TP2 - Probabilités discrètes

1 Quelques bases utiles

1.1 Tableaux numpy

Dans numpy, nous travaillons presque toujours exclusivement avec des tableaux numpy. Pour rappel, ces tableaux peuvent avoir une ou deux (ou plus) dimensions. Il sont utiles particulièrement pour représenter des matrices (dans ce cas, ils ont deux dimensions) ou des séries de données statistiques, ce que nous ferons aujourd'hui.

Exercice 1 - Rappels tableaux numpy ⋆

Décrire de tête le résultat de chacune des lignes suivantes puis le vérifier dans Pyzo :

```
1 import numpy as np
  np.array([[1,2,3],[4,5,6],[7,8,9]])
  np.array([5,8,9,-1,3,4])
  np.zeros((3,4))
5 np.zeros(5)
```

```
1 np.ones((4,3))
  np.ones(7)
  np.arange(2,7)
  np.linspace(2,10,9)
5 np.linspace(2,4,5)
```

1.2 Simulations avec numpy.random

La biblothèque numpy possède une série de fonctions permettant de simuler des variables aléatoires suivant un certain nombre de lois connues.

Exercice 2 - Simulation de lois usuelles ⋆

Décrire le résultat de chacune des lignes suivantes :

```
1 import numpy.random as rd
  rd.binomial(15,0.5,30)
  rd.binomial(50,0.2,15)
```

```
1 rd.geometric(0.2,15)
  rd.geometric(0.5,30)
  rd.poisson(0.2,20)
  rd.poisson(2,25)
```

Si besoin, se reporter à la fiche rappel Python de début d'année. Vous pouvez aussi exécuter les fonctions dans Pyzo, éventuellement en modifiant les paramètres, et voir le résultat obtenu.
Quelle loi usuelle manque-t-il dans les lignes ci-dessus ?

1.3 Analyse d'une simulation

Une fois que l'on a produit un tableau numpy contenant des résultats de simualtion, on peut les traiter avec des fonctions spécialisées.

Exercice 3 ⋆

Celles qui nous intéressent sont :

- np.min ;
- np.max ;
- np.mean ;
- np.median ;
- np.var ;
- np.std.

- la moyenne de la série ;
- la variance de la série ;
- la plus petite valeur de la série ;
- la plus grande valeur de la série ;
- la médiane de la série ;
- l'écart-type de la série.

1. Relier chaque fonction dans la colonne de gauche à son rôle dans la colonne de droite.
2. On écrit une fonction :

```
1 def simul_binom(n,p,k):
    simulations = rd.binomial(n,p,k)
    print(np.mean(simuations))
```

Cette fonction sera complétée dans la question suivante. En l'état qu'affiche-t-elle ?

3. Compléter la fonction pour qu'elle affiche également la variance de la série simulée.
4. Rappeler la valeur théorique de la moyenne et de la variance d'une loi binomiale. Vérifier que ces valeurs apparaissent bien avec la fonction précédente en l'exécutant sur quelques exemples. A-t-on exactement les valeurs théoriques ?

2 Exercices

2.1 Applications du cours

Exercice 4 ⋆

Écrire une fonction d'en-tête :

```
1 def simul_poisson(mu,n):
```

qui prend deux paramètres `mu` et `n` et qui affiche la moyenne et la variance d'une série de n tirages d'une variable aléatoire suivant une loi de Poisson de paramètre `mu`.
Quelles valeurs devraient s'afficher ? A-t-on toujours exactement cela ? Obtient-on un résultat plus précis si n est petit ? grand ?

Exercice 5 - Programmer une loi de Bernoulli ⋆

Un des problèmes de `numpy` est qu'il n'y a pas de simulation de loi de Bernoulli facilement utilisable. Nous allons en programmer une. Le but est de faire une fonction d'en-tête :

```
1 def bernoulli(p,k):
```

qui prend deux paramètres et qui renvoit un tableau `numpy` contenant k simulations d'une variable suivant une loi de Bernoulli de paramètre p.
On propose de partir du code suivant :

```
1 def bernoulli(p,k):
    simulations = np.zeros(k)
    for i in range(...):
      if rd.random() < p :
5       simulations[...] = 1
    return simulations
```

où les points de suspensions sont à compléter.

1. Quel est le rôle de la ligne `simulations = np.zeros(k)` ?
2. Rappeler ce que fait l'instruction `rd.random()`. Avec quelle probabilité l'instruction `simulations[...] = 1` s'exécute-t-elle ? Combien de fois faut-il donc que la boucle s'exécute ?
3. À quoi servent les crochets dans la ligne `simulations[...] = 1` ? À quel numéro faut-il commencer ? En déduire comment compléter la fonction proposée.
4. Exécuter votre fonction sur quelques exemples. Fonctionne-t-elle comme attendu ? Vérifier en particulier que la moyenne et la variance ont bien les valeurs attendues.

Exercice 6 - Reprogrammer une loi géométrique ⋆⋆

La loi géométrique est déjà très bien simulée par `rd.geometric`. Mais dans cette exercice, nous allons chercher à la simuler *à la main*[1] en partant de sa définition.
Compléter la fonction suivante afin qu'elle simule un unique tirage d'une variable suivant une loi géométrique :

1. C'est-à-dire sans faire appel à la fonction `numpy` dédiée.

```
def tirage_geom(p):
  comptage = ...
  while rd.random() ... :
    ...
  return comptage
```

Exercice 7 - Simulation de mobile - EDHEC 2018 (adapté et modifié) ⋆⋆

Cette exercice s'inspire de l'exercice 3 du second [2] sujet de l'EDHEC 2018.
Un mobile se déplace aléatoirement sur un axe dont l'origine est la point O d'abscisse 0. Au départ, à l'instant 0, le mobile se situe sur le point O. Le mobile se déplace selon la règle suivante : à l'instant n ($n \in \mathbb{N}^\star$), il se place de façon équiprobable sur l'un des points d'abscisses 0, 1, ..., n.
Pour tout entier naturel n, on note X_n l'abscisse de ce point à l'instant n. On a donc $X_0 = 0$. On admet que $(X_n)_{n\in\mathbb{N}}$ est une suite de variables aléatoires mutuellement indépendantes.
On note enfin T l'instant auquel, pour la première fois, le mobile revient à l'origine. On a donc $X_T = 0$.

1. On rappelle que l'on peut simuler un tirage d'une loi uniforme avec `rd.randint`. Compléter la fonction suivante suivante afin qu'elle simule un tirage de la variable aléatoire T :

```
def simul_T():
  val = 1
  while ... :
    val = val + 1
  return val
```

2. Compléter la fonction suivant pour qu'elle remplisse un tableau `numpy` avec k simulations de T :

```
def simul_Ts(k):
  simulations = np.zeros(k)
  for i in range(...):
    simulations[...] = simul_T()
  return simulations
```

   Utiliser cette fonction pour générer 10 000 simulations de T et les stocker dans un tableau que l'on nommera A.

3. Dans le sujet de l'EDHEC, il a été prouvé que $P(T = n) = \frac{1}{n(n+1)}$. **On ne cherchera pas à prouver cette formule.** Nous allons plutôt utiliser l'ordinateur pour la vérifier sur des simulations.

   (a) On considère les codes suivants :

```
u = np.zeros(10)
for i in range(10000):
  if A[i] <= 10:
    index = int(A[i])-1
    u[index] = u[index] + 1
u = u / 10000
```

```
v = np.zeros(10)
for i in range(1,11):
  v[i-1] = 1/(i*(i+1))
```

   Expliquer ce que fait chaque code. Les recopier et les exécuter.

   (b) On peut tracer des courbes en utiliser la bibliothèque `matplotlib`. On l'importe avec :

```
import matplotlib.pyplot as plt
```

2. Suite à des problèmes en série, 3 sujets différents ont été donnés l'année 2018 pour l'EDHEC. Le premier sujet a été diffusé par erreur avant le concours. Un second sujet (dont s'inspire cet exercice) a été utilisé pour le jour de l'épreuve. Mais comme une erreur a été commise dans un centre qui a donné le premier sujet, l'épreuve a été annulée et un troisième sujet a été utilisé pour une nouvelle épreuve.

Une fois importée, et en vous reportant à la fiche rappel de `Python` de début d'année, tracer sur le même graphique la loi de T et la simulation de cette loi. Les probabilités doivent apparaître en ordonnées et les valeurs possibles pour T en abscisses.

Les commandes importantes sont `plt.plot` et `plt.show`.

4. (a) Montrer que T n'admet pas d'espérance.
   (b) On peut pourtant calculer l'espérance de T sur un grand nombre de simulations. Écrire une fonction qui remplit un tableau avec les espérances d'un nombre croissant de simulation et utiliser `matplotlib` pour en tracer la courbe. Que remarque-t-on ?

2.2 Pour aller plus loin

Exercice 8 - Loi de Borel ⋆⋆⋆⋆

On considère une caisse de supermarché fonctionnant sur le principe suivant : chaque client passe une minute à la caisse et pendant qu'un client est servi, le nombre de clients arrivant à la caisse est une variable aléatoire suivant une loi de Poisson $\mathcal{P}(\mu)$.

1. Écrire une fonction `caisse` de paramètre `mu` estimant le nombre de minutes nécessaires après l'arrivée du premier client avant que la caisse soit vide. Essayer cette fonction à l'aide de plusieurs valeurs de μ.
2. Constater que pour $\mu < 1$, la moyenne est de l'ordre de $\frac{1}{1-\mu}$. Essayer d'expliquer ce résultat de manière intuitive en considérant le nombre moyen de clients se présentant à la caisse pendant que le premier est servi, puis le nombre moyen de clients arrivant à la caisse pendant que ces « clients de première génération » sont servis, etc.
3. La probabilité que ce nombre de clients soit égal à n est égale à $\frac{e^{-\mu n}(\mu n)^{n-1}}{n!}$. Vérifier cela à l'aide d'un graphique comme dans l'exercice 7.

3 Travail à préparer pour le prochain TP

Exercice 9 - Viande ou poisson ⋆⋆

Le nombre de clients qui entrent dans un restaurant un soir de semaine est une variable aléatoire X qui suit une loi de Poisson de paramètre μ. Chaque client choisit indépendemment du choix de ses voisins de la viande avec une probabilité p et du poisson avec une probabilité $1-p$. On note Y le nombre de clients qui choisissent de ma viande.

1. Écrire une fonction d'en-tête :

```
1 def restaurant_simul(mu,p):
```

qui prend les deux paramètres pertinents et renvoient la valeur simulée de X et Y pour une soirée au restaurant.

Note : pour renvoyer deux valeurs à la fois, il faut les séparer par des virgules. Par exemple :

```
1 return x,y
```

2. Écrire une fonction :

```
1 def restaurant_esp_y(mu,p,k):
```

qui calcule, sur un échantillon de k soirées simulées, l'espérance de Y.

Note : pour récupérer deux valeurs à la fois, il faut séparer par des virgules les deux variables utilisées pour les stocker. Par exemple :

```
1 x,y = soiree_restaurant(mu,p)
```