

Plaquette 1 - Seuils, Monte-Carlo et marches aléatoires

Algorithmique et Python — Terminale STI2D

Corrections détaillées

Boîte à outils

Syntaxe et schémas.

- **Fonction** : `def f(x): ... return ...`. Le `return` est indispensable pour réutiliser le résultat.
- **Boucle bornée** : `for i in range(n):` ; **boucle non bornée** : `while condition:` (elle doit progresser vers l'arrêt).
- **Hasard** : `random()` donne un réel de $[0 ; 1[$; `randint(a, b)` un entier de a à b inclus.
- Un événement de probabilité p se simule par la condition `random() < p`.
- **Monte-Carlo** : on répète N fois l'expérience, on compte les succès, et on estime

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

- **Marche aléatoire** sur une droite : à chaque pas, la position augmente ou diminue de 1 avec la probabilité $\frac{1}{2}$.
- Loi binomiale (rappel) : $P(X = k) = (\text{coefficient de Pascal}) \times p^k(1 - p)^{n-k}$, et $E(X) = np$.
- Plus N est grand, plus l'estimation de Monte-Carlo est précise (fluctuation d'échantillonnage en $\frac{1}{\sqrt{N}}$).

Correction de l'exercice 1 - Dérouler une boucle while

La boucle multiplie u par 0,85 tant que u dépasse 2 000.

Tour	4	5	6
u	2 610,0	2 218,5	1 885,7

- Valeurs : $u_4 \approx 2\,610,0$; $u_5 \approx 2\,218,5$; $u_6 \approx 1\,885,7$.
- Après le 6^e tour, $u \approx 1\,885,7 \leq 2\,000$: la boucle s'arrête.
- La fonction renvoie donc 6.
- Contrôle par le calcul : $5\,000 \times 0,85^6 \approx 1\,885,7$. Cohérent.

Réponse. 6.

Correction de l'exercice 2 - Écrire un algorithme de seuil

```
def seuil():
    p = 2000
    n = 0
    while p <= 10000:
        p = p * 1.09
        n = n + 1
    return n
```

- Contrôle par le calcul : $1,09^{18} \approx 4,717$ donc $p_{18} \approx 9\,434$; $1,09^{19} \approx 5,142$ donc $p_{19} \approx 10\,283$.

- La fonction renvoie 19 : la population dépasse 10 000 au bout de 19 heures.
- On pouvait aussi résoudre avec le logarithme : $x > \frac{\log 5}{\log 1,09} \approx 18,68$.

Réponse. 19 heures.

Correction de l'exercice 3 - Simuler une probabilité simple

- Un dé a six faces : `randint(1, 6)`.
- La fonction doit renvoyer une fréquence : `return succes / N`.
- Valeur attendue : la probabilité théorique $\frac{1}{6} \approx 0,167$.
- Pour $N = 10\,000$, la simulation donne une valeur proche de 0,167, différente à chaque exécution (fluctuation d'échantillonnage).

Réponse. `randint(1, 6)` et `return succes / N` ; on approche $\frac{1}{6} \approx 0,167$.

Correction de l'exercice 4 - Simuler une loi binomiale

On imbrique deux boucles : la boucle interne simule les 25 prélèvements, la boucle externe répète l'expérience.

```
from random import random
```

```
def simulation(N):
    succes = 0
    for essai in range(N):
        defaults = 0
        for piece in range(25):
            if random() < 0.03:
                defaults = defaults + 1
        if defaults >= 1:
            succes = succes + 1
    return succes / N
```

- Valeur théorique : $P(X \geq 1) = 1 - 0,97^{25} \approx 1 - 0,467 = 0,533$.
- Pour $N = 10\,000$, la simulation donne une valeur proche de 0,533.

Réponse. La simulation approche $1 - 0,97^{25} \approx 0,533$.

Correction de l'exercice 5 - Monte-Carlo : estimer une aire

- a) Le carré a une aire de 1. La fréquence de points sous la courbe $y = x^2$ estime donc l'**aire** de la région comprise entre la courbe et l'axe des abscisses sur $[0; 1]$.
- b) On complète par x^2 , soit la condition *if* $y < x^2$.
- Valeur théorique : une primitive de x^2 est $\frac{x^3}{3}$, donc l'aire vaut $\frac{1}{3} \approx 0,333$.
- Pour $N = 100\,000$, la simulation donne une valeur proche de 0,333.
- C'est le principe de la **méthode de Monte-Carlo** : estimer une aire (ou une intégrale) par un comptage aléatoire.

Réponse. a) l'aire sous $y = x^2$ sur $[0; 1]$ · b) *if* $y < x^2$; valeur attendue $\frac{1}{3} \approx 0,333$.

Correction de l'exercice 6 - Monte-Carlo : estimer π

- a) La condition $x^2 + y^2 < 1$ décrit le **quart de disque** de rayon 1 contenu dans le carré.

- Son aire vaut $\frac{\pi \times 1^2}{4} = \frac{\pi}{4}$, et le carré a une aire de 1 : la probabilité cherchée est donc $\frac{\pi}{4} \approx 0,785$.
- b) Si f est la fréquence obtenue, alors $f \approx \frac{\pi}{4}$, donc

$$\pi \approx 4f$$
- Avec $N = 100\,000$ tirages, on obtient typiquement $\pi \approx 3,14$, avec deux décimales fiables.
- Remarque : la précision augmente comme $\frac{1}{\sqrt{N}}$; gagner une décimale exige de multiplier N par 100.

Réponse. a) $\frac{\pi}{4} \approx 0,785$ · b) $\pi \approx 4f$.

Correction de l'exercice 7 - Marche aléatoire : lecture

- À chaque pas, la position augmente ou diminue de 1, avec la probabilité $\frac{1}{2}$ chacune : c'est une **marche aléatoire** sur une droite.
- Pour $n = 4$ pas, si k est le nombre de pas « +1 », la position finale vaut $k - (4 - k) = 2k - 4$.
- Avec $k \in \{0; 1; 2; 3; 4\}$, les positions possibles sont $-4, -2, 0, 2, 4$.
- La position finale est donc toujours **paire** après un nombre pair de pas : on ne peut pas obtenir 1 ou 3.

Réponse. Une marche aléatoire ; positions possibles pour $n = 4$: $-4, -2, 0, 2, 4$.

Correction de l'exercice 8 - Marche aléatoire : loi de la position

Le nombre k de pas « +1 » suit la loi binomiale $B(4; 0,5)$, et la position finale vaut $2k - 4$.

- Terminer en 0 correspond à $k = 2$: le coefficient de Pascal est 6, donc

$$P = 6 \times 0,5^4 = 6 \times 0,0625 = 0,375$$

- Terminer en 4 correspond à $k = 4$ (quatre pas vers la droite) :
 $P = 1 \times 0,0625 = 0,0625$.
- La position 0 est donc six fois plus probable que la position extrême 4.

Réponse. $P(\text{position} = 0) = 0,375$ et $P(\text{position} = 4) = 0,0625$.

Correction de l'exercice 9 - Moyenne de simulations

On réutilise la fonction *marche* et on accumule les valeurs absolues.

```
from random import randint

def marche(n):
    position = 0
    for i in range(n):
        if randint(0, 1) == 1:
            position = position + 1
        else:
            position = position - 1
    return position

def distanceMoyenne(n, N):
    total = 0
    for essai in range(N):
```

```
total = total + abs(marche(n))
return total / N
```

- Résultats typiques : pour $n = 100$, on obtient environ 8 ; pour $n = 400$, environ 16.
- Interprétation : la distance moyenne à l'origine croît comme $\sqrt{n}$ (quadrupler le nombre de pas double la distance), et non comme n .
- La position **moyenne** (sans valeur absolue) est en revanche proche de 0, par symétrie.

Réponse. La distance moyenne croît comme $\sqrt{n}$ (≈ 8 pour $n = 100$, ≈ 16 pour $n = 400$).

Correction de l'exercice 10 - Corriger un programme

- **Erreur 1** : la fonction utilise *print* au lieu de *return* — elle n'est donc pas réutilisable dans un calcul.
- **Erreur 2** : elle renvoie l'**effectif** et non la **fréquence** : il faut diviser par N .

```
from random import random
```

```
def frequenceFace(N):
    faces = 0
    for i in range(N):
        if random() < 0.5:
            faces = faces + 1
    return faces / N
```

- Pour N grand, le résultat s'approche de 0,5.

Réponse. Remplacer *print(faces)* par *return faces / N*.

Correction de l'exercice 11 - Problème : simuler un contrôle de lot

a)

```
from random import random
```

```
def frequenceAcceptation(N):
    acceptes = 0
    for essai in range(N):
        defaults = 0
        for piece in range(10):
            if random() < 0.05:
                defaults = defaults + 1
        if defaults <= 1:
            acceptes = acceptes + 1
    return acceptes / N
```

b) Valeur théorique avec $X \sim B(10; 0,05)$:

- $P(X = 0) = 0,95^{10} \approx 0,5987$;
- $P(X = 1) = 10 \times 0,05 \times 0,95^9 \approx 0,3151$;
- $P(X \leq 1) \approx 0,914$.
- La simulation doit donc donner une fréquence proche de 0,914.

Réponse. b) $\approx 0,914$.

Correction de l'exercice 12 - Une somme d'inverses

a) Le compteur k prend les valeurs entières de 1 à n (*range(1, n + 1)* s'arrête avant $n + 1$), et l'on ajoute $\frac{1}{k}$ à chaque tour :

$$\text{somme}(n) = \sum_{k=1}^n \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

$$\text{b) } 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} = \frac{12 + 6 + 4 + 3}{12} = \frac{25}{12} \approx 2,083.$$

c) s est l'**accumulateur** : il vaut 0 au départ (initialisation) et reçoit un terme à chaque tour. k est le **compteur** : ses valeurs extrêmes, 1 et n , sont les bornes de la somme. La fonction renvoie s après le dernier tour (sortie de boucle).

Réponse. a) $\sum_{k=1}^n \frac{1}{k}$ · b) $\frac{25}{12} \approx 2,083$ · c) s accumule, k compte.

Correction de l'exercice 13 - La somme des premiers cubes

a) On initialise un accumulateur à 0, puis on ajoute k^3 pour k allant de 1 à n :

```
def cubes(n):
    s = 0
    for k in range(1, n + 1):
        s = s + k**3
    return s
```

b) La fonction calcule $1 + 8 + 27 + 64 + 125 = 225$.

c) On calcule le carré de la somme des cinq premiers entiers :

$$\left(\frac{5 \times 6}{2}\right)^2 = 15^2 = 225$$

On retrouve le même nombre. C'est un cas d'une propriété générale : la somme des n premiers cubes est le carré de la somme des n premiers entiers.

Réponse. a) boucle *for k in range(1, n + 1)* avec $s = s + k**3$ · b) 225 · c) $15^2 = 225$.

Correction de l'exercice 14 - Intercaler un point par les moyennes

a) La fonction renvoie la **moyenne arithmétique** des abscisses et la **moyenne géométrique** des ordonnées :

$$\frac{0 + 4}{2} = 2 \quad \sqrt{100 \times 400} = \sqrt{40\,000} = 200$$

Elle renvoie le couple (2 ; 200).

b) $f(2) = 100 \times \sqrt{2}^2 = 100 \times 2 = 200$: le point est bien sur la courbe. **Contrôle** : $f(0) = 100$ et $f(4) = 100 \times 4 = 400$, donc A et B sont bien sur la courbe.

Sur une courbe exponentielle, l'ordonnée est multipliée par le même nombre quand l'abscisse avance du même pas : le point d'abscisse moyenne a pour ordonnée la moyenne géométrique. En répétant la fonction, on peut construire autant de points de la courbe qu'on veut.

Réponse. a) (2 ; 200) · b) oui, car $f(2) = 200$.

Correction de l'exercice 15 - Synthèse : seuil sur une simulation

a) On lance **tant que** l'on n'a pas obtenu un 6 : c'est une boucle non bornée.

```
from random import randint
```

```
def premierSix():
    n = 0
    de = 0
    while de != 6:
```

```
de = randint(1, 6)
n = n + 1
return n
```

b)

```
def moyennePremierSix(N):
    total = 0
    for essai in range(N):
        total = total + premierSix()
    return total / N
```

c) La valeur théorique est $\frac{1}{p} = \frac{1}{\frac{1}{6}} = 6$ lancers en moyenne.

- Pour $N = 10\,000$, la simulation donne une valeur proche de 6.
- Remarque : la boucle *while* s'arrête **presque sûrement**, mais le nombre de lancers n'est pas borné : certaines simulations peuvent demander 20 lancers ou plus.

Réponse. c) 6 lancers en moyenne.