TP3A - Probabilités discrètes

1 Tableaux 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 séries de données statistiques.

Exercice 1 - Rappels tableaux numpy ⋆

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

```
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))
np.zeros(5)
```

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

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 :

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

```
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 ?

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 :

```
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 ?

4 Exercices d'application

Exercice 4 ⋆

Écrire une fonction d'en-tête `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 def tirage_geom(p):
    comptage = ...
    while rd.random() ... :
      ...
5   return comptage
```

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