Y = \frac{1}{n}\sum_{k=1}^{n} X_k$, Par linearité de l'espérance et propriété de la variance (les v.a.r étant mutuellement indépendantes):

$$E(Y) = E\left(\frac{1}{n}\sum_{k=1}^{n} X_k\right) = \frac{1}{n} E\left(\sum_{k=1}^{n} X_k\right)$$

$$= \frac{1}{n}\sum_{k=1}^{n} E(X) = \frac{1}{n} \cdot n \cdot E(X) = E(X)$$

$$V(Y) = V\left(\frac{1}{n}\sum_{k=1}^{n} X_k\right) = \frac{1}{n^2} V\left(\sum_{k=1}^{n} X_k\right)$$

$$= \frac{1}{n^2}\sum_{k=1}^{n} V(X) = \frac{1}{n^2} \cdot n \cdot V(X)$$

$$= \frac{1}{n} V(X)$$

Ainsi $\forall \varepsilon > 0 : P\left(\left|\frac{1}{n}\sum_{k=1}^{n} X_k - E(X)\right| \geq \varepsilon\right) \leq \frac{V(X)}{\varepsilon^2}\frac{1}{n} \xrightarrow[n \to +\infty]{} 0$

3.Diagrammes en bâtons en Python

(S) la fonction position qui cherche un élément dans une liste:

```
def position(elt, L):
    n = len(L)
    for i in range(n):
        if L[i] == elt:
            return i
```

(S) la fonction calcEffectif qui renvoie le tableau des effectifs

```
def calcEffectif(cl, Obs):
    n = len(cl)
    tab = np.zeros(n)
    for elt in Obs:
        i = position(elt, cl)
        tab[i] = tab[i] + 1
    return tab
```

4.Loi uniforme discrète

(T) signification d'une v.a.r. X qui suit la loi uniforme discrète sur {a,.., b} :

La loi discrète uniforme est une loi de probabilité discrète indiquant une probabilité de se réaliser identique (équiprobabilité) à chaque valeur d'un ensemble fini de valeurs possibles.

Une variable aléatoire qui peut prendre n valeurs possibles k_1 , k_2 , …, k_n, suit une loi uniforme lorsque la probabilité de n'importe quelle valeur k_i est égale à 1/n.

On considère le programme suivant.

```
import random
import math
def uniforme(a,b) :
return (a + math.floor(random.random()*((b-a)+1)))
```

Programme 1 Simulation de la loi uniforme discrète sur $\{a, ..., b\}$.

(S) le rôle de la fonction uniforme :

La fonction uniforme prend comme entré un intervalle [a,b] et retourne un nombre entier aléatoire dans cet intervalle.

On prend la distance entre a et b « *(a-b)+1* » et on le multiple par un nombre réelle entre 0 et 1 en utilisant la commande *random.random()*, on obtient un nombre réelle qu'on utilise seulement la partie entière a l'aide de la commande *math.floor()*, puis on ajoute ce nombre à a pour retourner un nombre entier compris entre a et b.

Diagrammes en bâtons associés :

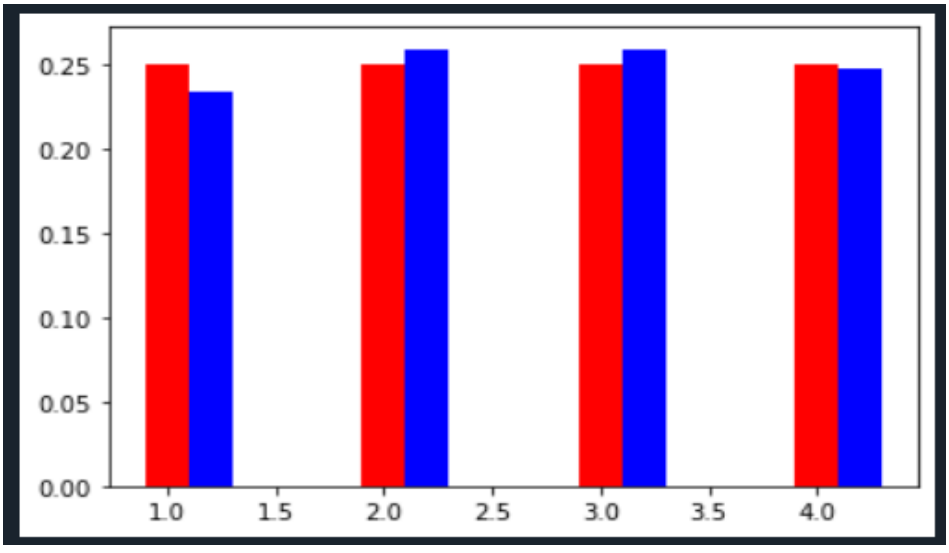
En ligne 9, on utilise une compréhension de liste

```
import random
import math

def uniforme(a,b):
    return (a + math.floor(random.random()*((b-a)+1)))
Obs=[]
for k in range(5):
    Obs = Obs + [uniforme(1, 4)]
```

Le code du diagramme :

```python
import random
import math
import numpy as np
import matplotlib.pyplot as plt
def position(elt, L):
    n = len(L)
    for i in range(n):
        if L[i] == elt:
            return i
def calcEffectif(cl, Obs):
    n = len(cl)
    tab = np.zeros(n)
    for elt in Obs:
        i = position(elt, cl)
        tab[i] = tab[i] + 1
    return tab

def uniforme(a,b):
    return (a + math.floor(random.random()*((b-a)+1)))
Obs=[]

N = 1000 # 1000 simulations
n = 4 # loi uniforme sur [[1,4]]
Obs = [uniforme(1,n) for k in range(N)]
cl = np.linspace(1, n, n)
efectif = calcEffectif(cl, Obs)
P = np.zeros(n)
for k in range(n) :
    P[k] = 1/n

absc = np.linspace(1,n,n)
plt.bar(absc, P, color = 'r', width = 0.2)
plt.bar(absc + 0.2, efectif / N, color = 'b', width = 0.2)
plt.show()
```

Le diagramme :



(S) On peut adapter le programme précédent afin d'obtenir le diagramme associé à la simulation d'une loi uniforme discrète sur {a, ..., b} (et plus seulement {1, ..., n}) par :

Il faut commencer par introduire a et b puis remplacer :

- `uniforme(1,n)` par `uniforme(a,b)`
- `np.linspace(1,n,n)` par `np.linspace(a,b,b-a+1)`

5.Loi Binomiale :

(T) Pour quel type d'expérience définit-on une v.a.r. suivant une loi binomiale ? :

On considère une expérience aléatoire qui consiste en une succession de n épreuves indépendantes, chacune d'entre elles ayant deux issues : succès obtenu avec probabilité p et échec obtenu avec probabilité q = 1 − p.

Autrement dit, l'expérience consiste à effectuer n épreuves de Bernoulli indépendantes et de même paramètre p.

Alors la v.a.r. donnant le nombre de succès de l'expérience suit la loi binomiale de paramètre (n, p).

(T) Que signifie qu'une v.a.r. X suit la loi binomiale de paramètre (n; p) ?

Une variable aléatoire X qui suit la loi binomiale de parametre (n,p) signifie que:

$X(\Omega) = [0, n]$ et $\forall k \in [0, n]$

On a

$$\mathbb{P}(X = k) = \binom{n}{k} p^k (1-p)^{n-k}.$$

Simulation à l'aide de la fonction random

(S) La fonction Bernoulli(p):

```
def Bernoulli(p):
    if random.random() < p:
        return 1
    else:
        return 0
```

(S) fonction binomiale(n,p) simulant une variable aléatoire de loi B(n; p):

```
def binomiale(n,p):
    S=0
    for i in range(n):
        S = S + Bernoulli(p)
    return S
```

Diagrammes en bâtons associés

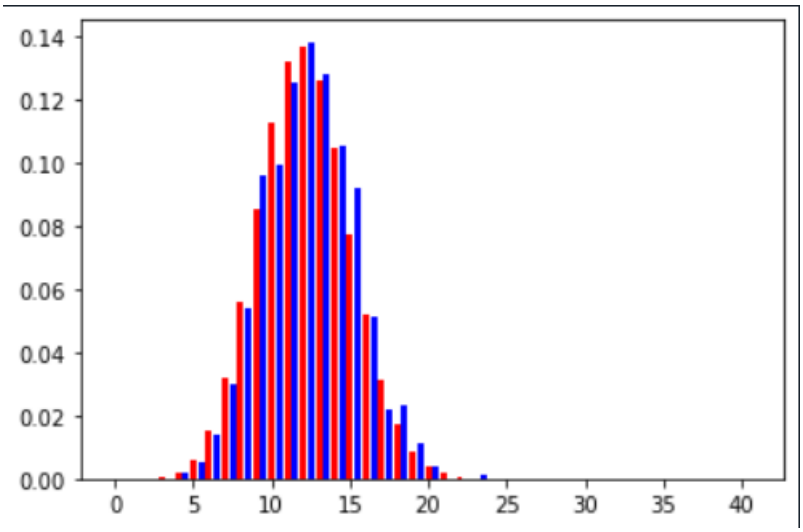
(S) Ligne 18 du programme précédent :

```
for k in range(n+1) :
    comb = math.factorial(n) / math.factorial(k) * math.factorial(n-k)
    P[k] = comb * (p**k) * ((1-p)**(n-k))
```

Le code du diagramme :

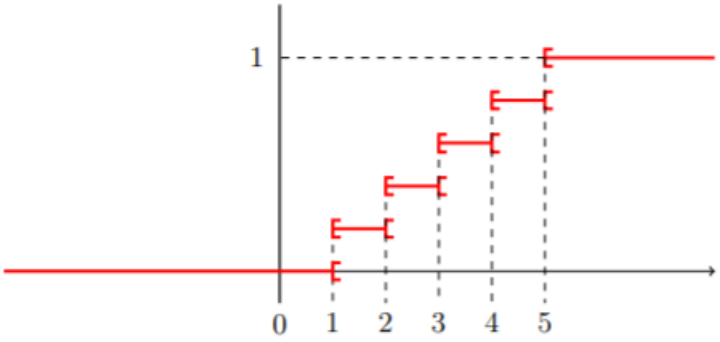
```
P = np.zeros(n+1)
for k in range(n+1):
    comb = math.factorial(n)/(math.factorial(k) * math.factorial(n-k))
    P[k] = comb * (p**k) * ((1-p)**(n-k))

absc = np.linspace(0, n, n+1)
plt.bar(absc, P, color = 'r', width = 0.4)
plt.bar(absc + 0.5, effectif / N, color = 'b', width = 0.4)
plt.show()
```

Le diagramme :



7 .Loi discrète quelconque

(T) la fonction de répartition FX:



les valeurs de (xi), (pk) et (rk) pour la variable X précédente :

- x1 = 1, x2 = 2, x3 = 3, x4 = 4, x5 = 5.
- p1 = ... = p5 = 1 5 .
- r0 = 0, r1 = 1 5 , r2 = 2 5 , r3 = 3 5 , r4 = 4 5 , r5 = 1.

Les valeurs (rk), valeurs cumulées des probabilités, peuvent se lire sur le graphe de FX puisque : rk = F(xk). Ce constat se retrouve dans la dénomination anglaise des fonctions de répartition : elles sont nommées cumulative distribution function.

Méthode d'inversion

(S) La fonction sommeCumulee :

```
def sommeCumulee(P):
    R = []
    r=0
    for p in P:
        r=r+p
        R.append(r)
    return R
```

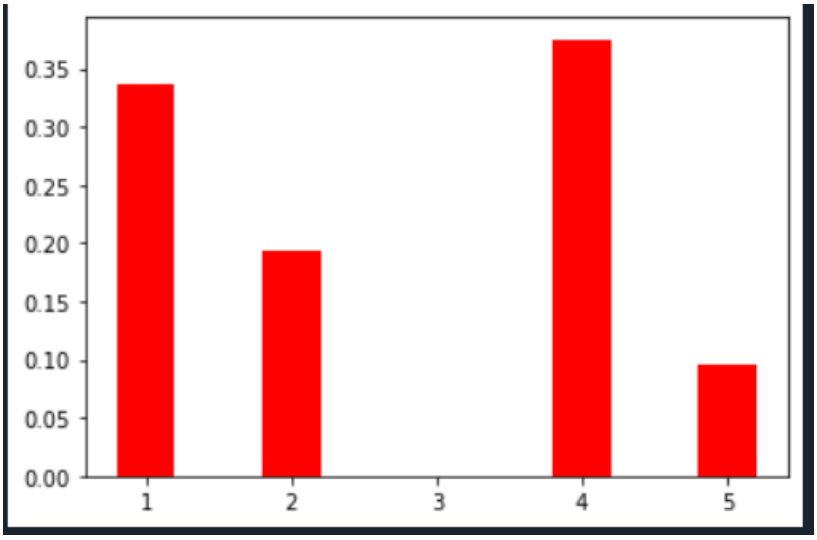
(S) La fonction discrete Q(X; P) prenant en paramètres les listes [p1; :::; pn]

```
def discreteQ(X, P):
    R = sommeCumulee(P)
    u = random.random()
    i=0
    while (u >= R[i]):
        i=i+1
    return X[i]
```

(S) le diagramme en bâtons :

```
def histoQ(X, P):
    N = 10000 # 10000 simulations
    Obs = [discreteQ(X, P) for k in range(N)]
    cl = X
    plt.bar(cl, Obs/N, color = 'b', width = 0.8)
```

Le diagramme :



8 .Simulation d'une v.a.r. suivant une loi géométrique :

(T) Pour quel type d'expérience définit-on une v.a.r. suivant une loi géométrique ?

On considère une expérience aléatoire qui consiste en une succession infinie d'épreuves indépendantes, chacune d'entre elles ayant deux issues : succès obtenu avec probabilité p et échec obtenu avec probabilité q = 1 − p.

Autrement dit, l'expérience consiste à effectuer une infinité d'épreuves de Bernoulli indépendantes et de même paramètre p. Alors la v.a.r. donnant le rang d'apparition du premier succès de l'expérience suit la loi géométrique de paramètre p .

(S) Signification qu'une v.a.r. X suit la loi géométrique de paramètre p

$X \rightarrow G(p)$ signifie : $X(\Omega) = N*$ et $\forall k \in N*, P(X = k) = p(1-p)^{k-1}$

8.1 Simulation à l'aide de la fonction bernoulli :

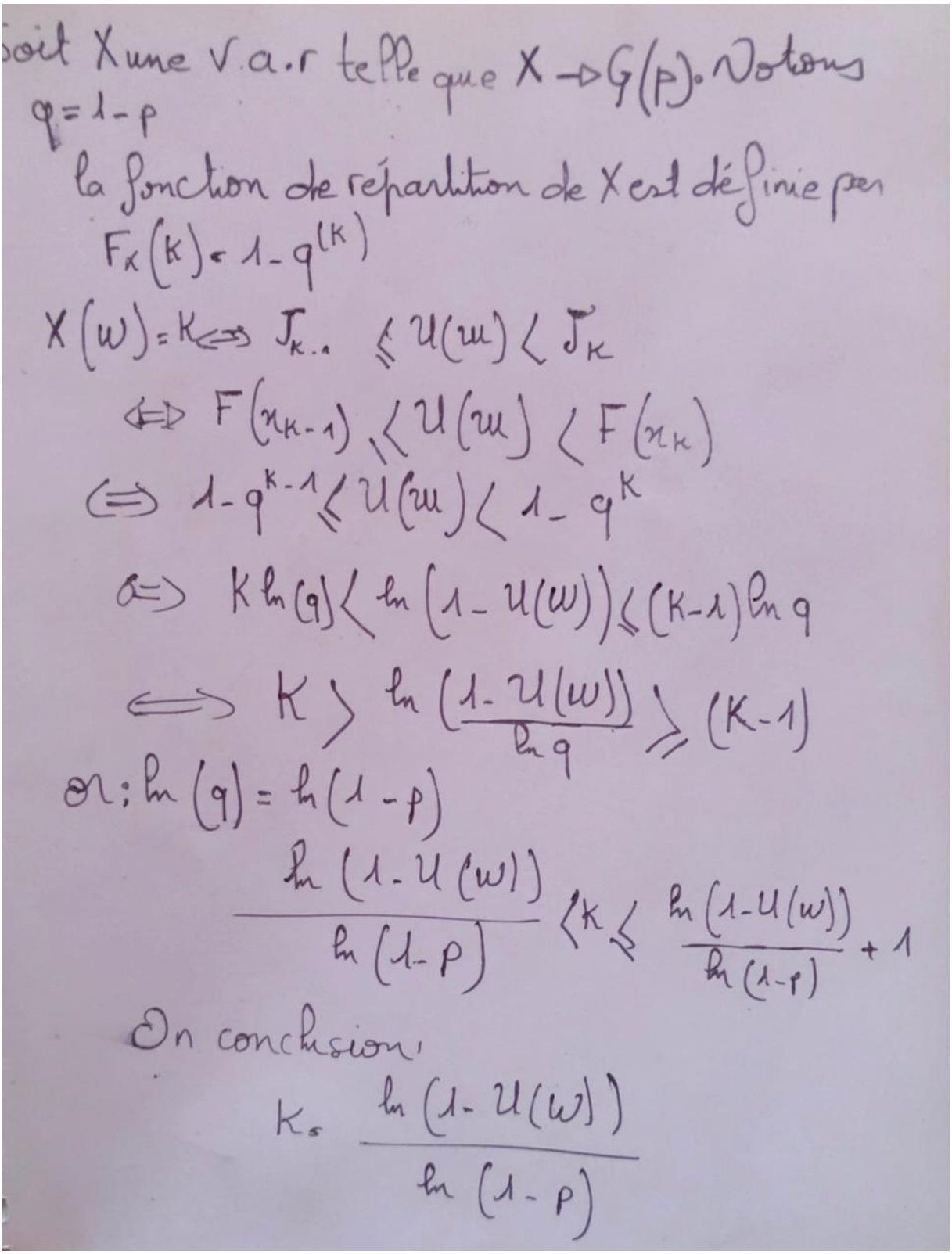
(S) la fonction geom(p) :

```
def geom(p):
    rang = 0
    while Bernoulli(p) == 0 :
        rang = rang + 1
    return rang
```

(S) illustration de la loi des grands nombres dans le cas de la loi géométrique

```
import random
import numpy as np
import math

def geom(p):
    rang = 0
    while Bernoulli(p) == 0 :
        rang = rang + 1
    return rang

N = 10000
p = 0.3
Obs = [np.random.geometric(p) for k in range(N)]
moyE = np.sum(Obs)/N
print(moyE)
```

(S) La fonction précédente permet de simuler une variable aléatoire de loi G(p):



9 .Approximation de la loi de Poisson :

```
def poissonRare(mu, n):
    return binomiale(n, mu/n)
```