

Plaque 1 - Boucles, dichotomie, suites, intégration numérique et simulation

Python et calcul scientifique — Licence 1 (L1)

Corrections détaillées

Boîte à outils

Structures de base.

- **Affectation** : le membre de droite est calculé avec les **anciennes** valeurs.
- **Boucle bornée** : `for i in range(n)` parcourt $0, 1, \dots, n - 1$ (soit n valeurs).
- **Boucle conditionnelle** : `while condition` — attention à la **terminaison**.
- **Fonction** : `def f(x): return ...`

Algorithmes au programme.

- **Recherche de seuil** : on itère tant que la condition n'est pas atteinte, en comptant les étapes.
- **Dichotomie** : pour f continue changeant de signe sur $[a; b]$, on divise l'intervalle par 2 à chaque étape. Après n étapes, la précision est $\frac{b-a}{2^n}$, d'où le nombre d'itérations :

$$\frac{b-a}{2^n} \leq \varepsilon \Leftrightarrow n \geq \log_2 \frac{b-a}{\varepsilon}$$

- **Méthode de Newton** : $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ — convergence **quadratique** (le nombre de décimales double).
- **Méthode des rectangles** : $\int_a^b f \approx \frac{b-a}{n} \sum_{k=0}^{n-1} f\left(a + k \frac{b-a}{n}\right)$, erreur en $\frac{1}{n}$.
- **Méthode des trapèzes** : erreur en $\frac{1}{n^2}$ — bien meilleure pour le même coût.
- **Monte-Carlo** : on estime une aire ou une probabilité par une **fréquence**, avec une erreur en $\frac{1}{\sqrt{n}}$.

Bonnes pratiques : toujours **valider** sur un cas dont on connaît la réponse exacte, et surveiller les **erreurs d'arrondi** (les flottants ne sont pas les réels).

Correction de l'exercice 1 - Dérouler une affectation

On suit les instructions **dans l'ordre**, en notant les valeurs.

Instruction	a	b
<code>a = 5</code>	5	—
<code>b = 3</code>	5	3
<code>a = a + b</code>	8	3
<code>b = a - b</code>	8	5

- **Ligne 3** : a devient $5 + 3 = 8$.
- **Ligne 4** : b devient $8 - 3 = 5$ — on utilise la **nouvelle** valeur de a .
- **Résultat** : $a = 8$ et $b = 5$.

- **Piège** : les valeurs sont calculées **séquentiellement**, pas simultanément. Si l'on inversait les deux dernières lignes, on obtiendrait $b = 2$ puis $a = 7$.

Réponse. $a = 8$ et $b = 5$.

Correction de l'exercice 2 - Dérouler une boucle for

- `range(1, 6)` fournit 1, 2, 3, 4, 5 (la borne 6 est **exclue**).
- **Accumulation** : `s` reçoit successivement 1, $1 + 4 = 5$, $5 + 9 = 14$, $14 + 16 = 30$, $30 + 25 = 55$.
- **Affichage** : 55.
- **Contrôle par la formule** : $\sum_{k=1}^5 k^2 = \frac{5 \times 6 \times 11}{6} = 55$ ✓
- **Attention à la borne** : `range(1, 6)` donne cinq valeurs de 1 à 5 ; `range(6)` en donnerait six, de 0 à 5.
- **Variante** : écrire le carré avec l'opérateur puissance plutôt qu'un produit donne le même résultat.

Réponse. Le programme affiche 55.

Correction de l'exercice 3 - Recherche de seuil

Programme.

```
n = 0
u = 1.0
while u <= 1000:
    u = u * 1.05
    n = n + 1
print(n)
```

Résolution mathématique. $1,05^n > 1000$ équivaut à $n \ln 1,05 > \ln 1000$, soit

$$n > \frac{\ln 1000}{\ln 1,05} = \frac{6,9078}{0,048790} \approx 141,6$$

- **Réponse** : $n = 142$.
- **Contrôles** : $1,05^{141} \approx 972,1 \leq 1000$ et $1,05^{142} \approx 1020,7 > 1000$ ✓
- **Terminaison** : la suite $1,05^n$ est **croissante non majorée**, donc la boucle s'arrête nécessairement.
- **Interprétation financière** : à 5 % par an, un capital est multiplié par 1 000 en 142 ans.

Réponse. $n = 142$.

Correction de l'exercice 4 - Dichotomie

Programme.

```
def f(x):
    return x ** 3 + x - 1

a, b = 0.0, 1.0
while b - a > 1e-3:
    m = (a + b) / 2
    if f(a) * f(m) <= 0:
        b = m
    else:
        a = m
```

```
print((a + b) / 2)
```

Analyse.

- **Validité** : $f(0) = -1 < 0$ et $f(1) = 1 > 0$ — le TVI garantit une racine, et $f' = 3x^2 + 1 > 0$ assure l'**unicité**.
- **Nombre d'itérations** : la précision après n étapes est 2^{-n} ; il faut $2^{-n} < 10^{-3}$, soit $n \geq 10$ (car $2^{10} = 1024 > 1000$).
- **Résultat** : $\alpha \approx 0,6823$.
- **Contrôles** : $f(0,682) = -0,0007$ et $f(0,683) = 0,0016$ — la racine est bien entre les deux ✓
- **Comparaison avec Newton** : Newton atteindrait cette précision en **3 itérations** au lieu de 10, mais exige la dérivée et un bon point de départ.

Réponse. $\alpha \approx 0,682$; 10 itérations suffisent.

Correction de l'exercice 5 - Méthode de Newton

Pour $f(x) = x^2 - 5$, la relation devient $x_{n+1} = x_n - \frac{x_n^2 - 5}{2x_n} = \frac{x_n}{2} + \frac{5}{2x_n}$.

```
x = 2.0
for i in range(5):
    x = x / 2 + 5 / (2 * x)
    print(x)
```

Itérations.

- $x_1 = 1 + 1,25 = 2,25$
- $x_2 = 1,125 + 1,11111 \approx 2,236111$
- $x_3 = 2,2360680$
- **Valeur exacte** : $\sqrt{5} = 2,23606798$.
- **Erreurs** : 0,0139, puis $4,3 \times 10^{-5}$, puis $4,2 \times 10^{-10}$ — l'erreur est **élevée au carré** à chaque étape.
- **Convergence quadratique** : le nombre de décimales exactes **double** à chaque itération (1, puis 4, puis 9...).
- **Contrôle** : $2,2360680^2 \approx 5,0000000$ ✓ — trois itérations suffisent pour la précision machine usuelle.

Réponse. $x_3 \approx 2,2360680$, soit $\sqrt{5}$ à 10^{-9} près.

Correction de l'exercice 6 - Suite récurrente**Programme.**

```
u = 1.0
for n in range(20):
    u = u / 2 + 1
    print(u)
```

Étude mathématique.

- **Point fixe** : $\ell = \frac{\ell}{2} + 1$ donne $\ell = 2$.
- **Forme explicite** : $u_n - 2 = \left(\frac{1}{2}\right)^n (1 - 2) = -2^{-n}$, donc $u_n = 2 - 2^{-n}$.
- **Valeur** : $u_{20} = 2 - 2^{-20} = 2 - 9,5 \times 10^{-7} \approx 1,99999905$.
- **Limite** : $u_n \rightarrow 2$, par valeurs **inférieures** (la suite est croissante).

- **Contrôles** : $u_1 = 1,5$, $u_2 = 1,75$, $u_3 = 1,875$ ✓ — conformes à la formule.
- **Remarque numérique** : dès $n = 53$, l'écart 2^{-n} devient inférieur à la précision des flottants et le programme affiche exactement 2,0.

Réponse. $u_{20} = 2 - 2^{-20} \approx 1,9999990$; limite 2.

Correction de l'exercice 7 - Intégration par rectangles

Programme.

```
def integrale(n):
    s = 0.0
    for k in range(n):
        s = s + (k / n) ** 2
    return s / n

print(integrale(10), integrale(1000))
```

Résultats.

- $n = 10$: 0,285.
- $n = 1000$: 0,3328335.
- **Valeur exacte** : $\int_0^1 x^2 dx = \frac{1}{3} \approx 0,333333$.
- **Erreurs** : 0,0483 puis 0,0005 — l'erreur est divisée par 100 quand n est multiplié par 100, soit une décroissance en $\frac{1}{n}$ ✓
- **Sous-estimation systématique** : f étant **croissante**, les rectangles à gauche sont toujours **sous** la courbe.
- **Amélioration** : la méthode des **trapèzes** (moyenne des rectangles gauche et droit) donne une erreur en $\frac{1}{n^2}$ — pour $n = 10$, on obtiendrait déjà 0,335.

Réponse. 0,285 puis 0,33283 — convergence en $\frac{1}{n}$ vers $\frac{1}{3}$.

Correction de l'exercice 8 - Simulation de Monte-Carlo

Programme.

```
import random

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

print(estime_pi(1000000))
```

Analyse.

- **Principe** : la probabilité de tomber dans le quart de disque est $\frac{\pi}{4} \approx 0,7854$; la fréquence observée F_n converge vers cette valeur (loi des grands nombres), donc $4F_n \rightarrow \pi$.
- **Précision** : $\sigma(F_n) = \sqrt{\frac{p(1-p)}{n}}$ avec $p = 0,7854$, soit $\sqrt{\frac{0,1685}{10^6}} \approx 4,1 \times 10^{-4}$.

- $\sigma(4F_n) \approx 1,6 \times 10^{-3}$: on obtient environ **deux à trois décimales** de π .
- **Limite de la méthode** : la précision progresse en $\frac{1}{\sqrt{n}}$ — gagner une décimale coûte **cent fois** plus de tirages.
- **Quand Monte-Carlo est pertinent** : en **grande dimension**, où les méthodes déterministes deviennent inutilisables.

Réponse. $\hat{\pi} = 4F_n$, précision $\approx 1,6 \times 10^{-3}$ pour $n = 10^6$.

Correction de l'exercice 9 - Boucle while et terminaison

- **Condition d'arrêt** : la boucle continue tant que $\frac{1}{n} > 0,01$, c'est-à-dire $n < 100$.
- **Terminaison** : n **croît strictement** à chaque passage, et la suite $\frac{1}{n}$ tend vers 0 — la boucle s'arrête nécessairement.
- **Sortie** : la boucle s'arrête au premier n avec $\frac{1}{n} \leq 0,01$, soit $n = 100$.
- **Affichage** : 100.
- **Vérification** : pour $n = 99$, $\frac{1}{99} = 0,0101 > 0,01$ (on continue) ; pour $n = 100$, $0,01 \leq 0,01$ (on sort) ✓
- **Attention aux flottants** : si l'on testait $1/n > 0,1$, la comparaison exacte pourrait surprendre car 0,1 n'a pas de représentation binaire exacte — c'est pourquoi on évite les tests d'égalité entre flottants.
- **Règle de sûreté** : toute boucle *while* doit reposer sur une quantité **monotone** franchissant un seuil.

Réponse. Oui, il s'arrête et affiche 100.

Correction de l'exercice 10 - Calcul d'une somme

Programme.

```
def harmonique(n):
    s = 0.0
    for k in range(1, n + 1):
        s = s + 1 / k
    return s

print(harmonique(1000))
```

Résultat : $H_{1000} \approx 7,485471$.

- **Comparaison théorique** : avec $\gamma \approx 0,5772$ (constante d'Euler),

$$H_n = \ln n + \gamma + o(1)$$
- $\ln 1000 + 0,5772 = 6,9078 + 0,5772 = 7,4850$ ✓ — l'accord est excellent.
- **Divergence** : $H_n \rightarrow +\infty$, mais **très lentement** — il faudrait $n \approx e^{100} \approx 2,7 \times 10^{43}$ termes pour dépasser 100.
- **Contrôle pour de petites valeurs** : $H_1 = 1$, $H_2 = 1,5$, $H_4 \approx 2,083$ ✓
- **Précaution numérique** : sommer de très nombreux petits termes accumule des erreurs d'arrondi ; pour de grands n , on additionne de préférence les **petits termes d'abord** (k décroissant), ou l'on utilise une sommation compensée.

Réponse. $H_{1000} \approx 7,4855$, cohérent avec $\ln 1000 + \gamma$.

Correction de l'exercice 11 – Recherche d'un extremum

Programme.

```
def f(x):
    return x ** 2 - 3 * x + 5

xmin = 0.0
for i in range(301):
    x = i * 0.01
    if f(x) < f(xmin):
        xmin = x
print(xmin, f(xmin))
```

Résultat du balayage : $x_{\min} = 1,50$ et $f = 2,75$.

Valeur exacte. $f(x) = 2x - 3$ s'annule en $x = 1,5$, et $f(1,5) = 2,25 - 4,5 + 5 = 2,75$ ✓

- **Forme canonique** : $f(x) = (x - 1,5)^2 + 2,75$ — le minimum est **exact** ici, le pas de 0,01 tombant pile sur 1,5.
- **Limite du balayage** : si le minimum était en $\sqrt{2} \approx 1,4142$, le balayage donnerait 1,41 — précision limitée par le **pas**.
- **Coût** : 301 évaluations pour une précision de 0,01 ; une dichotomie sur f en demanderait une dizaine.
- **Bonne pratique** : le balayage est simple et robuste (il trouve un minimum **global** sur l'intervalle), mais peu précis — on l'utilise pour **localiser**, puis on affine.

Réponse. Minimum 2,75 en $x = 1,5$, conforme au calcul exact.

Correction de l'exercice 12 – Comparaison de méthodes

— **Valeur exacte** : $e - 1 \approx 1,718282$.

Rectangles à gauche : $\frac{1}{10} \sum_{k=0}^9 e^{k/10} \approx 1,633799$.

— **Erreur** : 0,0845.

Trapezes : $\frac{1}{10} \left[\frac{e^0 + e^1}{2} + \sum_{k=1}^9 e^{k/10} \right] \approx 1,719713$.

- **Erreur** : 0,00143 — soit **60 fois** meilleure, pour le même nombre d'évaluations.
- **Explication théorique** : l'erreur des rectangles est en $\frac{1}{n}$, celle des trapèzes en $\frac{1}{n^2}$.
- **Vérification de l'ordre** : avec $n = 100$, l'erreur des trapèzes tomberait à $1,4 \times 10^{-5}$, soit divisée par 100 ✓
- **Sens du biais** : e^x étant **croissante**, les rectangles à gauche **sous-estiment** ; étant **convexe**, les trapèzes **sur-estiment** légèrement.
- **Conclusion pratique** : à coût égal, préférer toujours les trapèzes (ou Simpson, en $\frac{1}{n^4}$).

Réponse. Rectangles : erreur 0,085 ; trapèzes : 0,0014 — bien meilleurs.

Correction de l'exercice 13 – Méthode d'Euler pour $y' = y$

a)

```
import math

def euler(n):
    h = 1 / n
```

```

y = 1.0
for k in range(n):
    y = y + h * y
return y

for n in [10, 100, 1000]:
    print(n, euler(n), math.e - euler(n))

```

b) Chaque pas multiplie y par $1 + h$: après n pas, $y_n = (1 + h)^n = \left(1 + \frac{1}{n}\right)^n$, qui tend vers $y(1) = e$.

— Sorties : 2,5937 ($n = 10$), 2,7048 ($n = 100$), 2,71692 ($n = 1000$).

c) Quand n est multiplié par 10, l'erreur est divisée par 10 : erreur en $\frac{1}{n}$, méthode **d'ordre 1**.

d) $n \ln\left(1 + \frac{1}{n}\right) = 1 - \frac{1}{2n} + O\left(\frac{1}{n^2}\right)$, donc

$$\left(1 + \frac{1}{n}\right)^n = e e^{-\frac{1}{2n} + O(1/n^2)} = e - \frac{e}{2n} + O\left(\frac{1}{n^2}\right)$$

— **Contrôle** : $\frac{e}{2000} \approx 0,001359$, très proche de l'erreur mesurée 0,001358 ✓

Réponse. $y_n = \left(1 + \frac{1}{n}\right)^n \rightarrow e$; méthode d'ordre 1 ; erreur $\sim \frac{e}{2n}$.

Correction de l'exercice 14 - Les flottants ne sont pas les réels

a) 0,1 n'a pas d'écriture binaire finie (comme $\frac{1}{3}$ en décimal) : il est **arrondi** à 53 bits. Les arrondis de 0,1 et 0,2 ne s'additionnent pas exactement en l'arrondi de 0,3. **Règle** : ne jamais tester l'égalité de flottants, mais $|a - b| < \varepsilon$.

b) $\cos x = 1 - \frac{x^2}{2} + o(x^2)$, donc

$$\frac{1 - \cos x}{x^2} \xrightarrow{x \rightarrow 0} \frac{1}{2}$$

c) Pour $x = 10^{-8}$, $\cos x = 1 - 5 \times 10^{-17}$, écart inférieur à la précision relative des flottants ($\approx 1,1 \times 10^{-16}$) : le calcul donne exactement 1, et $1 - 1 = 0$. On soustrait deux nombres **très proches** : tous les chiffres significatifs disparaissent.

d) $\cos x = 1 - 2 \sin^2 \frac{x}{2}$ (formule de duplication). La nouvelle expression ne contient **aucune soustraction** de nombres voisins : $\sin \frac{x}{2}$ est calculé avec une excellente précision relative, et $\frac{2 \sin^2(x/2)}{x^2} \approx \frac{2(x/2)^2}{x^2} = \frac{1}{2}$ ✓

Réponse. Arrondis binaires ; limite $\frac{1}{2}$; annulation catastrophique (résultat 0) ; la forme $\frac{2 \sin^2(x/2)}{x^2}$ donne 0,5.

Correction de l'exercice 15 - Synthèse : simulation d'une loi

Programme.

```

import random

def simule(n):
    compte = 0
    for i in range(n):
        d1 = random.randint(1, 6)
        d2 = random.randint(1, 6)

```

```
    if d1 + d2 == 7:
        compte = compte + 1
    return compte / n

print(simule(100000))
```

Valeur théorique : la somme 7 s’obtient de 6 façons sur 36, soit $\frac{1}{6} \approx 0,166667$.

Précision attendue. La fréquence F_n vérifie

$$\sigma(F_n) = \sqrt{\frac{p(1-p)}{n}} = \sqrt{\frac{0,1667 \times 0,8333}{10^5}} \approx 1,18 \times 10^{-3}$$

- On attend donc une estimation à environ $\pm 0,0024$ près (deux écarts types), soit un résultat entre 0,164 et 0,169.
- **Validation** : si la simulation donne systématiquement une valeur hors de cet intervalle, c’est le signe d’une **erreur de programmation** (par exemple *randint(0, 6)* au lieu de *randint(1, 6)*).
- **Méthode générale** : toujours confronter la simulation à une **valeur théorique** sur un cas simple avant de l’appliquer à un problème inconnu.

Réponse. $\mathbb{P} = \frac{1}{6} \approx 0,1667$, estimée à $\pm 0,0024$ près.