

Plaquette 4 - Simuler le hasard : Monte-Carlo et fréquences

Algorithmique et Python — Terminale STI2D

Corrections détaillées

Boîte à outils

Simuler une expérience aléatoire.

- `from random import random, randint` donne accès aux deux générateurs.
- `random()` renvoie un réel au hasard dans $[0 ; 1[$, chaque partie de l'intervalle ayant la même chance.
- `randint(a, b)` renvoie un entier au hasard entre a et b , **bornes comprises** — contrairement à `range`.
- **Simuler un événement de probabilité p** : la condition `random() < p` est vraie avec la probabilité exactement p .
- **Schéma de Monte-Carlo** :

```
def simulation(N):
    succes = 0
    for i in range(N):
        if experience():
            succes = succes + 1
    return succes / N
```

- **Estimation** : la fréquence obtenue approche la probabilité cherchée.

$$p \approx \frac{\text{nombre de succès}}{N}$$

- **Estimer une aire** : on tire un point au hasard dans un rectangle de surface connue et on compte ceux qui tombent sous la courbe.

$$\text{aire} \approx (\text{aire du rectangle}) \times \frac{\text{points sous la courbe}}{N}$$

- **Fluctuation d'échantillonnage** : deux exécutions ne donnent jamais le même résultat. Pour N répétitions, l'écart typique à la vraie valeur est de l'ordre de $\frac{1}{\sqrt{N}}$: il faut multiplier N par 100 pour gagner un facteur 10 de précision.
- **Estimer une moyenne** (espérance) : on additionne les valeurs obtenues et on divise par N .

Correction de l'exercice 1 - Deux générateurs à ne pas confondre

- a) `random()` renvoie un nombre **décimal** de l'intervalle $[0 ; 1[$.
- La valeur 0 est possible, la valeur 1 ne l'est pas.
- b) `randint(1, 6)` renvoie un **entier** parmi 1, 2, 3, 4, 5, 6 : les deux bornes sont incluses.
- C'est exactement la simulation d'un dé à six faces.
- c) `randint(0, 1)` renvoie 0 ou 1, chacun avec la probabilité $\frac{1}{2}$: c'est une pièce de monnaie.
- d) `2 * random()` renvoie un décimal de $[0 ; 2[$.
- **Le piège à connaître** : `randint(a, b)` inclut b , alors que `range(a, b)` s'arrête à $b - 1$. Écrire `randint(1, 5)` pour un dé donnerait un dé à cinq faces.

Réponse. a) $[0; 1[$ décimal · b) entiers de 1 à 6 · c) 0 ou 1 · d) $[0; 2[$ décimal.

Correction de l'exercice 2 - Simuler un événement de probabilité donnée

```
from random import random
```

```
def piece():
    return random() < 0.03
```

Justification. *random()* tire un nombre au hasard dans $[0; 1[$, de façon uniforme.

- La probabilité que ce nombre tombe dans $[0; 0,03[$ est égale à la longueur de cet intervalle, soit 0,03.
- La condition *random()* < 0.03 est donc vraie avec la probabilité 0,03 : elle simule exactement l'événement voulu.
- **Généralisation** : *random()* < *p* simule un événement de probabilité *p*, pour tout *p* de $[0; 1]$. C'est l'instruction de base de toutes les simulations du chapitre.
- **Erreur à éviter** : *random()* == 0.03 serait pratiquement toujours faux, puisqu'il faudrait tomber exactement sur cette valeur parmi une infinité de décimaux.

Réponse. *return random()* < 0.03 : la probabilité de tomber dans $[0; 0,03[$ vaut 0,03.

Correction de l'exercice 3 - Compter des succès

- Un dé a six faces : *randint(1, 6)*.
- Un nombre est pair lorsque son reste dans la division par 2 vaut 0 : *if de % 2 == 0:*.
- La fonction doit renvoyer une fréquence : *return succes / N*.

Valeur attendue. Les faces paires sont 2, 4 et 6, soit 3 faces sur 6.

- La probabilité théorique vaut $\frac{3}{6} = 0,5$.
- Pour $N = 10\,000$, la simulation donne une valeur voisine de 0,5, par exemple 0,497 ou 0,503 selon l'exécution.
- **Contrôle systématique** : quand la valeur théorique est calculable, on la calcule. Une simulation qui donnerait 0,33 signifierait une erreur de programme, pas un coup de malchance.

Réponse. *randint(1, 6), de % 2 == 0, succes / N* ; on approche 0,5.

Correction de l'exercice 4 - Simuler une loi binomiale

a) Il faut **deux** boucles imbriquées : la boucle interne simule les 10 prélèvements d'une expérience, la boucle externe répète l'expérience *N* fois.

```
from random import random
```

```
def simulation(N):
    succes = 0
    for essai in range(N):
        retouches = 0
        for piece in range(10):
            if random() < 0.2:
                retouches = retouches + 1
        if retouches >= 3:
            succes = succes + 1
    return succes / N
```

b) *X* suit $B(10; 0,2)$ et l'on passe par l'événement contraire.

— $P(X \leq 2) \approx 0,107\,374 + 0,268\,435 + 0,301\,990 = 0,677\,800$.

$$P(X \geq 3) \approx 1 - 0,677\,800 = 0,322\,200$$

— La simulation doit donner une valeur voisine de 0,322.

— **Erreur classique** : remettre *retouches* = 0 à zéro **en dehors** de la boucle externe. Le compteur cumulerait alors toutes les expériences et la fréquence obtenue vaudrait pratiquement 1.

Réponse. b) $P(X \geq 3) \approx 0,322$.

Correction de l'exercice 5 - Estimer une aire par Monte-Carlo

a) Le carré a une aire de 1. Les points tirés se répartissent uniformément, donc la fréquence des points situés sous la courbe d'équation $y = x^2$ estime la **proportion d'aire** occupée par cette région.

— Comme l'aire du carré vaut 1, cette fréquence estime directement l'aire sous la courbe, c'est-à-dire $\int_0^1 x^2 dx$.

b) On tire les deux coordonnées séparément, puis on teste la condition.

```
from random import random
```

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

c) On calcule l'intégrale avec une primitive de x^2 , qui est $\frac{x^3}{3}$.

$$\int_0^1 x^2 dx = \left[\frac{x^3}{3} \right]_0^1 = \frac{1}{3} - 0 = \frac{1}{3}$$

— Le programme doit approcher $\frac{1}{3} \approx 0,333$.

— **Point délicat** : il faut deux appels distincts à *random()*, un pour *x* et un pour *y*. Écrire *x = y = random()* donnerait deux coordonnées égales, donc uniquement des points de la diagonale.

Réponse. a) l'aire sous la courbe · c) $\frac{1}{3} \approx 0,333$.

Correction de l'exercice 6 - Estimer le nombre pi

a) La condition $x^2 + y^2 < 1$ décrit l'intérieur du disque de centre *O* et de rayon 1. Dans le carré unité, on n'en voit que le **quart** situé dans le premier quadrant.

— Aire du quart de disque : $\frac{\pi \times 1^2}{4} = \frac{\pi}{4}$.

— Aire du carré : 1.

— La probabilité vaut donc $\frac{\pi}{4} \approx 0,785\,398$.

b) Si *f* désigne la fréquence obtenue, alors $f \approx \frac{\pi}{4}$, d'où :

$$\pi \approx 4f$$

c) On compte les points du quart de disque, puis on multiplie la fréquence par 4.

```
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
```

- Pour $N = 10\,000$, on obtient typiquement une valeur entre 3,11 et 3,17 : deux décimales exactes seulement.
- **Enseignement** : la méthode est simple et générale, mais peu précise. Elle sert surtout à estimer des aires qu'on ne sait pas calculer, pas à obtenir π avec précision.

Réponse. a) $\frac{\pi}{4} \approx 0,785$ · b) $\pi \approx 4f$ · c) compter les points vérifiant $x^2 + y^2 < 1$.

Correction de l'exercice 7 - Pourquoi deux exécutions différentes

a) **Non.** Chaque exécution tire de nouveaux nombres au hasard : il serait au contraire suspect d'obtenir trois fois exactement la même valeur.

- Ce phénomène porte un nom : la **fluctuation d'échantillonnage**.
- Les trois valeurs sont regroupées autour de 0,5, ce qui suggère que la probabilité cherchée vaut 0,5.

b) Il faut **augmenter** N , seul paramètre qui réduit la fluctuation.

- L'écart typique à la vraie valeur est de l'ordre de $\frac{1}{\sqrt{N}}$.
- Pour $N = 1\,000$: $\frac{1}{\sqrt{1\,000}} \approx 0,032$ — ce qui correspond bien à la dispersion observée.
- Pour $N = 100\,000$: $\frac{1}{\sqrt{100\,000}} \approx 0,003$, soit dix fois mieux.
- **Prix à payer** : il faut multiplier N par 100 pour gagner un facteur 10 de précision, donc 100 fois plus de calculs. C'est la limite structurelle de la méthode de Monte-Carlo.

Réponse. a) non, c'est la fluctuation d'échantillonnage · b) augmenter N ; l'écart typique est en $\frac{1}{\sqrt{N}}$.

Correction de l'exercice 8 - Choisir la taille de la simulation

On utilise l'ordre de grandeur de l'écart typique, $\frac{1}{\sqrt{N}}$.

a) On veut $\frac{1}{\sqrt{N}} \approx 0,01$.

- Donc $\sqrt{N} \approx 100$, soit $N \approx 100^2 = 10\,000$.

b) On veut $\frac{1}{\sqrt{N}} \approx 0,001$.

- Donc $\sqrt{N} \approx 1\,000$, soit $N \approx 1\,000^2 = 1\,000\,000$.
- **Ordre de grandeur à retenir** : 10 000 répétitions donnent environ deux décimales, un million en donne environ trois.
- **Comparaison utile** : lorsqu'un calcul exact est possible (loi binomiale, aire d'un domaine simple), il donne toutes les décimales en une ligne. La simulation ne se

justifie que lorsque le calcul exact est hors de portée.

Réponse. a) $N \approx 10\,000$ · b) $N \approx 1\,000\,000$.

Correction de l'exercice 9 - La somme de deux dés

```
from random import randint
```

```
def sommeSept(N):
    succes = 0
    for i in range(N):
        de1 = randint(1, 6)
        de2 = randint(1, 6)
        if de1 + de2 == 7:
            succes = succes + 1
    return succes / N
```

Valeur théorique. Deux dés donnent $6 \times 6 = 36$ résultats équiprobables.

- Ceux dont la somme vaut 7 : (1 ; 6), (2 ; 5), (3 ; 4), (4 ; 3), (5 ; 2), (6 ; 1), soit 6 cas.
- Probabilité : $\frac{6}{36} = \frac{1}{6} \approx 0,167$.
- **Point essentiel** : il faut **deux** appels à *randint*. Écrire *de = randint(1, 6)* puis *2 * de* simulerait un seul dé doublé, qui ne donne que des sommes paires — et jamais 7.
- La somme 7 est la plus probable des sommes : c'est celle qui admet le plus de décompositions.

Réponse. $\frac{1}{6} \approx 0,167$.

Correction de l'exercice 10 - Estimer un gain moyen

a) Ici on n'accumule pas des succès mais des **gains**, et on divise par N pour obtenir une moyenne.

```
from random import randint
```

```
def gainMoyen(N):
    total = 0
    for i in range(N):
        de = randint(1, 6)
        if de == 6:
            total = total + 10 - 2
        else:
            total = total - 2
    return total / N
```

b) Le gain net vaut 8 € avec la probabilité $\frac{1}{6}$ et -2 € avec la probabilité $\frac{5}{6}$.

$$E(X) = 8 \times \frac{1}{6} + (-2) \times \frac{5}{6} = \frac{8-10}{6} = -\frac{1}{3} \approx -0,33$$

- Le joueur perd en moyenne environ 0,33 € par partie : le jeu n'est pas équitable.
- La simulation doit donner une valeur proche de -0,33.
- **À retenir** : estimer une **espérance** se fait avec un accumulateur de valeurs, estimer une **probabilité** avec un compteur de succès. Le schéma de boucle est le même, mais ce qu'on additionne change.

Réponse. b) $E(X) = -\frac{1}{3} \approx -0,33$ € par partie.

Correction de l'exercice 11 – Trouver l'erreur

Diagnostic. Le compteur *succes* est incrémenté à **chaque lancer** qui donne un 6, et non une seule fois par expérience.

- Une expérience qui donne deux 6 compte pour 2, alors qu'elle ne doit compter que pour 1.
- Le programme calcule en réalité le **nombre moyen de 6** en trois lancers, soit $3 \times \frac{1}{6} = 0,5$ — exactement la valeur observée.

Correction. Il faut un indicateur propre à chaque expérience, testé après la boucle interne.

```
from random import randint

def simulation(N):
    succes = 0
    for i in range(N):
        six = 0
        for j in range(3):
            if randint(1, 6) == 6:
                six = six + 1
        if six >= 1:
            succes = succes + 1
    return succes / N
```

Valeur théorique.

$$P(\text{au moins un 6}) = 1 - \left(\frac{5}{6}\right)^3 = 1 - \frac{125}{216} = \frac{91}{216} \approx 0,421$$

- **Leçon** : une valeur simulée cohérente avec une **autre** question est le signe d'une erreur de modèle, pas d'un hasard défavorable. Toujours confronter la simulation au calcul théorique.

Réponse. Le compteur comptait chaque 6 au lieu de chaque expérience ; la valeur correcte est $\frac{91}{216} \approx 0,421$.

Correction de l'exercice 12 – Simuler un contrôle de production

```
from random import random

def refus(N):
    refuses = 0
    for essai in range(N):
        mauvaises = 0
        for piece in range(50):
            if random() < 0.06:
                mauvaises = mauvaises + 1
        if mauvaises > 4:
            refuses = refuses + 1
    return refuses / N
```

Espérance. Le nombre X de pièces non conformes suit $B(50; 0,06)$.

$$E(X) = np = 50 \times 0,06 = 3$$

- On attend 3 pièces non conformes par lot, et le seuil de refus est fixé juste au-dessus.
- La simulation donne une fréquence de refus d'environ 0,18 : près d'un lot sur cinq est refusé alors que la ligne fonctionne normalement.

- **Attention à la condition** : « plus de 4 » s'écrit *mauvaises* > 4, c'est-à-dire au moins 5. Écrire >= 4 rendrait le contrôle nettement plus sévère.

Réponse. $E(X) = 3$ pièces non conformes par lot ; le seuil de refus est à 5.

Correction de l'exercice 13 - Compter les essais jusqu'au succès

a) Le nombre d'essais n'est pas connu à l'avance : c'est une boucle *while*.

```
from random import random

def essais():
    n = 0
    reussi = False
    while not reussi:
        n = n + 1
        reussi = random() < 0.25
    return n
```

b) On accumule les résultats de la fonction précédente.

```
def moyenne(N):
    total = 0
    for i in range(N):
        total = total + essais()
    return total / N
```

- La simulation donne une valeur voisine de 4.
- **Interprétation** : avec une probabilité de succès de $\frac{1}{4}$, il faut en moyenne 4 essais. Plus généralement, le nombre moyen d'essais vaut $\frac{1}{p}$.
- **Précaution** : cette boucle *while* s'arrête toujours en pratique, mais rien ne borne le nombre d'essais. Pour un p très petit, on ajoute souvent une limite de sécurité.

Réponse. b) la moyenne obtenue vaut environ 4, soit $\frac{1}{p}$.

Correction de l'exercice 14 - Simulation ou calcul ?

a) **Calculer.** La formule du cours donne le résultat exact en une ligne :

$$P(X = 3) = 120 \times 0,4^3 \times 0,6^7$$

- Une simulation ne donnerait que deux ou trois décimales, pour bien plus de travail.
- b) **Simuler.** Dénombrer les résultats de cinq dés dont la somme dépasse 20 est fastidieux et hors programme.
- La simulation donne une bonne estimation en quelques lignes de code : c'est exactement le cas d'usage de Monte-Carlo.
- c) **Calculer.** La probabilité est directement donnée par l'énoncé : 0,02.
- Aucune simulation n'est nécessaire — la simuler reviendrait à estimer un nombre qu'on connaît déjà.
- **Règle de conduite** : la simulation ne remplace pas un calcul faisable. Elle sert quand le dénombrement est impraticable, ou pour **vérifier** un calcul déjà mené.

Réponse. a) calculer · b) simuler · c) calculer (la probabilité est donnée).

Correction de l'exercice 15 - Comparer simulation et théorie

a) On compare l'écart observé à l'ordre de grandeur de la fluctuation.

- Écart observé : $|0,329 - 0,322| = 0,007$.
- Fluctuation typique : $\frac{1}{\sqrt{10\,000}} = \frac{1}{100} = 0,01$.
- L'écart (0,007) est **inférieur** à la fluctuation typique (0,01) : il est parfaitement normal. La simulation confirme le calcul.
- b) Écart : $|0,450 - 0,322| = 0,128$, soit près de **treize fois** la fluctuation typique.
- Un tel écart n'est pas explicable par le hasard : il signale une **erreur**, soit dans le programme, soit dans le calcul théorique.
- Pistes habituelles : condition mal écrite ($\geq$ au lieu de $>$), compteur non réinitialisé, ou mauvaise valeur de p .
- **Méthode générale** : comparer l'écart à $\frac{1}{\sqrt{N}}$. Au-delà de deux ou trois fois cette valeur, on cherche une erreur plutôt qu'une explication par le hasard.

Réponse. a) oui : $0,007 < \frac{1}{\sqrt{10\,000}} = 0,01$ · b) écart treize fois trop grand : il y a une erreur.

Correction de l'exercice 16 - Synthèse : estimer un risque industriel

a) Une boucle sur les trois pompes, et un compteur de pannes.

```
from random import random

def mois():
    pannes = 0
    for pompe in range(3):
        if random() < 0.08:
            pannes = pannes + 1
    return pannes
```

b) On réutilise la fonction précédente dans le schéma de Monte-Carlo.

```
def arretTotal(N):
    succes = 0
    for i in range(N):
        if mois() == 3:
            succes = succes + 1
    return succes / N
```

c) Le nombre X de pompes en panne suit $B(3; 0,08)$. Ligne $n = 3$ du triangle de Pascal : $1 \cdot 3 \cdot 3 \cdot 1$.

— **Arrêt total** ($X = 3$) :

$$P(X = 3) = 0,08^3 = 0,000\,512$$

— **Mode dégradé** ($X = 2$) :

$$P(X = 2) = 3 \times 0,08^2 \times 0,92 = 3 \times 0,006\,4 \times 0,92 = 0,017\,664$$

— Le mode dégradé est environ 34 fois plus fréquent que l'arrêt total.

d) Nombre d'occasions dans l'année : $500 \text{ installations} \times 12 \text{ mois} = 6\,000$ mois-installations.

— Arrêts totaux attendus : $6\,000 \times 0,000\,512 = 3,072$, soit environ 3 **arrêts totaux par an**.

La simulation suffirait-elle ? Non.

- Avec $N = 1\,000$, on attend $1\,000 \times 0,000\,512 \approx 0,5$ occurrence : la plupart des exécutions renverraient 0 ou 0,001, sans aucune précision utile.
- La fluctuation typique, $\frac{1}{\sqrt{1\,000}} \approx 0,032$, est **soixante fois plus grande** que la probabilité cherchée.
- **Conclusion méthodologique** : Monte-Carlo est inadapté aux événements rares. Il faudrait ici N de l'ordre du million — alors que la formule binomiale donne la valeur exacte immédiatement. Pour un risque rare, on calcule, on ne simule pas.

Réponse. c) arrêt total 0,000 512, mode dégradé 0,017 664 · d) environ 3 arrêts totaux par an ; $N = 1\,000$ est très insuffisant pour un événement aussi rare.