Simulation de lois discrètes - Statistique descriptive univariée

Dans ce chapitre, on se propose de faire quelques rappels sur les simulations de variables aléatoires discrètes ainsi que sur la statistique descriptive univariée (on parle aussi de statistique simple). L'objectif de la statistique descriptive univariée est d'étudier le comportement d'une population donnée à partir d'observations effectuées sur cette population. Cela passe notamment par le calcul, à partir de données observées sur le terrain, de nouvelles données permettant de décrire rapidement les tendances générales de la population étudiée.

1 Rappels sur les simulations de lois discrètes

En Python, on est souvent amené à écrire des listes d'instructions ou des fonctions permettant de créer des nombres réels, des vecteurs ou des matrices dont les entrées sont des valeurs d'une variable aléatoire suivant une loi donnée au départ. On parle alors de *simulation d'une loi de probabilité*, voire de *simulation d'une loi discrète* si la loi en question est celle d'une variable aléatoire discrète. Pour ce faire, on aura besoin d'importer au préalable la bibliothèque `numpy.random`, via la commande :

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

Définition 1.1. *La fonction* `rd.random()` *permet de créer un nombre réel aléatoire choisi dans* $[0,1[$. *Plus généralement :*

1. *la fonction* `rd.random(r)` *permet de créer un vecteur à* r *composantes, lesquelles sont des nombres réels aléatoires choisis de façon indépendante dans* $[0,1[$.
2. *la fonction* `rd.random([r,s])` *permet de créer une matrice de* $\mathcal{M}_{r,s}(\mathbb{R})$, *dont les composantes sont des nombres réels aléatoires choisis de façon indépendante dans* $[0,1[$.

En fait, la fonction `rd.random()` permet de simuler une variable aléatoire qui suit la loi uniforme sur l'intervalle $[0,1[$. On aura l'occasion d'y revenir quand on abordera le chapitre sur les "variables à densité".

Définition 1.2. (Simulation des lois discrètes usuelles)

— *La commande* `rd.randint(n)` *simule la loi uniforme sur* $[\![0, n-1]\!]$.

— *La commande* `rd.randint(a,b+1)` *simule la loi uniforme sur* $[\![a, b]\!]$.

— *La commande* `rd.binomial(n,p)` *simule la loi binomiale* $\mathcal{B}(n,p)$.

— *La commande* `rd.geometric(p)` *simule la loi géométrique* $\mathcal{G}(p)$.

— *La commande* `rd.poisson(lambda)` *simule la loi de Poisson* $\mathcal{P}(\lambda)$.

A noter que la loi de Bernoulli peut se simuler à l'aide de la commande `rd.binomial(1,p)`. De plus, si l'on veut réaliser k simulations indépendantes d'une certaine loi, on pourra rajouter l'indice k comme dernier paramètre. Par exemple, la commande `rd.geometric(p,k)` retourne un vecteur formé de k réels choisis de façon indépendante et suivant la loi géométrique $\mathcal{G}(p)$. Enfin, si l'on veut créer une matrice de taille $r \times s$ formée de simulations indépendantes d'une certaine loi, on pourra rajouter le couple d'indices $[r,s]$ comme dernier paramètre. Par exemple, la commande `rd.geometric(p,[r,s])` retourne une matrice de taille $r \times s$ formée de réels choisis de faqn indépendante et suivant la loi géométrique $\mathcal{G}(p)$.

A noter enfin que Python considère les booléens comme des entiers, en attribuant la valeur 1 si le booléen en question est vrai et 0 s'il est faux. Ceci permet de simuler dans certains cas des lois discrètes.

Exemple 1.3. *Supposons que l'on veuille simuler la loi d'une variable binomiale de paramètres* 10 *et* 0.5, *et ce sans utiliser la commande* `rd.binomial`. *Dans ce cas, on peut utiliser le fait qu'une variable binomiale de paramètres* n, p *compte le nombre de succès lors de* n *épreuves de Bernoulli indépendantes, où la probabilité de succès est toujours égale à* p. *Dès lors, on pourra utiliser la commande* `x=np.sum(rd.random(n)<=p)`.

2 Notion de série statistique

Dans ce qui suit, une *population* est l'ensemble des éléments dont on étudie les données, et ses éléments sont appelés des *individus*. Un *échantillon* est alors une liste finie d'individus pour lesquels on observe une certaine caractéristique, que l'on appelle une *variable* et que l'on désigne par la lettre X (comme pour une variable aléatoire). On dira que la variable en question est *quantitative* si X prend des valeurs numériques, et *qualitative* sinon. Par la suite, on s'intéressera essentiellement à des variables quantitatives. Commençons par quelques définitions.

Définition 2.1. *Les valeurs $x_1, ..., x_N$ prises par une variable X (sur un échantillon) sont appelées les modalités de X. Une série statistique simple (d'un échantillon) est la donnée de la liste $(x_i)_{1\leq i\leq N}$ des modalités de X.*

On parle de *série statistique brute* si les modalités sont données dans un ordre quelconque, et de *série statistique ordonnée* si elles sont données dans l'ordre croissant, c'est-à-dire si : $x_1 \leq ... \leq x_N$.

Définition 2.2. *Etant donnée une série statistique, l'effectif n_i de la modalité x'_i ou de la i-ème classe est le nombre d'individus correspondant à cette modalité ou à cette classe.*

Définition 2.3. *Etant donnée une série statistique, l'effectif cumulé d'une modalité x'_i est la somme des effectifs des modalités $\leq x'_i$.*

Définition 2.4. *Etant donnée une série statistique, la fréquence d'une modalité x'_i ou de la i-ème classe est le réel $f_i = \frac{n_i}{N}$, où n_i est l'effectif de la modalité ou de la classe, et N est l'effectif total de la population.*

Définition 2.5. *Etant donnée une série statistique, la fréquence cumulée d'une modalité x'_i est la somme des fréquences des modalités $\leq x'_i$.*

A noter que, pour calculer l'effectif de la modalité x_i d'une série statistique X, on pourra utiliser la commande `a=np.sum(X==X[i])`. Si l'on cherche à calculer la fréquence de cette modalité, on pourra utiliser la liste de commandes :

```
n=np.shape(X)[0]
f=np.sum(X==X[i])/n
```

De même, pour calculer l'effectif cumulé de la modalité x_i d'une série statistique x, on pourra utiliser la commande `a=np.sum(X<=X[i])`. Si l'on cherche à calculer la fréquence cumulée de cette modalité, on pourra utiliser la liste de commandes :

```
n=np.shape(X)[0]
f=np.sum(X<=X[i])/n
```

Dans certains cas, on peut représenter la série statistique sous la forme d'un ***diagramme en bâtons***. Dans ce cas, on se place dans un repère $(O, \vec{i}, \vec{j})$, on indique sur l'axe (Ox) les modalités et on trace à la verticale de chacune d'entre elles un bâton de hauteur égale à son effectif (ou à sa fréquence). Pour obtenir un tel diagramme, on utilise la fonction `plt.bar(X,n)`, où X est la liste des modalités d'une série statistique et n est le vecteur des effectifs correspondants. A noter que la commande `plt.bar` n'est pas exigible des étudiants. Elle sera donc rappelée dans chaque énoncé où on en aura besoin.

Exemple 2.6. *Considérons la série statistique donnée par le vecteur $X = (1, 5, 1, 2, 3, 4, 4, 2, 1)$. Si l'on veut tracer le diagramme en bâtons de la série statistique donnée par X, on tapera dans la fenêtre de commande la liste de commandes suivante :*

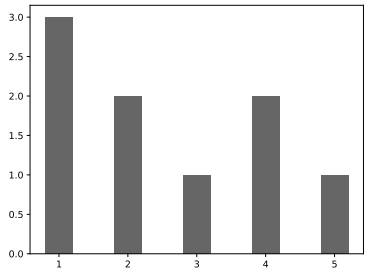
```
> x=np.array([1,5,2,3,4])    (liste des modalités)

> n=np.array([3,1,2,1,2])    (liste des effectifs correspondants)

> plt.bar(x,n, width=0.4, color='0.4')

> plt.show()
```

Dans ce cas, Python retournera le diagramme suivant :



A noter que l'on a spécifié ici la largeur et la couleur des bâtons (`width=0.4, color='0.4'`). Par défaut, la couleur des bâtons est bleue, et on peut la changer par une autre couleur, voire par une certaine teinte de gris avec un nombre compris entre 0 *et* 1*, selon la convention "0" pour "noir" et "1" pour "blanc".*

Enfin, il existe un deuxième mode de représentation graphique d'une série statistique, plus classique et exigible des étudiants, donné par la :

Définition 2.7. *Etant donnée une série statistique groupée en classes* $[a_1, a_2],]a_2, a_3], ...,]a_{q-1}, a_q]$*, l'histogramme de cette série est le diagramme du plan obtenu en plaçant les* a_i *sur l'axe* (Ox) *et en traçant les rectangles de base* $[a_i, a_{i+1}]$ *et de hauteur proportionnelle à l'effectif* n_i *de la classe en question.*

Pour construire l'histogramme d'une série statistique, on utilisera la commande `plt.hist` (après avoir importé la librairie `matplotlib.pyplot` au préalable, comme pour la commande `plt.bar`). Plus précisément, si X est une série statistique brute et si l'on veut tracer l'histogramme de X en n classes de même amplitude, on utilisera la commande `plt.hist(X,n)`. De même, si l'on veut tracer l'histogramme de X selon des classes dont les extrémités sont les composantes d'un vecteur c, on utilisera la commande `plt.hist(X,c)`.

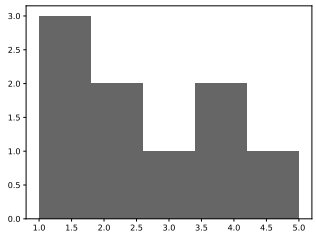
Exemple 2.8. *Considérons la série statistique donnée par le vecteur* $X = (1, 5, 1, 2, 3, 4, 4, 2, 1)$*. Si l'on veut tracer l'histogramme de* x *en* 5 *classes de même amplitude, on tapera dans la fenêtre de commande la liste de commandes suivante :*

```
> x=np.array([1,5,1,2,3,4,4,2,1])

> plt.hist(x,5)

> plt.show()
```

Dans ce cas, Python affichera l'histogramme suivant :



Si l'on veut maintenant tracer l'histogramme de X *suivant les classes* $[1, 2.5[, [2.5, 3.5[, [3.5, 4.5[, [4.5, 5]$*, alors on tapera dans la fenêtre de commande la liste de commandes suivante :*

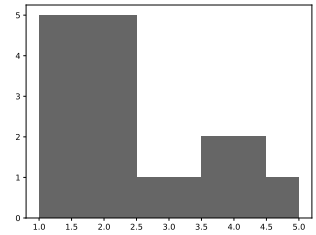
```
> x=np.array([1,5,1,2,3,4,4,2,1])

> c=np.array([1,2.5,3.5,4.5,5])

> plt.hist(x,c, color='0.4')

> plt.show()
```

Dans ce cas, Python affichera l'histogramme suivant :



A noter que l'on a spécifié ici la couleur de l'histogramme suivant la même convention que pour les diagrammes en bâtons. A noter enfin que, par défaut, les hauteurs des colonnes de l'histogramme en Python sont proportionnelles aux effectifs de chaque classe. Si l'on veut que les aires des colonnes obtenues soient proportionnelles aux effectifs de chaque classe, on doit rajouter une option à la commande `plt.hist(X,c)`, sous la forme `plt.hist(X,c,density=True)`.

3 Commandes statistiques

3.1 Paramètres de position

Définition 3.1. *Etant donnée une série statistique brute $(x_i)_{1\leq i\leq N}$, la moyenne de cette série est le réel :*

$$\overline{X} = \frac{1}{N}\sum_{i=1}^{N} x_i.$$

Pour calculer la moyenne d'une série statistique brute X, on utilisera la commande `np.mean(X)`.

Définition 3.2. *Etant donnée une série statistique ordonnée, la médiane de cette série est le réel M_e partageant la série en deux séries de mêmes effectifs.*

Plus précisément, considérons une série statistique ordonnée $X = (x_1, x_2, ..., x_N)$, avec $x_1 \leq x_2 \leq ... \leq x_N$. Si N est pair, de la forme $N = 2k$, alors $M_e = \frac{1}{2}(x_k + x_{k+1})$. Si N est impair, de la forme $N = 2k+1$, alors $M_e = x_{k+1}$. La commande `np.median(X)` permet de calculer la médiane d'une série statistique brute X.

Exemple 3.3. *Considérons la série statistique donnée par le vecteur $X = (1, 5, 1, 2, 3, 4, 4, 2, 1, 7)$. Si l'on veut calculer la moyenne et la médiane de X, il suffit de taper dans la fenêtre de commande la liste de commandes suivante :*

```
> x=np.array([1,5,1,2,3,4,4,2,1,7])

> m,n=np.array([np.mean(x),np.median(x)]

> print(m,n)
```

Dans ce cas, Python affichera le résultat `3.0 2.5`.

3.2 Paramètres de dispersion

Définition 3.4. *Etant donnée une série statistique, l'étendue de cette série est la différence entre la plus grande et la plus petite des modalités.*

Pour calculer l'étendue d'une série statistique brute X, on pourra utiliser les commandes `np.min` et `np.max`, sous la forme `e=np.max(X)-np.min(X)`.

Définition 3.5. *Etant donnée une série statistique brute $X = (x_i)_{1\leq i\leq N}$, la variance et l'écart-type de de cette série sont définis comme les réels :*

$$V(X) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{X})^2 \quad et \quad \sigma_X = \sqrt{V(X)}.$$

Pour calculer la variance (respectivement l'écart-type) d'une série statistique brute x, on utilisera la commande `np.var(X)` (respectivement `np.std(X)`).

Exemple 3.6. *Considérons la série statistique donnée par le vecteur $X = (1, 5, 1, 2, 3, 4, 4, 2, 1, 7)$. Si l'on veut calculer l'écart-type de x, il suffit de taper dans la fenêtre de commande les commandes suivantes :*

```
> x=np.array([1,5,1,2,3,4,4,2,1,7])

> s=np.std(x)

> print(s)
```

Dans ce cas, Python affichera le résultat `1.89736659`.