ALGORITMOS EN ÁLGEBRA LINEAL

Notas de curso (UBA - 2do cuatrimestre de 2005)

http://atlas.mat.ub.es/personals/sombra/curso.html

Michelle Schatzman

Martín Sombra

Institut Camille Jordan (Mathématiques), Université de Lyon 1 ; 43 Bd. du 11 Novembre 1918, 69622 Villeurbanne Cedex, Francia

Universitat de Barcelona, Departament d'Àlgebra i Geometria ; Gran Via 585, 08007 Barcelona, España

CHAPITRE 3

Représentation des données

1. Représentation exacte

L'avantage évident des algorithmes travaillant avec la représentation exacte des données c'est qu'on n'a pas de perte de précision ; ou tout au moins la seule instabilité est celle induite par la précision de l'entrée et non pas par les troncages internes de l'algorithme. Le désavantage est que la taille des calculs intermédiaires peut exploser, conduisant souvent à une augmentation sensible du temps d'exécution et/ou de l'occupation de place mémoire.

La multiplication des entiers est, à une constante près, du même coût que la division des entiers et du calcul de réciproques. Le coût du gcd est $O(\log(N)$ fois le coût de la multiplication. Introduisons un peu de notation :

$M(N)$	la complexité de la multiplication de deux entiers de taille N ;
$D(N)$	la complexité de la division avec reste de deux entiers de taille N ;
$R(N)$	la complexité de N bits du réciproque d'un entier de taille N ;
$GCD(N)$	la complexité du gcd de deux entiers de taille N.

L'algorithme de Schönhage-Strassen montre que

$$M(N) = O(N \log(N) \log(\log(N))).$$

On a

THÉORÈME 1.1. *$M(N)$, $R(N)$ et $D(N)$ sont comparables à un facteur constant près ; en particulier*

$$R(N), D(N) = O(N \log(N) \log(\log(N))).$$

Dans les exercices on fera quelques unes de ces comparaisons ; on renvoie à [**1**, Ch. 9] pour la démonstration de la division avec reste. La complexité du gcd nécessite d'une analyse plus fine, qui est développée dans la même référence :

THÉORÈME 1.2. *$GCD(N) = O(M(N) \log(N))$.*

Soit $\xi \in \mathbb{Q}$ un nombre rationnel et soit $\xi = p/q$ sa *représentation réduite* avec $p \in \mathbb{Z}$ et $q \in \mathbb{N}^\times$ premiers entre eux. La *hauteur exponentielle* de ξ est

$$H(\xi) := \max\{|p|, q\}.$$

On considérera surtout la *hauteur (logarithmique)* de ξ définie par

$$h(\xi) := \log_2(H(\xi)) = \max\{\log_2(|p|), \log_2(q)\} \in \mathbb{R}_+.$$

Ceci est la hauteur considérée usuellement en théorie des nombres, qu'on préférera à la complexité binaire

$$\tau(\xi) := \tau(p) + \tau(q) = \lfloor \log_2(|p|) \rfloor + 1 + \lfloor \log_2(q) \rfloor + 1.$$

Ces notions sont comparables :

$$h(\xi) \leq \ell(\xi) \leq 2\, h(\xi) + 2$$

mais la hauteur a un comportement plus agréable par rapport aux opérations arithmétiques. Soient $\xi, \eta \in \mathbb{Q}^\times$, alors

(1) $h(\xi+\eta) \leq h(\xi) + h(\eta) + \log(2)$.
Si $\xi, \eta \in \mathbb{Z}$, alors $h(\xi+\eta) \leq \max\{h(\xi), h(\eta)\} + 1$.

(2) $h(\xi \cdot \eta^{\pm 1}) \leq h(\xi) + h(\eta)$.

Comme conséquence de notre étude du coût des opérations arithmétiques entre nombres entiers, on voit que la complexité binaire des opérations arithmétiques entre nombres rationnels de hauteurs bornées par h est

$$O(h \log(h) \log(\log(h)))$$

pour l'addition, la soustraction, la multiplication et la division. Si l'on veut travailler uniquement avec des expressions réduites, il faut simplifier les facteurs redondants, ce qui demande un calcul de gcd. Le coût binaire des opérations arithmétiques avec représentation réduite des rationnels est donc

$$O(h \log^2(h) \log(\log(h))).$$

La notation $f = O^*(g)$ veut dire "à des facteurs logarithmiques près", c'est-à-dire $f \leq c\, g \log^\nu(g)$ pour certains $c, \nu > 0$.

COROLLAIRE 1.3. *Soit A un algorithme sur $\mathbb{Q}$, et notons $h_A(\tau)$ la hauteur maximale des calculs intermédiaires effectués par A sur une entrée de taille τ, alors*

$$\mathcal{C}_A(\tau) \leq O^*(\mathcal{C}_A(\mathbb{Q}; \tau) \cdot h_A(\tau)).$$

Posons $\mathbb{Q}_{\leq H} := \{\xi \in \mathbb{Q} : H(\xi) \leq x\}$ la filtration de la droite rationnelle par la hauteur. Son aspect est vachement différent de la droite flottante. Pour $H = 4$ les points sont

$$\frac{0}{1}, \frac{1}{1}; \frac{1}{2}, \frac{2}{1}; \frac{1}{3}, \frac{2}{3}, \frac{3}{2}, \frac{3}{1}; \frac{1}{4}, \frac{3}{4}, \frac{4}{3}, \frac{4}{1};$$

et les négatifs correspondants. La figure suivante représente ces rationnels :

FIG. 1. Les rationnels de hauteur exponentielle au plus 4.

EXERCICE 3.1. ◁ Un comptage naïf donne l'estimation

$$\mathrm{Card}(\mathbb{Q}_{\leq H}) \leq (2H+1)H;$$

le but de cet exercice est de démontrer l'asymptotique

$$\mathrm{Card}(\mathbb{Q}_{\leq H}) = \frac{12}{\pi^2} H^2 + O(H \log(H)) = 1,216 H^2 + O(H \log(H)). \tag{1}$$

Un couple $z = (p, q) \in \mathbb{Z}^2 \setminus \{(0,0)\}$ est *visible* si l'intérieur du segment $\overline{0\,z}$ ne contient aucun point de $\mathbb{Z}^2$, ce qui équivaut à ce que p, q soient premiers entre eux.

(1) Montrer que $\mathbb{Q}^\times$ s'identifie aux points visibles depuis le $(0,0)$ quand on "regarde" vers le nord. Dans ce modèle, la hauteur correspond à la norme ℓ^∞.

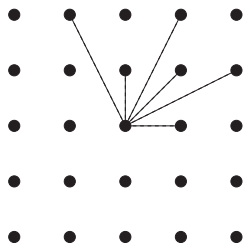

FIG. 2. Points visibles depuis le $(0,0)$

(2) Posons

$$m(t) := \#\{(p,q) \in \mathbb{Z}^2 \setminus \{(0,0)\} \,:\, |p|,|q| \leq t\},$$

$$n(t) := \#\{(p,q) \in \mathbb{Z}^2 \setminus \{(0,0)\} \,:\, \gcd(p,q)=1,\ |p|,|q| \leq t\}.$$

Montrer que $m(t) = (2t+1)^2 - 1$ et que

$$m(t) = \sum_{d \geq 1} n(t/d).$$

(3) Soit $\mu : \mathbb{N} \to \mathbb{N}$ la *fonction de Möbius*, définie par $\mu(d) = (-1)^k$ si $d = p_1 \cdots p_k$ avec $p_1, \ldots, p_k$ nombres premiers différents deux à deux, et $\mu(d) = 0$ sinon. Prouver la formule d'inversion

$$n(t) = \sum_{d \geq 1} \mu(d)\, m(t/d).$$

(4) En déduire l'asymptotique pour $t \to \infty$

$$n(t) = 4 \Big(\sum_{d \geq 1} \frac{\mu(d)}{d^2} \Big) t^2 + O(t \log(t)).$$

(5) Soit

$$\zeta(s) := \sum_{n \geq 1} n^{-s} = \prod_{p \text{ premier}} \frac{1}{1-p^{-s}}, \quad (Re(s) > 1)$$

la fonction zeta de Riemann. Observer que

$$\sum_{d \geq 1} \frac{\mu(d)}{d^2} = \frac{1}{\zeta(2)};$$

et que $\mathrm{Card}(\mathbb{Q}_{\leq H}) = \frac{1}{2}\, n(t) - 1$; en déduire (1), en utilisant que $\zeta(2) = \pi^2/6$.

▷

2. Nombres flottants

Références pour cette section : [**2**], [**3**]. La forme générale des nombres flottants est :

$$f = \pm .d_1 d_2 \ldots d_t \times \beta^e, \quad 0 \leq d_i < \beta, \quad d_1 \neq 0, \quad L \leq e \leq U; \tag{2}$$

$d_1 d_2 \ldots d_t$ est la *mantisse*, t la *précision*, β la *base*, L le *dépassement inférieur* (en anglais : *underflow*) et U le *dépassement supérieur* (en anglais : *overflow*). On impose $0 \leq d_i < \beta$ et $d_1 \neq 0$, ce qui assure l'unicité de l'expression.

Pour $\beta \geq 2, t \geq 1$ et $(L,U) \in \mathbb{Z}^2$ donnés, on désigne par

$$F(\beta, t, L, U)$$

l'ensemble des flottants associés plus le 0. Prenons un cas particulier très simple :

$$\beta = 2, \quad t = 3, \quad U = -1, \quad L = 1. \tag{3}$$

Par conséquent :

$$1 \leq d_1 \leq \beta - 1 \Longrightarrow d_1 = 1.$$

Les mantisses possibles sont

$$0.100, \quad 0.101, \quad 0.110, \quad 0.111,$$

soit les fractions $1/2, 5/8, 3/4, 7/8$, puis on a le droit de les multiplier par ± 2, ± 1, ou $\pm 1/2$. Voici la représentation graphique de ces flottants :

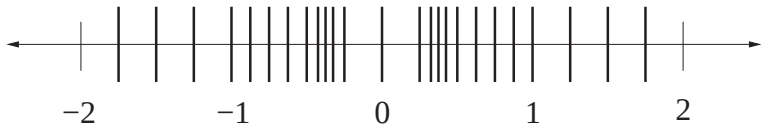

FIG. 3. Les flottants définis par les données (3).

Cette droite est pleine de trous, on a donc besoin d'une fonction arrondi. Soit $F := F(\beta, l, L, U)$, si $f \in F \setminus \{0\}$ alors

$$m \leq |f| \leq M$$

avec

$$m = \beta^{L-1}, \quad M = \beta^U(1 - \beta^{-t}). \tag{4}$$

Autre nom de m : ε de la machine, et de M : capacité de la machine.

EXERCICE 3.2. ◁ Vérifier les formules dans (4). ▷

Valeurs typiques de $(\beta,\ t,\ L,\ U)$: IBM 370 (une antiquité) : $(16, 14, -64, 43)$; Cray 1 (pas tout jeune, mais nettement plus récent) : $(2, 48, -16384, 8191)$.

Aujourd'hui le *standard IEEE pour l'arithmétique binaire* est le plus répandu. Il est utilisé dans les workstations Sun, DEC, HP, IBM, et dans toutes les PCs. L'arithmétique IEEE inclut deux systèmes de flottants : *simple précision* (32 bits) et *double précision* (64 bits).

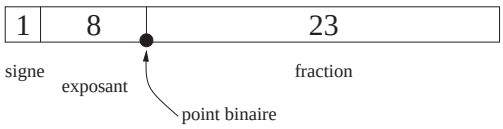

FIG. 4. Standard IEEE simple précision.

Dans le format simple précision, le signe s est codé par 1 bit, l'exposant e par 8 bits, et la fraction q par les 23 bits restants ; le nombre codé est

$$(-1)^s(1 + q)2^{e-127}.$$

Puisque la base es 2, le premier bit de la mantisse sera toujours 1, donc on n'a pas besoin de le coder. En passant à la représentation standard des flottants (1), on a

$$t = 24, \quad \beta = 2, \quad L = -126, \quad U = 129.$$

L'erreur relative maximale est

$$2^{-24} \cong 6 \cdot 10^-8$$

et le rang va de $2^{-127} \cong 6 \cdot 10^{-38}$ jusqu'à $2^{129}(1 - 2^{-24}) \cong 7 \cdot 10^{39}$.

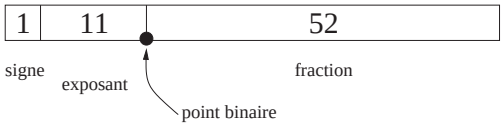

FIG. 5. Standard IEEE double précision.

Dans le format double précision, le signe s est codé par 1 bit, l'exposant e par 11 bits, et la fraction q par les 52 bits restants ; le nombre codé est

$$(-1)^s(1+q)2^{e-1023}.$$

Dans la représentation standard des flottants on a

$$t = 53, \quad \beta = 2, \quad L = -1022, \quad U = 1026.$$

L'erreur relative maximale est

$$2^{-53} \cong 6 \cdot 10^-16$$

et le rang va de $2^{-1022} \cong 2 \cdot 10^{-308}$ jusqu'à $2^{129}(1-2^{-24}) \cong 7 \cdot 10^{309}$.

Le modèle d'arithmétique des flottants, et en particulier du standard IEEE, est assez compliqué dans les détails. Ici on présentera une version simplifiée mais suffisante à nos besoins.

Une fonction

$$\mathrm{fl} : \mathbb{R} \to F(\beta, t, L, U) \cup \{E\}$$

qui envoie les réels sur les flottants est une fonction *arrondi* si elle vérifie :

(1) $\mathrm{fl}([-M, M]) \subset F(\beta, t, L, U)$;
(2) pour $x \in \mathbb{R}$ soient $f, f' \in F(\beta, t, L, U)$ tels que $x \in [f, f']$, alors $\mathrm{fl}(x) \in [f, f']$ (en particulier, fl laisse F invariant) ;
(3) si $|x| > M$ alors $\mathrm{fl}(x) = E$, si $|x| < m$ alors $\mathrm{fl}(x) = 0$; dans les deux cas on dit que x dépasse la capacité de la machine.

Dans le cas de l'*arithmétique arrondie* (qui est celle utilisée par le standard IEEE), $\mathrm{fl}(x)$ est défini comme le nombre dans F le plus proche de x ; si x est exactement à mi-chemim entre deux flottants, on choisit celui qui est le plus proche de 0.

Soit $\square$ une des quatre opérations arithmétiques $+, -, \times, /$; le modèle du calcul revient à supposer que la version "ordinateur" de chacune de ces quatre opérations est donnée par $\mathrm{fl}(a \square b)$.

EXEMPLE 2.1. Addition dans le système flottant jouet (3) :

$$\begin{aligned} 0.100 \times 2^1 + 0.110 \times 2^{-1} &= 0.10000 \times 2^1 + 0.00110 \times 2^1 \\ &= 0.10110 \times 2^1 \\ &\to \mathrm{fl}(0.100 \times 2^1 + 0.110 \times 2^{-1}) = 0.101 \times 2^1 \end{aligned}$$

Cependant les opérations sur les flottants ne sont pas nécessairement associatives. Dans le système jouet avec $\beta = 2$, $t = 3$, $L = -1$ et $U = 2$, si on utilise l'arithmétique tronquée :

$$\mathrm{fl}(0.1 \times 2^{-1} + 0.100 \times 2^2) = \mathrm{fl}((0.000100 + 0.100000) \times 2^2) = 0.100 \times 2^2;$$

et donc

$$\mathrm{fl}(\mathrm{fl}((0.1 \times 2^{-1} + 0.100 \times 2^2) + \mathrm{fl}(-0.100 \times 2^2))) = 0.$$

En revanche,

$$\mathrm{fl}(0.100 \times 2^2 + -0.100 \times 2^2) = 0,$$

si bien que

$$\mathrm{fl}(0.100 \times 2^{-1} + \mathrm{fl}(0.100 \times 2^{2} + (-0.100 \times 2^{2}))) = 0.100 \times 2^{-1}.$$

On peut montrer que l'opérateur fl satisfait

$$\mathrm{fl}(x) = x(1+\varepsilon), \quad |\varepsilon| \leq \mu,$$

avec $\mu = \mu(F) = \beta^{1-t}/2$. Par conséquent $\mathrm{fl}(a \square b) = (a \square b)(1+\varepsilon)$, $|\varepsilon| \leq \mu$ et donc en termes d'erreur relative, chaque opération arithmétique *individuelle* en flottant a un comportement satisfaisant :

$$\frac{|\mathrm{fl}(a \square b) - (a \square b)|}{a \square b} \leq \mu, \quad \text{si } a \square b \neq 0.$$

Tous les nombres flottants occupent la même place mémoire dans la machine. Le coût d'un calcul dépend des facteurs suivants :

(1) le nombre d'opérations arithmétiques $\oplus, \ominus, \otimes, \oslash$ et de tests $x < y$;

(2) la lecture et écriture en mémoire ;

(3) la place mémoire.

Dans la pratique de l'analyse numérique, on considère surtout (1) et (3).

3. Perte de précision catastrophique

Si on additionne deux nombres flottants de signe opposé et de taille comparable, on n'a plus guère de chiffres significatifs. L'un des meilleurs exemples mettent ce phénomène en évidence est le calcul de l'exponentielle au moyen de son développement en série entière :

$$\exp(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!}.$$

On utilise l'algorithme suivant, écrit en `scilab` (`http://www.scilab.org/`), qui est un clone gratuit de matlab :

```
x=input("donner la valeur de l'argument");
n=input("donner le nombre de termes apres 1");
s=1;y=1;
for j=1:n,
    y=y*x/j;s=s+y;
end
reponse=[s, exp(x),abs(s-exp(x))]
```

L'exécution de ce programme pour $x = 1$ et $n = 10$ donne

sommation	exponentielle	différence
2.7182818	2.7182818	$2.731E-08$

On refait le même essai avec $x = 10$ et $n = 100$; on trouve

sommation	exponentielle	différence
22026.466	22026.466	$7.276E-12$

Que se passe-t-il si l'on passe à des exposants négatifs ? Pour $x = -1$ et $n = 10$, on trouve

sommation	exponentielle	différence
0.3678795	0.3678794	$2.311E-08$

Jusqu'ici, tout va bien. Passons à $x = -10$ et $n = 10$:

sommation	exponentielle	différence
1342.5873	0.0000454	1342.5873

C'est mauvais ; peut-être n'a-t-on pas assez pris de termes... Pour le même $x = 10$, prenons successivement $n = 20$, $n = 30$, $n = 40$:

exponentielle	$n = 20$	différence	$n = 30$	différence
0.0000454	13.396866	13.396821	0.0009703	0.0009249

exponentielle	$n = 40$	différence
$0.2061154E-08$	0.0000454	$2.412E-09$

Prenons $x = -20$, et de la même façon, faisons les calculs pour n prenant les valeurs de 40 à 80, de 20 en 20 :

exponentielle	$n = 40$	différence	$n = 60$	différence
$0.2061154E-08$	4442.0344	4442.0344	0.0000343	0.0000343

exponentielle	$n = 80$	différence
$0.2061154E-08$	$0.5621885E-08$	$0.3560731E-08$

Rien ne nous empêche de chercher à mettre encore plus de termes. Avec 100 termes, la somme calculée est toujours de $0.5621884E-08$, et nous voyons qu'il est impossible d'obtenir ne serait-ce qu'un seul chiffre significatif de $\exp(-20)$ par sommation de la série, en utilisant `scilab`.

Bibliographie

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley Publishing Co., Reading, Mass.-London-Amsterdam, 1975. Second printing, Addison-Wesley Series in Computer Science and Information Processing.

[2] James W. Demmel. *Applied numerical linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.

[3] Michelle Schatzman. *Analyse numérique*. InterEditions, Paris, 1991. Cours et exercices pour la licence. [Course and exercises for the bachelor's degree].