Chapitre # (ALGO) 12 Aléatoire

1 Rappel des commandes principales .....

2 Exercices .....

3 Solutions des exercices .....

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.

1. RAPPEL DES COMMANDES PRINCIPALES

La bibliothèque `random` contient de nombreuses fonctions pré-programmées permettant de simuler des variables aléatoires. On pourra l'importer via la commande `import random as rd`. Nous utiliserons essentiellement les fonctions :

1. `rd.random()` *(sans paramètre)* qui renvoie un flottant pris aléatoirement dans l'intervalle $[0,1[$: pour importer cette fonction, écrire `from random import random`
2. `rd.randint(a,b)` *(où a et b sont deux paramètres entiers vérifiant $a \leqslant b$)* qui renvoie un entier pris aléatoirement dans l'intervalle $[\![a;b]\!]$: pour importer cette fonction, écrire `from random import randint`.

2. EXERCICES

2.1. Expériences aléatoires

On commence ici par quelques exercices sur les lois usuelles (la loi binomiale notamment).

Exercice 1 | 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 $Q = 5$. *La fonction `plt.bar(X,Y)` génère l'histogramme des points définis par les listes X et Y (abscisses et ordonnées).*
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 2 | 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. Dynamique des populations & Génétique

Exercice 3 | 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. Écrire 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 $[X_0, X_1, \dots, X_n]$.
2. 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 ?
3. 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 ?
4. **4.1)** 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 éteint au bout de n générations et 0 sinon.
   4.2) 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)`.
   4.3) 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.

2.3. Tracés de farctales

Exercice 4 | La fougère de Barnsley Solution

Dans cette partie, on souhaite construire (et tracer) une suite aléatoire de points $(A_n)_{n\in\mathbb{N}}$ du plan. On considère donc ici des variables aléatoires à valeurs dans $\mathbb{R}^2$ (et non dans $\mathbb{R}$ comme dans les exemples précédents).

- Le premier point de cette suite est $A_0(0,0)$.
- Pour tout $n \in \mathbb{N}$, on note (x_n, y_n) les coordonnées de A_n.
  On obtient les coordonnées (x_{n+1}, y_{n+1}) du point suivant A_{n+1} selon le principe suivant :

(1) $\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0,16 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix}$ avec probabilité $0,01$. *(Transformation 1)*

(2) $\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 0,85 & 0,04 \\ -0,04 & 0,85 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} 0 \\ 1,6 \end{pmatrix}$ avec probabilité $0,85$. *(Transformation 2)*

(3) $\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 0,2 & -0,26 \\ 0,23 & 0,22 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} 0 \\ 1,6 \end{pmatrix}$ avec probabilité $0,07$. *(Transformation 3)*

(4) $\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} -0,15 & 0,18 \\ 0,26 & 0,24 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} 0 \\ 0,44 \end{pmatrix}$ avec proba $0,07$. *(Transformation 4)*

La suite de points ainsi obtenue forme une *fractale* appelée **fougère de Barnsley**.

1. Écrire quatre fonctions `transformation1(x,y)`, `transformation2(x,y)`, `transformation3(x,y)` et `transformation4(x,y)`. Chaque fonction renvoie les coordonnées (x_{n+1}, y_{n+1}) du point A_{n+1} si le point A_n a pour coordonnées (x, y) pour la transformation correspondante.
2. En utilisant la fonction d'en-tête `random()` , comment peut-on choisir quelle transformation effectuer pour construire le point A_{n+1} à partir du point A_n ?

3. Écrire une fonction d'en-tête `fougere(n)` qui construit et affiche la suite de points $(A_0, \cdots, A_n)$. Tester cette fonction pour $n = 10000$.

3. SOLUTIONS DES EXERCICES

Solution (exercice 1) Énoncé

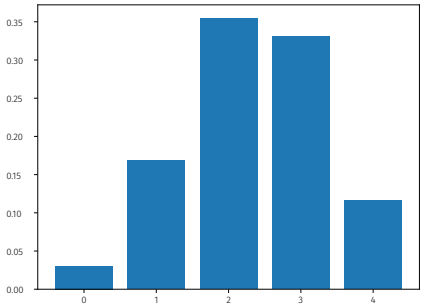
1.
```
def X(n, M, Q):
    p = M/(M+Q)
    X = 0
    for i in range(n):
        if rd.random() < p:
            X += 1
    return X
```

2.
```
def frequenceX(k, n, M, Q, nb_simu):
    S = 0
    for i in range(nb_simu):
        if X(n, M, Q) == k:
            S += 1
    return S/nb_simu
>>> frequenceX(3, 4, 7, 5, 1000)
0.315
>>> frequenceX(3, 4, 7, 5, 10000)
0.3288
```

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
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 i in range(nb_simu):
        S += X(n, M, Q)
    return S/nb_simu
>>> moyenneX(4, 7, 5, 10000) # valeur approchée de \
↪ l'espérance
2.3317
>>> 4*(7/12)
2.3333333333333335
```

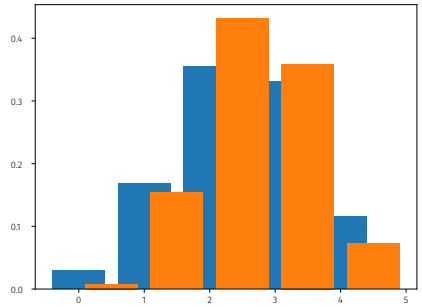
Solution (exercice 2) Énoncé

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

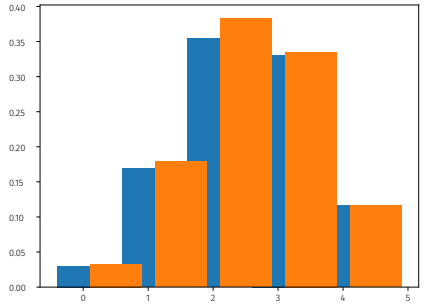
2.
```
def frequenceY(k, n, M, Q, nb_simu):
    S = 0
    for i in range(nb_simu):
        if Y(n, M, Q) == k:
            S += 1
    return S/nb_simu
```

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)
```

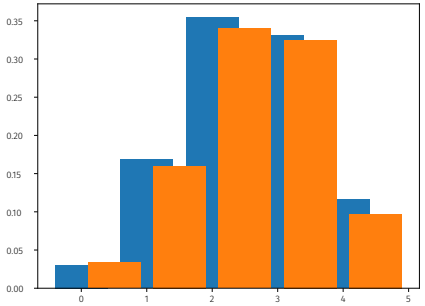
4.
```
n, M, Q = 4, 7, 5
diagramme2(n, M, Q)
```




```
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 3) Énoncé

```
def suiteX(p,n) :
    pop = 1
    L = [1]
    for k in range(n) :
        newpop = 0
        for i in range(pop) :
            if rd.random() < p :
                newpop = newpop + 2
        pop = newpop
        L.append(pop)
    return L
>>> p = 1/3
>>> suiteX(p, 10)
[1, 2, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> suiteX(p, 10)
[1, 2, 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, 0, 0, 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 $p = \frac{2}{3}$, il semblerait que cette fois la population ne s'éteigne pas systématiquement au bout de 10 générations mais seulement une fois sur deux ?

```
>>> 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, 2, 2, 4, 2, 2, 2, 2, 2, 4, 6]
>>> suiteX(p, 10)
[1, 2, 2, 4, 6, 10, 12, 16, 22, 26, 38]
>>> suiteX(p, 10)
[1, 2, 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, 6, 8, 14, 18, 24, 26, 40, 58]
>>> suiteX(p, 10)
[1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> suiteX(p, 10)
[1, 2, 2, 4, 6, 12, 20, 32, 34, 36, 48]
def extinction(p, n):
    L = suiteX(p, n)
    return 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, q = 1/3, 2/3
>>> freqExtinction(p, 10, 10000), 1
(0.9951, 1)
>>> p, q = 2/3, 1/3
>>> freqExtinction(2/3,10,10000), q/p
(0.5008, 0.5)
```

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

Solution (exercice 4) Énoncé

1.

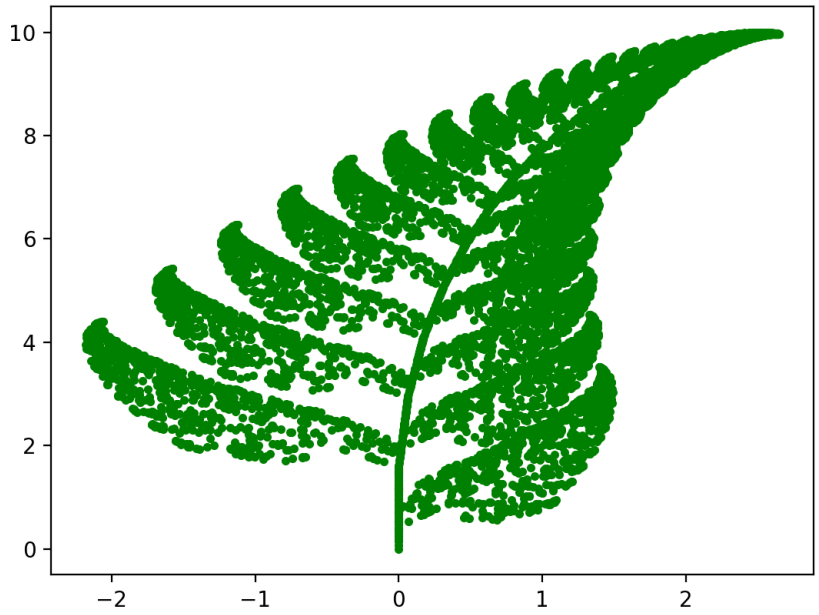
```
def transformation1(x,y) :
    return 0 , 0.16*y

def transformation2(x,y):
    return 0.85*x + 0.04*y , -0.04*x + 0.85*y+1.6

def transformation3(x,y):
    return 0.2*x - 0.26*y , 0.23*x + 0.22*y+1.6

def transformation4(x,y):
    return -0.15*x + 0.18*y , 0.26*x + 0.24*y+0.44
```

2.

```
def fougere(n):
    x,y=0,0
    abscisses = [0]
    ordonnees = [0]
    for i in range(n):
        r = rd.random()
        if r < 0.01 :
            x,y = transformation1(x,y)
        elif r < 0.86 :
            x,y = transformation2(x,y)
        elif r < 0.93 :
            x,y = transformation3(x,y)
        else :
            x,y = transformation4(x,y)
        abscisses.append(x)
        ordonnees.append(y)
    plt.plot(abscisses,ordonnees,'g.')
    plt.show()
```

3.