Lycée Pierre-Gilles de Gennes 2025-2026

TP Python – 3

Simulation de variables aléatoires discrètes

L'objet de ce TP est essentiellement de mettre en place une méthodologie pour effectuer des simulations probabilistes.

On se donne un espace probabilisé[1] $(\Omega, \mathcal{A}, \mathbb{P})$. Les fonctions Python de tirage au sort que nous allons rencontrer sont des modèles de variables aléatoires au sens où chaque appel à l'une de ces fonctions équivaut à tirer au sort, **indépendamment** des tirages précédents, un nombre flottant, suivant une loi donnée.

Pour ce TP, nous aurons besoin d'importer le module `numpy.random` sous l'alias `rd` :

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

La fonction de tirage au sort fondamentale que nous utiliserons est la commande `rd.rand()` qui tire un nombre au hasard **uniformément** dans l'intervalle $[0,1[$[2].
Cela signifie que pour tout intervalle I contenu dans $[0,1[$,

$$\mathbb{P}(\texttt{rd.rand()} \in I) = \text{longueur}(I).$$

Pour tirer un nombre au hasard, Python dispose de plusieurs fonctions qui diffèrent essentiellement par la *loi* du nombre aléatoire retourné.
Le programme du concours cependant impose que vous sachiez simuler des v.a. suivant des lois classiques à partir d'une variable uniforme sur $[0,1[$. De toutes ces belles fonctions `numpy` qui simulent des lois classiques ou beaucoup plus exotiques, vous n'avez donc le droit, dans un premier temps, de n'en utiliser qu'une : `rd.rand()`.

1 Simulation des lois classiques

1.1 Simulation de lois de Bernoulli et de lois binomiales

Le but de cette partie est de simuler des variables suivant une loi de Bernoulli ou binomiale seulement en utilisant `rd.rand()`. On considère le script suivant :

```
1 # Bernoulli manuelle
2
3 def Bernoulli ( p ):
4 """ Entrée : un réel p dans ]0,1[
5         Sortie : 1 avec proba p et 0 avec proba 1-p
6 """
7     u = rd.rand()
8     .....
9 return .....
```

1. On verra la définition dans le cours de mathématiques.
2. C'est-à-dire simule un loi uniforme à densité sur $[0,1[$.

Travail demandé

1. (a) Cours
   (b) Simulation d'une Bernoulli de paramètre $p \in]0,1[$

```
def Bernoulli ( p ):
    u = rd.rand()
    if u<p:
        return 1
    else:
        return 0
```

2. (a) Cours.
   (b) La somme de n variables aléatoires indépendantes de loi $\mathcal{B}(p)$ suit la loi $\mathcal{B}(n,p)$.
   (c) Simulation d'une variable aléatoire de loi $\mathcal{B}(n,p)$.

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

3. (a) Écrire une fonction `Rep_binomiale(N,n,p)` prenant en entrées des entiers naturels `N`, `n` et un réel `p`$\in]0,1[$ et simulant N variables aléatoires indépendantes de loi $\mathcal{B}(n,p)$.

```
def Rep_binomial(N,n,p):
    L = []
    moy = 0
    var = 0
    for k in range(N):
        x = Binomiale(n,p)
        moy += x
        var += x**2
        L.append(x)
    moy = moy/N
    var = var/N-moy**2
    return L,moy,var
```

   (b) On remarque que la moyenne et la variance empiriques sont proches de l'espérance et la variance (c'est la loi des grands nombres).

Le script suivant affiche l'histogramme des fréquences des valeurs prises lors de 10 000 simulations d'une $\mathcal{B}(11,0.3)$:

```
N = 10000
p = 0.3
n = 11

# Affichage histogramme des fréquences
plt.hist(Rep_binomiale(N,n,p)[0],density = True,rwidth = 0.5)
```

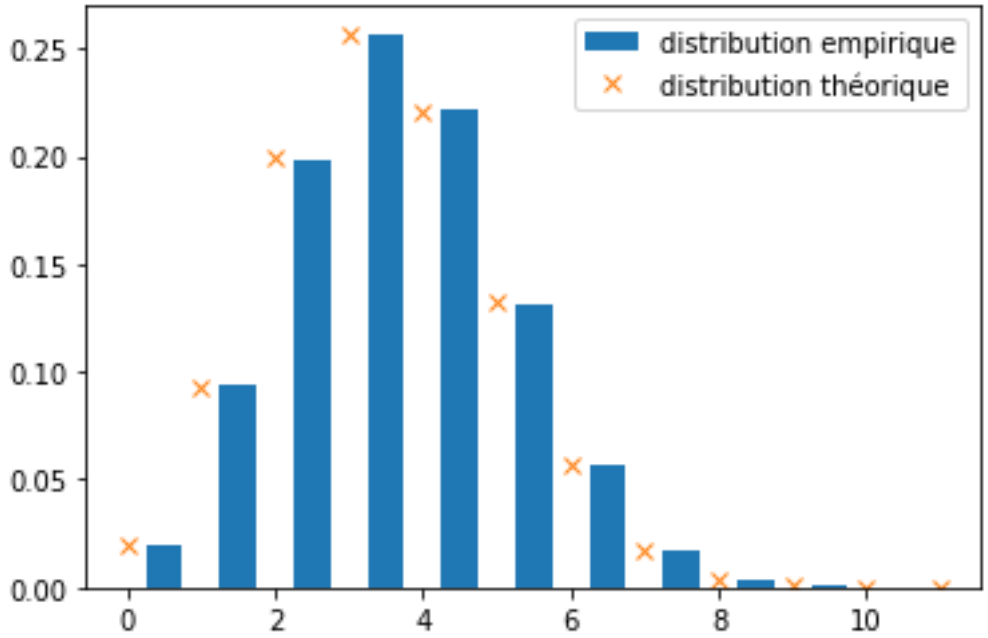
4. (a) Recopier le script ci-dessus.

(b) Soit X une variable aléatoire de loi $\mathcal{B}(11, 0.3)$. Tracer, sur la même fenêtre graphique que l'histogramme, la représentation graphique de $(\mathbb{P}(X = k))_{k\in[\![0,11]\!]}$.

```
def Binomiale_theo(n,p):
    L=[]
    for k in range(n+1):
        L.append(math.comb(n,k)*p**k*(1-p)**(n-k))
    return L

plt.hist(Rep_binomiale(N,n,p)[0],density = True,rwidth = 0.5)
plt.plot (Binomiale_theo(n,p) , 'x')
```

(c) On obtient le graphique suivant :



On constate que pour tout $k \in [\![0, n]\!]$, la proportion des simulations ayant pris la valeur k sur les 10 000 simulations (barres bleues) est proche de la valeur $\mathbb{P}(X = k)$ (croix oranges) : cela s'explique par la loi des grands nombres.

1.2 Simulation d'une loi uniforme discrète

Le but de cette partie est de simuler une variable aléatoire suivant une loi uniforme sur $[\![0, n-1]\!]$, $n \in \mathbb{N}^*$.
On rappelle que si x est un réel positif, sa partie entière, notée $\lfloor x \rfloor$ est l'unique entier tel que :

$$n \leq x < n + 1.$$

Travail demandé

1. Cours.
2. Soit x un nombre tiré au hasard dans $[0, 1[$ avec la commande `rd.rand()`.

(a) Soit $k \in [\![0, n-1]\!]$. Par définition de la partie entière :

$$\lfloor nx \rfloor = k \iff k \leq nx < k+1 \iff \frac{k}{n} \leq x < \frac{k+1}{n}.$$

(b) Soit $k \in [\![0, n-1]\!]$. D'après la question précédente :

$$\begin{aligned}\mathbb{P}(\lfloor n * \texttt{rd.rand()} \rfloor = k) &= \mathbb{P}\left(\frac{k}{n} \leq \texttt{rd.rand()} < \frac{k+1}{n}\right) \\ &= \mathbb{P}\left(\texttt{rd.rand()} \in \left[\frac{k}{n}, \frac{k+1}{n}\right[\right) \\ &= \frac{1}{n}\end{aligned}$$

la dernière égalité provenant de la définition de la commande `rd.rand()`.

(c) Écrire une fonction `Uniforme(n)` qui prend en entrée un entier $n \in \mathbb{N}^*$ et simule une variable de loi uniforme sur $[\![0, n-1]\!]$.

```
1 def EntierUniforme(n) :
2     u = np.random.rand()
3     return int( np.floor(n* u) )
```

3. Utiliser la fonction précédente pour tracer l'histogramme d'une suite de `N = 1000` tirages uniformément en 0 et 9.

```
1 hist_unif =[]
2 for k in range(1000):
3     hist_unif.append(EntierUniforme(10))
4
5 plt . hist (hist_unif  ,density = True,rwidth = 0.8)
```

On obtient les graphiques suivants :

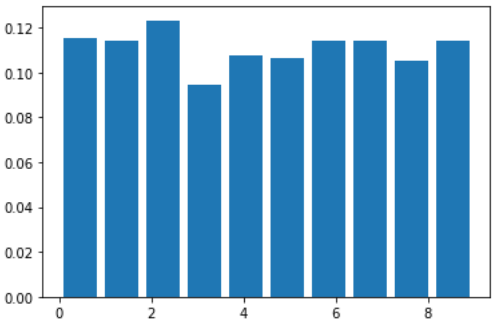


FIGURE 1 – Avec $N = 1000$

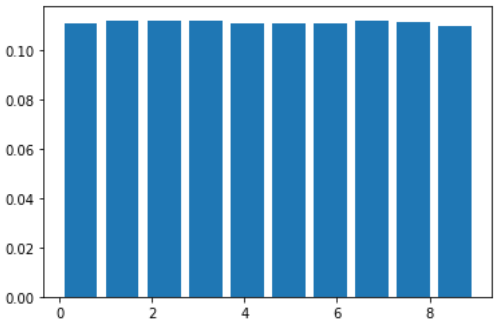


FIGURE 2 – Avec $N = 100000$

On remarque que les proportions de chaque valeur sont proches (et proche de $\frac{1}{n}$). C'est d'autant plus le cas lorsque N est grand : c'est encore une fois la loi des grands nombres.

1.3 Variables aléatoires discrètes finies

Travail demandé

1. Écrire une fonction `VaDiscrete(p)` où `p` est une liste de `float` positifs dont la somme vaut 1 et retournant un nombre entier (un `int` donc) entre 0 et `len(p)-1` tiré suivant la loi décrite par `p` c'est-à-dire :

$$\mathbb{P}(\texttt{VaDiscrete(p)} = k) = \texttt{p[k]}.$$

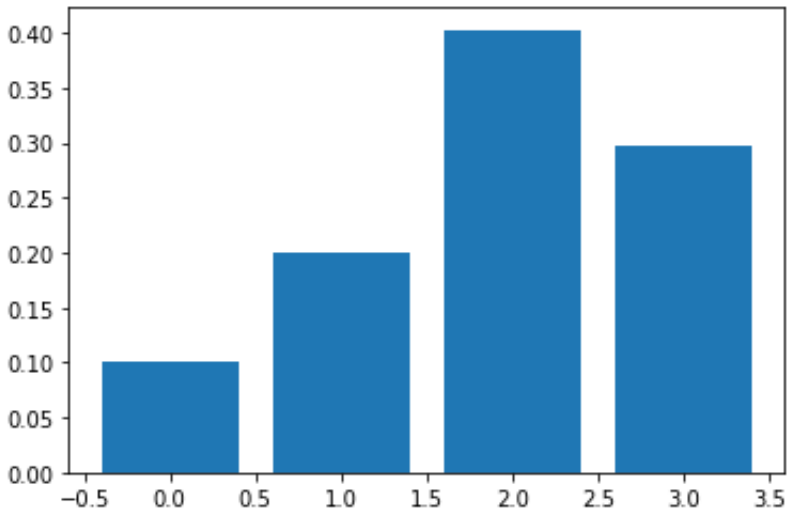
```
def VaDiscrete(p):
        n = len(p)
        x = rd.rand()
        cumul = 0
        for i in range(n):
                if cumul<x<= cumul+p[i]:
                        return i
                cumul+=p[i]
```

2. Écrire un bloc de code calculant une liste de `N = 10000` nombres tirés suivant la loi `p=[0.1,0.2,0.4,0.3]` et afficher l'histogramme de cette liste.

```
list_p=[0.1,0.2,0.4,0.3]

test = []
for k in range(10000):
    test.append(VaDiscrete(list_p))
plt.hist( test,[-0.5,0.5,1.5,2.5,3.5],density = True,rwidth = 0.8)
```

On obtient le graphique suivant :



3. Donner la moyenne et la variance de cette liste. Faire imprimer ces nombres ainsi que les valeurs théoriques de l'espérance et de la variance de la variable `VaDiscrete(p)`.

```
moy = 0
var = 0
```

```
for elt in test:
    moy += elt
    var += elt**2
moy = moy/10000
var = var/10000-moy**2
esp = 0.2+2*0.4+3*0.3
var_theo = 0.2+4*0.4+9*0.3-esp**2
print(moy,esp)
print(var, var_theo)
```

On constate que les valeurs empiriques (moyenne et variance) sont approximativement égale aux valeurs théoriques (espérance et variance).

2 Vers la loi géométrique

On considère une urne contenant $n \in \mathbb{N}^*$ boules numérotées de 1 à n indiscernables au toucher.
On tire uniformément avec remise dans l'urne (une infinité de fois).
Pour tout $i \in [\![1, n]\!]$, on note X_i la variable aléatoire donnant le numéro du tirage où on obtient la boule numéro i pour la première fois [3].

Travail demandé

1. Écrire une fonction `Geometrique(n,i)` qui prend en entrées un entier $n \in \mathbb{N}^*$ et un entier $i \in [\![1, n]\!]$ et qui simule la variable X_i.

```
def Geometrique(n,i):
    boule = 0
    nb_essai =0
    while boule != i:
        boule = EntierUniforme(n)+1
        nb_essai += 1
    return nb_essai
```

2. Écrire un bloc de code donnant la liste des résultats de `N = 1000` simulation de X_1 pour $n = 3, n = 10$, $n = 5$.

```
def Rep_Geometrique(N,n,i):
    liste = []
    for k in range(N):
                liste.append(Geometrique(n,i))
    return liste
```

3. On procède comme précédemment et on compare les valeurs empiriques avec l'espérance et la variance d'une loi géométrique de paramètre $\frac{1}{n}$.

3. X_i suit la loi géométrique de paramètre $\frac{1}{n}$.