

# Analyse Numérique avec Python - Aide-Mémoire

Basé sur les notes de Yehor Korotenko (Prof. Jean-Baptiste APOUNG KAMGA)

April 28, 2025

# 1. Interpolation Polynomiale

## Définition

Soit  $(x_i, y_i)_{i=0, \dots, N-1}$  un nuage de  $N$  points distincts. Interpoler consiste à trouver un polynôme  $P_{N-1}(x)$  de degré au plus  $N-1$  tel que  $P_{N-1}(x_i) = y_i$  pour tout  $i$ . Si  $y_i = f(x_i)$ ,  $P_{N-1}$  est le polynôme d'interpolation de Lagrange de  $f$ .

## Polynôme de Lagrange

**Existence et Unicité (Thm 2.17):** Il existe un unique polynôme  $P \in \mathbb{R}_{N-1}[X]$  tel que  $P(x_i) = y_i$ .

$$P(x) = \sum_{i=0}^{N-1} y_i L_i(x) \quad \text{où} \quad L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{N-1} \frac{x - x_j}{x_i - x_j}$$

**Erreur d'interpolation (Thm 2.19):** Si  $f \in C^N([a, b])$  et  $P_{N-1}$  interpole  $f$  aux points  $x_0, \dots, x_{N-1} \in [a, b]$ :

$$\forall x \in [a, b], \exists \xi_x \in ]a, b[ \text{ tel que } f(x) - P_{N-1}(x) = \frac{f^{(N)}(\xi_x)}{N!} \prod_{i=0}^{N-1} (x - x_i)$$

On note  $\omega_N(x) = \prod_{i=0}^{N-1} (x - x_i)$ . **Python:** `from scipy.interpolate import lagrange; p = lagrange(x_nodes, y_nodes)`

## Méthode des Différences Divisées (Newton)

Soit  $f[x_0, \dots, x_k]$  la différence divisée d'ordre  $k$ .  $f[x_i] = y_i$   $f[x_i, \dots, x_j] = \frac{f[x_{i+1}, \dots, x_j] - f[x_i, \dots, x_{j-1}]}{x_j - x_i}$  Le polynôme de Newton est:

$$P(x) = f[x_0] + f[x_0, x_1](x - x_0) + \dots + f[x_0, \dots, x_{N-1}] \prod_{i=0}^{N-2} (x - x_i)$$

$$P(x) = \sum_{k=0}^{N-1} a_k \omega_k(x) \quad \text{où} \quad a_k = f[x_0, \dots, x_k] \quad \text{et} \quad \omega_k(x) = \prod_{j=0}^{k-1} (x - x_j).$$

## Phénomène de Runge et Polynômes de Tchebychev

L'interpolation avec des points équidistants peut mal converger. Les **points de Tchebychev** sur  $[-1, 1]$  minimisent  $\|\omega_N(x)\|_\infty$ :

$$x_j = \cos\left(\frac{(2j+1)\pi}{2N}\right) \quad \text{pour } j = 0, \dots, N-1 \quad (\text{racines de } T_N(x))$$

Sur  $[a, b]$ :  $t_j = \frac{a+b}{2} + \frac{b-a}{2} x_j$ . Polynômes de Tchebychev:  $T_0(x) = 1, T_1(x) = x, T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x)$ .

# 2. Intégration Numérique (Quadrature)

**But:** Calculer  $I(f) = \int_a^b f(x)dx$ . **Formule de quadrature:**  $\hat{I}(f) = \sum_{i=0}^{N-1} w_i f(x_i)$ . Une formule est d'ordre  $p$  si elle est exacte pour tout polynôme de degré  $\leq p-1$ .

## Formules de Newton-Cotes (points $x_i$ équidistants)

**Erreur (générale):** Si la formule est d'ordre  $p$ , l'erreur  $E(f) = I(f) - \hat{I}(f)$  est:  $E(f) = C \cdot h^{p+1} f^{(p)}(\xi)$  (pour formule élémentaire,  $C$  constante) ou  $E(f) = C \cdot (b-a)h^p \|f^{(p)}\|_\infty$ .

- **Rectangle à gauche (élémentaire  $\alpha, \beta$ ):**  $\int_\alpha^\beta f(x)dx \approx (\beta - \alpha)f(\alpha)$ . Ordre 1. Erreur:  $E_e(f) = \frac{f'(\zeta)}{2}(\beta - \alpha)^2$ .

- **Point Milieu (élémentaire  $\alpha, \beta$ ):**  $\int_{\alpha}^{\beta} f(x)dx \approx (\beta - \alpha)f\left(\frac{\alpha+\beta}{2}\right)$ . Ordre 2. Erreur:  $E_e(f) = \frac{f''(c)}{24}(\beta - \alpha)^3$ .
- **Trapèze (élémentaire  $\alpha, \beta$ ):**  $\int_{\alpha}^{\beta} f(x)dx \approx \frac{\beta-\alpha}{2}(f(\alpha) + f(\beta))$ . Ordre 2. Erreur:  $E_e(f) = -\frac{f''(c)}{12}(\beta - \alpha)^3$ .
- **Simpson (élémentaire  $\alpha, \beta$ ):**  $\int_{\alpha}^{\beta} f(x)dx \approx \frac{\beta-\alpha}{6}\left(f(\alpha) + 4f\left(\frac{\alpha+\beta}{2}\right) + f(\beta)\right)$ . Ordre 4. Erreur:  $E_e(f) = -\frac{f^{(4)}(c)}{2880}(\beta - \alpha)^5$ .

**Formules composites:** On subdivise  $[a, b]$  en  $M$  sous-intervalles de longueur  $h = \frac{b-a}{M}$  et on applique la formule élémentaire sur chaque sous-intervalle. Ex. Trapèze composite:  $I_T(f) = h\left(\frac{f(x_0)+f(x_M)}{2} + \sum_{i=1}^{M-1} f(x_i)\right)$ . Erreur:  $E_T(f) = -\frac{(b-a)h^2}{12}f''(\xi)$ .

## Quadrature de Gauss-Legendre

Points  $x_i$  et poids  $w_i$  choisis pour maximiser l'ordre. Pour  $N$  points, la formule est exacte pour les polynômes de degré  $\leq 2N - 1$ . Sur  $[-1, 1]$ :  $\int_{-1}^1 f(x)dx \approx \sum_{i=1}^N w_i f(x_i)$ . Les  $x_i$  sont les racines du  $N$ -ième polynôme de Legendre  $L_N(x)$ .  $L_0(x) = 1, L_1(x) = x, (n+1)L_{n+1}(x) = (2n+1)xL_n(x) - nL_{n-1}(x)$ .

**Python:** `from scipy.integrate import quad; I, err = quad(f, a, b)`

### 3. Résolution Approchée d'Équations Différentielles Ordinaires (EDO)

Problème de Cauchy:  $y'(t) = f(t, y(t))$ ,  $y(t_0) = y_0$ , pour  $t \in [t_0, T]$ . Discrétisation:  $t_n = t_0 + n\Delta t$ . On cherche  $x_n \approx y(t_n)$ . Forme générale des schémas à un pas explicites:  $x_{n+1} = x_n + \Delta t \Phi(t_n, x_n, \Delta t)$ .

#### Schémas Explicites à un Pas

- **Euler explicite**

$$x_{n+1} = x_n + \Delta t f(t_n, x_n)$$

Ordre local de consistance: 1. Erreur globale:  $O(\Delta t)$ .

- **Euler implicite**

$$x_{n+1} = x_n + \Delta t f(t_{n+1}, x_{n+1})$$

- **Clank-Nicolas**

$$x_{n+1} = x_n + \frac{\Delta t}{2} (f(t_n, x_n) + f(t_{n+1}, x_{n+1}))$$

- **Point-Milieu**

$$x_{n+1} = x_n + \Delta t f\left(t_n + \frac{\Delta t}{2}, x_n + \frac{\Delta t}{2} f(t_n, x_n)\right)$$

Ordre local: 2. Erreur globale:  $O(\Delta t^2)$ .

- **de Heun**

$$x_{n+1} = x_n + \frac{\Delta t}{2} (f(t_n, x_n) + f(t_{n+1}, x_n + \Delta t f(t_n, x_n)))$$

Ordre local: 2. Erreur globale:  $O(\Delta t^2)$ .

#### Consistance, Stabilité, Convergence

- **Erreur locale de troncature:**  $\xi_n(\Delta t) = y(t_{n+1}) - y(t_n) - \Delta t \Phi(t_n, y(t_n), \Delta t)$ . Le schéma est **consistant d'ordre  $q$**  si  $\|\xi_n(\Delta t)\| = O(\Delta t^{q+1})$ .
- **Stabilité:** Un schéma est stable si de petites perturbations des données initiales ou des calculs n'entraînent pas de grandes déviations de la solution numérique. Pour  $\Phi$  Lipschitzienne en  $y$  de constante  $A$ : stable avec constante  $S = e^{A(T-t_0)}$ .
- **Convergence (Thm de Lax):** Pour un problème bien posé, un schéma consistant est convergent si et seulement si il est stable. Si le schéma est stable et consistant d'ordre  $q$ , alors il est convergent d'ordre  $q$ :  $\|y(t_n) - x_n\| = O(\Delta t^q)$ .

## 4. Résolution Approchée d'Équations Ordinaires (EO): $f(x) = 0$

### Méthode de Dichotomie (Bissection)

**Principe:** Soit  $f$  continue sur  $[a, b]$  avec  $f(a)f(b) < 0$ . À l'étape  $k$ ,  $I_k = [a_k, b_k]$ .  $c_k = (a_k + b_k)/2$ . Si  $f(a_k)f(c_k) < 0$ ,  $I_{k+1} = [a_k, c_k]$ . Sinon  $I_{k+1} = [c_k, b_k]$ . **Convergence:** Linéaire.  $|x^* - c_k| \leq \frac{b_0 - a_0}{2^{k+1}}$ . **Coût:** 1 éval. de  $f$  par itération.

### Méthode de Fausse Position (Regula Falsi)

**Principe:** Similaire à la dichotomie, mais  $c_k$  est l'intersection de la sécante passant par  $(a_k, f(a_k))$  et  $(b_k, f(b_k))$  avec l'axe des  $x$ .  $c_k = b_k - f(b_k) \frac{b_k - a_k}{f(b_k) - f(a_k)}$ . **Convergence:** Linéaire (souvent plus rapide que la dichotomie, mais peut être lente si une extrémité reste fixe).

### Méthode du Point Fixe

**Problème:** Trouver  $x^*$  tel que  $x^* = g(x^*)$  (équivalent à  $f(x^*) = 0$  si  $g(x) = x - f(x)$  ou autre transformation). **Itération:**  $x_{k+1} = g(x_k)$ . **Convergence (Thm du Point Fixe):** Si  $g(I) \subset I$ ,  $I$  fermé, et  $g$  est contractante sur  $I$  (i.e.,  $\exists K < 1$  t.q.  $|g(x) - g(y)| \leq K|x - y|$ ), alors  $g$  a un unique point fixe  $x^*$  dans  $I$  et la suite  $(x_k)$  converge vers  $x^*$ . Si  $g \in C^1(I)$  et  $|g'(x)| \leq K < 1$  sur  $I$ , alors  $g$  est contractante. **Ordre de convergence:**

- Si  $g'(x^*) \neq 0$ , convergence linéaire avec  $K_1 = |g'(x^*)|$ .
- Si  $g'(x^*) = \dots = g^{(p-1)}(x^*) = 0$  et  $g^{(p)}(x^*) \neq 0$ , convergence d'ordre  $p$ .

### Méthode de Newton-Raphson

**Principe:** Approximation de  $f$  par sa tangente en  $x_k$ .  $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$ . C'est un cas particulier de point fixe avec  $g(x) = x - \frac{f(x)}{f'(x)}$ . On a  $g'(x^*) = 0$  si  $f'(x^*) \neq 0$ . **Convergence:**

- Quadratique (ordre 2) si  $f'(x^*) \neq 0$  (racine simple).
- Linéaire si  $f'(x^*) = 0$  (racine multiple d'ordre  $m > 1$ ).  $K_1 = 1 - 1/m$ .

**Coût:** 1 éval. de  $f$  et 1 éval. de  $f'$  par itération. **Python (pour point fixe):**

```
def PointFixe(g, x0, eps, IterMax):
    # ... (implementation)
def Newton(f, f_prime, x0, eps, IterMax):
    g = lambda x: x - f(x)/f_prime(x)
    return PointFixe(g, x0, eps, IterMax)
```