

Plaquette 5 - Simuler une expérience aléatoire

Algorithmique et Python — Première STI2D

Corrections détaillées

Boîte à outils

Tirer au hasard.

```
from random import randint, random
```

```
randint(1, 6)    # un entier entre 1 et 6, bornes comprises
random()         # un réel de l'intervalle [0 ; 1[
```

- **Simuler un succès de probabilité p** : la condition $random() < p$ est vraie avec la probabilité p .
- **Simuler un dé équilibré** : $randint(1, 6)$.

Le schéma d'une simulation.

```
def experience(p):
    ...          # une répétition, renvoie 0 ou 1, ou un gain

def frequence(nb, p):
    s = 0
    for k in range(nb):
        s = s + experience(p)
    return s / nb
```

- La **fréquence observée** est $f = \frac{\text{nombre de succès}}{\text{nombre de simulations}}$.
- Elle **fluctue** : deux exécutions ne donnent pas le même résultat.
- Plus le nombre de simulations est grand, plus f se rapproche de la probabilité.
- Une **moyenne** de gains simulés se rapproche de l'espérance $E(X)$.

Probabilités utiles pour vérifier une simulation.

- Répétition de n épreuves indépendantes de probabilité p :
 $P(\text{aucun succès}) = (1 - p)^n$, donc

$$P(\text{au moins un succès}) = 1 - (1 - p)^n$$

- Nombre moyen de succès : $E(X) = n \times p$.

⚠ Une simulation **ne démontre rien** : elle donne un ordre de grandeur qu'on compare au calcul exact quand il existe.

Correction de l'exercice 1 - Lire un tirage au hasard

- $randint(1, 6)$ renvoie un entier entre 1 et 6, **bornes comprises** : 1, 2, 3, 4, 5 ou 6, chacun avec la probabilité $\frac{1}{6}$, comme un dé équilibré.
- $random()$ renvoie un nombre décimal de l'intervalle $[0; 1[$: 0 est possible, 1 ne l'est pas.
- Le nombre tiré est réparti uniformément sur $[0; 1[$, de longueur 1 : la probabilité qu'il tombe dans $[0; 0,03[$ est la longueur de cet intervalle,

$$P(random() < 0,03) = 0,03$$

C'est ainsi qu'on simule un défaut de fabrication au taux de 3 % : plus généralement, $\text{random}() < p$ simule un succès de probabilité p .

Réponse. a) les entiers de 1 à 6 · b) un réel de $[0 ; 1[$ · c) 0,03.

Correction de l'exercice 2 - Simuler une pièce défectueuse

La condition $\text{random}() < p$ est vraie avec la probabilité p : on renvoie alors 1 (pièce défectueuse), et 0 sinon.

```
from random import random
def piece(p):
    if random() < p:
        return 1
    else:
        return 0
```

La fonction simule une variable de Bernoulli de paramètre p , d'espérance

$$E(X) = 1 \times p + 0 \times (1 - p) = p$$

Le codage par 0 et 1 est commode : en additionnant les résultats de plusieurs appels, on obtient directement le nombre de pièces défectueuses.

Réponse. Une fonction renvoyant 1 si $\text{random}() < p$, et 0 sinon.

Correction de l'exercice 3 - Fréquence sur un échantillon

On compte les défauts dans s , initialisée à 0 avant la boucle, puis on divise par n **après** la boucle.

```
from random import random
def frequence(n, p):
    s = 0
    for i in range(n):
        if random() < p:
            s = s + 1
    return s / n
```

Sur 2 000 pièces à 3 % de défauts, on attend en moyenne

$$n \times p = 2\,000 \times 0,03 = 60 \text{ défauts}$$

soit une fréquence proche de 0,03. Chaque exécution donnera pourtant une valeur différente, par exemple 0,028, 0,031 ou 0,034 : c'est la **fluctuation d'échantillonnage**.

Réponse. Une valeur proche de 0,03 (environ 60 défauts sur 2 000 pièces), différente à chaque exécution.

Correction de l'exercice 4 - Quatre épreuves indépendantes

On répète quatre fois le test et on compte les réussites.

```
from random import random
def quatre():
    s = 0
    for i in range(4):
        if random() < 0.25:
            s = s + 1
    return s
```

La fonction renvoie un entier de 0 à 4. Pour n épreuves indépendantes de probabilité de succès p , le nombre moyen de succès vaut $n \times p$:

$$E(X) = 4 \times 0,25 = 1$$

En moyenne, une réussite sur quatre tests. Pourtant, aucune réussite arrive avec la probabilité $0,75^4 \approx 0,316$: près d'un tiers des séries n'en contient aucune.

Réponse. Une fonction renvoyant un entier de 0 à 4 ; en moyenne 1 réussite.

Correction de l'exercice 5 - Au moins un défaut

a) On contrôle les 10 pièces une à une, et on renvoie *True* **dès** qu'un défaut apparaît ; *return False* n'est atteint que si la boucle se termine sans défaut.

```
from random import random
def au_moins_un():
    for i in range(10):
        if random() < 0.02:
            return True
    return False
```

b) L'événement contraire est « aucune pièce défectueuse ». Chaque pièce est saine avec la probabilité 0,98, et les pièces sont indépendantes :

$$P(\text{au moins un}) = 1 - 0,98^{10} \approx 1 - 0,8171 = 0,1829$$

Environ 18 % des prélèvements contiennent au moins un défaut ; sur 1 000 appels, la simulation doit renvoyer *True* environ 180 fois. Calculer $10 \times 0,02 = 0,2$ serait faux : c'est le nombre **moyen** de défauts, pas une probabilité.

Réponse. b) $1 - 0,98^{10} \approx 0,183$, soit environ 18 %.

Correction de l'exercice 6 - Simuler un gain

a) On tire **un seul** nombre au hasard dans $[0; 1[$, puis on regarde dans quel intervalle il tombe : $[0; 0,7[$ (longueur 0,7), $[0,7; 0,95[$ (longueur 0,25) ou $[0,95; 1[$ (longueur 0,05). Ces longueurs sont les probabilités voulues, et leur somme vaut 1.

```
from random import random
def gain():
    x = random()
    if x < 0.7:
        return 0
    elif x < 0.95:
        return 3
    else:
        return 10
```

b) L'espérance est la somme des gains pondérés par leurs probabilités :

$$E(X) = 0 \times 0,7 + 3 \times 0,25 + 10 \times 0,05 = 0,75 + 0,5 = 1,25 \text{ €}$$

La moyenne de 10 000 parties simulées doit s'approcher de 1,25 €. Tirer un nouveau nombre à chaque test fausserait les probabilités.

Réponse. b) $E(X) = 1,25 \text{ €}$: la moyenne simulée doit s'en approcher.

Correction de l'exercice 7 - Estimer une probabilité inconnue

a) Sur un grand nombre de lancers, la fréquence observée estime la probabilité :

$$P(6) \approx \frac{9\,730}{50\,000} = 0,1946 \approx 0,195$$

b) Pour un dé équilibré, on attendrait $\frac{1}{6} \approx 0,1667$, soit environ $50\,000 \times \frac{1}{6} \approx 8\,333$ fois la face 6. On en observe 1 397 de plus, un écart de près de 0,028 en fréquence.

Sur 50 000 lancers, la fluctuation d'échantillonnage est très faible : un tel écart n'est pas dû au hasard, le dé favorise le 6. Sur 60 lancers seulement, le même écart n'aurait rien prouvé.

Réponse. a) $P(6) \approx 0,195$ · b) non : c'est nettement au-dessus de $\frac{1}{6} \approx 0,167$, sur un très grand nombre de lancers.

Correction de l'exercice 8 - L'effet du nombre de simulations

On mesure l'écart entre chaque fréquence et la probabilité 0,4 :

Simulations	50	500	50 000
Écart	0,08	0,026	0,0013

L'écart diminue nettement quand le nombre de simulations augmente : il est divisé par plus de soixante entre 50 et 50 000 simulations. C'est la règle observée en cours : la fréquence se rapproche de la probabilité quand l'échantillon grandit.

$$|f - p| : 0,08 \rightarrow 0,026 \rightarrow 0,0013$$

Pour estimer une probabilité inconnue, on simule donc le plus grand nombre d'expériences que le temps de calcul permet. Relancer une simulation courte jusqu'à obtenir le résultat qui arrange n'a aucune valeur.

Réponse. L'écart passe de 0,08 à 0,0013 : pour estimer une probabilité, on simule le plus grand nombre possible d'expériences.

Correction de l'exercice 9 - Compléter une simulation

Ligne 1. « Au moins une réussite » signifie que le nombre de réussites s de la série vaut 1 ou plus : *if s >= 1*: (ou *if s > 0*:). Écrire *if s == 4*: compterait seulement les séries entièrement réussies.

Ligne 2. La fréquence cherchée est le nombre de séries retenues divisé par le nombre de séries simulées : *return c / nb*. Diviser par 4 confondrait le nombre de tests d'une série avec le nombre de séries.

Contrôle : pour $p = 0,25$, la probabilité d'au moins une réussite vaut

$$1 - 0,75^4 \approx 0,684$$

et la fréquence simulée doit s'en approcher.

Réponse. ligne 1 : *if s >= 1* : · ligne 2 : *return c / nb*.

Correction de l'exercice 10 - Corriger une simulation

L'initialisation $s = 0$ est placée **dans** la boucle : le compteur est remis à zéro à chaque tour. À la fin, s ne contient que le résultat du dernier tirage, 0 ou 1, et la fonction renvoie 0 ou $\frac{1}{n}$.

Un accumulateur s'initialise **avant** la boucle, qu'il s'agisse d'une somme, d'un produit, d'un compteur ou d'un extremum.

```
from random import random
def frequence(n, p):
    s = 0
```

```

for i in range(n):
    if random() < p:
        s = s + 1
return s / n

```

Test de contrôle : *frequence(10000, 0.5)* doit renvoyer une valeur proche de 0,5, ce que la version fautive ne fait jamais.

Réponse. L'initialisation $s = 0$ doit être placée **avant** la boucle.

Correction de l'exercice 11 - Deux dés

Chaque dé est tiré **séparément** par un appel à *randint(1, 6)* ; on compte les lancers dont la somme vaut 7.

```

from random import randint
def freq_sept(nb):
    c = 0
    for k in range(nb):
        if randint(1, 6) + randint(1, 6) == 7:
            c = c + 1
    return c / nb

```

Écrire $2 * \text{randint}(1, 6)$ serait faux : cela ne donnerait que des sommes paires.

Calcul exact. Les $6 \times 6 = 36$ couples sont équiprobables, et 6 donnent la somme 7 : (1 ; 6), (2 ; 5), (3 ; 4), (4 ; 3), (5 ; 2), (6 ; 1).

$$P = \frac{6}{36} = \frac{1}{6} \approx 0,167$$

Sur 60 000 lancers, on attend environ 10 000 sommes égales à 7, la somme la plus probable.

Réponse. $P = \frac{1}{6} \approx 0,167$; la simulation doit s'en approcher.

Correction de l'exercice 12 - Compter jusqu'au premier défaut

On ne sait pas d'avance combien de pièces il faudra contrôler : on utilise une boucle *while*, qui contrôle une pièce de plus à chaque tour et s'arrête au premier défaut.

```

from random import random
def premier_defaut(p):
    n = 0
    trouve = False
    while trouve == False:
        n = n + 1
        if random() < p:
            trouve = True
    return n

```

La fonction renvoie le rang de la première pièce défectueuse : 1, 2, 3... sans borne supérieure. Avec $p = 0,05$, on rencontre en moyenne un défaut pour

$$\frac{1}{0,05} = 20 \text{ pièces}$$

ce qui donne l'ordre de grandeur attendu. Si $p = 0$, la boucle ne s'arrêterait jamais : il faut ajouter une sécurité, par exemple $n < 100000$.

Réponse. Une boucle *while* qui compte les pièces jusqu'au premier défaut ; environ 20 pièces en moyenne pour $p = 0,05$.

Correction de l'exercice 13 - Simuler pour décider

Simulation du contrat A : le coût fixe vaut 900 €, et l'on ajoute 1 500 € avec la probabilité 0,3.

```
from random import random

def cout_a():
    c = 900
    if random() < 0.3:
        c = c + 1500
    return c
```

Comparaison par le calcul. Le coût A vaut 900 € avec la probabilité 0,7 et 2 400 € avec la probabilité 0,3 :

$$E(\text{coût A}) = 900 \times 0,7 + 2\,400 \times 0,3 = 630 + 720 = 1\,350 \text{ €}$$

Le contrat B coûte 1 500 € de façon certaine. En moyenne, A est moins cher ; mais B supprime le risque de payer 2 400 € une mauvaise année. Une simulation sur 10 000 années donnerait une moyenne proche de 1 350 €, avec environ 30 % d'années à 2 400 €.

Réponse. $E(\text{coût A}) = 1\,350 \text{ €}$ contre 1 500 € pour B : A est meilleur en moyenne, B supprime le risque.

Correction de l'exercice 14 - Une règle d'alerte simulée

a) Le nombre moyen de rebuts dans un lot de n pièces vaut $n \times p$:

$$E(X) = 250 \times 0,04 = 10 \text{ rebuts par lot}$$

b) Deux boucles imbriquées : l'extérieure parcourt les 1 000 lots, l'intérieure les 250 pièces d'un lot. Le compteur s du lot est remis à zéro à **chaque nouveau lot**, donc dans la boucle extérieure.

```
from random import random

def lots_au_dessus():
    c = 0
    for k in range(1000):
        s = 0
        for i in range(250):
            if random() < 0.04:
                s = s + 1
        if s > 15:
            c = c + 1
    return c
```

Le programme effectue 250 000 tirages. Un résultat de 58 signifierait que 5,8 % des lots dépassent 15 rebuts : de quoi fixer un seuil d'alerte qui ne se déclenche pas trop souvent.

Réponse. a) 10 rebuts par lot · b) deux boucles imbriquées, le compteur du lot étant remis à zéro à chaque lot.

Correction de l'exercice 15 - Lire un résultat de simulation

a) La fréquence des lots recontrôlés vaut $\frac{47}{1\,000} = 0,047$, soit 4,7 %. C'est cohérent : on attend en moyenne $200 \times 0,05 = 10$ défauts par lot, et dépasser 15 reste rare.

b) Le coût de recontrôle vaut 120 € avec une probabilité estimée à 0,047, et 0 € sinon :

$$120 \times 0,047 + 0 \times 0,953 = 5,64 \text{ € par lot}$$

Sur 1 200 lots par an, il faut prévoir $1\,200 \times 5,64 = 6\,768 \text{ €}$ de recontrôles. Ce budget

repose sur une **fréquence simulée**, pas sur une probabilité exacte : une autre simulation donnerait un résultat un peu différent.

Réponse. a) 0,047, soit 4,7 % · b) 5,64 € par lot, soit 6 768 € pour 1 200 lots.

Correction de l'exercice 16 - Synthèse : monter une campagne de simulation

a) Une soudure est bonne avec la probabilité $1 - 0,03 = 0,97$. Les six soudures étant indépendantes, on multiplie :

$$P(\text{accepté}) = 0,97^6 \approx 0,8330$$

b) Le refus est l'événement contraire : $P(\text{refusé}) \approx 1 - 0,8330 = 0,1670$, environ 16,7 %.

c) Pour chaque assemblage, un indicateur *refuse* passe à *True* dès qu'une soudure est défectueuse ; on compte les assemblages refusés.

```
from random import random
def freq_refus():
    c = 0
    for k in range(10000):
        refuse = False
        for i in range(6):
            if random() < 0.03:
                refuse = True
        if refuse == True:
            c = c + 1
    return c / 10000
```

On attend une valeur proche de 0,167, soit environ 1 670 refus.

d) Le coût moyen vaut $85 \times 0,167 \approx 14,20$ € par assemblage, soit environ 56 800 € pour 4 000 assemblages (668 refus à 85 €). Avec 1 % de défaut par soudure, le refus tomberait à $1 - 0,99^6 \approx 0,0585$, soit environ 19 890 € : près de 36 900 € d'économie, ce qui montre qu'agir sur la qualité de chaque soudure vaut mieux que trier en fin de ligne.

Réponse. a) $0,97^6 \approx 0,833$ · b) $\approx 0,167$ · c) deux boucles imbriquées et un indicateur de refus · d) environ 14,20 € par assemblage, soit près de 56 800 € sur 4 000.