Math ECG 2. 2023-2024

Mathématiques Appliquées - F. Gaunard
http://frederic.gaunard.com
ENC Bessières, Paris 17e.

Chapitre 1. Simulation de variables aléatoires finies et discrètes. Diagrammes à bâtons.

1 Avant-propos

Ce premier chapitre a plusieurs motivations. Déjà, il permet quelques révisions sur les variables aléatoires finies et discrètes et notamment sur les lois usuelles, qu'on va rencontrer tout le temps.
Ensuite, il donne le ton dès le premier chapitre; c'est un cours de *mathématiques appliquées*; en probabilités, la simulation sous Python est une composante incontournable du programme et sera très présente tout au long de l'année.

☞ On va donc considérer que, dans tous les programmes et exercices venir, auront déjà été importées sous leurs alias habituels les bibliothèques et libraires nécessaires à chaque situation:

```
import numpy as np
import numpy.random as rd
import matplotlib.pyplot as plt
```

2 Simulation d'une variable aléatoire avec `rd.random( )`

Commençons par quelques rappels.

Résultat du cours de première année

Soit $(\Omega, \mathcal{P}(\Omega))$ un espace probabilisable. On appelle **variable aléatoire réelle** (ou v.a.r) sur $(\Omega, \mathcal{P}(\Omega))$ toute application

$$X : \Omega \to \mathbb{R}$$

L'ensemble $X(\Omega)$ s'appelle **l'univers image de** X. C'est l'ensemble des valeurs prises par X.

Une variable aléatoire est caractérisée par sa *loi*.

À retenir!

☞ **En pratique**, dans le cas d'un univers Ω discret (fini ou infini), la loi d'une variable aléatoire sera la donnée de tous les couples

$$(x, P(X = x)), \qquad x \in X(\Omega).$$

☞ La loi sera la principale information dont on disposera sur une variable aléatoire, l'univers Ω étant le plus souvent implicite et non précisé.

Avant de simuler une variable aléatoire, il est important d'avoir compris comment simuler un évènement de probabilité p.

À retenir!

La commande `rd.random( )` renvoie un nombre réel aléatoire entre 0 et 1 *suivant la loi uniforme (continue)* $\mathcal{U}([0;1])$[a]. Plus précisément, si $p \in [0;1]$, la probabilité que le nombre renvoyé soit inférieure ou égale à p (ou dans un intervalle de longueur p) vaut exactement p.

On **identifie** et décide de représenter donc un évènement de probabilité p par le fait que le nombre aléatoire ainsi renvoyé soit inférieur ou égal à p (ou dans un intervalle de longueur p).

[a]Cette loi usuelle sera introduite et étudiée dans la chapitre sur les variables aléatoires à densité

Exercice 1. Le score d'un joueur lors d'un lancer de fléchette est modélisé par une variable aléatoire X telle que $X(\Omega) = \{0, 2, 5, 10\}$ et

$$P(X=0) = \frac{1}{5}, \qquad P(X=2) = \frac{1}{2}, \qquad P(X=5) = \frac{1}{5}, \quad \text{et} \quad P(X=10) = \frac{1}{10}.$$

(1) Calculer l'espérance et la variance de X.
(2) Découper l'intervalle $[0;1]$ en quatre intervalles où pourrait *tomber* un nombre aléatoire généré avec `rd.random()` dans le but de simuler X. Écrire alors une fonction `simul_X()` qui renvoie une simulation de la variable X.
(3) On ajoute les commandes suivantes. Les exécuter et commenter l'affichage.

```
def sample_X(n):
    S=[]
    for k in range(n):
        S.append(simul_X())
    return S

print(np.mean(sample_X(1000)))
```

À retenir!

Un $n-$échantillon d'une variable aléatoire X est un $n-$uplet dont chaque composante est une variable aléatoire X_i de même loi que X et qui sont toutes mutuellement indépendantes.

La **moyenne empirique** des réalisations d'un $n-$échantillon (que l'on obtient grâce à la commande `np.mean( )`) fournit alors une *estimation* (valeur approchée) de l'espérance de X (quand celle-ci existe). Cette observation prendra tout son sens avec le chapitre *Estimation* de fin d'année.

Exercice 2. On désigne par n un entier naturel supérieur ou égal à 2. On lance n fois une pièce équilibrée (c'est-à-dire donnant *Pile* avec la probabilité 1/2, les lancers étant supposés indépendants.
On note Z la variable aléatoire qui vaut 0 si l'on n'obtient aucun *Pile* pendant ces n lancers et qui, dans le cas contraire, prend pour valeur le rang du premier pile.

(1) Recopier et compléter la fonction suivante permettant de simuler la variable aléatoire Z.

```
def simul_Z(n) :
    for i in range(1, n+1):
        if ........ :
            return ...
    return ...
```

(2) Réécrire cette fonction en utilisant une boucle `while`.
(3) Déterminer $P(X = k)$ pour $k \in [\![0; n]\!]$. (On pourra différencier deux cas selon que $1 \leq k \leq n$ et $k = 0$.)

3 (Simulation de) Lois usuelles finies et discrètes

3.1 Loi uniforme $\mathcal{U}([\![m; n]\!])$

Résultat du cours de première année

On dit que X suit une loi uniforme sur $[\![m; n]\!]$), ce qu'on note $X \hookrightarrow \mathcal{U}([\![m; n]\!])$ si $X(\Omega) = [\![m; n]\!]$ et

$$\forall k \in [\![m; n]\!], \qquad P(X = k) = \frac{1}{n - m + 1}$$

De plus,

$$E(X) = \frac{n + m}{2}, \qquad V(X) = \frac{(n - m + 1)^2 - 1}{12}.$$

☞ On peut *reconnaître* une loi uniforme lors d'un tirage dont les issues sont équiprobables (lancer d'un dé non truqué, tirage de boules indiscernables ...).

☞ La commande `rd.randint(m, n+1)` renvoie une simulation d'une variable $X \hookrightarrow \mathcal{U}([\![m; n]\!])$.

Exercice 3. On dispose de n urnes, numérotées de 1 à n. Pour chaque $k \in [\![1, n]\!]$, l'urne k est composée de boules numérotées de 1 à k. On choisit une urne au hasard puis on tire une boule dans cette urne et on note X_n la variable aléatoire qui prend la valeur du numéro de la boule piochée.
Écrire une fonction `simul_X(n)` qui simule X_n.

3.2 Loi de Bernoulli $\mathcal{B}(p)$

Résultat du cours de première année

On dit que X suit une loi de Bernoulli de paramètre p, ce qu'on note $X \hookrightarrow \mathcal{B}(p)$ si $X(\Omega) = \{0; 1\}$ et

$$P(X = 1) = p, \qquad P(X = 0) = 1 - p.$$

De plus,

$$E(X) = p, \qquad V(X) = p(1 - p).$$

☞ La loi de Bernoulli correspond à une situation à deux alternatives, l'une qu'on appelle *succès* (et lors de laquelle la variable prend la valeur 1) et l'autre qu'on appelle *échec*.

Exercice 4. Écrire, à l'aide de la commande `np.rand( )` une fonction `Bernoulli(p)` permettant de simuler une variable $X \hookrightarrow \mathcal{B}(p)$.

3.3 Loi binomiale $\mathcal{B}(n,p)$

Résultat du cours de première année

On dit que X suit une loi binomiale de paramètres n et p, ce qu'on note $X \hookrightarrow \mathcal{B}(n,p)$ si $X(\Omega) = [\![0,n]\!]$ et

$$\forall k \in [\![0,n]\!], \qquad P(X=k) = \binom{n}{k} p^k (1-p)^{n-k}$$

De plus,

$$E(X) = np, \qquad V(X) = np(1-p).$$

☞ Si $n=1$, la loi binomiale $\mathcal{B}(1,p)$ n'est rien d'autre que la loi de Bernoulli $\mathcal{B}(p)$.

☞ On *reconnait* une loi binomiale de paramètre n et p lorsque l'on compte le nombre de succès (dont la probabilité de chaque est p) lors de n répétitions **indépendantes** d'épreuves de Bernoulli.

☞ La commande `rd.binomial(n,p)` renvoie une simulation d'une variable $X \hookrightarrow B(n,p)$.

☞ On peut alors très rapidement simuler une variable de Bernoulli de paramètre p avec la commande `rd.binomial(1,p)`.

Exercice 5. Écrire, à l'aide de la commande `np.rand( )` et d'une boucle `for`, une fonction `Binomiale(n,p)` permettant de simuler une variable $X \hookrightarrow \mathcal{B}(n,p)$.

3.4 Loi géométrique $\mathcal{G}(p)$

Résultat du cours de première année

On dit que X suit une loi géométrique de paramètre p, ce qu'on note $X \hookrightarrow \mathcal{G}(p)$ si $X(\Omega) = \mathbb{N}^*$ et

$$\forall k \in \mathbb{N}^*, \qquad P(X=k) = (1-p)^{k-1} p$$

De plus, X admet espérance et variance et

$$E(X) = \frac{1}{p}, \qquad V(X) = \frac{1-p}{p^2}.$$

☞ Une loi géométrique correspond au *temps d'attente* du premier succès (de probabilité p) lors de répétitions (infinies) **indépendantes** d'une même épreuve de Bernoulli.

☞ La commande `rd.geometric(p)` renvoie une simulation d'une variable $X \hookrightarrow G(p)$.

Exercice 6. Écrire, à l'aide de la commande `rd.rand( )` et d'une boucle `while` une fonction `Geometrique(p)` permettant de simuler une variable $X \hookrightarrow \mathcal{G}(p)$.

3.5 Loi de Poisson $\mathcal{P}(\lambda)$

Résultat du cours de première année

On dit que X suit une loi de Poisson de paramètre $\lambda > 0$, ce qu'on note $X \hookrightarrow \mathcal{P}(\lambda)$ si $X(\Omega) = \mathbb{N}$ et

$$\forall k \in \mathbb{N}, \qquad P(X = k) = e^{-\lambda}\frac{\lambda^k}{k!}$$

De plus, X admet espérance et variance et

$$E(X) = \lambda, \qquad V(X) = \lambda^2.$$

☞ La commande `rd.poisson(m)` renvoie une simulation d'une variable $X \hookrightarrow P(m)$.

Exercice 7. Chaque jour, le nombre de personnes subissant un test de dépistage dans une certaine pharmacie est modélisé par une variable X suivant une loi de Poisson de paramètre $m = 5$. Chaque test a une certaine probabilité $p = 1/10$ d'être positif et les tests sont indépendants les uns des autres. On note N le nombre de tests positifs un jour donné.

(1) Écrire une fonction permettant simuler la variable N.
(2) Créer un échantillon de taille 1000 de cette variable aléatoire et le comparer, avec un histogramme, à un échantillon de même taille d'une loi de Poisson de paramètre $mp = 0,5$. Conjecturer.
(3) (*) En commençant par expliciter $P_{X=n}(N = k)$ (en faisant attention sur le couplage des indices n et k), démontrer le résultat précédemment conjecturé.

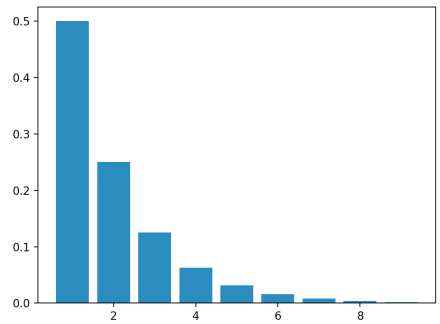
4 Diagrammes à bâtons

Rappel de commande en Python

La commande `bar(x, y)` du package `matplotlib.pyplot` permet de représenter un **diagramme à bâtons**.
Plus précisément, en notant les arguments `x` $= [x_1, ..., x_n]$ et `y` $= [y_1, ..., y_n]$ (listes de mêmes longueurs) elle permet de positionner un bâton de hauteur y_i en face de la valeur x_i.

```
p=1/2
K=[k for k in range(1, 10)]
p_th=[p*(1-p)**(k-1) for k in K]
plt.bar(K, p_th)
plt.show( )
```

Affichage Python



Exercice 8. Représenter le diagramme à bâtons dont les hauteurs des bâtons sont les valeurs **théoriques** de la loi uniforme $\mathcal{U}(\llbracket 1; N \rrbracket)$.

Remarque

☞ On ne fera pas de confusion avec la commande `hist(U)` (du même package) qui représente l'*histogramme* des valeurs d'une liste `U`.

Si les histogrammes sont très utiles pour représenter graphiquement la *répartition empirique* d'une loi simulée avec un échantillon (de grande taille) lorsque la variable simulée est à densité (car les données sont *continues*), les **diagrammes à bâtons** sont plus adaptés lorsque la loi est finie.

Il est souvent très efficace d'utiliser de tels diagrammes pour conjecturer sur la loi suivie (par exemple si tous les bâtons ont à peu près la même hauteur, il est raisonnable de penser que la loi est uniforme - sur l'ensemble des valeurs prises, ou bien en mettant en parallèle des bâtons d'une autre loi - usuelle simulée ou hauteur théorique).
Les diagrammes à bâtons sont aussi pratiques pour visualiser (ou conjecturer graphiquement d') une *convergence en loi*[1]. Par exemple, les commandes ci-dessous permettent d'afficher les bâtons dont les hauteurs sont les valeurs théoriques de la loi géométrique de paramètre $p = 1/2$ (pour les 10 premiers entiers).

À retenir!

L'idée est alors de créer un $n-$échantillon (avec n grand) et de créer un diagramme à bâtons dont la liste `x` en abscisses est celles des valeurs obtenues par simulation et la liste `y` en ordonnées celle des **fréquences** de chaque valeur dans l'échantillon.

4.1 Premier exemple

Une urne contient des boules numérotées de 1 à n, indiscernables au toucher. On effectue alors n tirages sans remise dans l'urne et on note, pour tout $k \in [\![1, n]\!]$, X_k la variable aléatoire qui prend la valeur de la boule obtenue au $k-$ième tirage.

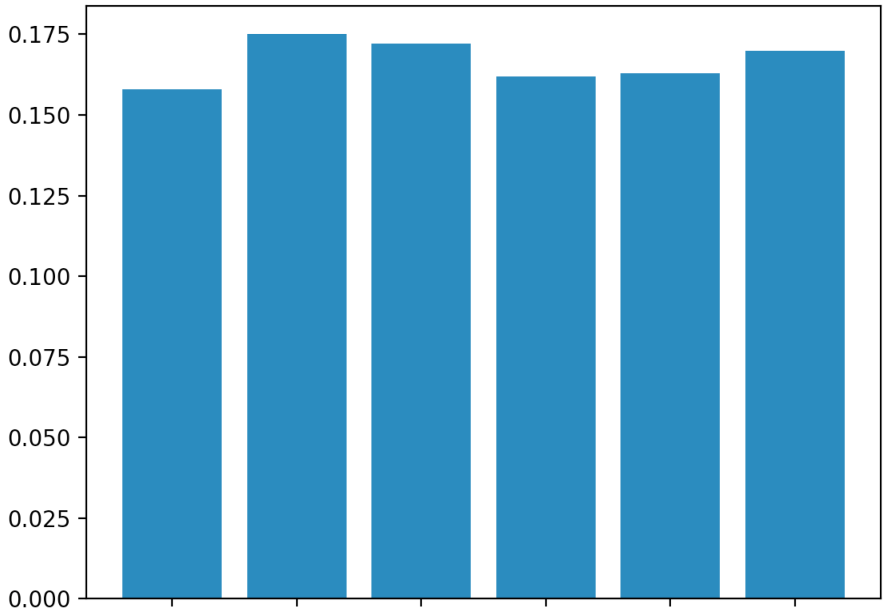
(1) Écrire une fonction Python d'en-tête `def simul_X(k,n):` qui renvoie une simulation de X_k.
(2) On ajoute les commandes suivantes. Expliquer ce qu'elles font. On insistera que la commande de la ligne 6.

```
n=6
k=4
echantillon=[simul_X(k, n) for j in range(1000)]
freq=[0]*6
for j in range(1000):
    freq[echantillon[j]-1]+=1 # ligne 6
freq=[freq[j]/1000 for j in range(n)]
plt.bar([j for j in range(1, n+1)], freq)
plt.show()
```

(3) Leur exécution permet l'affichage suivant. Émettre une conjecture.

[1] Voir le Chapitre *Convergence(s) et approximations.*

Affichage Python



4.2 Deuxième exemple

On lance indéfiniment une pièce donnant *Pile* avec la probabilité p et *Face* avec la probabilité $q = 1 - p$. On suppose que $p \in]0, 1[$ et on admet que les lancers sont mutuellement indépendants.

Pour tout entier naturel k, supérieur ou égal à 2, on dit que le $k^{ième}$ lancer est un changement s'il amène un résultat différent de celui du $(k-1)^{ième}$ lancer.

On note P_k (resp. F_k) l'événement : "on obtient *Pile* (resp. *Face*) au $k^{ième}$ lancer".
Pour tout entier naturel n supérieur ou égal à 2, on note X_n la variable aléatoire égale au nombre de changements survenus durant les n premiers lancers.

(1) Écrire une fonction d'en-tête `def simul_X(n,p):` qui renvoie une simulation de X_n.

(2) Quelles commandes peut-on ajouter pour représenter le diagramme à bâtons des fréquences des valeurs prises par un 1000−échantillon de X_n?

(3) Représenter le diagramme à bâtons susmentionné dans le cas $p = 1/2$ et $n = 3$ puis $n = 4$ puis $n = 5$. Représenter en parallèle des batôns correspondant aux hauteurs théoriques de la loi $\mathcal{B}(n-1, 1/2)$. Émettre une conjecture.

Sélection d'exercices - Travaux dirigés

Exercice 9. On considère la variable aléatoire X telle que $X(\Omega) = \{-1, 0, 1\}$ et

$$P(X = -1) = P(X = 1) = 1/4, \qquad P(X = 0) = 1/2.$$

Écrire une fonction Python permettant de simuler X

Exercice 10. Une urne contient 1 boule bleue, 2 boules blanches et 3 boules rouges. On pioche successivement et avec remise 3 boules dans cette urne. Si on obtient les 3 couleurs, on marque 2 points, si on obtient une seule couleur 1 point et 0 point sinon. On note X la variable aléatoire qui prend la valeur du nombre de points marqués sur une partie.

(1) Déterminer la loi de X. Écrire une fonction Python qui simule X.
(2) Calculer $E(X)$ puis $V(X)$.

Exercice 11. Une urne contient initialement deux boules rouges et une boule bleue indiscernables au toucher. On appelle "tirage" la séquence suivant:

On pioche, au hasard, une boule de l'urne, puis :

- Si la boule piochée est bleue, on la remet dans l'urne.
- Si la boule piochée est rouge, on ne la remet pas dans l'urne mais on remet une boule bleue dans l'urne à sa place.

Pour tout entier naturel n non nul, on note Y_n la variable aléatoire discrète égale au nombre de boules rouges présentes dans l'urne à l'issue du $n-$ième tirage.

(1) Compléter la fonction ci-dessous afin qu'elle simule la variable Y_n.

```
def simul_Y(n):
    r=2 # nombre de boules rouges dans l'urne
    for  k in range(n):
        if ... :
            if  ... :
                ...
    return ....
```

(2) On cherche à *estimer* $\lim_{n\to+\infty} P(Y_n = 0)$. On introduit la variable T_n définie par

$$T_n = \begin{cases} 1, & \text{si } [Y_n = 0] \\ 0, & \text{sinon} \end{cases}$$

(a) Quelle est la loi de la variable T_n?
(b) Écrire une fonction qui simule T_n.
(c) On admet que la *moyenne empirique* d'un échantillon de 1000 réalisations de T_n renvoie une estimation de $P(Y_n = 0)$. Écrire une suite d'instruction permettant de visualiser graphiquement l'évolution de (l'estimation de) $P(Y_n = 0)$ en fonction de n. Que peut-on conjecturer quant à la limite ci-dessus?
(d) Justifier que $(Y_n = 0) \subset (Y_{n+1} = 0)$. En déduire que la conjecture précédente se reformule comme

$$P\left(\bigcup_{n=2}^{+\infty}(Y_n = 0)\right) = 1.$$

Interpréter.

(3) On note Z la variable aléatoire égale au numéro du "tirage" amenant la dernière boule rouge.
(a) Donner $Z(\Omega)$.
(b) Écrire une fonction qui simule la variable Z.

Exercice 12. Soit N un entier supérieur ou égal à 3. On considère une urne qui contient $(N-1)$ boules blanches et une seule boule noire.
On effectue des tirages **sans remise** jusqu'à l'obtention de la boule noire et on note X la variable aléatoire qui prend pour valeur le nombre de tirages nécessaires.

(1) Écrire une fonction `simul_X(N)` qui renvoie une simulation de la variable X.
(2) Recopier et exécuter les instructions ci-contre. Comparer avec le diagramme théorique de l'Exercice 8. Commenter, conjecturer, démontrer.

```
import matplotlib.pyplot as plt

N=5 # on fera varier les valeurs de N
freq_val_X=np.zeros(N)
for k in range(10000):
    i=simul_X(N)
    freq_val_X[i-1]+=1
freq_val_X=freq_val_X/10000
plt.bar(range(1, N+1), freq_val_X)
plt.show()
```

Exercice 13. On donne le code suivant. Le recopier et exécuter `fig_mystere` pour plusieurs valeurs de n. Expliquer, commenter.

```
import numpy as np
import matplotlib.pyplot as plt

def mystere(n,k):
    return np.prod([range(n-k+1, n+1)])/np.prod(range(1, k+1))

def fig_mystere(n):
    Y=[mystere(n,k) for k in range(0, n+1)]
    X=[k for k in range(n+1)]
    plt.bar(X,Y)
    plt.show()
```

Exercice 14. On effectue une succession infinie de lancers indépendants d'une pièce équilibrée, donnant pile avec la probabilité $p = 1/2$ et face avec la probabilité $q = 1 - p = 1/2$.
On s'intéresse aux successions de lancers amenant un même côté.

On dit que la première série est de longueur k (avec $k \in \mathbb{N}^*$) si les k premiers lancers ont amené le même côté de la pièce et le $(k+1)$-ième l'autre côté.
De même, la deuxième série commence au lancer suivant la fin de la première série et se termine (si elle se termine) au lancer précédant un changement de côté. On définit de même les séries suivantes.
Pour tout i de $\mathbb{N}^*$, on note P_i (resp. F_i) l'événement : " le i-ième lancer amène pile (resp. face) ".
Pour tout n de $\mathbb{N}^*$, on note N_n la variable aléatoire égale au nombre de séries obtenus lors des n premiers lancers.

(1) Soit $n \in \mathbb{N}^*$.
  (a) Justifier que $N_n(\Omega) = [\![1; n]\!]$.
  (b) Calculer les probabilités $P(N_n = 1)$ et $P(N_n = n)$.

(2) (a) Déterminer les lois des variables aléatoires N_1 et N_2 et calculer leurs espérances.
  (b) Déterminer la loi de N_3 puis vérifier que $E(N_3) = 2$.

(3) Simulation informatique sous Python.

(a) Écrire une fonction `def lancer(p):` qui simule un lancer d'une pièce en renvoyant 1 si on obtient pile (avec probabilité $p \in]0;1[$) et 0 si on obtient face (avec probabilité $q = 1-p$).

(b) Compléter la fonction suivante qui, étant donnés un entier n de $\mathbb{N}^*$, simule n lancers de la pièce et renvoie la valeur de N_n obtenue. *On rappelle que dans cet exercice p vaut $1/2$.*

```
def simul_N(n) :
    # Simulation des n lancers stockés dans la liste L
    L = [ ]
    for k in range(..........)
        L.append(.......)
    end
  # Calcul de la valeur de Nn stockée dans la variable N
    N = .........
    for i in range(........, ..........)
        if  ................ :
            N = .........
    return N
```

(4) (*) On pose, pour tout n de $\mathbb{N}^*$ et pour tout s de $[0;1]$

$$G_n(s) = \sum_{k=1}^{n} P(N_n = k)s^k.$$

La fonction G_n s'appelle **fonction génératrice** de la variable N_n.

(a) Calculer, pour tout n de $\mathbb{N}^*$, $G_n(0)$ et $G_n(1)$.

(b) Montrer que $G_n'(1) = E(N_n)$.

(c) Soit $n \in \mathbb{N}^*$. En utilisant un système complet d'évènements associé à la variable N_n, montrer que :

$$\forall k \in [\![1; n+1]\!], \quad P(N_{n+1} = k) = \frac{1}{2}P(N_n = k) + \frac{1}{2}P(N_n = k-1).$$

(d) En déduire : $\forall n \in \mathbb{N}^*,\ \forall s \in [0;1]$,

$$G_{n+1}(s) = \left(\frac{1+s}{2}\right) G_n(s).$$

(e) En déduire, pour tout n de $\mathbb{N}^*$ et pour tout s de $[0;1]$,

$$G_n(s) = s \cdot \left(\frac{1+s}{2}\right)^{n-1}.$$

(f) En déduire, à l'aide de la question 4b, que le nombre moyen de séries lors des n premiers lancers est $(n+1)/2$.

Exercice 15. On effectue une succession infinie de lancers indépendants d'une pièce donnant `Pile` avec probabilité p. On note $q = 1-p$.
On dit que la première série est de longueur $n \geq 1$ si les n premiers lancers ont amené le même côté de la pièce et le $(n+1)$-ième l'autre côté. De même la deuxième série commence au lancer suivant la fin de la première série et se termine au lancer précédant un changement de côté.

Par exemple si les lancers donnent les résultats $FFPPPPPPFFFPP\ldots$ alors la première série est de longueur 2 et la deuxième est de longueur 6.
Soient X_1 et X_2 les variables aléatoires égales aux longueurs de la première et deuxième série.

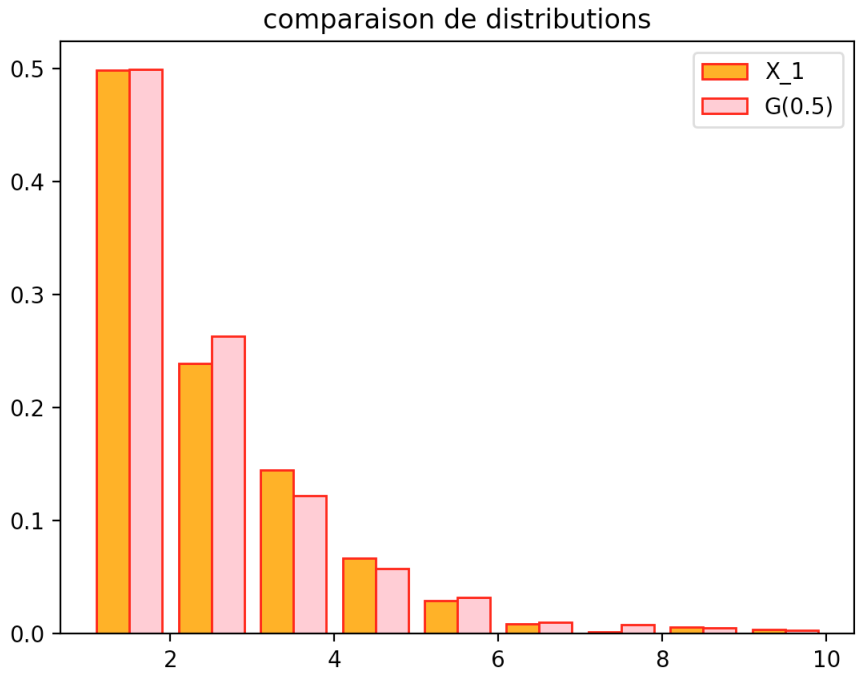
(1) Recopier et compléter la fonction suivante, prenant en argument la probabilité p d'obtenir `Pile` et permettant de simuler la variable aléatoire X_1.

```
def simul_X1(p):
    X=1
    t_old=rd.binomial(1,p)
    t_new=rd.binomial(1,p)
    while .... :
        X= ...
        t_old= ...
        t_new= ...
    return X
```

(2) On ajoute le code suivant qui affiche la figure reproduite ci-après. Interpréter.

```
p=0.5
N=1000
x1=[simul_X1(p) for k in range(N)]
x2=[rd.geometric(p) for k in range(N)]
plt.hist([x1, x2], color = ['orange', 'pink'], density=True,
         label = ['X_1', 'G(0.5)'])
plt.title('comparaison de distributions')
plt.legend()
plt.show()
```

Affichage Python



(3) Déterminer la loi de X_1. Retrouver le résultat conjecturé à la question précédente. Montrer qu'elle admet une espérance que l'on explicitera.

(4) Écrire une fonction d'entête `simul_X2(p)` permettant de simuler la variable X_2.

(5) Déterminer, pour $(k, j) \in \mathbb{N}^* \times \mathbb{N}^*$, $P(X_1 = k \cap X_2 = j)$. En déduire, à l'aide de la formule des probabilités totales, la loi de X_2.

Exercice 16. On considère la suite (u_k) définie, pour $k \geq 2$ par $u_k = \dfrac{1}{k-1} - \dfrac{1}{k}$.

(1) Vérifier la suite (u_k) peut être considérée comme une distribution de probabilité. On note alors Y une variable aléatoire telle que $Y(\Omega) = [\![2, +\infty[\![$ et $P(Y = k) = u_k$.

Soit $n \in \mathbb{N}^*$. On dispose d'une urne contenant n boules, indiscernables au toucher, numérotées de 1 à n et on effectue des tirages successifs, avec remise, dans cette urne.
On introduit la variable aléatoire X_n qui prend la valeur du rang du tirage pour lequel, pour la première fois, la boule piochée a un numéro supérieur ou égal à celle piochée lors du tout premier tirage.
Pour tout $k \in \mathbb{N}^*$, on note T_k la variable aléatoire qui correspond au numéro de la boule piochée au $k-$ième tirage.

(2) (a) Recopier et compléter la fonction suivante de sorte qu'elle renvoie une simulation de X_n.

```
import numpy as np
import numpy.random as rd

def simul_X(n) :
    t = .........
    k = ........
    while .......... :
        k=k+1
    return k
```

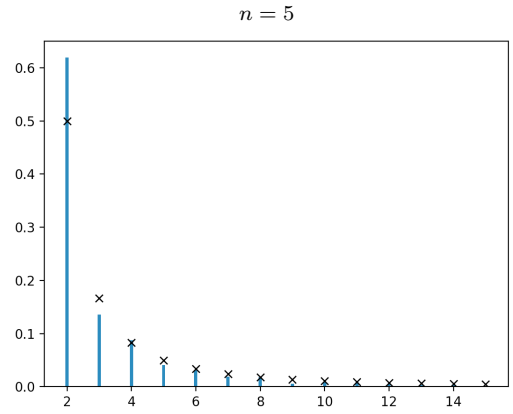
(3) Compléter les commandes suivantes qui permettent d'obtenir une liste de fréquences des valeurs obtenues pour un $1000-$échantillon de X_n.

```
def frequence(n):
    S=[simul_X(n) for k in range(1000)]
    m=max(S)
    f=np.zeros(m)
    for k in range(1000):
        f[......]+=1
    return f/1000
```

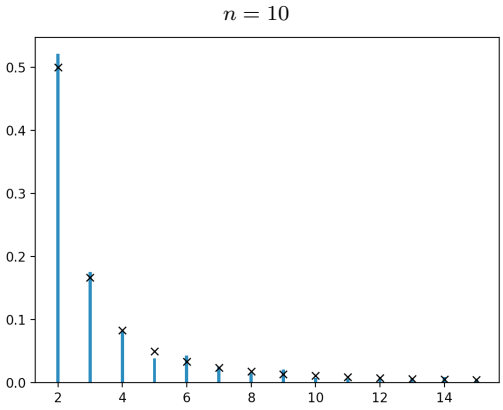
(4) On ajoute les commandes suivantes dont l'exécution permet l'affichage ci-après. Interpréter.

```
def u(k):
    return 1/(k-1)-1/k

import matplotlib.pyplot as plt

n=5 # puis n=10, puis n=25 puis n=100
obs=frequence(n)
N=min(len(obs), 14)
x=[k for k in range(2, N+2)]
U=[u(k) for k in x]
h=[obs[k] for k in range(N)]
plt.plot(x, U, 'kx')
plt.bar(x, h, width=0.1)
plt.show()
```

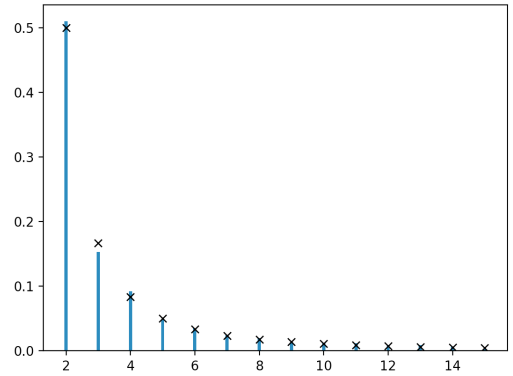
Affichage Python



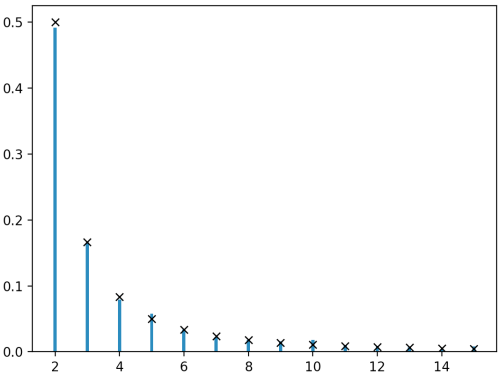
Affichage Python



Affichage Python



Affichage Python



(4) (a) Soient $k \geq 2$ et $1 \leq i \leq k$. Écrire l'évènement $[X_n = k] \cap [T_1 = i]$ à l'aide d'évènements dépendant uniquement des variables aléatoires T_j.

(b) En déduire que, pour tout $k \geq 2$,

$$P(X_n = k) = \frac{1}{n} \sum_{i=1}^{n} \left(\left(\frac{i-1}{n} \right)^{k-2} - \left(\frac{i-1}{n} \right)^{k-1} \right).$$

(5) (*) Montrer que (X_n) converge en loi vers Y. (*Cette question est pour l'instant réservée aux khubes.*)

Exercice 17.

On considère la matrice M définie par $M = \frac{1}{6}\begin{pmatrix} 6 & 3 & 0 \\ 0 & 3 & 4 \\ 0 & 0 & 2 \end{pmatrix}$ et les trois vecteurs de $\mathcal{M}_{3,1}(\mathbb{R})$ suivants

$$V_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \; V_2 = \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}, \; V_3 = \begin{pmatrix} 3 \\ -4 \\ 1 \end{pmatrix}$$

(1) La matrice M est-elle inversible ?
(2) Calculer MV_1, MV_2 et MV_3 en fonction de V_1, V_2 et V_3.

Une urne contient une boule rouge et deux boules blanches. On effectue dans cette urne une succession de tirages d'une boule selon le protocole suivant :

- si la boule tirée est rouge, elle est remise dans l'urne.
- si la boule tirée est blanche, elle n'est pas remise dans l'urne.

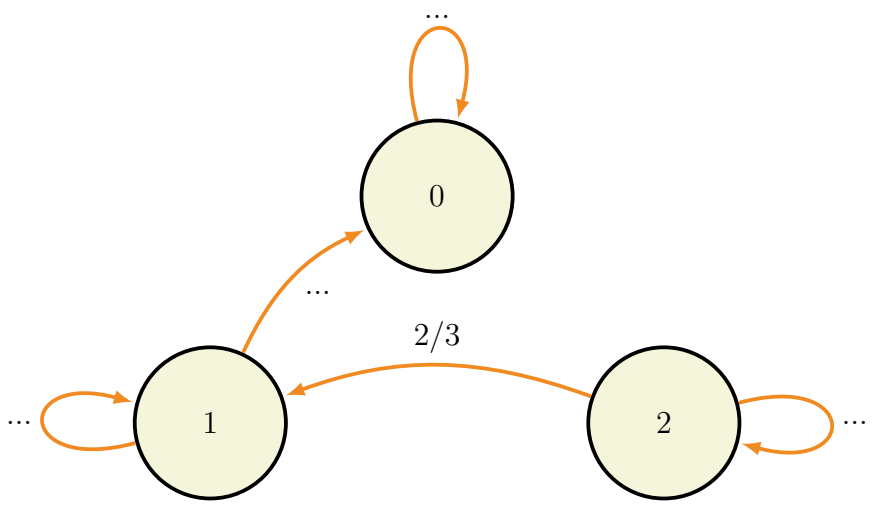
Pour tout entier i supérieur ou égal à 1, on note B_i (respectivement R_i) l'événement "on obtient une boule blanche (respectivement une boule rouge) lors du $i^{ème}$ tirage".

Pour tout entier n supérieur ou égal à 1, on note X_n le nombre de boules blanches contenues dans l'urne à l'issue du $n^{ème}$ tirage et on pose $X_0 = 2$.

On introduit la matrice colonne $U_n = \begin{pmatrix} P[X_n = 0] \\ P[X_n = 1] \\ P[X_n = 2] \end{pmatrix}$.

(3) (a) Déterminer pour tout entier naturel n, l'ensemble des valeurs prises par la variable X_n (on distinguera les trois cas : $n = 0, n = 1$ et $n \geq 2$).

(b) Recopier et compléter **en justifiant** le *diagramme de transition* ci-contre de la suite (X_n) (pour $n \geq 2$). On précisera notamment à quelles probabilités conditionnelles correspondent les valeurs sur les flèches du diagramme.



(c) En utilisant la formule des probabilités totales avec un système complet d'évènements construit avec la variable X_n, montrer que pour tout entier n supérieur ou égal à 2, on a l'égalité suivante :

$$P(X_{n+1} = 1) = \frac{1}{2}P(X_n = 1) + \frac{2}{3}P(X_n = 2).$$

Montrer de même qu'on a

$$U_{n+1} = MU_n.$$

Vérifier que l'égalité précédente reste valable pour $n = 0$ et $n = 1$.

(d) En déduire par récurrence, pour tout entier naturel n, la relation suivante :

$$U_n = V_1 + 4\left(\frac{1}{2}\right)^n V_2 + \left(\frac{1}{3}\right)^n V_3.$$

(e) Donner la loi de la variable X_n .

(4) Calculer $E(X_n)$, espérance de X_n , ainsi que sa limite lorsque n tend vers $+\infty$.

(5) Recopier et compléter la fonction suivante permettant de représenter graphiquement la *trajectoire* de la v.a. X_n.

```
def traj_X(n):
    X=np.zeros(n+1)
    X[0]=..........
    for i in range (1, n+1):
        if X[i-1]==0 :
            X[i]=.....
        elif ...... :
            X[i]= ...
        else:
            ........
    N=[k for k in range(n+1)]
    plt.grid()
    plt.plot(N, X, 'ko')
    plt.show()
```