かけ算・わり算の深遠な世界

野呂 正行
(神戸大学理学部)

概要

大きな整数の乗除算

CPU が提供する機能は限られていて, 大きな整数の計算はプログラムとして実現しなければならない.

一見複雑に見えるが実は高速なアルゴリズムが考案され, 実用化されている.

いくつかの応用

大きな数の計算を基礎として, 多項式演算が可能になり, コンピュータ上で厳密な数学 (代数学) が扱える.

いくつか簡単な例を紹介する.

コンピュータにできること

指示された手順に従って, **データの操作**をすること
処理を行うのが CPU

データの操作

- メモリの読み書き
- 条件判断
- 演算
  データに対する加減乗除
- 入出力
  外部装置からのデータの読み出し, 外部装置へのデータの書き出し

もともとの使用目的は「計算」

人がやるには手間がかかり過ぎる計算を行う
性能はどんどん向上 $\Rightarrow$ 数万円のパソコンでも, 大変な計算ができる
しかし, CPU の機能自体は昔から変化ない

扱える数

限られた範囲の 「整数」と「浮動小数」
例えば, 整数は -2^{31} 以上 2^{31} 未満だけ, など

$\Rightarrow$ 科学技術計算では, 誤差の問題が避けられない

CPU の提供する演算機能

基本的には加減乗除

$$2000000000 + 2000000000 = -294967296$$

という結果を平気で返す. (**実は**, 2^{32} **で割った余り**)

こういうものの上で数学は可能か?

可能 — 基本演算を使ってプログラムを書けばよい

数学ソフトウェア： Mathematica (**高価**), Maple (**高価**), Risa/Asir (**無料**) etc.

整数の表現

いくらでも大きい整数

適当に区切って, 数列で表現 — 例えば 2^{32} 進数とする.

例: $B = 2^{32}$ **とすると**

$$111111111111111111111111111111111$$

$$= 140 \times B^3 + 1039118829 \times B^2$$

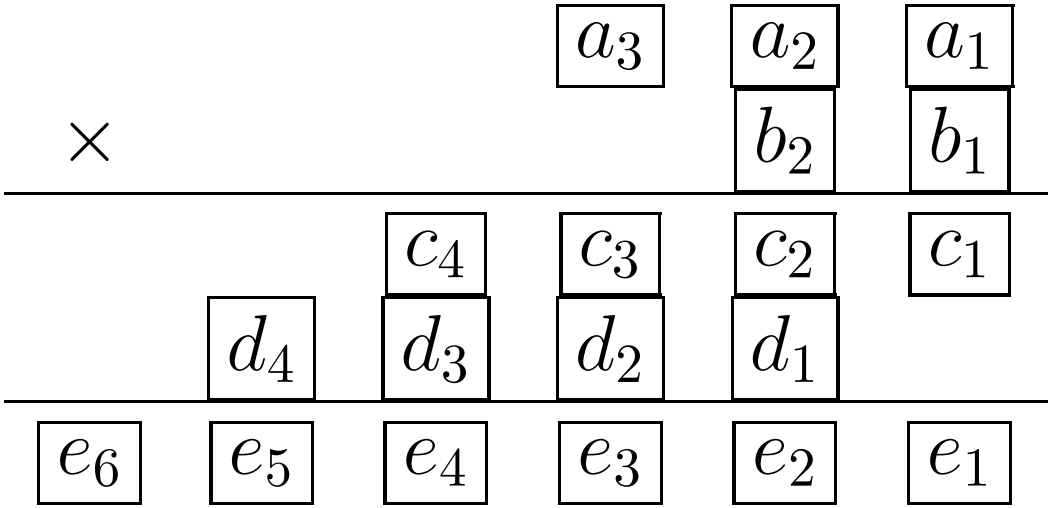
$$+3112500778 \times B + 3340530119$$

$$\Rightarrow (140|1039118829|3112500778|3340530119)$$

と表現できる.

かけ算のアルゴリズム — 筆算法

筆算



において，「1 **桁** $\times$ 1 **桁** $\rightarrow$ 2 **桁」を，**CPU **のかけ算で実行すればよい**

わり算のアルゴリズム — 筆算法

前処理

除数, 被除数に同じ数をかけて, 除数の最上桁をなるべく大きくする— 商を正確に見積もるため

商 1 桁の見積り

CPU 提供のわり算 (2 桁 わる 1 桁 $\longrightarrow$ 商, 余り)
$\Rightarrow$ 商の誤差は最大 2
さらに 1 桁余分に使うと誤差最大 1 (ほとんど正確)

あとは人間がやる筆算と同じ

「深遠」には程遠いが . . .

問題点

計算効率 (スピード) : 手間は, 桁数の積に比例する

N 桁 $\times$ N 桁なら手間は N^2

$\Rightarrow$ 桁数が大きくなるとどんどん大変になる.

アルゴリズムを工夫する

どんなに複雑でもかまわない

速ければよい

(計算するのはコンピュータなので)

高速アルゴリズム — Karatsuba 法

2 桁 × 2 桁 のかけ算

B **進数** $a = a_1B + a_0, b = b_1B + b_0$ **に対し**

$$ab = (\underline{a_1b_1})B^2 + (\underline{a_1b_0} + \underline{a_0b_1})B + \underline{a_0b_0}$$

と計算すると, 1 **桁の**かけ算は 4 回**. これを**

$$ab = (\underline{a_1b_1})B^2 + (\underline{(a_1 - a_0)(b_0 - b_1)} + a_1b_1 + a_0b_0)B + \underline{a_0b_0}$$

とすればかけ算は 3 回 **(** a_0b_0, a_1b_1 **を再利用)**

$\Rightarrow$ **これを繰り返すと,** n **桁のかけ算が** $n^{1.58}$ **に比例する手間で計算できる.**

例 — 2345×9876

$$
\begin{array}{rrrrr}
 & & 2 & 3 & 4 & 5 \\
\times & & 9 & 8 & 7 & 6 \\
\hline
 & & 3 & 4 & 2 & 0 \\
 & 3 & 4 & 2 & 0 & \\
 & 2 & 2 & 5 & 4 & \\
 & & 4 & 8 & 4 & \\
\hline
2 & 2 & 5 & 4 & & \\
\hline
2 & 3 & 1 & 5 & 9 & 2 & 2 & 0
\end{array}
\qquad
\begin{array}{rrrr}
 & & 4 & 5 \\
\times & & 7 & 6 \\
\hline
 & & 3 & 0 \\
 & 3 & 0 & \\
 & 2 & 8 & \\
 & & 1 & \\
\hline
2 & 8 & & \\
\hline
3 & 4 & 2 & 0
\end{array}
\qquad
\begin{array}{rrrr}
 & & 2 & 3 \\
\times & & 9 & 8 \\
\hline
 & & 2 & 4 \\
 & 2 & 4 & \\
 & 1 & 8 & \\
 & & 1 & \\
\hline
1 & 8 & & \\
\hline
2 & 2 & 5 & 4
\end{array}
$$

$$(23 - 45) \times (76 - 98) = (-22) \times (-22) = 484$$

実際のソフトウェアで比較

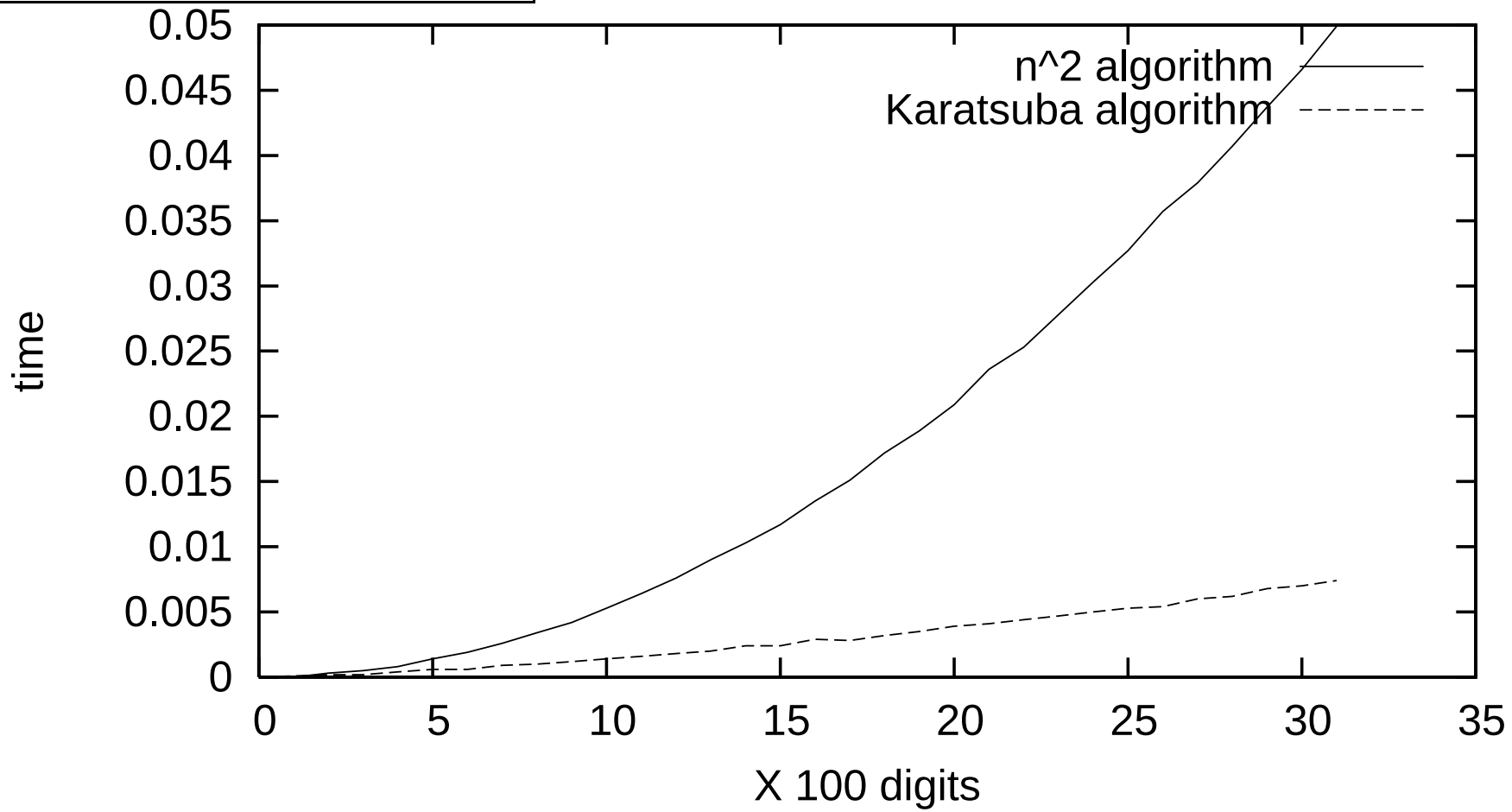
gmp (GNU MP library)

多倍長整数, 浮動小数演算のためのソフトウェア

- **フリーソフトウェア**
  ソースも公開
- **高性能**
  CPU **ごとに細かく最適化されている**
- **事実上の標準**
  Maple, Mathematica **でも使われている**.

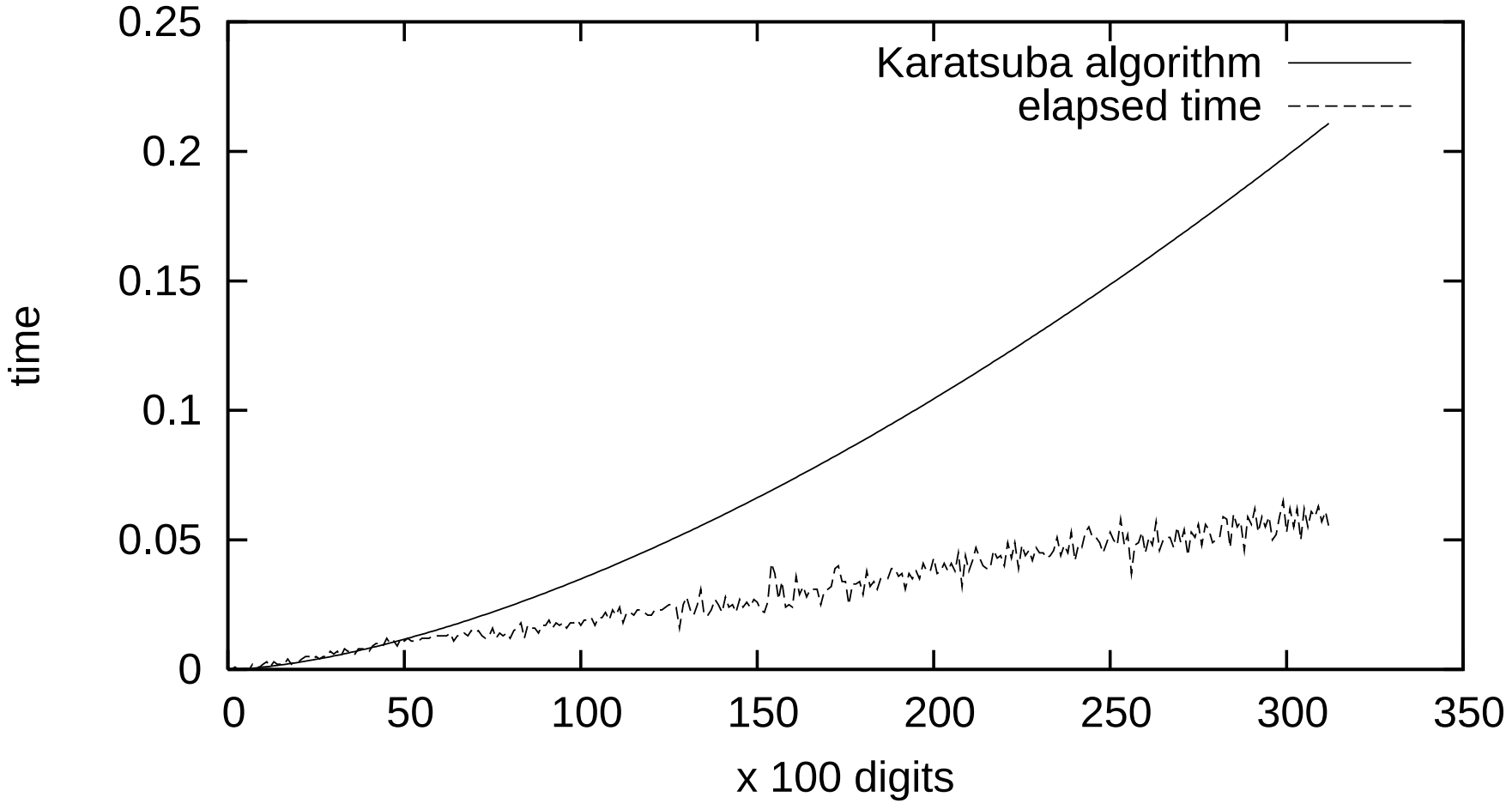
筆算法と Karatsuba 法の比較

2^{32} 進 n 桁 $\times$ n 桁



もっと桁数が多い場合

Karatsuba 法の推定値と実測値の比較



実測値が Karatsuba 法より高速なのはなぜか?

FFT 法

n 桁 $\times$ n 桁 $= 2n$ 桁
$\Rightarrow$ n に比例する程度の手間で計算できないか
$\Rightarrow$ $C \cdot n \log n$ 程度の手間のアルゴリズムがある.
(C は比例定数)

FFT 法の基本

多項式の積に持ち込む

整数を多項式で表現

B 進 n 桁の数 $a = (a_{n-1}a_{n-2}\cdots a_1a_0)_B$ は

$$a = a_{n-1}B^{n-1} + a_{n-2}B^{n-2} + \cdots + a_1B + a_0$$

のことである.

例: $12345 = 1\times 10^4 + 2\times 10^3 + 3\times 10^2 + 4\times 10 + 5$

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0$$

とおけば,

$$a = f(B)$$

である.

整数の積から多項式の積へ

同様に $b = (b_{n-1}b_{n-2}\cdots b_1 b_0)_B$ **に対し**

$$g(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_1 x + b_0$$

とおけば

$$ab = f(B)g(B)$$

である. よって,

$$h(x) = f(x)g(x)$$

とおけば

$$ab = h(B).$$

(多項式の積を計算してから代入する)

m 次多項式は $m+1$ 点での値で決まる

$h(x)$ は $2n-2$ 次式

$2n-1$ 個の異なる x_i における値

$$h(x_i) = f(x_i)g(x_i)$$

を与えれば, $h(x)$ は決まる (= 補間法)

ただし, でたらめにとった x_i を使うと

- $f(x_i)$, $g(x_i)$ の計算の手間
- $h(x_i)$ から $h(x)$ を計算する手間

それぞれに n^2 の手間がかかる

FFT (Fast Fourier Transform)

x_i を 1 のベキ根にとると

$C \cdot n \log n$ ($n \log n$ に比例する手間) で計算できる.
(計算途中で, 値の再利用がうまくいく.)

比例定数 C は小さくないが

$$\frac{n \log n}{n^{1.58}} = \frac{\log n}{n^{0.58}} \to 0 \quad (n \to \infty)$$

だから, 桁数が大きくなると効果を発揮するはず

ハイブリッドアルゴリズム

桁数に応じて, アルゴリズムを選択

gmp では次のようになっている. (桁数は全て 2^{32} 進)

- 23 桁まで筆算法
- 164 桁まで Karatsuba 法
- 604 桁まで Toom-Cook 法
- それ以降は FFT 法

切替え桁数は CPU 毎に実験で決める

わり算 (Divide and Conquer 法)

整数 A, B に対し, $A = BQ + R, \quad 0 \leq R < B$ を満たす整数 Q, R を求めるのがわり算

$A : 2N$ 桁, $B : N$ 桁の場合

1. **$N/2$ 桁を 1 桁とみなす**
2. **4 桁 わる 2 桁と考え, 商を 1 桁ずつ求める.**
3. **「2 桁 わる 1 桁」に対し, この手順を繰り返す**

$\Rightarrow$ 剰余の計算 $(R = A - BQ)$ に高速乗算が使える

実用性

得られた方法

人間が行うのとは程遠い (複雑, 場合分け etc.)

実行するのはコンピュータ

命令されたことはどんなに複雑でも間違いなく実行してくれる

$\Rightarrow$ 結果として, あらゆる整数に対して最良と思われるパフォーマンスを提供する実装が得られている

そんな大きな数の計算が本当に必要か?

はい, 必要な場合があります.

以下でいくつか例をお見せしましょう.

応用 その 1 — 種々の数値の高精度計算

- $f(x)$ の値, $f(x) = 0$ の解の計算
  倍精度浮動小数では間に合わない場合に必要
- π の値の計算
  金田研 (東大)
  10 進で 1 兆 2411 億桁計算してあるそうです.

応用 その 2 — 多項式の因数分解

人間的方法

- **足して** a**, かけて** b

$$x^2 + ax + b = (x + \alpha)(x + \beta)$$

- **因数定理**

$$f(a) = 0 \Rightarrow f(x) = (x - a)g(x)$$

では

$$x^3 + 9026048x^2 + 11184961774x + 3993821097$$

はどうする?

整数係数多項式 $f(x)$ の因数分解

コンピュータ的方法

1. **法 p での因数分解**
   $f(x) = \prod_{i=1}^{m} f_{1i}(x) \bmod p$　(p：**素数**)
2. **法 p^k への持ち上げ** (k：$f(x)$ **で決まる**)
   $f(x) = \prod_{i=1}^{m} f_{ki}(x) \bmod p^k$
3. **因子の候補を作る**
   $g(x) = \prod_{i \in I} f_{ki}$　($I \subset \{1, \ldots, m\}$)
   $g(x)$ **の各係数** c_j **を** $-\frac{p^k}{2} < c_j < \frac{p^k}{2}$ **に調節**
4. **試し割り**
   $g(x)$ **が** $f(x)$ **を割り切れば, 因子**

例 — 因数分解

$$f(x) = x^{32} + 8x^{28} + 20x^{24} + 2632x^{20} - 30490x^{16}$$

$$+74872x^{12} + 23316x^{8} + 14904x^{4} + 1$$

$$f(x) = f_{11}(x)f_{12}(x)f_{13}(x)f_{14}(x) \bmod 13$$

$$f_{11}(x) = x^8 + 6x^6 + 7x^4 + 4x^3 + 7x^2 + 7x + 12$$

$$f_{12}(x) = x^8 + 6x^6 + 7x^4 + 9x^3 + 7x^2 + 6x + 12$$

$$f_{13}(x) = x^8 + 7x^6 + 7x^4 + 6x^3 + 6x^2 + 9x + 12$$

$$f_{14}(x) = x^8 + 7x^6 + 7x^4 + 7x^3 + 6x^2 + 4x + 12$$

mod 13^{21} への持ち上げ

$$f(x) = f_{21,1}(x)f_{21,2}(x)f_{21,3}(x)f_{21,4}(x) \bmod 13^{21}$$

$$f_{21,1}(x) = x^8 - 3129060631382307673196 0x^6 - 6x^4 - 6258121262764615346392 8x^3$$

$$+3129060631382307673196 0x^2 - 6258121262764615346391 2x - 1$$

$$f_{21,2}(x) = x^8 - 3129060631382307673196 0x^6 - 6x^4 + 6258121262764615346392 8x^3$$

$$+3129060631382307673196 0x^2 + 6258121262764615346391 2x - 1$$

$$f_{21,3}(x) = x^8 + 3129060631382307673196 0x^6 - 6x^4 + 6258121262764615346391 2x^3$$

$$-3129060631382307673196 0x^2 + 6258121262764615346392 8x - 1$$

$$f_{21,4}(x) = x^8 + 3129060631382307673196 0x^6 - 6x^4 - 6258121262764615346391 2x^3$$

$$-3129060631382307673196 0x^2 - 6258121262764615346392 8x - 1$$

試し割り

$f_{21,1}, f_{21,2}, f_{21,3}, f_{21,4} \Rightarrow$ **因子でない**

$$f_{21,1} f_{21,2} = x^{16} - 625812126276461534639 20x^{14} - 28x^{12}$$

$$-5606056975337771116138 6x^{10} + 66x^{8}$$

$$-7888281981331725922025 5x^{6} + 252x^{4}$$

$$-4953992687910926885885 2x^{2} + 1$$

$\Rightarrow$ **因子でない**

$$f_{21,1} f_{21,3} = x^{16} + 4x^{12} - 16x^{11} + 80x^{9} + 2x^{8} + 160x^{7} + 128x^{6} - 160x^{5}$$

$$+28x^{4} - 48x^{3} + 128x^{2} - 16x + 1$$

$\Rightarrow$ **因子**

因数分解完了

$$f_{21,2}f_{21,4} = x^{16}+4x^{12}+16x^{11}-80x^9+2x^8-160x^7+128x^6+160x^5$$

$$+28x^4+48x^3+128x^2+16x+1$$

$\Rightarrow$ **因子.**

結局, $f(x)$ **の因数分解は**

$$f(x) = (x^{16}+4x^{12}-16x^{11}+80x^9+2x^8+160x^7+128x^6$$

$$-160x^5+28x^4-48x^3+128x^2-16x+1)$$

$$(x^{16}+4x^{12}+16x^{11}-80x^9+2x^8-160x^7+128x^6$$

$$+160x^5+28x^4+48x^3+128x^2+16x+1)$$

応用 その 3 — 連立代数方程式の求解

n 変数多項式 $f_i(x_1, \ldots, x_n)$ に対し, 連立方程式

$$f_1(x_1, \ldots, x_n) = 0, \ldots, f_n(x_1, \ldots, x_n) = 0$$

を考える.
これが, 多項式をかけて, 足したり引いたりして

$$g(x_n) = 0, x_1 = g_1(x_n), \ldots, x_{n-1} = g_{n-1}(x_n)$$

と変形できれば, 解けたとみなす.

($g(x_n) = 0$ を解けば, 他の値が決まるから.)

例 — 円と双曲線の交点

$$f_1(x,y) = x^2 + y^2 - 1 = 0$$

$$f_2(x,y) = xy - \frac{1}{2} = 0$$

これから,

$$f_3(x,y) = 2yf_1 - 2xf_2 = x + 2y^3 - 2y = 0$$

$$f_4(y) = 2yf_3 - 2f_2 = 4y^4 - 4y^2 + 1 = (2y^2 - 1)^2 = 0$$

解は

$$(x,y) = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}), (-\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})$$

Shape base

Shape lemma

解が有限個の場合，「大抵の」連立方程式は

$$g(x_n) = 0, x_1 = g_1(x_n), \ldots, x_{n-1} = g_{n-1}(x_n)$$

と変形できる．

Shape base

$\{g(x_n), x_1 - g_1(x_n), \ldots, x_{n-1} - g_{n-1}(x_n)\}$

のこと

グレブナー基底 — 自動的式変形

計算法

元の式 $f_1, f_2, \ldots, f_m$ **に, 多項式をかけて足して, 新しい (よさそうな) 多項式を作る**

$$f_{m+1} = g_1 f_1 + \ldots + g_m f_m$$

$$f_{m+2} = h_1 f_1 + \ldots + h_m f_m + h_{m+1} f_{m+1}$$

$$\ldots$$

- **いつ終るか分からない (いずれは終る)**
- **係数は, どれだけ大きくなるか分からない**
  shape base **の場合,** $g_i(x_n)$ **の係数は巨大.**

例 — 3 変数連立代数方程式

$$f_0(x,y,z) = -3x^3 + 4y^2 + (-2z-3)y + 3z^2$$

$$f_1(x,y,z) = -2x^2 - 3x - 2y^2 + 2zy - z^2$$

$$f_2(x,y,z) = (-8y-4)x + (2z+3)y$$

これを変形して,

$$f_3(x,y,z) = (32y+16)f_1 + (-8x-2z-15)f_2$$

$$f_4(x,y,z) = (48x-72)f_1 + (-12y+12z+6)f_2 + (6z-3)f_3 - 32f_0$$

$$f_5(x,y,z) = 39/2 f_3 + (-z-3/2)f_4$$

$$\cdots$$

方程式の解 — shape base

$$f_{31}(z) = 5184z^{12} - 340416z^{11} + 8486640z^{10} - 6150816z^{9} - 136089060z^{8} + 133408548z^{7}$$

$$+801811649z^{6} - 949460920z^{5} - 982229846z^{4} + 64801908z^{3} + 1892201103z^{2}$$

$$f_{32}(y,z) = -29345757916289417092111295433120584675959353 6y$$

$$-5857424554025583974676111233415355149 12z^{11}$$

$$+3711615820568688014303958668655901821844 8z^{10}$$

$$-8757326873421779440619359900248286377388 32z^{9} - \cdots$$

$$f_{33}(x,z) = 2445479826357451424342607952760048722996612 8x$$

$$-1503468383682055779097754549555570716 80z^{11}$$

$$+9689812416148882251923768916254920775 472z^{10}$$

$$-2343485366978359358234665526544773884 64448z^{9} - \cdots$$

例 — 量子コンピュータに関する方程式

(よく知らないので) 詳細は省略するが, 量子コンピュータのある実現方法の可能性が, 7 変数連立方程式

$$F_1(r_1, \ldots, r_7) = 0, \cdots, F_7(r_1, \ldots, r_7) = 0$$

の, $|r_i| = 1$ $(i = 1, \ldots, 7)$ を満たす複素数解の存在に帰着される (F_i は有理式).

解くべき連立代数方程式

F_i の分母, 分子をとって整理すれば,

$$d_1(r_1, \ldots, r_7) \neq 0, \ldots, d_8(r_1, \ldots, r_7) \neq 0$$

のもとで

$$f_1(r_1, \ldots, r_7) = 0, \cdots, f_7(r_1, \ldots, r_7) = 0$$

を解く問題に帰着される (d_i, f_i は多項式).
d_i は小さいが, f_i は巨大.

各 f_i の項数

f_1,f_6:18, f_2,f_3,f_4:315, f_5:280, f_6:816

最初の求解

[D.P. DiVincenzo et al., Nature 408, 339-342 (2000)]
数値計算で近似的に求められた

$$r_1 = -0.8473417699 + 0.5310479498\sqrt{-1}, r_2 = 0.4420588109 - 0.8969860688\sqrt{-1},$$
$$r_3 = 0.9478117929 - 0.3188303705\sqrt{-1}, r_4 = -0.6349720565 - 0.7725351043\sqrt{-1},$$
$$r_5 = -0.03076749785 + 0.9995265685\sqrt{-1}, r_6 = 0.5996455576 + 0.8002657091\sqrt{-1},$$
$$r_7 = 0.2432074262 + 0.9699743027\sqrt{-1}$$

問題点

- **解の存在は近似的にしか示せない**
- **解の精度が不明**

厳密解を求める

終結式による変数消去 (前処理)

$f(x,y), g(x,y)$ に対し, $a(x,y), b(x,y)$ をうまくとって

$$a(x,y)f(x,y) + b(x,y)g(x,y) = R(y)$$

となるようにする $\Rightarrow$ $R(y)$ が**終結式**.

$f(x,y) = 0, g(x,y) = 0$ ならば $R(y) = 0$ だから, y の条件が得られる.

場合によっては $R(y)$ が因数分解できる $\Rightarrow$ **場合分け**

厳密解を求める (つづき)

グレブナー基底による方程式の自動変形

終結式で変数を減らす $\Rightarrow$ 場合分け
$\Rightarrow$ グレブナー基底計算

- **ここで巨大係数が発生する可能性が高い**
- **計算できれば, 扱いやすい形になっている可能性が高い.**
- **いくつかの方程式が, さらに因数分解できる場合がある.**
- **$d_i \neq 0$ の条件もとりこんで計算できる.**

解 — shape base

$$g(r_2) = 0, r_1 = g_1(r_2), r_3 = g_3(r_2), \ldots, r_7 = g_7(r_2)$$

$g(r_2) =$

$97617599813625575r_2^{24} - 450156972572535951 0r_2^{23} + 3742531908134471069 4r_2^{22} +$

$1699230477780000702 58r_2^{21} - 8008726737739418240 4r_2^{20} -$

$1558477174734663322356r_2^{19} - 5144039248697655218797r_2^{18} -$

$1247898067305431866236 0r_2^{17} - 2442578674016616619554 9r_2^{16} -$

$3889249736086013501681 0r_2^{15} - 5277577796340835184324 1r_2^{14} -$

$6372695353931333219165 4r_2^{13} - 6820673167722542627767 6r_2^{12} -$

$6372695353931333219165 4r_2^{11} - 5277577796340835184324 1r_2^{10} - \cdots$

g_i：10 進 150 桁程度の係数を持つ.

(途中の計算で, 巨大な係数が多数現れている.)

絶対値 1 の解を探す

$r_i = x_i + \sqrt{-1}y_i$ を代入して, $x_i^2 + y_i^2 = 1$ を満たす解を探せばよい

- 解の存在が示せた.
  第三者のチェックも可能 (もとの方程式に代入して 0 になることが確かめられる.)
- 解も原理的には任意精度で求まる.
  $g(r_2) = 0$ を必要な精度で解けばよい.

まとめ

誤差なしの計算を支えるもの

高速, 大容量コンピュータ
効率よいアルゴリズム, ソフトウェアの実現
(といっても, 結局は加減乗除のくり返し)

数学ソフトウェア

既に数多く開発されている
Mathematica, Maple **以外はほとんどフリー**

Risa/Asir **もよろしくお願いします**.
http://www.math.kobe-u.ac.jp/Asir/asir-ja.html