TP 4

Simulation de variables aléatoires discrètes

Compétences attendues.

- ✓ Savoir simuler une loi discrète usuelle à l'aide des fonctions de la librairie `numpy.random`, ou uniquement à partir de la fonction `rd.random()`.
- ✓ Vérifier graphiquement la pertinence d'une simulation d'une loi.

Liste des commandes `Python` exigibles aux concours.

- Dans la librairie `numpy.random` : `rd.random`, `rd.binomial`, `rd.randint`, `rd.geometric`, `rd.poisson`.
- Dans la librairie `matplotlib.pyplot` : `plt.bar`, `plt.hist`, `plt.show`.

Objectifs.

- On souhaite *simuler* une loi donnée, c'est-à-dire créer une fonction `Python` qui, à chaque appel, renvoie une réalisation d'une variable aléatoire X suivant cette loi.

  Par exemple, simuler la loi $\mathcal{U}(\llbracket 1,6 \rrbracket)$, c'est créer une fonction `Python` qui renvoie les valeurs 1,2,3,4,5,6, chacune d'entre elles apparaissant avec une fréquence égale à 1/6.
- On souhaite vérifier graphiquement la pertinence des simulations obtenues.

Anthony Mansuy
Professeur de Mathématiques en deuxième année de CPGE filière ECG au Lycée Clemenceau (Reims)
Page personnelle : `http://anthony-mansuy.fr`
E-mail : `mansuy.anthony@hotmail.fr`

1 Simulation des lois discrètes usuelles

1.1 La bibliothèque `numpy.random`

La bibliothèque `numpy.random` est dédiée aux simulations de variables aléatoires.

Définition.

On importe la bibliothèque `numpy.random` en écrivant l'une ou l'autre des instructions suivantes (on privilégiera la deuxième) :

`from numpy.random import *` ou `import numpy.random as rd`

Définition.

- `rd.randint(n)` simule la loi uniforme sur $[\![0, n-1]\!]$ avec $n \in \mathbb{N}^*$.
- `rd.randint(a,b)` simule la loi uniforme sur $[\![a, b-1]\!]$ avec $a < b$.
- `rd.binomial(n,p)` simule la loi binomiale $\mathcal{B}(n,p)$.
- `rd.geometric(p)` simule la loi géométrique $\mathcal{G}(p)$.
- `rd.poisson(lambda)` simule la loi de Poisson $\mathcal{P}(\lambda)$.

Remarques.

- On peut simuler la loi de Bernoulli de paramètre p en écrivant l'instruction `rd.binomial(1,p)`.
- On peut obtenir r simulations d'une loi usuelle sous la forme d'un vecteur ou $r \times s$ simulations sous la forme d'une matrice de $\mathscr{M}_{r,s}(\mathbb{R})$. Par exemple :
  - `rd.geometric(p,r)` renvoie un vecteur contenant r simulations de la loi géométrique $\mathcal{G}(p)$;
  - `rd.poisson(lambda,[r,s])` renvoie $r \times s$ simulations de la loi de Poisson de paramètre λ.

Exemple. Après avoir importé le sous-module `numpy.random`, exécuter plusieurs fois les instructions suivantes :

```
>>> rd.randint(1,7,10)
>>> rd.poisson(5,[3,3])
```

Cette section pourrait s'arrêter là, étant donné qu'on dispose de commandes pour simuler toutes les lois usuelles discrètes. Notre but dans la suite est de proposer des simulations à l'aide uniquement de la fonction `random`.

1.2 La fonction `random`

Définition.

- `rd.random()` simule la loi uniforme continue $\mathcal{U}([0,1])$. Elle renvoie donc un nombre aléatoire de $[0,1]$.
- `rd.random(r)` simule r réalisations de la loi $\mathcal{U}([0,1])$ sous la forme d'un vecteur de taille r.
- `rd.random([r,s])` simule $r \times s$ réalisations de la loi $\mathcal{U}([0,1])$ sous la forme d'une matrice de $\mathscr{M}_{r,s}(\mathbb{R})$.

Propriété 1

(1) Pour tout $(a,b) \in [0,1]^2$ tel que $a < b$, la probabilité que le réel renvoyé par `rd.random()` soit compris (strictement ou non) entre a et b vaut $b-a$.

(2) Pour tout $p \in [0,1]$, la probabilité que le réel renvoyé par `rd.random()` soit inférieur (strictement ou non) à p vaut p.

Preuve. *Pour les 5/2.* Commençons par rappeler que si $U \hookrightarrow \mathcal{U}([0,1])$, alors sa fonction de répartition F_U est :

$$F_U : x \in \mathbb{R} \mapsto \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \in [0,1] \\ 1 & \text{si } x > 1 \end{cases}.$$

La probabilité de l'évènement (1) est (puisque U est à densité) :

$$P(a \leq U \leq b) = P(a < U \leq b) = P(a \leq U < b) = P(a < U < b)$$

et elle vaut :

$$P(a < U \leq b) = P(U \leq b) - P(U \leq a) = F_U(b) - F_U(a) \underset{a,b \in [0,1]}{=} b - a.$$

Si $p \in [0,1]$, alors la probabilité de l'évènement (2) est $P(U < p) \underset{U \text{ à densité}}{=} P(U \leq p) \underset{p \in [0,1]}{=} p$. □

Méthode.

Pour tout $p \in [0,1]$, l'instruction `rd.random()<=p` renvoie un booléen qui prend la valeur `True` avec la probabilité p et la valeur `False` avec la probabilité $1-p$.
Cette instruction permet de simuler un événement de probabilité p.

Remarque. Pour simuler un événement de probabilité p, on pourra indifféremment utiliser l'une ou l'autre des instructions suivantes : `rd.random()<p` ou `rd.random()<=p`.

Les commandes suivantes pourront être utiles dans la suite.

Définition.

Si `u` est un vecteur dont les composantes sont des booléens, alors la commande :

- `np.sum(u)` renvoie le nombre de booléens qui ont pris la valeur `True` ;
- `np.mean(u)` renvoie la proportion de booléens qui ont pris la valeur `True`.

Exemple. Après avoir importer la bibliothèque `numpy`, taper les instructions suivantes :

```
>>> u = rd.random(100) ; u
>>> v = (u <= 0.5) ; v
>>> np.mean(v)
```

1.3 Loi uniforme

Exercice 1 (★)

1. Soit $n \in \mathbb{N}^*$. Justifier que la commande `np.floor(n*rd.random())` simule une loi uniforme $\mathcal{U}([\![0, n-1]\!])$.
2. Soient a et b deux entiers tels que $a < b$.
   (a) Écrire une fonction `uniforme(a,b)` simulant la loi $\mathcal{U}([\![a, b-1]\!])$.
   (b) Compléter la fonction suivante afin qu'elle renvoie un vecteur contenant N réalisations indépendantes de la loi $\mathcal{U}([\![a, b-1]\!])$.

```
1 def Uniforme(a,b,N):
2     v = np.zeros(N) ;
3     for k in range(N):
4         v[k] = ...
5     return v
```

Exercice 2 (★★)

On lance indéfiniment un dé équilibré à 6 faces numérotées de 1 à 6.
Écrire le script d'un programme qui permet de simuler ce lancer de dé et qui renvoie le rang du lancer où l'on obtient pour la première fois un numéro déjà obtenu.

1.4 Loi de Bernoulli

Exercice 3 (★)

1. Pour simuler une loi de Bernoulli, on procèdera comme évoqué plus haut : on tire au hasard un nombre dans l'intervalle $[0,1]$ avec la fonction `random()`. On a alors deux cas :
   - si `rd.random()<=p`, ce qui arrive avec une probabilité de p, on renvoie la valeur 1 ;
   - si `rd.random()>p`, ce qui arrive avec une probabilité de $1-p$, on renvoie 0.

   Compléter la fonction suivante afin qu'elle simule une loi de Bernoulli de paramètre p.

```
def bernoulli(p):
    u = 0
    if .........
        u = ...
    return u
```

2. Écrire une fonction `Bernoulli(p,N)` renvoyant un vecteur contenant N réalisations indépendantes de la loi $\mathcal{B}(p)$.

Exercice 4 (★★)

On dispose d'une urne contenant sept boules dont trois sont blanches et quatre sont noires. On effectue dans cette urne deux tirages successifs d'une boule sans remise. On note X (respectivement Y) la variable aléatoire prenant la valeur 1 si la première (respectivement la deuxième) boule tirée est blanche et 0 sinon.

Compléter le programme suivant afin qu'il simule l'expérience et affiche la valeur du couple (X,Y).

```
if rd.random()<3/7 :
    X = ...
else :
    X = ...
if rd.random()< ... :
    Y = ...
else :
    Y = ...
print (X,Y)
```

1.5 Loi binomiale

Propriété 2 (Somme de Bernoulli indépendantes)

Soit $n \in \mathbb{N}^*$ et $p \in]0,1[$.
Si $X_1, \dots, X_n$ sont n variables aléatoires telles que :

- elles sont mutuellement indépendantes,
- elles suivent toutes la même loi $\mathcal{B}(p)$,

alors $\displaystyle\sum_{i=1}^{n} X_i \hookrightarrow \mathcal{B}(n,p)$.

Exercice 5 (★)

1. Écrire une fonction `binomiale(n,p)` simulant la loi $\mathcal{B}(n,p)$.
2. Écrire une fonction `Binomiale(n,p,N)` donnant un échantillon de taille N de la loi $\mathcal{B}(n,p)$
3. Simuler un échantillon de taille $N = 10000$ de la loi $\mathcal{B}(10, 0.2)$, puis vérifier que la valeur moyenne de cet échantillon est cohérente avec ce qu'on attend.

Exercice 6 (★)

Soit n un entier naturel non nul. On dispose de n urnes notées $U_1, \dots, U_n$. Pour tout $k \in [\![1, n]\!]$, l'urne U_k contient k boules blanches et $n-k$ boules noires. On lance un dé équilibré à n faces numérotées de 1 à n. On note k le numéro obtenu et on effectue n tirages successifs d'une boule avec remise dans l'urne U_k. On note X la variable aléatoire égale au nombre de boules blanches obtenues.

Écrire une fonction d'en-tête `def simul(n)` qui simule X pour une valeur de n entrée par l'utilisateur.

1.6 Loi géométrique

Propriété 3 (de la loi géométrique)

Soit $p \in]0, 1[$.

Si X est le rang du premier succès lors de la répétition d'épreuves de Bernoulli indépendantes, toutes de probabilité de succès p, alors $X \hookrightarrow \mathcal{G}(p)$.

Exercice 7 (★)

1. Écrire une fonction `geom(p)` simulant la loi $\mathcal{G}(p)$.
2. Écrire une fonction `Geom(p,N)` afin de simuler un échantillon de taille N de la loi $\mathcal{G}(p)$.
3. Simuler un échantillon de taille $N = 10000$ de la loi $\mathcal{G}(0.2)$, puis vérifier que la valeur moyenne de cet échantillon est cohérente avec ce qu'on attend.

---

2 Représentations graphiques

2.1 Comparaison diagramme en bâtons des fréquences / probabilités théoriques

Soit X une variable aléatoire discrète. Supposons qu'on dispose d'une fonction `Loi` permettant de simuler la loi de X. Pour juger de la pertinence des simulations, on peut utiliser des représentations graphiques. Pour cela, on procèdera comme suit :

- on crée un *échantillon* de taille N, c'est-à-dire un vecteur ligne `x` contenant N réalisations de la fonction `Loi`.
- on compare graphiquement les fréquences empiriques obtenues (c'est-à-dire grâce à l'échantillon) avec les probabilités théoriques pour vérifier la pertinence de la simulation.

Dans le cas de simulation de variables aléatoires discrètes, on va comparer plus particulièrement :

- le *diagramme en bâtons des fréquences* de notre échantillon de taille N ;
- le *diagramme en bâtons des probabilités théoriques* $P(X = k)$ pour $k \in X(\Omega)$.

Si notre simulation est bonne, on doit constater que :

Théorème 4 (Théorème d'Or de Bernoulli)

Pour N « suffisamment grand », le diagramme en bâtons des fréquences de l'échantillon doit être proche de celui des probabilités théoriques.

2.2 Commandes Python

On suppose avoir importé la bibliothèque `matplotlib.pyplot` à l'aide de l'instruction :

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

Diagramme en bâtons des probabilités théoriques

Pour dessiner le diagramme en bâtons des probabilités théoriques, on définit `x` le vecteur contenant (une partie de) $X(\Omega)$ et `y` le vecteur contenant les probabilités théoriques correspondantes. On utilise alors la commande suivante :

Définition.

Si `x` et `y` sont des vecteurs, `plt.bar(x,y)` trace le diagramme en bâtons d'abscisse `x` et d'ordonnée `y`.

Diagramme en bâtons des fréquences de l'échantillon

Pour dessiner le diagramme en bâtons des fréquences de l'échantillon `x`, il nous faut trier notre série par modalités/fréquences, puis tracer le diagramme en bâtons associé. On utilisera pour cela (de manière un peu détournée) la commande suivante.

Définition.

Si `x` est un vecteur contenant une série statistique et `c` un vecteur contenant les classes choisies, la commande `plt.hist(x,c)` dessine l'histogramme associé à la série statistique `x` triée selon les classes définies par `c`.

Méthode.

Pour tracer le diagramme des fréquences d'un échantillon $\boldsymbol{x}$ (qu'on suppose à valeurs entières), on procède ainsi :

(i) on décide des modalités $m_1 < m_2 < \cdots < m_k$ qu'on souhaite représenter (dans le cas où elles seraient en nombre infini) ;

(ii) on définit les classes $c = (m_1 - 0,5 < m_1 + 0,5 < m_2 - 0,5 < m_2 + 0,5 < \cdots < m_k - 0,5 < m_k + 0,5)$;

(iii) on dessine l'histogramme (le « diagramme en bâtons des fréquences ») à l'aide de la commande :

```
plt.hist(x,c,density='True',edgecolor='k',color='...', label="...")
```

où l'on a ajouté les options de tracé suivantes (non exigibles) :

- normalisation des rectangles (la surface totale vaut 1) : `density='True'`
- contours des rectangles en noir : `edgecolor='k'`
- couleur des rectangles : `color='...'` (mettre le nom de la couleur en anglais)
- légende associée à chaque histogramme : `label="..."` (mettre la légende choisie)

Remarque. Pour réaliser plusieurs graphiques dans une même fenêtre et ainsi pouvoir mieux les comparer, on peut utiliser l'instruction `plt.subplot(1,m,k)` avant chaque instruction de tracé de graphique, qui découpe la fenêtre graphique en 1 ligne et m colonnes, k indiquant le numéro de la colonne souhaitée pour chaque graphique.

2.3 Exercices

Exercice 8 (★)

On se donne le code suivant :

```
# Echantillon
x = Uniforme(1,7,100000)

#DeB des fréquences
c = np.arange(0.5,7)
plt.subplot(1,2,1)
plt.hist(x,c,density='True',edgecolor='k',
color='blue',label="DeB des fréq")

#DeB des probas théoriques
u = np.arange(1,7)
v = (1/6)*np.ones(6)
plt.subplot(1,2,2)
plt.bar(u,v,label="DeB des prob
theo")

plt.show()
```

Exécuter ce code et interpréter le graphique ainsi obtenu. A quoi correspondent les vecteurs `x`, `c` et `u` ?

Exercice 9 (★★)

1. On considère les instructions suivantes :

   ```
   >>> c = np.ones(n+1)
   >>> c[1:(n+1)] = np.cumprod(np.arange(n,0,-1)/np.arange(1,n+1))
   ```

   Que contient le vecteur `c` à la suite de ces instructions ? Expliquer.

   On pourra commencer par exécuter ces instructions pour `n = 3`, `n = 4`.

2. À l'aide d'un diagramme en bâtons, tester la simulation de la loi binomiale $\mathcal{B}(10, 0.2)$ obtenue à l'aide de la fonction `Binomiale`.

Exercice 10 (★★)

1. Donner les instructions `Python` pour définir un vecteur de taille 20 contenant les premières probabilités théoriques d'une loi géométrique de paramètre p.

2. À l'aide d'un diagramme en bâtons, tester la simulation de la loi géométrique $\mathcal{G}(0.2)$ obtenue à l'aide de la fonction `Geom`.

Exercice 11 (★★)

À un guichet, des clients peuvent venir expédier ou retirer un colis. Au cours d'une journée, le nombre de clients N qui s'y présentent suit la loi de Poisson de paramètre $\lambda = 10$. Chaque client a une probabilité $p = 0.2$ de venir pour expédier un colis et $1 - p = 0.8$ pour en retirer un.
On note C le nombre de colis expédiés dans la journée.

1. Quelle est la loi de C sachant $(N = k)$? Justifier.

2. Écrire un programme qui simule la variable aléatoire C.

3. En déduire un programme renvoyant un vecteur `C` contenant 10000 réalisations de la variable aléatoire C.

4. On souhaite représenter le diagramme en bâtons des fréquences de l'échantillon.

   (a) Exécuter la commande `np.mean(C<=10)`. Définir alors un vecteur classe `c` adapté à l'échantillon.

   (b) Représenter le diagramme en bâtons des fréquences de l'échantillon.

5. (a) Donner les instructions `Python` pour définir un vecteur de taille 10 contenant les premières probabilités théoriques d'une loi de Poisson de paramètre 2.

   (b) Tracer le diagramme en bâtons associé. Que peut-on conjecturer ?

---

3 Méthode d'inversion discrète

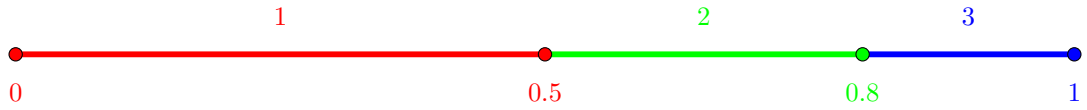
3.1 Principe

La méthode d'inversion est une méthode générale pour simuler la loi d'une variable discrète X à partir d'une variable aléatoire $U \hookrightarrow \mathcal{U}([0,1])$. Elle consiste à « inverser » la fonction de répartition de X.

Exemple. On souhaite simuler une variable aléatoire X ayant pour support $X(\Omega) = \{1,2,3\}$ et pour probabilités ponctuelles :

$$P(X=1) = 0.5, \quad P(X=2) = 0.3 \quad \text{et} \quad P(X=3) = 0.2.$$

On va pour cela découper l'intervalle $[0,1]$ en trois intervalles de longueurs 0.5, 0.3 et 0.2.



Pour simuler la variable X, on tire alors au hasard un nombre t de $[0,1]$ avec la fonction `rd.random()` et on retourne :

- $x = 1$ si $t \in [0, 0.5]$, ce qui se produit avec la probabilité 0.5 ;
- $x = 2$ si $t \in]0.5, 0.8]$, ce qui se produit avec la probabilité 0.3 ;
- $x = 3$ si $t \in]0.8, 1]$, ce qui se produit avec la probabilité 0.2.

On peut ainsi simuler la loi de la variable X à l'aide de la fonction suivante :

```
def simulation():
    t = rd.random()
    x = 3
    if t <= 0.5 :
        x = 1
    elif t <= 0.8 :
        x = 2
    return x
```

Cas général. Soit à présent X une variable discrète qui prend les valeurs $x_1 < x_2 < \cdots < x_n < \ldots$ Rappelons que sa fonction de répartition est en escalier, donnée par :

$$\forall x \in \mathbb{R}, \quad F(x) = P(X \leq x) = \begin{cases} 0 & \text{si } x < x_1, \\ \displaystyle\sum_{i=1}^{n} P(X = x_i) & \text{si } x_n \leq x < x_{n+1}. \end{cases}$$

La méthode d'inversion présentée sur les exemples précédents se généralise de la manière suivante.

Méthode.

Pour simuler la variable discrète X, on procèdera comme suit :

(i) on choisit le paramètre t aléatoirement dans $[0,1]$ à l'aide de la fonction `rd.random()` ;

(ii) on détermine l'intervalle $I_k =]F(x_{k-1}), F(x_k)]$ tel que $t \in I_k$ (si $k = 1$, $I_1 = [0, F(x_1)]$) ;

(iii) on retourne x_k.

Remarque. *Pour les 5/2.* Le programme est sensé retourner x_k avec une probabilité $P(X = x_k)$ pour tout k. Vérifions le pour $k \geq 2$: x_k est retourné si et seulement si $t \in]F(x_{k-1}), F(x_k)]$, où t est une réalisation d'une variable $U \hookrightarrow \mathcal{U}([0,1])$. Or ceci se réalise avec une probabilité :

$$P(F(x_{k-1}) < U \leq F(x_k)) = P(U \leq F(x_k)) - P(U \leq F(x_{k-1})) = F_U(F(x_k)) - F_U(F(x_{k-1}).$$

Comme de plus $F_U(x) = x$ pour tout $x \in [0,1]$, et que $F(x_k), F(x_{k-1}) \in [0,1]$, on obtient :

$$P(F(x_{k-1}) < U \leq F(x_k)) = F(x_k) - F(x_{k-1}) = \sum_{i=1}^{k} P(X = x_i) - \sum_{i=1}^{k-1} P(X = x_i) = P(X = x_k).$$

Le même argument vaut aussi pour $k = 1$. Ainsi la probabilité qu'un tel programme retourne x_k est bien $P(X = x_k)$.

3.2 Exercices

Exercice 12 (★)

On souhaite simuler une loi uniforme $\mathcal{U}(\llbracket 1,6 \rrbracket)$.

1. Déterminer le découpage correspondant de l'intervalle $[0,1]$ pour cette loi.
2. Écrire une fonction `unif()` simulant la loi $\mathcal{U}(\llbracket 1,6 \rrbracket)$ à l'aide de la méthode d'inversion.

Exercice 13 (★★)

On souhaite simuler une loi de Poisson de paramètre $\lambda > 0$ par la méthode d'inversion.

1. On tire au hasard un nombre $t \in [0,1]$. Dans quel intervalle I_k le nombre t doit se trouver pour qu'on retourne k ?
2. On se donne le code suivant :

```
def poisson(lbd):
    t = rd.random()
    k = 0
    u = np.exp(-lbd)
    s = u ;
    while s<t :
        k = k+1
        u = u*(lbd/k)
        s = s+u
    return k
```

   Expliquer le fonctionnement de la fonction `poisson`.

3. Écrire une fonction `Poisson(lbd,N)` renvoyant un vecteur contenant N réalisations indépendantes de la loi $\mathcal{P}(\lambda)$.
4. À l'aide d'un diagramme en bâtons, juger de la pertinence de cette simulation pour $\lambda = 5$. On admet pour cela que les cas où la variable prend une valeur supérieure à 10 sont négligeables.

Exercice 14 (★★★)

Écrire une fonction `geom3` qui prend en paramètre un nombre $p \in]0,1[$, puis à l'aide de la méthode d'inversion, simule une loi géométrique de paramètre p.
