

Python et calcul scientifique — L1

Fiche de révision — syntaxe, méthodes numériques et pièges

1. Python essentiel

Types : `int`, `float`, `bool`, `str`, `list`.
 Affectation : `x = 3`. Test : `x == 3`.
`range(a,b)` parcourt $a, \dots, b-1$.

```
def f(x):
    return x**2 - 2
for k in range(5):
    print(k, f(k))
```

2. Listes, boucles, fonctions

```
L = [1, 4, 9]
L.append(16)
carres = [k**2 for k in range(10)]
```

for : nombre d'itérations connu.
while : arrêt défini par une condition.
 Une récursion doit avoir un **cas d'arrêt**.

3. Complexité

Ordres usuels :

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

Dichotomie : $\mathcal{O}(\log n)$. Parcours : $\mathcal{O}(n)$. Double boucle : $\mathcal{O}(n^2)$.

4. Flottants

Ne pas tester en général `a == b`. Utiliser

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

NumPy : `np.isclose(a,b)`, `np.allclose(A,B)`.

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

5. NumPy

```
import numpy as np
x = np.linspace(0, 1, 101)
y = np.exp(-x**2)
```

`A * B` : produit terme à terme.

`A @ B` : produit matriciel.

Vectoriser plutôt que boucler quand l'opération est composante par composante.

6. Matplotlib

```
import matplotlib.pyplot as plt
plt.plot(x, y, label="f")
plt.xlabel("x"); plt.ylabel("y")
plt.grid(True); plt.legend(); plt.show()
```

Nuage de points : `plt.scatter(x,y)`.

7. Dichotomie

Hypothèses : f continue sur $[a, b]$, $f(a)f(b) \leq 0$.

$$m = \frac{a+b}{2}, \quad \ell_n = \frac{b-a}{2^n}.$$

Erreur sur le milieu :

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

Nombre d'étapes pour largeur $\leq \varepsilon$:

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

8. Newton

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

Très rapide près d'une racine simple, mais pas garanti globalement. Vérifier $f'(x_n) \neq 0$ et imposer **nmax**.

```
xn = x - f(x)/df(x)
```

Python et calcul scientifique — L1

Fiche de révision — calcul numérique et réflexes

9. Rectangles et trapèzes

$$h = (b - a)/n, \quad x_k = a + kh.$$

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

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

Comparer l'approximation lorsque n augmente.

10. Euler explicite

Pour

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

avec pas h :

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

Un pas plus petit améliore souvent la précision mais augmente le coût.

11. Simulation aléatoire

```
rng = np.random.default_rng(123)
x = rng.integers(1, 7, size=100000)
freq = np.mean(x == 6)
```

La simulation **illustre** une loi ; elle ne la démontre pas. Fixer une graine rend l'expérience reproductible.

12. Critères d'arrêt

Toujours combiner si possible :

- une tolérance ε ;
- un nombre maximal d'itérations ;
- un contrôle des cas interdits.

Exemples : $|x_{n+1} - x_n| \leq \varepsilon$, $b - a \leq \varepsilon$.

13. Méthode type d'un exercice

1. Traduire précisément le problème mathématique.
2. Identifier entrées, sorties et préconditions.
3. Écrire l'algorithme avant le code.
4. Programmer en fonctions courtes.
5. Tester sur un cas connu.
6. Étudier précision et coût.
7. Tracer seulement si cela aide à interpréter.

14. Tableau des méthodes

But	Méthode
Racine robuste	Dichotomie
Racine rapide	Newton
Intégrale	Rectangles / trapèzes
EDO	Euler explicite
Courbe	Matplotlib
Calcul tableau	NumPy

15. Pièges à éviter

- `==` pour deux flottants.
- Oublier que `range(n)` finit à $n - 1$.
- Confondre `*` et `@` sous NumPy.
- Boucle `while` sans garantie d'arrêt.
- Newton avec f' nul ou point initial mal choisi.
- Conclure à partir d'un seul graphique ou d'une simulation.
- Pas d'Euler trop grand sans contrôle d'erreur.

16. Mini-syntaxe à connaître

```
# somme des carres
s = sum(k**2 for k in range(n))

# tableau regulier
x = np.linspace(a, b, N)

# test numerique
a_proche = np.isclose(a, b)
```