

Python et calcul scientifique

Licence 1 (L1) — Cours complet

Algorithmique, complexité, NumPy, Matplotlib et méthodes numériques fondamentales

Méthode — idée directrice

Le calcul scientifique ne remplace pas la démonstration : il sert à **expérimenter**, **approcher** et **visualiser**. En L1, on apprend surtout à traduire une définition mathématique en algorithme, puis à contrôler l'erreur, le coût et les limites du calcul numérique.

1 Premiers réflexes en Python

Python est un langage interprété à typage dynamique. Une variable est un nom associé à un objet ; son type peut être obtenu avec `type`.

Type	Exemple	Usage courant
<code>int</code>	<code>n = 12</code>	entier
<code>float</code>	<code>x = 3.14</code>	réel approché
<code>bool</code>	<code>x > 0</code>	condition logique
<code>str</code>	<code>"L1 maths"</code>	texte
<code>list</code>	<code>[1, 2, 3]</code>	collection mutable
<code>tuple</code>	<code>(1, 2)</code>	collection immuable

Les opérateurs usuels sont `+`, `-`, `*`, `/`, `//` (quotient entier), `%` (reste) et `**` (puissance). Les comparaisons sont `==`, `!=`, `<`, `<=`, `>` et `>=`.

Piège classique — affectation et égalité

`x = 3` affecte la valeur 3 à `x`. En revanche, `x == 3` construit le booléen qui teste l'égalité. Ce sont deux opérations différentes.

1.1 Conditions, boucles et fonctions

```

1 def signe(x):
2     if x > 0:
3         return 1
4     elif x < 0:
5         return -1
6     else:
7         return 0
8
9 s = 0
10 for k in range(1, 6):
11     s += k**2
12 print(s)  # 55

```

`range(a,b)` parcourt les entiers $a, a+1, \dots, b-1$. Une boucle `while` est adaptée lorsqu'on ne connaît pas à l'avance le nombre d'itérations.

Méthode — écrire une fonction mathématique

Pour traduire une fonction f en Python :

1. préciser son domaine utile ;
2. choisir les entrées et la valeur retournée ;
3. traiter les cas particuliers ;
4. tester sur quelques valeurs dont le résultat est connu.

2 Listes, compréhension et récursivité

Une liste Python est mutable et indexée à partir de 0. Ainsi `L[0]` est le premier élément et `L[-1]` le dernier.

```
1 L = [2, 5, 7, 11]
2 L.append(13)
3 carres = [k**2 for k in range(10)]
4 positifs = [x for x in L if x > 0]
```

Une fonction est **récursive** lorsqu'elle s'appelle elle-même. Elle doit posséder un cas d'arrêt.

```
1 def factorielle(n):
2     if n == 0:
3         return 1
4     return n * factorielle(n - 1)
```

Piège classique — récursivité

Une définition récursive mathématiquement correcte peut être informatiquement inefficace. Le Fibonacci naïf recalculant plusieurs fois les mêmes termes a un coût exponentiel ; une boucle itérative a un coût linéaire.

3 Complexité d'un algorithme

Définition

La **complexité temporelle** décrit l'ordre de grandeur du nombre d'opérations effectuées lorsque la taille n de l'entrée croît. La notation $T(n) = \mathcal{O}(g(n))$ signifie qu'à une constante multiplicative près, le coût ne croît pas plus vite que $g(n)$ pour n grand.

Ordres de grandeur usuels :

$$\mathcal{O}(1), \quad \mathcal{O}(\log n), \quad \mathcal{O}(n), \quad \mathcal{O}(n \log n), \quad \mathcal{O}(n^2), \quad \mathcal{O}(2^n).$$

Exemple	Nombre d'étapes	Complexité
Accéder à <code>L[k]</code>	constante	$\mathcal{O}(1)$
Dichotomie	proportionnel à $\log n$	$\mathcal{O}(\log n)$
Parcourir une liste	proportionnel à n	$\mathcal{O}(n)$
Deux boucles imbriquées	proportionnel à n^2	$\mathcal{O}(n^2)$

Exemple

Pour calculer $\sum_{k=0}^{n-1} L_k$, une boucle unique effectue n additions : son coût est $\mathcal{O}(n)$. Une double boucle parcourant toutes les paires (i, j) effectue environ n^2 opérations : coût $\mathcal{O}(n^2)$.

4 Calcul flottant et erreurs numériques

Un `float` représente un réel par un nombre fini de bits. Beaucoup de nombres décimaux ne sont donc pas stockés exactement.

```
1 0.1 + 0.2 == 0.3
2 # False dans l'arithmétique flottante usuelle
```

Méthode — comparer des flottants

On remplace l'égalité exacte par une tolérance :

$$|a - b| \leq \varepsilon.$$

En Python :

```
abs(a - b) <= 1e-12
```

Avec NumPy, préférer souvent `np.isclose(a,b)` ou `np.allclose(A,B)`.

Trois sources d'erreur sont à distinguer : erreur de modèle, erreur de méthode (discrétisation/troncature) et erreur d'arrondi machine.

5 NumPy : tableaux et calcul vectorisé

NumPy fournit des tableaux homogènes `ndarray` et des fonctions mathématiques appliquées composante par composante.

```
1 import numpy as np
2
3 x = np.array([0.0, 0.5, 1.0])
4 y = np.sin(x) + x**2
5 u = np.linspace(0.0, 1.0, 101)
```

`np.linspace(a,b,n)` crée n points régulièrement espacés entre a et b . La vectorisation évite souvent des boucles Python explicites.

Exemple — vectorisation

Pour calculer $f(x) = e^{-x^2}$ en 10^5 points :

```
1 x = np.linspace(-3, 3, 100_000)
2 y = np.exp(-x**2)
```

La formule est lue presque comme l'expression mathématique.

Piège classique — produit NumPy

Pour deux tableaux, `A * B` est un produit composante par composante. Le produit matriciel s'écrit `A @ B` (ou `np.matmul(A,B)`).

6 Matplotlib : tracer une courbe ou un nuage de points

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 x = np.linspace(-2, 2, 400)
5 y = np.exp(-x**2)
6
7 plt.plot(x, y, label="exp(-x^2)")
8 plt.axhline(0)
9 plt.axvline(0)
10 plt.xlabel("x")
11 plt.ylabel("y")
12 plt.grid(True)
13 plt.legend()
14 plt.show()
```

Pour un nuage de points, on utilise `plt.scatter(x,y)`. Un graphique scientifique doit comporter des axes identifiés, des unités si nécessaire et une légende lorsqu'il y a plusieurs courbes.

7 Dichotomie : recherche robuste d'une racine

Soit f continue sur $[a, b]$ avec $f(a)f(b) \leq 0$. Le théorème des valeurs intermédiaires garantit au moins une racine dans $[a, b]$.

Méthode — principe de la dichotomie

À chaque étape, on pose $m = (a + b)/2$. On conserve la moitié de l'intervalle sur laquelle le changement de signe subsiste. Après n étapes, la longueur vaut

$$\frac{b - a}{2^n}.$$

Pour obtenir une largeur au plus ε , il suffit de choisir

$$n \geq \log_2 \left(\frac{b - a}{\varepsilon} \right).$$

```

1 def dichotomie(f, a, b, eps=1e-10):
2     fa, fb = f(a), f(b)
3     if fa * fb > 0:
4         raise ValueError("Pas de changement de signe")
5
6     while b - a > eps:
7         m = (a + b) / 2
8         fm = f(m)
9         if fa * fm <= 0:
10            b, fb = m, fm
11        else:
12            a, fa = m, fm
13    return (a + b) / 2

```

Résultat — erreur

Si α est la racine conservée dans l'intervalle final $[a_n, b_n]$, alors le milieu m_n vérifie

$$|m_n - \alpha| \leq \frac{b_n - a_n}{2} = \frac{b - a}{2^{n+1}}.$$

8 Méthode de Newton

La méthode de Newton remplace localement la courbe de f par sa tangente. À partir de x_n , l'intersection de cette tangente avec l'axe des abscisses donne

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

```

1 def newton(f, df, x0, eps=1e-12, nmax=50):
2     x = x0
3     for _ in range(nmax):
4         d = df(x)
5         if abs(d) < 1e-14:
6             raise ZeroDivisionError("Derivee trop petite")
7         xn = x - f(x) / d
8         if abs(xn - x) <= eps:
9             return xn
10        x = xn
11    raise RuntimeError("Newton n'a pas converge")

```

Exemple

Pour $f(x) = x^2 - 2$, $f'(x) = 2x$ et $x_0 = 1$, Newton produit rapidement une approximation de $\sqrt{2}$.

Piège classique — Newton n'est pas infallible

La méthode peut diverger, osciller ou rencontrer $f'(x_n) = 0$. Elle exige un choix raisonnable du point initial et des hypothèses locales. La dichotomie est plus lente mais plus robuste lorsqu'un changement de signe est disponible.

9 Intégration numérique

Soit f continue sur $[a, b]$ et $h = (b - a)/n$, $x_k = a + kh$.

9.1 Méthode des rectangles à gauche

$$R_n = h \sum_{k=0}^{n-1} f(x_k).$$

```
1 def rectangles(f, a, b, n):
2     h = (b - a) / n
3     return h * sum(f(a + k*h) for k in range(n))
```

9.2 Méthode des trapèzes

$$T_n = h \left(\frac{f(a) + f(b)}{2} + \sum_{k=1}^{n-1} f(x_k) \right).$$

```
1 def trapezes(f, a, b, n):
2     h = (b - a) / n
3     s = 0.5 * (f(a) + f(b))
4     s += sum(f(a + k*h) for k in range(1, n))
5     return h * s
```

Pour une fonction suffisamment régulière, les trapèzes convergent généralement plus vite que les rectangles. L'idée importante est de distinguer la valeur exacte de l'intégrale de son approximation numérique.

10 Méthode d'Euler pour une équation différentielle

On considère le problème de Cauchy

$$y'(t) = F(t, y(t)), \quad y(t_0) = y_0.$$

Avec un pas $h > 0$, la méthode d'Euler explicite est

$$t_{n+1} = t_n + h, \quad y_{n+1} = y_n + hF(t_n, y_n).$$

Elle vient de l'approximation $y(t + h) \approx y(t) + hy'(t)$.

```
1 def euler(F, t0, y0, h, n):
2     t = [t0]
3     y = [y0]
4     for _ in range(n):
5         y.append(y[-1] + h * F(t[-1], y[-1]))
6         t.append(t[-1] + h)
7     return t, y
```

Exemple

Pour $y' = y$, $y(0) = 1$, la solution exacte est e^t . Euler donne

$$y_{n+1} = (1 + h)y_n, \quad y_n = (1 + h)^n,$$

qui approche e^{nh} lorsque h est petit.

Piège classique — pas trop grand

Diminuer h améliore généralement la précision mais augmente le nombre d'étapes. Un pas trop grand peut produire une approximation médiocre, voire instable pour certaines équations.

11 Simulation d'expériences aléatoires

NumPy permet de simuler des tirages pseudo-aléatoires. Une simulation n'est pas une preuve : elle donne une observation numérique compatible avec une loi théorique.

```
1 import numpy as np
2 rng = np.random.default_rng(12345)
3
4 tirages = rng.integers(1, 7, size=100_000)
5 freq_six = np.mean(tirages == 6)
6 print(freq_six)
```

La fréquence observée d'un six est proche de $1/6$ pour un grand nombre de tirages. Fixer la graine (ici 12345) rend l'expérience reproductible.

Méthode — démarche expérimentale

1. formuler une conjecture mathématique ;
2. écrire une expérience numérique reproductible ;
3. varier les paramètres et la taille de l'échantillon ;
4. mesurer l'erreur ou l'écart ;
5. revenir à la démonstration quand une preuve est demandée.

12 Un mini-projet complet : racine, intégrale et tracé

On étudie $f(x) = \cos x - x$ sur $[0, 1]$.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 f = lambda x: np.cos(x) - x
5 df = lambda x: -np.sin(x) - 1
6
7 alpha_dicho = dichotomie(f, 0.0, 1.0, eps=1e-12)
8 alpha_newton = newton(f, df, 0.7)
9
10 x = np.linspace(0, 1, 300)
11 plt.plot(x, f(x))
12 plt.axhline(0)
13 plt.scatter([alpha_newton], [0])
14 plt.grid(True)
15 plt.show()
```

On compare alors les deux approximations, le nombre d'itérations et la robustesse. Le calcul scientifique consiste précisément à relier algorithme, théorie et contrôle de l'erreur.

13 Checklist pour rédiger un programme de L1

Méthode — avant de rendre un code

- Donner des noms explicites aux variables et fonctions.
- Éviter de recopier du code : factoriser dans une fonction.
- Vérifier les préconditions (intervalle, dérivée non nulle, $n > 0$, etc.).
- Prévoir un critère d'arrêt pour les boucles itératives.

- Tester sur des cas simples ou connus exactement.
- Pour un résultat approché, afficher ou estimer l'erreur.
- Distinguer algorithme mathématique et détails d'affichage.
- Ne jamais conclure à partir d'un graphique seul lorsqu'une preuve est demandée.

Piège classique — les erreurs les plus fréquentes

- Tester des flottants avec `==` au lieu d'une tolérance.
- Oublier que `range(n)` s'arrête à $n - 1$.
- Modifier une liste pendant qu'on la parcourt sans maîtriser les effets.
- Confondre `*` et `@` avec des tableaux NumPy.
- Oublier un critère d'arrêt dans `while` ou une récursion.
- Utiliser Newton sans contrôler f' ou le point initial.
- Choisir un pas numérique sans étudier son effet sur l'erreur.

14 Exercices corrigés

Exemple — 1. Dichotomie pour $\sqrt{5}$

On veut résoudre $f(x) = x^2 - 5 = 0$ sur $[2, 3]$. Comme f est continue et $f(2) < 0 < f(3)$, la dichotomie s'applique. Pour une erreur sur le milieu inférieure à 10^{-6} , il suffit que

$$\frac{3 - 2}{2^{n+1}} \leq 10^{-6},$$

soit $n \geq 19$. Un appel possible est :

```
1 f = lambda x: x**2 - 5
2 r = dichotomie(f, 2.0, 3.0, eps=2e-6)
```

La valeur obtenue doit être proche de $\sqrt{5} \simeq 2.2360679$. Le contrôle théorique de l'erreur est plus important que le nombre de décimales affichées.

Exemple — 2. Lire une complexité

Considérons :

```
1 s = 0
2 for i in range(n):
3     for j in range(i):
4         s += i + j
```

Le nombre d'itérations de la boucle intérieure vaut

$$0 + 1 + \dots + (n - 1) = \frac{n(n - 1)}{2}.$$

Ainsi le coût est quadratique : $T(n) = \mathcal{O}(n^2)$. La présence de deux boucles imbriquées suggère $\mathcal{O}(n^2)$, mais c'est le comptage effectif qui le justifie.

Exemple — 3. Euler sur $y' = -2y$

Pour $y(0) = 1$, la solution exacte est $y(t) = e^{-2t}$. Euler donne

$$y_{k+1} = (1 - 2h)y_k.$$

Avec $h = 0.1$, on obtient $y_k = 0.8^k$. À $t = 1$, donc $k = 10$,

$$y_{10} = 0.8^{10} \approx 0.1074, \quad e^{-2} \approx 0.1353.$$

L'écart montre que la méthode est approchée ; recommencer avec $h = 0.05$ permet d'observer l'amélioration lorsque le pas diminue.