

X – Algorithmes itératifs fondamentaux

Les algorithmes exposés dans ce chapitre ont tous une complexité linéaire. En effet, ils sont constitués de boucles simples qui contiennent un nombre fixé d'opérations et dont le nombre d'itérations égale le paramètre de complexité.

1/ Manipulations élémentaires de listes

1.1) Somme et produit

Les calculs de sommes sont extrêmement fréquents et l'outil informatique s'avère bien souvent indispensable pour cela. L'algorithme suivant doit donc être parfaitement maîtrisé.

<pre>L : liste de nombres s ← 0 POUR e DANS L s ← s+e Renvoyer s</pre>	<pre>def Somme(L): s=0 for e in L: s=s+e return s</pre>
--	---

On part ainsi d'une variable initialement nulle et on lui ajoute un à un les termes de la somme. Pour les produits, c'est le même principe en partant d'une variable égale à 1.

<pre>L : liste de nombres p ← 1 POUR e DANS L p ← p× e Renvoyer p</pre>	<pre>def Produit(L): p=1 for e in L: p=p*e return p</pre>
---	---

1.2) Moyenne et variance

La moyenne m et la variance V d'une liste de nombres $L = [e_1, e_2, \dots, e_n]$ sont respectivement définies par $m = \frac{1}{n} \sum_{i=1}^n e_i$ et $V = \frac{1}{n} \sum_{i=1}^n (e_i - m)^2$. Voici comment on peut les calculer :

<pre>L : liste de nombres m ← 0, V ← 0, n ← taille(L) POUR e DANS L m ← m+e POUR e DANS L V ← V+(e-m)² Renvoyer (m/n,V/n)</pre>	<pre>def MoyVar(L): m , V, n = 0 , 0, len(L) for e in L: m=m+e for e in L: V=V+((e-m)**2) return (m/n,V/n)</pre>
--	--

Remarque : on effectue les divisions par n à la fin pour minimiser les opérations.

2/ Calcul des termes d'une suite récurrente

C'est un classique incontournable car les suites récurrentes sont partout en informatique.

2.1) Les récurrences simples

Le problème

Soient E un ensemble, $a \in E$ et f une application de $\mathbb{N} \times E$ dans E . On cherche alors à calculer les termes successifs de la suite $(u_n)_{n \in \mathbb{N}}$ d'éléments de E définie par
$$\begin{cases} u_0 = a \\ u_{n+1} = f(n, u_n) \end{cases}$$

Les solutions

Voici deux façons de calculer et stocker les $N + 1$ premiers termes de cette suite $(u_n)_{n \in \mathbb{N}}$ dans une liste U , de sorte que $U[n] = u_n$ pour tout $n \in \llbracket 0; N \rrbracket$.

- 1) On les calcule les uns après les autres (ce sont les valeurs prises par u) en les ajoutant au fur et à mesure à une liste U correctement initialisée.
- 2) On crée un vecteur nul U de longueur $N + 1$, dont on remplace les termes un à un par ceux de la suite en recopiant simplement la définition mathématique de la suite.

$\begin{aligned} &U \leftarrow [a] \\ &u \leftarrow a \\ &\text{Pour } n \text{ allant } 0 \text{ à } N-1 \\ &\quad u \leftarrow f(n, u) \\ &\quad \text{ajouter } u \text{ à la fin de } U \end{aligned}$	$\begin{aligned} &U = [a] \\ &u = a \\ &\text{for } n \text{ in range}(N): \\ &\quad u = f(n, u) \\ &\quad U.append(u) \end{aligned}$	$\begin{aligned} &\text{import numpy as np} \\ &U = \text{np.zeros}(N+1) \\ &U[0] = a \\ &\text{for } n \text{ in range}(N): \\ &\quad U[n+1] = f(n, U[n]) \end{aligned}$
--	---	---

Exemple : étudions de cette façon la suite $(u_n)_{n \in \mathbb{N}}$ définie par $u_0 = 1$ et $u_{n+1} = (n + 1) \sin(u_n)$.

2.2) Les récurrences multiples

Le principe

On peut introduire une suite auxiliaire à valeurs vectorielles $(v_n)_{n \in \mathbb{N}}$ qui vérifie une relation de récurrence simple. En pratique, il vaut mieux utiliser la seconde méthode et créer un vecteur nul dont on remplace les termes un par un par ceux de la suite.

Un exemple

Considérons la suite de Fibonacci $(u_n)_{n \in \mathbb{N}}$ définie par $u_0 = 0$, $u_1 = 1$ et $u_{n+2} = u_{n+1} + u_n$.

$\begin{aligned} &U = [0, 1] \\ &u, v = 0, 1 \quad (u : u_n / v : u_{n+1}) \\ &\text{for } n \text{ in range}(N-1): \\ &\quad u, v = v, u+v \\ &\quad U.append(v) \end{aligned}$	$\begin{aligned} &\text{import numpy as np} \\ &U = \text{np.zeros}(N+1) \\ &U[0], U[1] = 0, 1 \\ &\text{for } n \text{ in range}(N-1): \\ &\quad U[n+2] = U[n+1] + U[n] \end{aligned}$
--	---

3/ Calculs approchés d'intégrales

Soit une fonction f définie sur l'intervalle $[a; b]$. On cherche son intégrale sur $[a; b]$.

3.1) La méthode des rectangles

C'est l'approche la plus simple du calcul intégral. Elle est directement liée à la définition de l'intégrale en tant que calcul d'aire et à la notation $I = \int_a^b f(x) dx$. L'idée est la suivante :

- ❶ on découpe l'intervalle $[a; b]$ en n parties en posant $x_k = a + k(b-a)/n$ pour $k \in \llbracket 0; n \rrbracket$
- ❷ pour $k \in \llbracket 0; n-1 \rrbracket$, on approche la fonction f sur l'intervalle $[x_k; x_{k+1}]$ par $f(x_k)$
- ❸ l'intégrale I admet alors pour valeur approchée la somme de Riemann à gauche

$$R_n(f) = \frac{b-a}{n} \sum_{k=0}^{n-1} f(x_k) = h \sum_{k=0}^{n-1} f(a + kh) \quad \text{où } h = \frac{b-a}{n}$$

Lorsque f est continue sur l'intervalle $[a; b]$, le Théorème de Riemann nous assure que

$$\lim_{n \rightarrow +\infty} R_n(f) = \int_a^b f(t) dt$$

Si la fonction f est en outre de classe $\mathcal{C}^1$ et que $|f'| \leq M_1$ sur $[a; b]$, on montre que

$$\forall n \geq 1 \quad \left| \int_a^b f(t) dt - R_n(f) \right| \leq \frac{M_1 (b-a)^2}{2n}$$

Ainsi, l'erreur commise lors de l'approximation est en $1/n$.

Remarque : on a des résultats similaires avec les somme de Riemann à droite, obtenues en remplaçant $f(x_k)$ par $f(x_{k+1})$ dans l'approximation de f .

L'algorithme est très simple, puisqu'il utilise un bête calcul de somme :

f, a, b, n : fonction, réel, réel, entier $S \leftarrow 0$ $h \leftarrow (b-a)/n$ POUR k ALLANT DE 0 A $n-1$ $S \leftarrow S + f(a + kh)$ Renvoyer $h \times S$	<pre>def Rectangles(f, a, b, n): S=0 h=(b-a)/n for k in range(n): S=S+f(a+k*h) return h*S</pre>
--	---

Exemple : appliquer ceci au calcul de $\int_0^1 \frac{dt}{1+t^2}$. On représentera l'erreur commise en fonction de n puis de $1/n$ afin d'en proposer une estimation.

3.2) La méthode du point-milieu

On reprend l'idée précédente, en approchant la fonction f sur l'intervalle $[x_k; x_{k+1}]$ non plus par $f(x_k)$ ou par $f(x_{k+1})$ mais par $f\left(\frac{x_k+x_{k+1}}{2}\right)$. On peut aisément voir sur un dessin que l'on obtient ainsi une approximation plus précise de l'intégrale, donnée par

$$M_n(f) = \frac{b-a}{n} \sum_{k=0}^{n-1} f\left(\frac{x_k+x_{k+1}}{2}\right) = h \sum_{k=0}^{n-1} f\left(a + \left(k + \frac{1}{2}\right) h\right)$$

Si la fonction f est de classe $\mathcal{C}^2$ et que $|f''| \leq M_2$ sur $[a; b]$, alors

$$\forall n \geq 1 \quad \left| \int_a^b f(t) dt - M_n(f) \right| \leq \frac{M_2 (b-a)^3}{24n^2}$$

Cette fois, l'erreur commise lors de l'approximation est en $1/n^2$, comme on peut le voir en reprenant l'exemple précédent.

3.3) La méthode des trapèzes

Les méthodes précédentes consistaient à approcher la fonction f par une fonction constante sur l'intervalle $[x_k; x_{k+1}]$. Graphiquement, la courbe de f était approchée par une courbe en escalier et le domaine sous la courbe par une réunion de rectangles.

Pour être plus précis, on va maintenant approcher f sur ce même intervalle par une fonction affine qui coïncide avec la fonction f aux points x_k et x_{k+1} . D'un point de vue graphique, la courbe de f est alors approchée par une ligne brisée, et le domaine sous la courbe par une réunion de trapèzes dont l'aire totale est

$$T_n(f) = \frac{b-a}{n} \sum_{k=0}^{n-1} \left[\frac{f(x_k) + f(x_{k+1})}{2} \right] = h \left[\frac{f(a) + f(b)}{2} + \sum_{k=1}^{n-1} f(a + kh) \right]$$

Si f est de classe $\mathcal{C}^2$ et que $|f''| \leq M_2$ sur $[a; b]$, on prouve que

$$\forall n \geq 1 \quad \left| \int_a^b f(t) dt - T_n(f) \right| \leq \frac{M_2 (b-a)^3}{12n^2}$$

L'erreur commise lors de l'approximation est encore en $1/n^2$.

3.4) Pour aller plus loin

Les calculatrices emploient en général la **méthode de Simpson**, qui consiste à approcher la fonction f sur chaque intervalle $[x_k; x_{k+1}]$ par un polynôme du second degré coïncidant avec elle aux points x_k , $\frac{x_k+x_{k+1}}{2}$ et x_{k+1} . Assez étonnamment, l'approximation obtenue est

$$S_n(f) = \frac{2}{3} M_n(f) + \frac{1}{3} T_n(f)$$

et l'erreur commise lors de l'approximation est en $1/n^4$.

Enfin, la **méthode de Romberg** permet encore d'accélérer la convergence de ces procédés.