Plaquette 4 – Simulations aléatoires et statistiques

Algorithmique et Python — Seconde

Corrections détaillées

Boîte à outils

Les briques du hasard.

Instruction	Effet
from random import random	importe le générateur décimal
from random import randint	importe le générateur entier
random()	un décimal au hasard dans $[0\,;1[$
randint(a, b)	un entier au hasard de a à b **inclus**
random() < p	vrai avec la probabilité p

Simuler une expérience de probabilité p. On écrit *if random() < p:* — la condition est vraie exactement avec la probabilité p, puisque *random()* est uniforme sur $[0\,;1[$.

Structure d'une estimation de probabilité.

```
def estime(n):
    succes = 0
    for i in range(n):
        # une répétition de l'expérience
        if condition:
            succes = succes + 1
    return succes / n
```

La valeur renvoyée est une **fréquence**, et non un effectif :

$$f = \frac{\text{nombre de succès}}{n}$$

Précision d'une simulation. D'après la loi des grands nombres, quand n est grand, la fréquence obtenue est, sauf exception, proche de la probabilité. Les simulations montrent qu'il faut environ 100 fois plus de répétitions pour gagner un chiffre :

n	**Écart habituel entre f et p**
100	quelques centièmes à 0,1
10 000	quelques millièmes à 0,01
1 000 000	moins de 0,001

Indicateurs au fil de la simulation, sans stocker les résultats.

Indicateur	Calcul
moyenne	accumulateur s, puis s / n
minimum, maximum	variables mises à jour à chaque tour
fréquence d'une valeur	compteur c, puis c / n

Deux réflexes de contrôle.

— Comparer le résultat simulé à la valeur **théorique** quand on la connaît.

mathlvl.fr — Tous droits réservés. Toute reproduction, diffusion, revente ou exploitation commerciale, même partielle, est strictement interdite.
Tout contrevenant s'expose à des poursuites en vertu du Code de la propriété intellectuelle (art. L.335-2 et suivants).

— Relancer le programme : un résultat qui ne **varie pas** du tout révèle un bug.

Correction de l’exercice 1 – Simuler un dé

randint(1, 6) simule un lancer de dé ; on compte les 6 et l’on divise par le nombre de lancers pour obtenir une **fréquence**.

```
from random import randint

def frequenceSix(n):
    c = 0
    for i in range(n):
        if randint(1, 6) == 6:
            c = c + 1
    return c / n
```

La probabilité d’obtenir 6 est $\frac{1}{6} \approx 0{,}167$: sur 6000 lancers, on attend environ 1000 six. La fréquence observée fluctue autour de cette valeur, avec un écart de l’ordre de

$$\frac{1}{\sqrt{6000}} \approx 0{,}013$$

On obtiendra donc typiquement une fréquence entre 0,15 et 0,18, différente à chaque exécution. Trois erreurs fréquentes : *randint(1, 5)* ne donne jamais 6 ; *randint(0, 6)* simule un dé à 7 faces ; renvoyer *c* au lieu de *c / n* donne un effectif (environ 1000) et non une fréquence.

Réponse. La fréquence obtenue doit être proche de $\frac{1}{6} \approx 0{,}167$, soit environ 1000 six sur 6000 lancers.

Correction de l’exercice 2 – Somme de deux dés

Il faut simuler **deux** dés, donc appeler deux fois *randint*. Doubler un seul dé donnerait toujours une somme paire, donc jamais 7.

```
from random import randint

def estimeSept(n):
    c = 0
    for i in range(n):
        if randint(1, 6) + randint(1, 6) == 7:
            c = c + 1
    return c / n
```

Pour la valeur théorique, on dénombre les $6 \times 6 = 36$ couples équiprobables. Ceux de somme 7 sont $(1\,;6)$, $(2\,;5)$, $(3\,;4)$, $(4\,;3)$, $(5\,;2)$ et $(6\,;1)$:

$$P(\text{somme} = 7) = \frac{6}{36} = \frac{1}{6} \approx 0{,}167$$

C’est la somme la plus probable : les sommes 2 et 12 n’ont qu’un couple chacune, soit $\frac{1}{36}$. Avec 10 000 simulations, la fréquence obtenue s’écarte en général de moins de 0,01 de 0,167.

Réponse. La probabilité théorique est $\frac{6}{36} = \frac{1}{6} \approx 0{,}167$.

mathlvl.fr — Tous droits réservés. Toute reproduction, diffusion, revente ou exploitation commerciale, même partielle, est strictement interdite.
Tout contrevenant s’expose à des poursuites en vertu du Code de la propriété intellectuelle (art. L.335-2 et suivants).

Correction de l’exercice 3 – Tirage dans une urne

random() renvoie un décimal au hasard dans $[0\,;1[$: la condition *random() < 0.7* est vraie avec la probabilité 0,7, celle de tirer une rouge. Avec remise, les deux tirages se font dans la même urne, et l’on veut que les **deux** soient rouges.

```
from random import random

def deuxRouges(n):
    c = 0
    for i in range(n):
        if random() < 0.7 and random() < 0.7:
            c = c + 1
    return c / n
```

On numérote les 10 boules : avec remise, il y a $10 \times 10 = 100$ tirages équiprobables, dont $7 \times 7 = 49$ donnent deux rouges.

$$P = \frac{49}{100} = 0{,}49$$

Sans remise, il ne reste que 9 boules au second tirage : $\frac{7 \times 6}{10 \times 9} = \frac{42}{90} \approx 0{,}467$, un peu moins. Attention à l’écriture *if random() < 0.7 and 0.7* : elle ne fait qu’un tirage, car Python considère le nombre 0,7 comme toujours vrai.

Réponse. $0{,}7^2 = 0{,}49$ avec remise (contre $\frac{42}{90} \approx 0{,}467$ sans remise).

Correction de l’exercice 4 – Estimer π par Monte-Carlo

On tire des points au hasard dans le carré de côté 1 et l’on compte ceux qui tombent dans le quart de disque de centre l’origine et de rayon 1, c’est-à-dire ceux qui vérifient $x^2 + y^2 < 1$.

```
from random import random

def estimePi(n):
    dedans = 0
    for i in range(n):
        x = random()
        y = random()
        if x * x + y * y < 1:
            dedans = dedans + 1
    return 4 * dedans / n
```

La proportion de points dedans s’approche du rapport des aires : le carré a une aire de 1, le quart de disque une aire de $\frac{\pi \times 1^2}{4}$. Donc

$$\frac{\text{dedans}}{n} \approx \frac{\pi}{4} \quad \text{d'où} \quad \pi \approx 4 \times \frac{\text{dedans}}{n}$$

Avec 10 000 points, on obtient environ 7854 points dedans et une estimation proche de 3,14, à quelques centièmes près. Sans le facteur 4, on obtiendrait 0,785 ; avec la condition *x + y < 1*, on compterait les points d’un triangle d’aire 0,5 et l’on trouverait environ 2. La méthode est très lente : pour trois décimales fiables, il faudrait des millions de points.

Réponse. $\pi \approx 4 \times \frac{\text{points dedans}}{n}$, car la probabilité de tomber dans le quart de disque vaut $\frac{\pi}{4}$.

mathlvl.fr — Tous droits réservés. Toute reproduction, diffusion, revente ou exploitation commerciale, même partielle, est strictement interdite.
Tout contrevenant s’expose à des poursuites en vertu du Code de la propriété intellectuelle (art. L.335-2 et suivants).

Correction de l’exercice 5 – Marche aléatoire

À chaque pas, *random() < 0.5* décide de la direction, avec une chance sur deux pour chacune.

```
from random import random

def marche(n):
    x = 0
    for i in range(n):
        if random() < 0.5:
            x = x + 1
        else:
            x = x - 1
    return x
```

Les deux directions jouent des rôles symétriques : en moyenne sur un grand nombre de marches, la position finale vaut

$$0$$

Cela ne veut pas dire que le pion finit souvent en 0 : en relançant le programme, on constate qu’il ne revient exactement en 0 qu’environ 8 fois sur 100, et qu’il termine le plus souvent entre -20 et 20. La position finale est toujours **paire** : si l’on fait a pas vers la droite et $100 - a$ vers la gauche, on arrive en $a - (100 - a) = 2a - 100$. Atteindre 100 demanderait 100 pas à droite de suite : c’est possible en théorie, mais jamais observé en pratique.

Réponse. La position moyenne attendue est 0, mais l’éloignement typique est de l’ordre de $\sqrt{100} = 10$.

Correction de l’exercice 6 – Pile ou face : plus longue série

On utilise deux variables : *courant* mesure la série de piles en cours, *record* garde la plus longue vue jusqu’ici.

```
from random import random

def plusLongueSerie(n):
    record = 0
    courant = 0
    for i in range(n):
        if random() < 0.5:
            courant = courant + 1
            if courant > record:
                record = courant
        else:
            courant = 0
    return record
```

Un face remet *courant* à 0 ; un pile l’allonge, et l’on met à jour le record aussitôt. Sur la suite *PPPFFPP*, la série courante finit à 2, mais le record vaut 3 : si l’on ne comparait qu’à la fin, on perdrait ce record.

En lançant le programme de nombreuses fois avec $n = 50$, on obtient le plus souvent 4 ou 5, parfois 6 ou 7, et une série de 10 dans environ 2 % des cas. Sans la remise à zéro, la fonction compterait tous les piles, soit environ

$$\frac{50}{2} = 25$$

Les longues séries surprennent : quand on invente une suite de pile ou face « au

mathlvl.fr — Tous droits réservés. Toute reproduction, diffusion, revente ou exploitation commerciale, même partielle, est strictement interdite.
Tout contrevenant s’expose à des poursuites en vertu du Code de la propriété intellectuelle (art. L.335-2 et suivants).

hasard », on fait des séries trop courtes, et c'est ainsi qu'un professeur repère une suite inventée.

Réponse. Le plus souvent 4 ou 5 pour 50 lancers ; une série de 10 piles est rare (environ 2 % des simulations).

Correction de l'exercice 7 – Indicateurs d'une simulation

On n'a pas besoin de garder les 1000 sommes : un accumulateur *s* donne la moyenne, et deux variables *mini* et *maxi*, mises à jour à chaque lancer, donnent les extrêmes. On les initialise avec des valeurs que la première somme remplacera forcément.

```
from random import randint
```

```
s = 0
mini = 13
maxi = 1
for i in range(1000):
    somme = randint(1, 6) + randint(1, 6)
    s = s + somme
    if somme < mini:
        mini = somme
    if somme > maxi:
        maxi = somme
print(s / 1000, mini, maxi, maxi - mini)
```

Un dé donne en moyenne $\frac{1+2+3+4+5+6}{6} = 3{,}5$, donc la somme de deux dés vaut en moyenne

$$3{,}5 + 3{,}5 = 7$$

En relançant, on obtient des moyennes comme 6,93 ou 7,08, qui s'écartent de 7 de quelques centièmes seulement. Les sommes possibles vont de 2 à 12. Les sommes 2 et 12 ont chacune une probabilité de $\frac{1}{36}$, soit environ 28 apparitions sur 1000 lancers : il est pratiquement certain qu'elles apparaissent, et l'étendue vaut 10. Avec 20 lancers seulement, elle serait souvent plus petite.

Réponse. Moyenne proche de 7, minimum 2 et maximum 12 (étendue 10) presque à coup sûr.

Correction de l'exercice 8 – Réunion de deux événements

$A \cup B$ est réalisé quand le résultat est pair **ou** supérieur ou égal à 5 : on traduit ce « ou » par *or*.

```
from random import randint
```

```
def reunion(n):
    c = 0
    for i in range(n):
        d = randint(1, 6)
        if d % 2 == 0 or d >= 5:
            c = c + 1
    return c / n
```

Première façon : on liste les issues favorables. $A = \{2\,;4\,;6\}$ et $B = \{5\,;6\}$, donc $A \cup B = \{2\,;4\,;5\,;6\}$, soit 4 issues sur 6.

Deuxième façon : avec la formule de la réunion, sachant que $A \cap B = \{6\}$,

mathlvl.fr — Tous droits réservés. Toute reproduction, diffusion, revente ou exploitation commerciale, même partielle, est strictement interdite.
Tout contrevenant s'expose à des poursuites en vertu du Code de la propriété intellectuelle (art. L.335-2 et suivants).

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) = \frac{3}{6} + \frac{2}{6} - \frac{1}{6} = \frac{4}{6} = \frac{2}{3}$$

Le 6 appartient aux deux événements : sans le terme $-P(A \cap B)$, on le compterait deux fois et l'on trouverait $\frac{5}{6}$. Avec 10 000 lancers, la fréquence obtenue est proche de 0,667, à 0,01 près environ. En remplaçant *or* par *and*, on estimerait $P(A \cap B) = \frac{1}{6}$.

Réponse. $P(A \cup B) = \frac{4}{6} = \frac{2}{3} \approx 0{,}667$.

Correction de l'exercice 9 – Attendre deux « face » consécutifs

Il suffit de se souvenir du lancer **précédent**, dans une variable booléenne *precedent* : la partie s'arrête quand le lancer actuel et le précédent sont tous deux « face ».

```
from random import random
```

```
def attente():
    n = 0
    precedent = False
    while True:
        n = n + 1
        face = random() < 0.5
        if face and precedent:
            return n
        precedent = face
```

La boucle *while True* ne s'arrête que par le *return*. Au premier lancer, *precedent* vaut *False*, si bien qu'on ne peut pas s'arrêter avant le deuxième lancer : la fonction renvoie toujours au moins 2.

Les moyennes se stabilisent autour de 6 quand le nombre d'expériences augmente : on conjecture qu'il faut en moyenne 6 lancers pour obtenir deux « face » consécutifs, alors que la probabilité d'obtenir « face, face » sur deux lancers donnés vaut

$$\frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$$

On pourrait croire qu'il suffit de 4 lancers en moyenne ; ce n'est pas le cas, car les essais successifs se chevauchent et un « pile » oblige à repartir de zéro.

Réponse. Variable *precedent* pour retenir le lancer d'avant ; la moyenne semble tendre vers 6 lancers.

Correction de l'exercice 10 – Jeu équitable ?

On accumule le gain **net** de chaque partie : $8 - 2 = 6$ € si l'on gagne, -2 € sinon.

```
from random import randint
```

```
def gainMoyen(n):
    total = 0
    for i in range(n):
        s = randint(1, 6) + randint(1, 6)
        if s > 9:
            total = total + 6
        else:
            total = total - 2
    return total / n
```

Parmi les 36 couples équiprobables, la somme vaut 10 pour 3 couples, 11 pour 2 et 12

mathlvl.fr — Tous droits réservés. Toute reproduction, diffusion, revente ou exploitation commerciale, même partielle, est strictement interdite.
Tout contrevenant s'expose à des poursuites en vertu du Code de la propriété intellectuelle (art. L.335-2 et suivants).

pour 1 : on gagne dans 6 cas sur 36, soit avec la probabilité $\frac{1}{6}$. Sur 6 parties, on gagne donc en moyenne une fois et l'on perd cinq fois :

$$\frac{6 \times 1 - 2 \times 5}{6} = -\frac{4}{6} \approx -0{,}67 \text{ €}$$

Le jeu est **défavorable** : on perd en moyenne 0,67 € par partie, et le programme donne une valeur proche de $-0{,}67$. Pour un jeu équitable, il faudrait que le gain net $g - 2$ d'une victoire compense les cinq pertes de 2 €, soit $g - 2 = 10$, donc un gain de $g = 12$ €.

Réponse. Gain moyen théorique $= 6 \times \frac{1}{6} - 2 \times \frac{5}{6} \approx -0{,}67$ € : le jeu est défavorable (il faudrait 12 € de gain).

Correction de l'exercice 11 – Estimer une aire

Le carré a une aire de 1 : la proportion de points tirés au hasard qui tombent dans la région est donc une estimation de son aire.

```
from random import random

def aire(n):
    dedans = 0
    for i in range(n):
        x = random()
        y = random()
        if y < x * x:
            dedans = dedans + 1
    return dedans / n
```

Un point $(x\,;y)$ est dans la région quand il est **sous** la parabole d'équation $y = x^2$. Avec 10 000 points, on obtient environ 3333 points dedans, donc

$$\text{aire} \approx 0{,}333$$

La valeur exacte est $\frac{1}{3}$; on saura la calculer en Terminale. **Contrôle** de vraisemblance : la région est contenue dans le triangle situé sous la droite $y = x$, d'aire $\frac{1}{2}$, puisque $x^2 \leq x$ sur $[0\,;1]$. D'ailleurs, avec la condition $y < x$, le programme donnerait environ 0,5. L'intérêt de la méthode est qu'elle s'applique à des régions dont on ne sait pas calculer l'aire.

Réponse. La fréquence obtenue est proche de 0,333 : l'aire sous la parabole vaut environ le tiers du carré.

Correction de l'exercice 12 – Comparer deux stratégies

Une même fonction sert pour les deux stratégies : elle simule n parties et renvoie le gain moyen.

```
from random import random

def gain(n, p, montant):
    total = 0
    for i in range(n):
        if random() < p:
            total = total + montant
    return total / n

print(gain(100000, 0.4, 10))
print(gain(100000, 0.7, 5))
```

mathlvl.fr — Tous droits réservés. Toute reproduction, diffusion, revente ou exploitation commerciale, même partielle, est strictement interdite.
Tout contrevenant s'expose à des poursuites en vertu du Code de la propriété intellectuelle (art. L.335-2 et suivants).

Sur 10 parties, A gagne en moyenne 4 fois 10 €, et B 7 fois 5 € :

$$\text{A} : 0{,}4 \times 10 = 4 \text{ €} \qquad \text{B} : 0{,}7 \times 5 = 3{,}50 \text{ €}$$

La stratégie A rapporte donc 0,50 € de plus par partie en moyenne, bien que B gagne plus souvent. Avec 100 000 parties, la simulation sépare nettement les deux valeurs ; avec 100 parties seulement, la fluctuation pourrait masquer l'écart. A est cependant plus **risquée** : ses gains valent 0 ou 10 €, alors que B gagne plus régulièrement. Sur une seule partie, B est plus sûre ; sur un grand nombre de parties, A rapporte davantage.

Réponse. A rapporte $0{,}4 \times 10 = 4$ € en moyenne contre $0{,}7 \times 5 = 3{,}50$ € pour B : A est meilleure, mais plus risquée.

Correction de l'exercice 13 – Fluctuation d'échantillonnage

Pour chaque échantillon, on lance 100 fois la pièce, on calcule la fréquence de pile, puis on teste si elle sort de $[0{,}4\,;0{,}6]$.

```
from random import random

def sorties(repetitions, n):
    c = 0
    for k in range(repetitions):
        piles = 0
        for i in range(n):
            if random() < 0.5:
                piles = piles + 1
        f = piles / n
        if f < 0.4 or f > 0.6:
            c = c + 1
    return c
```

Les simulations de 1 000 échantillons de 100 lancers montrent que la fréquence de pile reste dans $[0{,}4\,;0{,}6]$ dans environ 96 % des cas. On peut donc s'attendre à ce qu'environ

$$0{,}04 \times 200 = 8$$

échantillons sur 200 sortent de cet intervalle, le nombre exact variant d'une exécution à l'autre (souvent entre 3 et 14). Un résultat comme 80 sorties sur 200 révélerait une erreur dans le programme, par exemple une division oubliée. Avec des échantillons de taille 400, les fréquences sont bien plus resserrées autour de 0,5 : presque aucune ne sort de $[0{,}4\,;0{,}6]$.

Réponse. Quelques sorties sur 200, de l'ordre de 8 (environ 4 % des échantillons).

Correction de l'exercice 14 – Tirage sans remise

La première boule a est tirée au hasard parmi 1 à 5. Pour la seconde, il ne reste que 4 boules : on tire un entier b entre 1 et 4, puis on « saute » le numéro déjà tiré en ajoutant 1 quand $b \geq a$. Par exemple, si $a = 3$, les valeurs $1, 2, 3, 4$ de b deviennent $1, 2, 4, 5$: ce sont exactement les quatre boules restantes, chacune avec la même probabilité. On simule ainsi le tirage sans remise sans représenter l'urne.

Pour la valeur théorique, on dénombre les 10 paires de boules : $\{1,2\}$, $\{1,3\}$, $\{1,4\}$, $\{1,5\}$, $\{2,3\}$, $\{2,4\}$, $\{2,5\}$, $\{3,4\}$, $\{3,5\}$, $\{4,5\}$. La somme est paire quand les deux numéros ont la même parité : $\{2,4\}$, $\{1,3\}$, $\{1,5\}$, $\{3,5\}$, soit

$$P = \frac{4}{10} = 0{,}4$$

Contrôle : les 6 autres paires mélangent un pair et un impair, et $4 + 6 = 10$. Avec remise, sur les 25 tirages possibles, $3 \times 3 + 2 \times 2 = 13$ donnent une somme paire, soit 0,52 :

mathlvl.fr — Tous droits réservés. Toute reproduction, diffusion, revente ou exploitation commerciale, même partielle, est strictement interdite.
Tout contrevenant s'expose à des poursuites en vertu du Code de la propriété intellectuelle (art. L.335-2 et suivants).

l'écart entre les deux situations est bien visible en simulation.

Réponse. Le décalage de b évite de retirer la boule a ; probabilité 0,4 (contre 0,52 avec remise).

Correction de l'exercice 15 – Temps d'attente

On ne sait pas combien de lancers il faudra : on lance **tant que** le résultat n'est pas 6. On initialise *resultat* à 0 pour que la boucle démarre.

```
from random import randint

def attente():
    n = 0
    resultat = 0
    while resultat != 6:
        resultat = randint(1, 6)
        n = n + 1
    return n
```

Pour estimer la moyenne, on appelle *attente()* un grand nombre de fois et l'on fait la moyenne des valeurs obtenues ; on trouve un nombre proche de

$$6 \text{ lancers}$$

C'est cohérent : le 6 sort en moyenne une fois tous les 6 lancers. Pourtant, la valeur la plus fréquente est 1 : environ 17 % des parties s'arrêtent au premier lancer. Les valeurs sont très étalées : la moitié des parties se terminent en 4 lancers ou moins, mais environ 16 % dépassent 10 lancers, et quelques-unes en demandent plus de 20. Ces attentes très longues tirent la moyenne vers le haut.

Réponse. Une moyenne proche de 6 lancers, alors que le cas le plus fréquent est d'obtenir le 6 dès le premier lancer.

Correction de l'exercice 16 – Synthèse : simuler un contrôle qualité

a) Chaque pièce est défectueuse avec la probabilité p : on fait n tirages et l'on compte les défauts.

```
from random import random

def lot(n, p):
    d = 0
    for i in range(n):
        if random() < p:
            d = d + 1
    return d
```

Pour notre machine, on appelle *lot(50, 0.04)*.

b) Sur 50 pièces, 4 % sont défectueuses en moyenne :

$$50 \times 0{,}04 = 2 \text{ défectueuses par lot}$$

C'est une moyenne : en relançant *lot(50, 0.04)*, on obtient 0, 1, 2, 3, parfois 5 ou plus.

c) On simule un grand nombre de lots et l'on compte ceux qui ont au moins 4 défectueuses.

```
def probaRejet(repetitions, n, p, seuil):
    c = 0
    for k in range(repetitions):
        if lot(n, p) >= seuil:
```

mathlvl.fr — Tous droits réservés. Toute reproduction, diffusion, revente ou exploitation commerciale, même partielle, est strictement interdite.
Tout contrevenant s'expose à des poursuites en vertu du Code de la propriété intellectuelle (art. L.335-2 et suivants).

```
            c = c + 1
    return c / repetitions
```

probaRejet(100000, 50, 0.04, 4) renvoie une valeur proche de 0,14 ; le calcul exact, vu en Première, donne 0,139.

d) Sur 1000 lots, on attend donc environ $0{,}139 \times 1000 \approx 140$ rejets, soit un lot sur sept. À 8 % de défauts, *probaRejet(100000, 50, 0.08, 4)* donne environ 0,575, soit à peu près 575 rejets. Le taux de défaut a doublé, mais les rejets sont multipliés par plus de 4 : le contrôle par seuil réagit très fortement à une dérive de la machine, ce qui est justement son but.

Réponse. b) 2 défectueuses par lot · c) environ 0,14 · d) environ 140 rejets sur 1000 lots à 4 %, contre environ 575 à 8 %.

mathlvl.fr — Tous droits réservés. Toute reproduction, diffusion, revente ou exploitation commerciale, même partielle, est strictement interdite.
Tout contrevenant s’expose à des poursuites en vertu du Code de la propriété intellectuelle (art. L.335-2 et suivants).