TP 4: Instructions utiles.

5 juillet 2023

Les fonctions permettant de faire des simulations d'expériences aléatoires se trouvent dans la bibliothèque `numpy.random`. On les importe en écrivant `import numpy.random as rd`.

1 La fonction `random()`.

L'instruction `rd.random()` renvoie un réel aléatoire de $[0, 1[$.

- **Lancer d'une pièce qui amène "pile" avec la proba p.**

```
def simule_de(p):
    if rd.random() < p :
        return "P"
    else :
        return "F"
```

On peut alors simuler facilement n lancers de la pièce à l'aide d'une boucle `for` :

```
for k in range(n) :
    r = simul_de(p)
    print(r)
```

On peut ajouter en option à la fonction `rd.random()` une option de taille qui permet de faire plusieurs simulations. Plus précisément `X = rd.random(10)` crée un vecteur ligne `X` contenant 10 réels aléatoires de $[0, 1[$. De même `M = rd.random([10,10])` crée une matrice `M` d'ordre $(10, 10)$ contenant 100 réels aléatoires de $[0, 1[$.
Ainsi pour simuler 1000 lancers d'un dé honnête et pour renvoyer le nombre de "pile" obtenus on écrit :

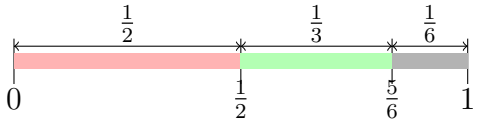
```
X = rd.random(size=1000)
print(np.sum(X<0.5))
```

- **Simulation d'un événement de probabilité p**

Méthode : Simulation d'un évènement de probabilité p.

Tout évènement de probabilité p se simule avec la condition **rd.random()** $< p$.

Exemple. Une urne contient 3 boules rouges, 2 boules vertes et 1 boule noire. Il y a donc une probabilité $\frac{3}{6} = \frac{1}{2}$ de tirer une rouge, une probabilité $\frac{2}{6} = \frac{1}{3}$ de tirer une verte et une probabilité $\frac{1}{6}$ de tirer une noire.
On décompose l'intervalle $[0, 1[$ en fonction de ces probabilités :



Pour simuler le tirage d'une boule on écrit donc :

```
1 alea = rd.random()
2 if alea < 1/2 :
3     print("Rouge")
4 elif alea < 5/6 :
5     print("Vert")
6 else :
7     print("Noir")
```

2 La fonction `randint()`.

L'instruction `rd.randint(a,b)` où a et b sont deux entiers tels que $a < b$ permet de simuler une loi uniforme sur $[\![a, b-1]\!]$.

- **Lancer d'un dé**
  Pour simuler le lancer d'un dé honnête on écrit donc : `de = rd.randint(1,7)`.
  Pour simuler 100 lancers d'un dé honnête on écrit : `rd.randint(1,7,100)`.

3 Simulation de lois usuelles.

On décrit dans le tableau suivant les instructions pour simuler les lois discrètes usuelles. Comme pour les fonctions `randint()` et `random()` on peut simuler plusieurs fois une même loi usuelle en ajoutant une option de taille.

Instructions	Simulation
`X=rd.randint(a,b+1)`	$\mathcal{U}([\![a, b]\!])$
`X=rd.geometric(p)`	$\mathcal{G}(p)$
`X=rd.binomial(n,p)`	$\mathcal{B}(n, p)$
`X=rd.poisson(lambda)`	$\mathcal{P}(\lambda)$

4 Tracé d'histogrammes.

Les fonctions de ce paragraphe se trouvent dans la bibliothèque `matplotlib.pyplot` que l'on importe en écrivant : `import matplotlib.pyplot as plt`.
On simule une expérience aléatoire un grand nombre de fois (disons 1000). Si X est une variable aléatoire relative à cette expérience, la simulation va produire une série statistique que l'on va mémoriser sous la forme d'un vecteur ligne `X` chacun des `X[i]` représentant la valeur de X lors de la i-ème occurrence de l'expérience.

- **La fonction `hist()`.**
  `plt.hist(X,bins=L)` permet de tracer l'histogramme de l'échantillon selon les classes entrées dans l'ordre croissant dans la matrice ligne L. Plus précisément, si $L = [a_0, a_1, \cdots, a_{n-1}]$ les classes seront $[a_0, a_1[$, $[a_1, a_2[$, etc.
  Ainsi si $X(\Omega) = [\![1, 10]\!]$ on pourra écrire `plt.hist(X,bins=[i+0.5 for in range(11)])`
- **La fonction `bar()`.**
  Elle a pour syntaxe `bar(X,Y)`. Où `X` est une matrice ligne d'abscisses et `Y` est la matrice ligne des ordonnées correspondantes. Cela a pour effet de tracer des bâtons de hauteur `Y[k]` à chaque abscisse `X[k]`.
  Exemple. Si X une variable aléatoire dont la loi théorique est donnée par $X(\Omega) = [\![1, n]\!]$ et
  $$\forall k \in [\![1, n]\!], \ \mathcal{P}(X = k) = p_k$$
  Pour tracer l'histogramme théorique de X ont peut écrire :

```
1 Xt = np.zeros(n+1)
2 for k in range(1,n+1):
3     Xt[k] = # valeur de pk
4 plt.bar(range(n+1),Xt)
5 plt.show()
```

5 Indicateurs de position.

- L'espérance empirique d'un échantillon X se calcule par `np.mean(X)`.
- La variance par `np.mean(X**2)-mean(X)**2`.

Fonction de répartition théorique.

Soit X une variable aléatoire dont la loi est donnée par : $X(\Omega) = [\![0, n]\!]$ et

$$\forall k \in [\![0, n]\!], \ \mathcal{P}(X = k) = p_k$$

La fonction de répartition de X est définie par : $\forall x \in \mathbb{R}, \ F_X(x) = \mathcal{P}(X \leq x)$.
Si on a stocké dans un vecteur ligne `X` les valeurs des p_k pour k compris entre 0 et n, alors les valeurs de F sont stockées dans la matrice ligne `F = np.cumsum(X)` qui mémorise dans `F[k]` la somme des valeurs de `X[0]` à `X[k]`.
Le tracé de la fonction de répartition empirique se fait donc par : `plt.step(range(n),np.cumsum(X))`