TP 16 : Histogrammes et diagrammes bâtons associés aux variables aléatoires

Commencer par importer les bibliothèque **matplotlib.pyplot** , **numpy** et **numpy.random**

1 Représentation graphique de lois

Dans cette partie on cherche à représenter des lois sous forme de diagrammes en bâtons. Pour ce faire on utilisera la commande Python `plt.hist`.

On lance deux dés équilibrés à 6 faces et on note X la somme des résultats obtenus.

1. Préciser $X(\Omega)$.
2. Compléter le tableau ci-dessous donnant la loi de X.

x_i	2	3	4								
$P(X=x_i)$	$\frac{1}{36}$	$\frac{2}{36}$	$\frac{3}{36}$	$\frac{4}{36}$	$\frac{5}{36}$	$\frac{6}{36}$					

3. Construire un vecteur X contenant les valeurs du support de X (c'est-à-dire les valeurs de $X(\Omega)$) et un vecteur P contenant les probabilités calculées à la question précédente.
4. Taper plt.bar(X,P) à la suite du code de la question 3.
5. On considère maintenant la variable aléatoire $Y = \max(X, 3)$ où X est la variable aléatoire définie ci-dessus.
   Préciser $Y(\Omega)$.
6. Compléter le tableau ci-dessous donnant la loi de Y.

y_i										
$P(Y=y_i)$										

7. Écrire quelques lignes de code permettant de tracer un diagramme en bâtons représentant la loi de Y.

2 Histogrammes

Le but de cette partie est de comprendre ce qu'est un histogramme et d'apprendre à les tracer. Un histogramme permet de visualiser rapidement la distribution d'une série statistique. Cela est particulièrement utile lorsque la loi d'une variable aléatoire n'est pas connue mais qu'on dispose d'un certain nombre d'observations de X.

2.1 Loi uniforme

1. Compléter le code ci-dessous pour simuler 10 fois une variable aléatoire $X \hookrightarrow \mathcal{U}([|-2,3|])$ et qui enregistre les résultats des simulations dans un tableau de taille 1×10.

```
1 n=...
2 X=rd.randint(...,...,....)
3 display("X= ", X)
```

2. À la suite du code de la question précédente taper

```
p=6
plt.hist(X,p)
```

3. Essayer de comprendre le graphique obtenu. Pour ce faire on pourra réfléchir aux questions suivantes.
   (a) Donner $X(\Omega)$. Quel est le cardinal de cet ensemble ?
   (b) Changer la valeur de p dans le code ci-dessus par un autre **entier naturel non-nul**.
   (c) Changer la valeur de n dans le code ci-dessus par un autre **entier naturel non-nul**.
   (d) Que remarque-t-on lorsque n est grand ?

4. Essayer maintenant la commande suivante :

   ```
   plt.hist(X,[-2,1,2,3])
   ```

   (a) Tester avec des nombres différents **classé par ordre croissant**.
   (b) Que permet cette commande ?

2.2 Loi binomiale

1. On rappelle que la commande `rd.binomial(5,1/2)` permet de simuler une loi binomiale.
2. Préciser $\text{card}(X(\Omega))$.
3. Faire 1000 simulations de $X \hookrightarrow \mathcal{B}(5, 1/2)$ puis tracer l'histogramme correspondant à ces simulations. On choisira le deuxième nombre dans la fonction plt.hist de manière judicieuse (cf. question 2).

2.3 Bernoulli

Réaliser 1000 simulations de la variable aléatoire X et dessiner l'histogramme du jeu de données obtenu lorsque X suit une loi de Bernoulli de paramètre $p = \frac{1}{3}$.

2.4 Lancés de dés

Soit $n \in \mathbb{N}^*$ un entier naturel non-nul fixé. On lance n dés équilibrés. Soit S la variable aléatoire correspondant à la somme des scores obtenus.

1. Donner $S(\Omega)$ et $\text{card}(S(\Omega))$.
2. Écrire une fonction Python, SommeDe(n), prenant en entrée un entier naturel non-nul n et qui permette de simuler S.
3. On itère m fois cette expérience.
   Ecrire une fonction qui prend n et m en argument et qui affiche un tableau ligne contenant m simulations de S.

```
def tableauS(n,m):
        y=np.ones(m)
        for i in range(m):
            y[i]=...
        return(y)
```

4. Compléter le code ci-dessus pour tracer l'histogramme du jeu de données obtenu.
5. Faire varier n et m. Lorsque $m >> n >> 1$ on doit obtenir un histogramme dont l'allure est proche de celle d'une gaussienne.