SIMULATIONS D'EXPÉRIENCES ALÉATOIRES DISCRÈTES

Les parties marquées d'un ⛰, ne doivent être abordées que si vous avez de l'avance.

Objectifs, motivations et premier pas

Dans ce TP nous allons créer des fonctions permettant de simuler les principales lois usuelles discrètes au programme et hors programme. Nous allons aussi utiliser des représentations graphiques pour visualiser les distributions des lois simulées.

Il faut bien distinguer

- *Simuler* une variable aléatoire ou une expérience aléatoire. On écrit une fonction python qui donne un résultat aléatoire qui correspond à l'expérience décrite. À chaque fois que nous utiliserons cette fonction le résultat obtenu peut être différent, mais les fréquences d'obtention de chaque résultat doivent s'approcher de la probabilité théorique.
- *Calculer la loi* d'une variable aléatoire, ce qui a été fait en cours de mathématiques.

Remarque : La plupart des fonctions que nous allons écrire existent déjà dans différents modules notamment `numpy`, `numpy.random`[1]. Quand cela est possible le corrigé indique la fonction `numpy` qui effectue la tache demandée.

Nous proposons ici des versions naïves qui ne correspondent pas toujours à celles utilisées en pratique mais utilisent les schémas classiques.

À programmer 1 (Préambule)**.**
Télécharger le fichier `Simulation.py`. Le compléter en important les modules :

1. `numpy` avec l'alias `np`
2. `numpy.random` avec l'alias `rd`
3. `matplotlib.pyplot` avec l'alias `plt`

La maîtrise du module de tracé de graphe n'est pas un attendu du programme et il peut être compliqué sans entraînement d'obtenir des graphiques exploitables, c'est pourquoi on vous donne une fonction `Graphe`, qui prend en argument un objet de type tableau et qui affiche un graphe représentant les fréquences.

À programmer 2 (Test de la fonction `Graphe`)**.**
Recopier la ligne suivante :

```
Graphe([0,1,2,1,2,1,0])
```

1. ou pour les distributions les plus récentes `numpy.random.Generator`

I Première simulation, et fonctions utiles

Consigne Dans la suite le seul générateur aléatoire que nous allons utiliser est la fonction :

```
numpy.random.random()
```

qui renvoie un réel choisi de façon uniforme dans l'intervalle $[0;1[$.

À programmer 3 (Bernoulli)**.**
Écrire une fonction `Ber(p)` qui, pour une valeur réelle $p \in]0;1[$, renvoie une simulation d'une variable aléatoire suivant la loi $\mathscr{B}(p)$; les valeurs renvoyées doivent être 0 ou 1.

Pour tester nos générateurs aléatoires, nous allons effectuer un grand nombre d'expériences pour estimer les fréquences et calculer la moyenne et l'écart-type.... Nous allons en profiter pour manipuler des tableaux numpy `np.array`, ce qui servira de rappel sur ce module.

Rappels numpy

- Pour créer un tableau, on utilise par exemple `np.array([1,2,0,3.5])`
- On obtient la taille d'une matrice `M` à `n` lignes et `p` colonnes grâce à `n,p=M.shape` ; pour un tableau unidimensionnel `T` de taille `n` on peut écrire `n=len(T)`.
- Pour accéder à l'élément d'indice `i` d'un tableau `T` : `T[i]`.
- On ne peut pas utiliser la méthode `append` sur ce type d'objet, la taille du tableau est fixée à la création.
- Pour créer un grand tableau, on peut partir d'un tableau de la bonne taille rempli de 0 avec la commandes `numpy.zeros`, puis modifier les éléments à l'aide d'une boucle : par exemple

```
T=np.zeros(10)
for i in range(10):
    T[i]=i
```

- On peut effectuer des opérations sur le tableau élément par élément[2] sans utiliser de boucle, par exemple : `T**2` (resp. `T+1`) renvoie un nouveau tableau dont les éléments sont les carrés de ceux de `T` (resp. ceux de `T` augmentés de 1)
- On peut obtenir facilement la somme des valeurs de `T` avec `S = T.sum()`. On peut aussi obtenir la moyenne et l'écart-type des valeurs, les noms des méthodes sont ceux utilisés en anglais.

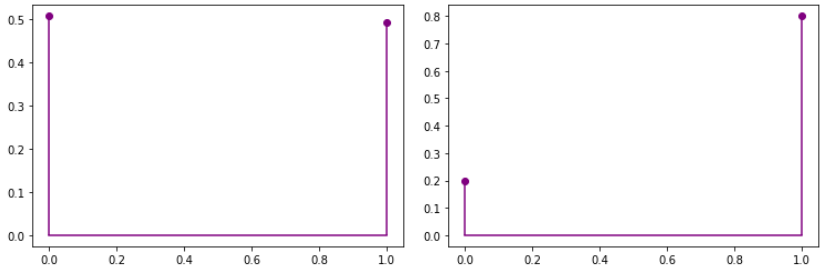
À programmer 4 (Test)**.**
Tester rapidement les commandes précédentes.

2. *elementwise* ou *pointwise* ou *pointed operator*

Nous commençons par créer une fonction générale qui permet de répéter un grand nombre de fois une simulation et qui renvoie un tableau des valeurs obtenues.
Par exemple `Repetition(100,Ber,0.5)` va retourner une série de 100 simulations d'une variable aléatoire suivant une loi de Bernoulli équilibrée. Comme il faut pouvoir passer des arguments à la fonction et que la syntaxe est compliquée, on vous fournit un début de réponse.

À programmer 5 (Complétion)**.**
Décommenter la fonction `Repetition` et la compléter.

À programmer 6 (Test)**.**
Utiliser les fonctions `Repetition` et `Graphe` pour simuler un grand nombre de résultats suivant une loi de Bernoulli. Choisir les paramètres pour obtenir un graphe très proche de

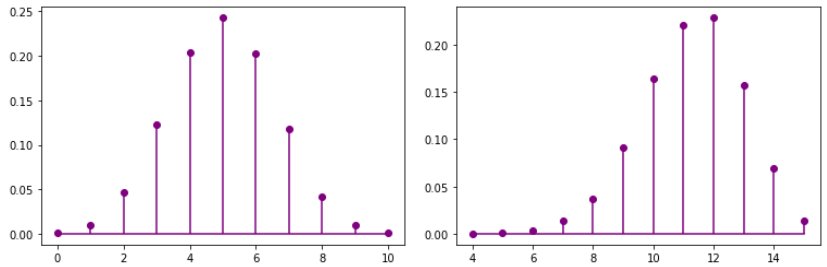


À programmer 7 (Fonctions Moyenne et EcartType)**.**
Écrire une fonction `Moyenne(L)` et une fonction `EcartType(L)` qui renvoient respectivement la moyenne est l'écart-type de la série de valeurs représentées par le tableau `L`.

II Les lois classiques

À programmer 8 (Loi binomiale)**.**
En utilisant le modèle classique de la loi binomiale, écrire une fonction `Binomiale(n,p)` qui simule le résultat d'une variable aléatoire suivant la loi $\mathscr{B}(n,p)$.

À programmer 9 (Test)**.**
En utilisant les fonctions `Repetition` et `Graphe`, tester votre fonction. Bien choisir les paramètres pour obtenir des graphes proches des graphes suivants :

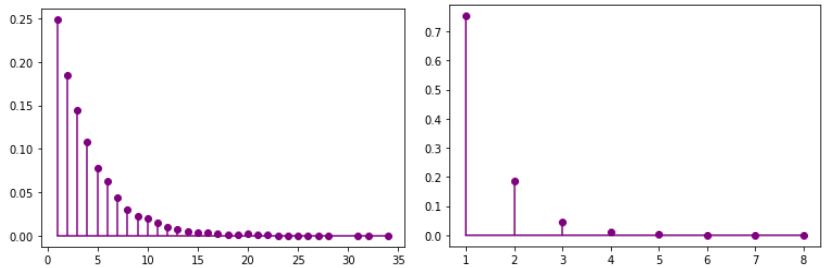


À programmer 10 (Estimation des paramètres)**.**
En utilisant les fonctions `Moyenne` et `EcartType`, vérifier que l'on obtient des résultats cohérents avec les résultats du cours.

À programmer 11 (Loi Géométrique)**.**
En utilisant le modèle classique de la loi géométrique, écrire une fonction `Geometrique(p)` qui simule le résultat d'une variable aléatoire suivant la loi $\mathscr{G}(p)$.

Remarque : Dans le cas où p est très petit, la boucle `while` peut se répéter un grand nombre de fois. nous verrons par la suite comment simuler cette loi autrement.

À programmer 12 (Test)**.**
En utilisant les fonctions `Repetition` et `Graphe`, tester votre fonction. Bien choisir le paramètre pour obtenir les graphes suivants :



À programmer 13 (Estimation des paramètres)**.**
En utilisant les fonctions `Moyenne` et `EcartType`, vérifier que l'on obtient des résultats cohérents avec les résultats du cours.

Loi de Poisson Pour simuler une loi de Poisson, qui n'est pas décrite par un schéma simple, il existe plusieurs procédés. Nous en proposons une qui anticipe un peu sur le programme du second semestre

Sur papier 1 (Calcul d'une Limite)**.**
Soit $\lambda > 0$, et $n \in \mathbb{N}^*$ on suppose que X_n est une variable aléatoire qui suit la loi $\mathscr{B}(n, \frac{\lambda}{n})$

1. Soit $n \in [\![0, n]\!]$, rappeler la valeur de $P(X_n = k)$.
2. calculer $\lim\limits_{n\to+\infty} \left(1 - \frac{\lambda}{n}\right)^{n-k}$
3. Montrer que $\binom{n}{k} \underset{n\to+\infty}{\sim} \frac{n^k}{k!}$
4. Montrer que $\lim\limits_{n\to+\infty} P(X_n = k) = \frac{\lambda^k}{k!} e^{-\lambda}$

On dira que X_n tend en loi vers une variable aléatoire suivant la loi $\mathscr{P}(\lambda)$

À programmer 14.
Pour simuler une variable aléatoire suivant la loi $\mathscr{P}(\lambda)$, on la remplace par une loi $\mathscr{B}(n, \frac{\lambda}{n})$ avec n "grand". En pratique si $\lambda < 15$ on doit choisir n plus grand que 30 et tel que $\frac{\lambda}{n} < 0.1$

Écrire une fonction `Poisson(lam)` qui simule le résultat d'une variable aléatoire suivant la loi $\mathscr{P}(\lambda)$. On commencera par calculer un n qui convient et on utilisera la simulation de Binomiale implémenter précédemment.

À programmer 15 (Test)**.**
En utilisant les fonctions `Repetition` et `Graphe`, tester votre fonction. Le faire pour plusieurs paramètre et décrire l'influence de ce paramètre sur l'allure du graphique.

II.1 Reconnaître une loi

À programmer 16 (Reconnaissance de loi)**.**
Le fichier fourni comporte deux séries de données `Mystere1` et `Mystere2`. Ces séries ont été générées en utilisant l'un des générateurs aléatoires précédents. Le but est de proposer une loi qui permette d'obtenir ce type de série. Pour chacune des deux séries :

- Tracer le graphe de la distribution, et proposer une loi.
- Estimer les paramètres de cette loi ; on pourra utiliser les fonction `Moyenne` et `EcartType`.

III Simuler une loi (finie) quelconque

III.1 Le principe

On se donne une variable aléatoire X dont le support est $[\![0, n]\!]$, et telle que

$$\forall i \in [\![0, n]\!],\ P(X = i) = p_i$$

On représente les probabilités par un tableau P tel que $P[i] = p_i$.

Attention : On suppose de plus dans la suite que toutes les probabilités p_i sont strictement positives

À programmer 17 (Validité)**.**
Écrire une fonction `TestValide(P)` qui renvoie `True` si `P` représente bien une probabilité de ce type, `False` sinon.

On note R la fonction de répartition de X, c'est une fonction à valeurs dans $[0; 1[$, croissante, qui vérifie

$$R(0) < R(1) < \cdots < R(n)$$

On choisit un nombre réel U au hasard[3] dans $[0; 1[$, il existe alors un unique indice i tel que

$$R(i-1) < U \leqslant R(i)$$

(ou si $i = 0$, $U \leqslant R(0)$)

Pour une loi uniforme sur $[0; 1[$, $P(X \in I) = \frac{\text{"longueur de l'intervalle I"}}{\text{"longueur de l'intervalle } [0; 1[\text{"}}$.

Donc la probabilité $P(R(i-1) < U \leqslant R(i))$ est égale à la longueur de l'intervalle $]R(i-1); R(i)]$ que divise 1 ; c'est à dire

$$P\left(R(i-1) \leqslant U < R(i)\right) = R(i) - R(i-1) = P(X = i)$$

Algorithme Pour simuler la variable aléatoire X

- Choisir un nombre réel u au hasard entre 0 et 1
- Trouver l'entier i qui vérifie $R[i-1] < u \leqslant R[i]$; on remarque que c'est le plus petit entier i tel que $R[i] \geqslant u$

III.2 Implémentations

À programmer 18 (Répartition)**.**
Écrire une fonction `Repartition(P)` qui pour un tableau `P`, de longueur $n+1$, représentant une loi d'une variable aléatoire, renvoie le tableau de même longueur représentant la fonction de répartition.

$$\forall i \in [\![0, n]\!] \qquad R[i] = P(X \leqslant i)$$

À programmer 19 (Simulation)**.**
Écrire une fonction `Simulation(R)` qui prend en paramètre un tableau représentant une fonction de répartition, choisit un réel u au hasard dans $[0; 1[$ et renvoie le premier entier tel que $R[i] \geqslant u$.

3. loi uniforme à densité

À programmer 20 (Test)**.**
Tester la fonction précédente (on utilisera les fonctions `Repetitions` et `Graphe`)

- sur une loi non uniforme sur $[\![0, 3]\!]$;
- sur une loi uniforme sur $[\![0, N]\!]$, pour N assez grand.

III.3 ⛰ Support plus général

On suppose maintenant que la loi de X est donnée par

Son support fini $\{x_0, x_1, \ldots, x_n\}$, valeurs toutes différentes, qui est représenté en mémoire par un tableau (ou une liste) S, tel que $S[i] = x_i$)

Les probabilités associées $p_0 = P(X = 0)$, $p_1 = P(X = 1)$,..., $p_n = P(X = n)$, qui sont représentées par un tableau P tel que $P[i] = p_i$

À programmer 21 (Fonction Choisir)**.**
Écrire une fonction `Choisir(Support,Probabilites)` qui renvoie un élément de `S` choisi selon la probabilité `P`. La fonction tient en une ligne. La tester.

III.4 ⛰⛰ Une extension à un support dénombrable

Soit X la variable aléatoire définie par

$$X(\Omega) = \mathbb{N}^* \qquad \forall i \in \mathbb{N}^* \; P(X = i) = \frac{6}{k^2\pi^2}$$

Sur papier 2 (Mathématique)**.**
Montrer que l'on a ainsi définit une variable aléatoire.

Dans ce cas-là on ne peut pas créer un tableau représentant la probabilité, mais en utilisant le même raisonnement on peut simuler cette variable aléatoire.

À programmer 22 (Implémentation)**.**
Écrire une fonction `Rieman()` qui simule cette variable aléatoire.

À programmer 23 (Test)**.**
Faire afficher le graphe des fréquences ; trouver une façon de vérifier que l'on a un résultat cohérent avec la théorie.

À programmer 24 (Espérance)**.**
Montrer qu'une telle variable aléatoire n'admet pas d'espérance, ni de variance. Illustrez ce résultat avec un programme.

IV Autres lois et expériences

Pour les lois qui suivent, vous pouvez vous renseigner sur les espérances, variances et lois dans des ouvrages, ou sur wikipedia.

IV.1 Loi de Pascal

On répète des expériences de Bernoulli indépendantes et identiques. La variable aléatoire qui donne le rang d'apparition du n-ième succès suit une loi de Pascal de paramètres n et p.

À programmer 25 (Simulation)**.**
Écrire une fonction `Pascal(n,p)` qui simule cette variable aléatoire. Utiliser les fonctions précédentes pour afficher la distribution, une estimation de l'espérance et de la variance. Vérifier la cohérence avec les valeurs théoriques.

IV.2 Loi Binomiale négative

On répète des expériences de Bernoulli indépendantes et identiques. La variable aléatoire qui donne le nombre d'échecs obtenus avant l'apparition du n-ième succès suit une loi binomiale négative de paramètres n et p.

À programmer 26 (Simulation)**.**
Écrire une fonction `BinNegative(n,p)` qui simule cette variable aléatoire. Utiliser les fonctions précédentes pour afficher la distribution, une estimation de l'espérance et de la variance. Vérifier la cohérence avec les valeurs théoriques.

IV.3 Loi hypergéométrique

Une urne contient b boules blanches et a boules noires. On tire n boules successivement et sans remise. La variable aléatoire X donnant le nombre de boules blanches obtenues suit une loi hypergéométrique de paramètres n, a et b.

À programmer 27 (Simulation)**.**
Écrire une fonction `Hyper(n,a,b)` qui simule cette variable aléatoire. Utiliser les fonctions précédentes pour afficher la distribution, une estimation de l'espérance et de la variance. Vérifier la cohérence avec les valeurs théoriques.

IV.4 Urnes de Pólya

Une urne contient a boules blanches et b boules noires, on pioche uniformément au hasard une boule dans l'urne, puis, on replace la boule piochée ainsi que h autres boules de la même couleur dans l'urne. On repète n fois cette expérience. La loi de Markov-Pólya de paramètres a, b, h et n est alors la loi de la variable aléatoire X qui compte le nombre total de boules blanches piochées au bout de ces n tirages.

À programmer 28 (Simulation)**.**
Écrire une fonction `Polya(n,a,b,h)` qui simule cette variable aléatoire. Utiliser les fonctions précédentes pour afficher la distribution, une estimation de l'espérance et de la variance. Vérifier la cohérence avec les valeurs théoriques.