Chapitre (NUM) 5 Aléatoire

L'aléatoire en Informatique n'existe pas
— Mathématicien

Résumé & Plan

Dans ce chapitre nous approfondissons les premières techniques de simulation de variables aléatoires réelles vues en cours.

- Les chapitres d'Informatique sont composés de cours et d'exercices intégrés. Le cours sera projeté au tableau.
- Il n'est pas attendu que toute la classe aborde tous les exercices. Traitez donc en priorité les exercices présents dans la liste donnée à chaque début de séance.
- Exercices 💣 / **Pour aller plus loin** : exercices plus difficiles, ou plus techniques. À ne regarder que si les autres sont bien compris.

Fichier externe ?

NON *pas de fichier externe dans ce TP*

1 INTRODUCTION

1.1 (faux)-aléatoire en Python

En cours de Maths, on a découvert la commande `rd.random()` qui permettait notamment de renvoyer un réel « au hasard » (cette loi sera formalisée en 2ème année) dans $[0,1]$. En fait, ce hasard est complètement factice comme l'illustre l'exercice qui suit.

Exercice 1 | [Solution] Testez deux fois les commandes ci-dessous dans une console, et comparez avec votre voisin.

```
import random as rd
rd.seed(0)
rd.random()
rd.random()
```

1.2 Estimer une espérance ou probabilité par simulation

Le cours de seconde année (sur la loi faible des grands nombres) permettra de justifier la méthode qui suit. Pour l'instant, elle est donc donnée sans justification supplémentaire mais est très intuitive.

Méthode (NUM) 5.1 (>_ Approcher une espérance (ou une probabilité) par simulation) Soient $X, X_1, \dots, X_n$ une suite de variables aléatoires indépendantes et de même loi, donc de même espérance $\mathbb{E}(X)$, et admettant une variance. Alors :

1. **[Pour l'espérance]**
$$\mathbb{E}(X) \approx \frac{\sum_{i=1}^{n} X_i}{n}, \quad \text{avec } n \text{ assez grand.}$$
On doit donc calculer la moyenne de simulations X_i pour approcher l'espérance.

2. **[Pour une probabilité/fonction de répartition]** Soit I un intervalle de $\mathbb{R}$. Alors comme $\mathbb{P}(X \in I) = \mathbb{E}\left(\mathbb{1}_{\{X \in I\}}\right)$ (rappel : nous avons établi en cours que pour tout évènement A, on a $\mathbb{E}(\mathbb{1}_A) = \mathbb{P}(A)$), on peut utiliser **1)** pour approcher une probabilité :
$$\mathbb{P}(X \in I) \approx \frac{\sum_{i=1}^{n} \mathbb{1}_{\{X_i \in I\}}}{n}, \quad \text{avec } n \text{ assez grand.}$$
On compte donc le nombre de simulations X_i dans I (c'est-à-dire la quan-

tité $\sum_{i=1}^{n} \mathbb{1}_{\{X_i \in I\}}$) et on renvoie la moyenne. Pour obtenir une approximation de $F_X(x)$, on compte le nombre de simulations « $\leqslant x$ ».

En résumé :

- pour estimer une probabilité d'évènement par simulation, on simule un grand nombre de fois l'expérience et on renvoie la fréquence où l'évènement a été réalisé.
- Pour estimer une espérance, on renvoie la moyenne de simulations.

2 EXERCICES

2.1 Expériences aléatoires

On commence ici par des exercices élémentaires sur les lois usuelles.

Exercice 2 | Tirage avec remise [Solution] On cherche à modéliser un tirage aléatoire de n boules **avec remise** dans une urne contenant M boules blanches et Q boules noires, avec N = M + Q le nombre total de boules dans l'urne, et compter le nombre X de boules blanches obtenues.

On reconnaît que X suit la loi binomiale de paramètres n et p, avec $p = \frac{M}{N}$. Pour une valeur de k donnée, on cherche à estimer la probabilité de l'évènement $\{X = k\}$ en l'approchant par une fréquence.

1. Définir une fonction d'en-tête `X(n, M, Q)` qui renvoie une simulation de X, le nombre de boules blanches.
2. On s'intéresse maintenant à la fréquence de réalisation de l'évènement $\{X = k\}$ lorsqu'on effectue un grand nombre de fois l'expérience (une expérience consiste en n tirages avec remise). Écrire une fonction d'en-tête `frequenceX(k, n, M, Q, nb_simu)` qui réalise `nb_simu` fois l'expérience et détermine la fréquence d'apparition de l'évènement $\{X = k\}$ sur ces `nb_simu` tests. Tester `frequenceX(3, 4, 7, 5, 1000)` et `frequenceX(3, 4, 7, 5, 10000)`.
3. Définir une fonction d'en-tête `probaX(k, n, M, Q)` qui renvoie la probabilité théorique de l'évènement $\{X = k\}$. Testez `probaX(3, 4, 7, 5)`. *Pour obtenir le coefficient binomial, on réalisera l'importation* `import scipy.special`, *puis on utilisera la fonction* `scipy.special.binom`
4. Écrire une fonction d'en-tête `diagramme(n, M, Q)` qui affiche le diagramme en bâtons représentant la loi de X. La tester pour $n = 4$, M = 7 et Q = 5. *On utilisera la fonction* `plt.bar` *comme présentée dans le cours de statistiques.*
5. Écrire une fonction d'en-tête `moyenneX(n, M, Q, nb_simu)` qui renvoie le **nombre moyen** de boules blanches obtenues pour une série de `nb_simu` tests, chaque test représentant une succession de n tirages. Quelle quantité a-t-on approchée par ce calcul ? Le vérifier.

Exercice 3 | Tirage sans remise [Solution] On cherche à modéliser un tirage aléatoire de n boules **sans remise** dans une urne contenant M boules blanches et Q boules noires, avec N = M + Q le nombre total de boules dans l'urne, et compter le nombre Y de boules blanches obtenues. *(pour information, on dit que* Y *suit une loi « hypergéométrique » associée à certains paramètres que je ne préciserai pas)* On cherche alors à estimer la probabilité de l'évènement $\{Y = k\}$, pour une valeur de k donnée. On rappelle que X a été définie dans l'exercice précédent.

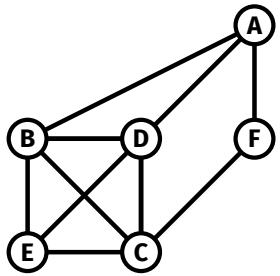
1. Définir une fonction d'en-tête `Y(n, M, Q)` qui renvoie le nombre de boules blanches obtenues lors de n tirages sans remise dans l'urne.
2. On s'intéresse maintenant à la fréquence de réalisation de l'évènement $\{Y = k\}$ lorsqu'on effectue un grand nombre de fois l'expérience (une expérience est un tirage de n boules sans remise). Écrire une fonction d'en-tête `frequenceY(k, n, M, Q, nb_simu)` qui réalise `nb_simu` fois l'expérience, compte à chaque fois le nombre de boules blanches obtenus (Y) et détermine la fréquence d'apparition de l'évènement $\{Y = k\}$ sur ces `nb_simu` tests.
3. Écrire une fonction d'en-tête `diagramme2(n,M,Q)` qui affiche sur le même graphique le diagramme en bâtons représentant la loi de X et celui (approché) représentant la loi de Y.
   On pourra décaler légèrement les bâtons correspondant à la loi de Y *pour une meilleure visualisation.*
4. Tracer sur le même graphique le diagramme en bâtons représentant la loi de X et celui représentant la loi de Y.
   - pour $n = 4$, M = 7 et Q = 5,
   - pour $n = 4$, M = 70 et Q = 50,
   - pour $n = 4$, M = 700 et Q = 500.

   Que remarque-t-on ? Sauriez-vous l'expliquer ?

2.2 Chaîne de Markov & Graphe

Exercice 4 | Marche aléatoire sur un graphe [Solution] Une fourmi est au point A sur le graphe $\mathscr{F}$ ci-contre. Elle se déplace sur le graphe de sommet en sommet en

choisissant au hasard une arête (y compris celle qu'elle vient d'emprunter) quand elle est sur un sommet.



Graphe $\mathscr{F}$

1. Implémenter le graphe $\mathscr{F}$ sous la forme d'un dictionnaire.

2. **[Probabilité de présence sur un sommet]**

2.1) Écrire une fonction `arrivee` prenant en argument un entier naturel n et qui renvoie la position de la fourmi sur le graphe $\mathscr{F}$ après n déplacements. *Vous pourrez vous servir de `rd.choice` qui permet de choisir un élément dans une liste selon une loi uniforme.*

2.2) Écrire une fonction `frequence(n, nb_simu, s)` prenant en argument deux entiers naturels n et `nb_simu`, le nom sommet d'un sommet `s` et qui renvoie la fréquence de fois où la fourmi se trouve sur le sommet `s` après n déplacements.

2.3) Estimer la probabilité que la fourmi soit au sommet C après 10 déplacements.

3. Les chemins de longueur minimale d'origine A et sortie E sont A → D → E ou encore A → B → E (que l'on peut trouver grâce au Chapitre (ALGO) 8). Déterminer une valeur approchée de la probabilité qu'elle trouve l'un des deux chemins précédents au bout de deux déplacements, sur 10^3 essais.

2.3 Dynamique des populations & Génétique

Exercice 5 | Processus de branchement de Galton-Watson [Solution] On souhaite modéliser l'effectif d'une population à reproduction assexuée contenant initialement 1 individu et suivant les règles d'évolution suivantes :

- À chaque génération, chacun des individus donne naissance avant de disparaitre :
  - à 2 descendants, avec probabilité $p \in]0,1[$,
  - à aucun descendant avec probabilité $q = 1-p \in]0,1[$.
- Le fait qu'un individu donne ou non naissance à 2 enfants est indépendant des naissances de tous les autres individus de la population.

On note X_n le nombre d'individus que contient la génération n.

1. Réecrire le code de la fonction `binomiale(n, p)` du cours permettant de renvoyer une simulation de la loi $\mathscr{B}(n,p)$.

2. Én déduire une fonction d'en-tête `suiteX(p, n)` qui simule l'évolution d'une population dont la génération 0 comporte 1 individu sur n générations selon le processus précédent et qui renvoie la liste $[X_0, X_1, \dots, X_n]$.

3. Tester au moins 10 fois la fonction d'en-tête `suiteX` en prenant à chaque fois $p = \frac{1}{3}$ et $n = 10$. Que remarquez-vous ?

4. Tester au moins 10 fois la fonction d'en-tête `suiteX` en prenant à chaque fois $p = \frac{2}{3}$ et $n = 10$. Que remarquez-vous ?

5. **5.1)** Écrire une fonction d'en-tête `extinction(p, n)` qui simule l'évolution d'une population dont la génération 0 comporte 1 individu sur n générations selon le processus précédent et qui renvoie 1 et si la population s'est éteinte au bout de n générations et 0 sinon.

5.2) Écrire une fonction d'en-tête `freqExtinction(p, n, nb_simu)` qui renvoie la fréquence d'extinction d'une telle population si l'on répète `nb_simu` le processus. Tester `freqExtinction(1/3,10,10000)` et `freqExtinction(2/3,10,10000)`.

5.3) Un calcul théorique permet de montrer que la probabilité d'extinction vaut $\frac{q}{p}$ si $p > \frac{1}{2}$ et 1 si $p \leqslant \frac{1}{2}$. Vérifier que ce résultat est conforme à ce que vous avez obtenu.

Exercice 6 | Modèle de Wright-Fisher [Solution]

Le modèle de Wright-Fisher décrit l'évolution au cours du temps de la fréquence d'un allèle A dans une population. On s'intéresse pour simplifier au cas d'une reproduction asexuée. Dans ce modèle, on fait deux hypothèses simplificatrices :

- la population est de taille constante égale à N,
- les générations ne se chevauchent pas.

On note X_k le nombre d'individus portant l'allèle A à la génération k. Si d'aventure à un certain rang on a $X_k = N$, on dit que l'allèle A s'est *fixé* (et de même si $X_k = 0$, l'allèle B est fixé).

On considère la règle d'évolution suivante de la suite (X_k).

- Initialement, on pose $X_0 = i$ avec $i \in [\![0, N]\!]$.
- Les N individus d'une génération $k+1$ sont les enfants des N individus de la génération k, on suppose que :
  - ◇ chacun de ces enfants a un parent tiré au hasard parmi les N individus de la génération précédente. Ce tirage se fait uniformément (pas d'avantage sélectif).
  - ◇ Un enfant hérite directement de l'allèle de son parent (on néglige la possibilité de mutations).

1. Dans cette question, on s'intéresse à la simulation de la suite (X_k).

1.1) D'après les hypothèses, en supposant qu'on ait déjà simulé X_k et qu'on ait $X_k = a$ avec $a \in [\![0, N]\!]$, quelle est la loi de X_{k+1} ?

1.2) Créer une fonction d'en-tête `wrightfisher(i, N)` qui simule la suite (X_k) et plus précisément renvoie la liste `[X_0, X_1, ..., X_T]` où T est le numéro de la génération où un des deux allèles s'est fixé. La simulation doit s'arrêter à la fixation de l'un ou l'autre des allèles. *On admet qu'avec probabilité 1 l'un ou l'autre des allèles sera fixé.*

2. On s'intéresse maintenant à la probabilité de fixation de l'allèle A. Cette probabilité dépend de i et N.

2.1) Créer une fonction d'en-tête `estim_proba(i, N, nb_ex)` qui renvoie une valeur approchée de p_A, en simulant `nb_ex` fois la suite (X_k).

2.2) Avec N = 50, tracer la courbe de p_A en fonction de $i \in [\![0, N]\!]$, où `nb_ex` est choisi assez grand de sorte que la courbe soit suffisamment régulière, mais suffisamment petit pour que le temps de calcul soit raisonnable.

2.3) Recommencez pour d'autres valeurs de N. Qu'observez-vous ? Conjecturer une expression de p_A en fonction de i et N.

3. On s'intéresse au temps moyen de fixation, on notera T la variable aléatoire donnant le temps de fixation, c'est-à-dire :

$$T = \min\{n \in \mathbb{N} \,|\, X_n \in \{0, N\}\}.$$

3.1) Créer une fonction d'en-tête `estim_tempsmoyen(i, N, nb_ex)` qui renvoie une valeur approchée de $\mathbb{E}(T)$.

3.2) Avec N = 50, tracer la courbe de $\mathbb{E}(T)$ en fonction de $i \in [\![0, N]\!]$, où `nb_ex` est choisi assez grand de sorte que la courbe soit suffisamment régulière.

3.3) Un calcul permet de montrer que pour N grand, et $i \in [\![1, N-1]\!]$:

$$\mathbb{E}(T) \approx -2N(x\ln(x) + (1-x)\ln(1-x)), \quad x = \frac{i}{n}.$$

Confronter ce résultat avec la courbe obtenue dans la question précédente.

3 SOLUTIONS DES EXERCICES

Solution (exercice 1) [Énoncé]

```
>>> import random as rd
>>> rd.seed(0)
>>> rd.random()
0.8444218515250481
>>> rd.random()
0.7579544029403025
```

Eh allez, on recommence :

```
>>> import random as rd
>>> rd.seed(0)
>>> rd.random()
0.8444218515250481
>>> rd.random()
0.7579544029403025
```

Donc on vous ment : avec la commande `seed` on aura toujours la même suite de valeurs, ce qui ne semble pas très « aléatoire ». En fait Python utilise des suites récurrentes bien calibrées qui « ressemblent » à une suite de simulations uniformément réparties entre 0 et 1.

Solution (exercice 2) [Énoncé]

1.
```
def X(n, M, Q):
    p = M/(M+Q)
    X = 0
    for _ in range(n):
        if rd.random() < p:
            X += 1
    return X
```

2.
```
def frequenceX(k, n, M, Q, nb_simu):
    S = 0
    for _ in range(nb_simu):
        if X(n, M, Q) == k:
            S += 1
    return S/nb_simu
```

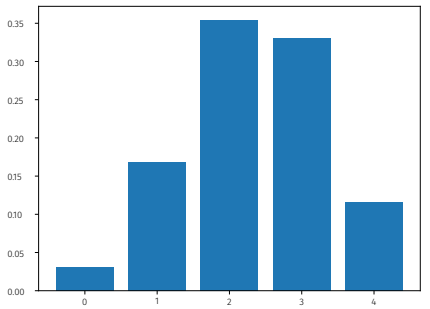
```
>>> frequenceX(3, 4, 7, 5, 1000)
0.335
>>> frequenceX(3, 4, 7, 5, 10000)
0.3401
```

3.
```
import scipy.special
def probaX(k, n, M, Q):
    p = M/(M+Q)
    return scipy.special.binom(n, k)*p**k*(1-p)**(n-k)
```

```
>>> probaX(3, 4, 7, 5) # proche de la précédente
np.float64(0.33082561728395066)
```

4.
```
def diagramme(n, M, Q):
    Support = range(0, n+1)
    Loi = [probaX(k, n, M, Q) for k in Support]
    plt.bar(Support, Loi)

diagramme(4, 7, 5)
```



5.
```
def moyenneX(n, M, Q, nb_simu):
    S = 0
    for _ in range(nb_simu):
        S += X(n, M, Q)
    return S/nb_simu
```

```
>>> moyenneX(4, 7, 5, 10000) # valeur approchée de \
↪ l'espérance
2.3334
>>> 4*(7/12)
2.3333333333333335
```

Solution (exercice 3) [Énoncé]

1.
```
def Y(n, M, Q):
    n_bl, n_no = M, Q
    Y = 0
    for _ in range(n):
```

```
            if rd.random() < n_bl/(n_bl+n_no):
                # blanche
                Y += 1
                n_bl -= 1
            else:
                n_no -= 1
        return Y
```

Cette fonction ne fonctionne pas lorsque le nombre de tirages dépasse le nombre de boules (division par zéro). On peut adapter ainsi :

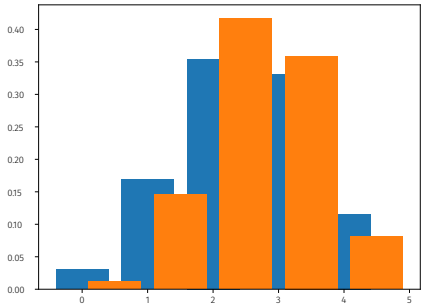
```
def Y(n, M, Q):
    n_bl, n_no = M, Q
    if n >= n_bl + n_no:
        return n_bl
    else:
        Y = 0
        for _ in range(n):
            if rd.random() < n_bl/(n_bl+n_no):
                # blanche
                Y += 1
                n_bl -= 1
            else:
                n_no -= 1
        return Y
```

```
>>> Y(20, 2, 3)
2
```

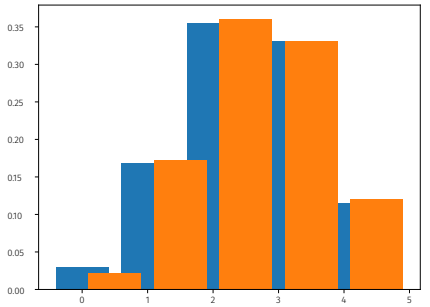
2.
```
def frequenceY(k, n, M, Q, nb_simu):
    S = 0
    for _ in range(nb_simu):
        if Y(n, M, Q) == k:
            S += 1
    return S/nb_simu
```

3.
```
def diagramme2(n, M, Q):
    diagramme(n, M, Q) # diagramme de X
    Support = range(0, n+1)
    Loi = [frequenceY(k, n, M, Q, 10**3) for k in Support]
    plt.bar([x+0.5 for x in Support], Loi)
```

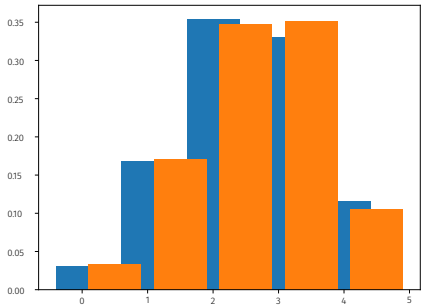
4.
```
n, M, Q = 4, 7, 5
diagramme2(n, M, Q)
```



```
n, M, Q = 4, 70, 50
diagramme2(n, M, Q)
```



```
n, M, Q = 4, 700, 500
diagramme2(n, M, Q)
```



Lorsque M + N grandit (le nombre total de boules), le tirage avec remise devient équivalent au tirage avec remise. Les deux lois seront alors très proches.

Solution (exercice 4) [Énoncé]

1.
```
F = {}
```

```
F['A'] = ['B','D','F']
F['B'] = ['A','C','D','E']
F['C'] = ['B','D','E','F']
F['D'] = ['A','B','C','E']
F['E'] = ['B','C','D']
F['F'] = ['A','C']
```

2. [Probabilité de présence sur un sommet]

2.1)
```
def arrivee(n):
    s = 'A'
    for _ in range(n):
        voisins = F[s]
        s = rd.choice(voisins)
    return s
```
```
>>> arrivee(10)
'E'
>>> arrivee(10)
'C'
>>> arrivee(10)
'F'
```

2.2)
```
def frequence(n, nb_simu, s):
    nb = 0
    for _ in range(nb_simu):
        arr = arrivee(n)
        if arr == s:
            nb += 1
    return nb/nb_simu
```

2.3) Estimer la probabilité que la fourmi soit au sommet C après 10 déplacements.
```
>>> frequence(10, 10**3, 'C')
0.21
```

3.
```
>>> frequence(2, 10**3, 'E')
0.157
```

Solution (exercice 5)

[Énoncé] Sachant $X_n = k$, le nombre d'individus de la génération $n+1$ a même loi que $2\times Y$ où $Y \hookrightarrow \mathscr{B}(k,p)$. On peut se servir de ce fait.

```
def suiteX(p, n):
    L = [1]
    for _ in range(1, n+1):
        X = L[-1] # valeur précédente
        L.append(2*binomiale(X, p))
    return L
```

On peut aussi faire une fonction indépendante.

```
def suiteX(p, n):
    L = [1]
    for _ in range(1, n+1):
        X = L[-1] # valeur précédente
        for _ in range(X):
            if rd.random() < p:
                X += 1 # disparition + 2 descendants -1 ascendant
            else:
                X -= 1 # disparition sans descendant
        L.append(X)
    return L
```
```
>>> p = 1/3
>>> suiteX(p, 10)
[1, 2, 4, 0, 0, 0, 0, 0, 0, 0, 0]
>>> suiteX(p, 10)
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> suiteX(p, 10)
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> suiteX(p, 10)
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> suiteX(p, 10)
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> suiteX(p, 10)
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> suiteX(p, 10)
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> suiteX(p, 10)
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> suiteX(p, 10)
[1, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0]
>>> suiteX(p, 10)
[1, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0]
```

On constate donc que la population finit très souvent par s'éteindre. Logique, puisqu'il y a souvent disparition sans descendant. Si $p = \frac{2}{3}$ on observe naturellement l'inverse :

```
>>> p = 2/3
```

```
>>> suiteX(p, 10)
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> suiteX(p, 10)
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> suiteX(p, 10)
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> suiteX(p, 10)
[1, 2, 2, 2, 4, 6, 8, 14, 22, 40, 54]
>>> suiteX(p, 10)
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> suiteX(p, 10)
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> suiteX(p, 10)
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> suiteX(p, 10)
[1, 2, 4, 2, 4, 6, 6, 10, 14, 24, 26]
>>> suiteX(p, 10)
[1, 2, 4, 4, 4, 6, 12, 22, 32, 40, 54]
>>> suiteX(p, 10)
[1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
def extinction(p, n):
    L = suiteX(p, n)
    return int(L[-1] == 0)
def freqExtinction(p, n, nb_simu):
    S = 0
    for _ in range(nb_simu):
        S += extinction(p, n)
    return S/nb_simu
```

```
>>> p = 1/3
>>> q = 1-p
>>> freqExtinction(p, 30, 10000), 1
(1.0, 1)
>>> p = 2/3
>>> q = 1-p
>>> freqExtinction(p, 30, 10000), q/p
(0.502, 0.5000000000000001)
```

On voit que les valeurs expérimentales sont bien proches des valeurs théoriques (mentionnées dans le second affichage).

Solution (exercice 6) [Énoncé]

1. **1.1)** Sachant que $X_k = a$, alors X_{k+1} est le nombre d'individus porteurs de l'allèle A à la génération suivante, mais cela correspond alors au nombre d'enfants parmi les N qui auront choisi un parent porteur de A. Or, sachant $X_k = a$, les parents porteurs de A sont en proportion $\frac{a}{N}$ et les choix sont indépendants (puisque chaque enfant choisit un parent de manière équiprobable), donc la loi (conditionnelle) cherchée est une $\boxed{\mathscr{B}\left(N, \frac{a}{N}\right)}$.

1.2) Pour simuler, on peut utiliser par exemple la fonction `binomiale` du cours.

```
def wrightfisher(i, N):
    L = [i]
    while L[-1] != 0 and L[-1] != N:
        L.append(binomiale(N, L[-1]/N))
    return L

L_1 = wrightfisher(10, 20)
L_2 = wrightfisher(50, 100) # la proportion initiale \
↪ d'allèle A est la même dans les deux populations
```

Testons cette fonction.

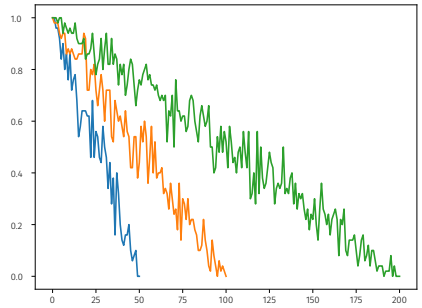
```
>>> L_1
[10, 10, 10, 11, 16, 13, 9, 5, 6, 7, 8, 7, 8, 8, 6, 2, \
↪ 0]
>>> L_2
```

```
[50, 49, 52, 52, 46, 51, 61, 67, 74, 71, 72, 73, 74, 6
8, 65, 53, 53, 53, 54, 50, 54, 49, 47, 50, 60, 54, 50, 4
7, 44, 42, 31, 38, 44, 45, 44, 38, 36, 27, 26, 25, 29, 3
1, 43, 46, 40, 37, 31, 31, 30, 28, 36, 35, 32, 37, 47, 4
9, 51, 47, 47, 43, 35, 39, 37, 43, 44, 41, 36, 29, 24, 2
4, 32, 32, 25, 15, 16, 17, 12, 12, 15, 11, 12, 11, 9, \
↪ 7, 9, 11, 16, 15, 14, 16, 14, 18, 16, 19, 12, 11, 13, \
↪ 14, 14, 21, 19, 21, 26, 24, 28, 24, 29, 25, 20, 24, 2
1, 24, 25, 26, 25, 23, 23, 22, 16, 21, 28, 25, 25, 24, 2
2, 26, 24, 22, 21, 13, 17, 18, 24, 20, 20, 16, 15, 12, 1
6, 19, 22, 22, 24, 18, 19, 19, 15, 12, 13, 9, 4, 6, 3, \
↪ 2, 3, 3, 1, 1, 2, 2, 3, 5, 8, 10, 9, 8, 9, 10, 12, 1
3, 11, 11, 10, 9, 6, 3, 2, 1, 3, 4, 4, 6, 2, 2, 5, 5, \
↪ 5, 9, 10, 13, 16, 15, 14, 17, 22, 20, 24, 23, 33, 37, \
↪ 32, 31, 27, 27, 30, 33, 29, 26, 23, 16, 18, 12, 11, \
↪ 6, 3, 2, 1, 1, 0]
```

2. **2.1)**

```
def estim_proba(i, N, nb_ex):
    nb = 0
    for _ in range(nb_ex):
        L = wrightfisher(i, N)
        if L[-1] == 0:
            nb += 1
    return nb/nb_ex

N = 50
T_1 = range(0, N+1)
P_1 = [estim_proba(i, N, 50) for i in T_1]
N = 100
T_2 = range(0, N+1)
P_2 = [estim_proba(i, N, 50) for i in T_2]
N = 200
T_3 = range(0, N+1)
P_3 = [estim_proba(i, N, 50) for i in T_3]
plt.plot(T_1, P_1)
plt.plot(T_2, P_2)
plt.plot(T_3, P_3)
```



On conjecture une dépendance affine de p_A en fonction de i, d'ordonnée à l'origine 1. Donc il existe $a_N \in \mathbb{R}$ de sorte que

$$\boxed{\forall i \in [\![0, N]\!], \quad p_A = a_N \cdot i + 1}.$$

```
def estim_tempsmoyen(i, N, nb_ex):
    moyenne = 0
    for _ in range(nb_ex):
        moyenne += len(wrightfisher(i, N))
    return moyenne/nb_ex

N = 50
T = range(1, N)
P_1 = [estim_tempsmoyen(i, N, 50) for i in T]
P_theo = [-2*N*(i/N*ma.log(i/N)+(1-i/N)*ma.log(1-i/N)) \
↪ for i in T]
plt.plot(T, P_1)
plt.plot(T, P_theo)
```