

1. SCIPY ET L'INTÉGRATION NUMÉRIQUE

Évidemment Scipy contient tout ce qu'il faut ! L'extension `scipy.integrate` propose pour une fonction

- intégrale simple `quad`
- intégrale double `dblquad`, triple `tplquad`
- quadrature de Gauss `fixed_quad` (on fixe l'ordre de la méthode), `quadrature` (on fixe ordre et tolérance)
- `romberg` méthode de Romberg (voir plus loin).

Si on dispose d'un échantillon d'une fonction, c'est à dire la discrétisation $(x_i)_{0 \leq i \leq n}$ et $(y_i = f(x_i))_{0 \leq i \leq n}$ alors on dispose aussi

- `trapez(y, x)` méthode des trapèzes
- `cumtrapz(y, x)` version cumulative ? (à explorer)
- `simsimp(y, x)` méthode de Simpson
- `romb(y, x)` méthode de Romberg avec `x` et `y` de taille spécifique $2^k + 1$

2. MÉTHODES USUELLES D'INTÉGRATION NUMÉRIQUE

Dans la suite f sera une fonction définie de $[a, b]$ dans $\mathbb{R}$ et suffisamment régulière. Pour $N \geq 1$, on pose $h = \frac{b-a}{N}$ le pas de discrétisation de l'intervalle $[a, b]$ et pour $i = 0 \dots N$, $x_i = a + i * h$.

Exercice 1. Trapèzes

Écrire un algorithme amélioré pour la méthode des trapèzes. Compter le nombre d'opérations (+, - et *, /). Rappel :

$$\int_{x_i}^{x_{i+1}} f(t) dt \simeq \frac{h}{2} (f(x_i) + f(x_{i+1})).$$

Exercice 2. Simpson

(a) Écrire un algorithme direct pour la méthode de Simpson. Compter le nombre d'opérations. Rappel :

$$\int_{x_i}^{x_{i+1}} f(t) dt \simeq \frac{h}{6} \left(f(x_i) + 4 * f\left(\frac{x_i + x_{i+1}}{2}\right) + f(x_{i+1}) \right).$$

(b) Écrire un algorithme amélioré. Compter le nombre d'opérations.

Exercice 3. Python

- (a) Écrire deux routines `trapez(f, a, b, n)` et `simsimp(f, a, b, n)`.
- (b) Essayer de « vectoriser » vos routines pour améliorer les performances. En effet Numpy et Scipy permettent l'appel `cos(x)` où `x` est un « array » et qui renvoie un array tandis que `np.sum(y)` renvoie la somme des éléments de `y`.
- (c) Tester vos routines sur des exemples d'intégrales dont vous pouvez calculer la valeur.
- (d) Soit $I(f) = \int_0^1 20(1-x^2)^3 dx$. Le but est de comparer les deux méthodes composites et de vérifier que l'erreur est conforme aux prévisions. Si $T(f, h)$ et $S(f, h)$ désignent respectivement les formules composites des trapèzes et de Simpson,

$$T(f, N) = h \left(\frac{f(x_0)}{2} + f(x_1) + \dots + f(x_{N-1}) + \frac{f(x_N)}{2} \right),$$

$$S(f, N) = \frac{h}{6} \left(f(x_0) + 2 \sum_{k=1}^{N-1} f(x_k) + 4 \sum_{k=0}^{N-1} f\left(\frac{x_k + x_{k+1}}{2}\right) + f(x_N) \right),$$

on rappelle qu'il existe ξ_1 et ξ_2 tels que

$$|I(f) - T(f, h)| = \frac{b-a}{12} h^2 |f''(\xi_1)|,$$

$$|I(f) - S(f, h)| = \frac{b-a}{180} \frac{h^4}{24} |f^{(4)}(\xi_2)|.$$

L'erreur est donc prévisible si f est assez régulière et à condition de connaître f'' ou $f^{(4)}$, ce qui n'est pas toujours le cas (on fait alors des méthodes dites adaptatives).

Avec Python, pour les valeurs $N = 2, 4, 8, \dots, 512, 1024$ calculer l'erreur exacte pour chaque méthode et tracer sur un même graphique à double échelle logarithmique $(n, \text{Erreur}(n))$. Tracer sur ce même graphique (n, n^{-2}) et (n, n^{-4}) . Conclure.

Exercice 4. Python Comparer vos routines avec `trapz` et `simps` de `scipy.integrate`.

3. MÉTHODE DE ROMBERG

La méthode de Romberg est un cas particulier de l'extrapolation de Richardson appliquée à la méthode composite des trapèzes et permet, à peu de frais, d'améliorer l'approximation. Nous admettrons que si f est suffisamment régulière alors pour tout $n \geq 1$ (attention ce n'est pas le N lié au pas de discrétisation $h = (b-a)/N$),

$$(1) \quad T(f, h) = \int_a^b f(t) dt + \alpha_1 h^2 + \alpha_2 h^4 + \alpha_3 h^6 + \dots + \alpha_n h^{2n} + O(h^{2n+2}).$$

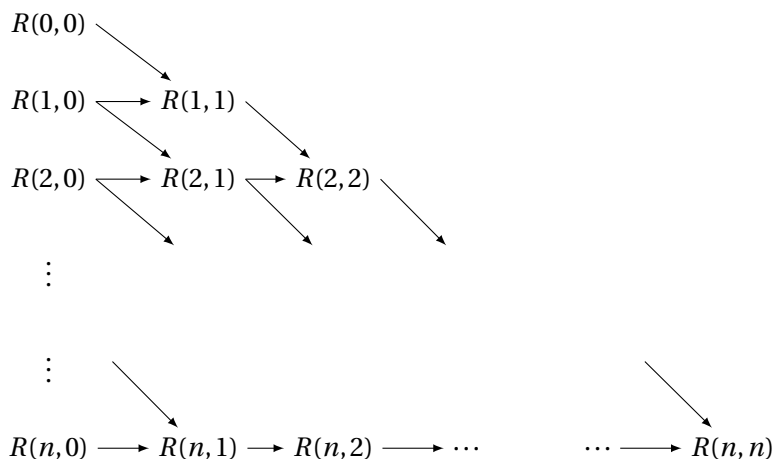
Les coefficients $\alpha_1, \alpha_2, \dots, \alpha_n$ sont évidemment inconnus et dépendent de f . Grosso modo l'idée du procédé d'extrapolation de Richardson est par exemple pour $n = 3$ d'évaluer $T(f, h)$ pour quatre valeurs différentes h_0, h_1, h_2, h_3 et de calculer en 0 la valeur du polynôme d'interpolation de Lagrange passant par $(h_i, T(f, h_i))$, $i \in \{0, 1, 2, 3\}$. Si tout se passe bien et comme par continuité $T(f, 0) = \int_a^b f(t) dt$, la valeur obtenue est plus précise que celles calculées en h_0, h_1, h_2, h_3 . Le calcul du polynôme d'interpolation de Lagrange n'est pas nécessaire, il suffit d'utiliser l'algorithme de Neville (version en un point des différences divisées de Newton).

La méthode de Romberg consiste à prendre $h_0 = h$, $h_1 = \frac{h}{2}$, $h_2 = \frac{h}{2^2}$, $\dots$, $h_n = \frac{h}{2^n}$ et les calculs sont assez simples. Soient $h = (b-a)/N$ et $n \geq 1$ fixés. On définit $R(i, k)$ pour $0 \leq k \leq n$ et $k \leq i \leq n$:

$$R(i, 0) = T(f, h2^{-i}), \quad 0 \leq i \leq n$$

$$R(i, k+1) = \frac{4^{k+1} R(i, k) - R(i-1, k)}{4^{k+1} - 1} \quad 0 \leq k \leq n-1 \text{ et } k+1 \leq i \leq n.$$

On présente généralement ces valeurs par le diagramme suivant,



Exercice 5. Pour $h = b-a$ et $n = 1$, vérifier que $R(1, 1)$ est la formule de quadrature élémentaire de Simpson.

Exercice 6. [Du calcul !] D'après la formule ((E)) et en ajoutant un indice

$$(2) \quad R(i, 0) = \int_a^b f(t) dt + \alpha_{1,0} (h2^{-i})^2 + \dots + \alpha_{n,0} (h2^{-i})^{2n} + O(h^{2n+2}).$$

Montrer par récurrence que pour tout $0 \leq k \leq n-2$ et $k+1 \leq i \leq n$

$$(3) \quad R(i, k+1) = \int_a^b f(t) dt + \alpha_{k+2,k+1} (h2^{-i})^{2(k+2)} + \dots + \alpha_{n,k+1} (h2^{-i})^{2n} + O(h^{2n+2})$$

et que

$$R(n, n) = \int_a^b f(t) dt + O(h^{2n+2}).$$

Exercice 7. [Python] Écrire une routine `romberg(f, a, b, N, n)` qui retourne le tableau des $R(i, k)$. Il faut remarquer que les $R(i, k)$ forment un tableau triangulaire (il y a donc une partie du tableau qui n'est pas utilisée). Tester cette routine sur un exemple avec $n = 5$ et vérifier que l'approximation s'améliore. Proposer une routine optimisée qui calcule $R(n, n)$. Pour la fonction $f(x) = \exp(1)$, $N = 1$ et $n = 5$ voici le tableau obtenu

$$\begin{pmatrix} 1.8591409 & 0 & 0 & 0 & 0 & 0 \\ 1.7539311 & 1.7188612 & 0 & 0 & 0 & 0 \\ 1.7272219 & 1.7183188 & 1.7182827 & 0 & 0 & 0 \\ 1.7205186 & 1.7182842 & 1.7182818 & 1.7182818 & 0 & 0 \\ 1.7188411 & 1.718282 & 1.7182818 & 1.7182818 & 1.7182818 & 0 \\ 1.7184217 & 1.7182818 & 1.7182818 & 1.7182818 & 1.7182818 & 1.7182818 \end{pmatrix}.$$

4. ÉQUATION INTÉGRALE DE FREDHOLM DE 2ÈME ESPÈCE, INTÉGRATION NUMÉRIQUE ET SYSTÈME LINÉAIRE

Soient $a < b$ et $I = [a, b]$ et deux applications continues, $K : I \times I \mapsto \mathbb{R}$ et $f : I \mapsto \mathbb{R}$. Cherchons une fonction u telle que

$$(E) \quad \forall x \in I, \quad u(x) = f(x) + \int_a^b K(x, t) u(t) dt$$

4.1. Indication pour la preuve de l'existence et l'unicité d'une solution de (E).. On suppose que $\forall (x, t) \in I \times I$ on a $K(x, t) \leq \alpha / (b - a)$ avec $\alpha < 1$. La preuve se fait (presque) comme pour l'existence d'un point fixe dans la cas réel. Soit $u_0 \equiv 0$ la fonction nulle. Pour tout $n \in \mathbb{N}$ soit $u_{n+1}(x) = f(x) + \int_a^b K(x, t) u_n(t) dt$. Les fonctions $f(\cdot)$ et $K(\cdot, \cdot)$ étant continues, la suite u_n est une suite parfaitement définie de fonctions continues. On note $\|g\|_\infty = \sup_{x \in I} |g(x)|$, montrer successivement,

- (a) Pour tout $n \in \mathbb{N}$, $\|u_{n+2} - u_{n+1}\|_\infty \leq \alpha \|u_{n+1} - u_n\|_\infty$.
- (b) Pour tout $n \in \mathbb{N}$, $\|u_{n+1} - u_n\|_\infty \leq \alpha^n \|u_1 - u_0\|_\infty$.
- (c) Pour tout $n, p \in \mathbb{N}$, $\|u_{n+p} - u_n\|_\infty \leq \alpha^n \|u_1 - u_0\|_\infty / (1 - \alpha)$.

En déduire que u_n est une suite de Cauchy dans $\mathcal{C}(I)$ muni de la norme de la convergence uniforme et que u_n converge uniformément vers une fonction u solution de (E). Montrer que cette solution est nécessairement unique (toujours sous l'hypothèse initiale faite sur la fonction K).

Remarque. Si $f \equiv 0$, $b - a = 1$ et $K \equiv 1$ toute fonction constante est solution.

4.2. Approximation d'une solution. L'idée est d'utiliser une formule de quadrature pour calculer $\int_a^b K(x, t) u(t) dt$, linéairement dépendante des valeurs de u aux points $x_0, \dots, x_n$, d'écrire (E) pour tout $x = x_i$ ($i \in \{0, \dots, n\}$) et de constater que l'on obtient un problème linéaire.

Choisissons comme formule de quadrature la formule des trapèzes. Soient $n \geq 1$ et $x_i = a + (b - a)i/n$ pour tout $i \in \{0, \dots, n\}$. Pour tout $i \in \{0, \dots, n\}$ écrivons l'approximation

$$\int_a^b K(x_i, t) u(t) dt \approx \frac{b-a}{n} \left(\frac{1}{2} K(x_i, x_0) u(x_0) + K(x_i, x_1) u(x_1) + \dots + K(x_i, x_{n-1}) u(x_{n-1}) + \frac{1}{2} K(x_i, x_n) u(x_n) \right)$$

La version discrète de (E) aux points x_i nous donne le système linéaire, $\forall i \in \{0, \dots, n\}$,

$$(4) \quad u(x_i) - \frac{b-a}{n} \left(\frac{1}{2} K(x_i, x_0) u(x_0) + K(x_i, x_1) u(x_1) + \dots + K(x_i, x_{n-1}) u(x_{n-1}) + \frac{1}{2} K(x_i, x_n) u(x_n) \right) = f(x_i).$$

Posons (attention nous avons un système linéaire de taille $n+1$)

$$(5) \quad U = \begin{pmatrix} u(x_0) \\ \vdots \\ u(x_n) \end{pmatrix}, \quad b = \begin{pmatrix} f(x_0) \\ \vdots \\ f(x_n) \end{pmatrix}, \quad \left\{ \begin{array}{l} M = (m_{ij})_{1 \leq i, j \leq n+1} \text{ avec } m_{i1} = \frac{1}{2} K(x_{i-1}, x_0), \\ m_{i, (n+1)} = \frac{1}{2} K(x_{i-1}, x_n), \quad m_{i,j} = K(x_{i-1}, x_{j-1}) \text{ si } 2 \leq j \leq n. \end{array} \right\},$$

La version discrète (4) se traduit matriciellement par $(I - \frac{b-a}{n} M)U = b$.

Exercice. Sous l'hypothèse en 2.1 sur $K(\cdot, \cdot)$, montrer la convergence de l'approximation quand n tend vers $+\infty$. (matrice à diagonale strictement dominante, u (exacte) est solution d'un système linéaire perturbé, d'où conditionnement...)

4.3. Avec Python. Écrire un programme Python `fredholm(f,K,a,b,n)` qui calcule l'approximation de u . Organisation :

- créer une matrice carrée A de taille $n+1$ et un vecteur b de taille $n+1$, tous deux remplis de zéros.
- à l'aide de boucle(s) et de (5) « remplir » A et b de façon à obtenir $A = I - (b-a)M/n$ et $b = b$. (évaluation numérique)
- résoudre le système avec `linalg.solve`. Soit U le vecteur solution.
- tracer l'approximation en traçant la ligne brisée $[[x_0, U[1]], [x_1, U[2]], \dots, [x_n, U[n+1]]]$.

Exemples. a) $a = 0$, $b = 1$, $f(x) = x^2$ et $K(x, t) = \exp|x - t|$.

b) $a = 0$, $b = 1$, $f(x) = \frac{3}{4} \exp(x)$, $K(x, t) = \exp(x - t)/4$ (solution exacte unique $u(x) = \exp(x)$)

c) $a = 0$, $b = 2\pi$, $f(x) = 1$, $K(x, t) = \frac{1}{10} \cos(x - t)$ (solution exacte unique $u(x) \equiv 1$).

d) $a = 0$, $b = 1$, $f(x) = \cos(x\pi)(1 + 1/(2\pi^2)) + 1/(2\pi^2)$, $K(x, t) = \sup(x, t)/2$.

4.4. Utilisation de la méthode de Simpson. Faire 2.2 et 2.3 avec la méthode de Simpson et comparer la précision.

Exercice 8. [Des idées pour la suite]

- un autre choix de suite h_i (suite de Burlisch)
- méthode adaptative de Simpson