数理解析研究所講究録
920巻 1995年 62-73

7.

Fraction-free による行列式の計算効率

工学院大電気系 木下 孝
(Takashi Kinoshita)
工学院大数学 牧野 潔夫
(Isao Makino)
工学院大電子 三好 和憲
(Kazunori Miyoshi)

We consider to calculate determinants of matrices whose coefficients are integers or polynomials with integral coefficients. Speed and precision are important factors for mathematical computing system. On the calculation of determinants, Gauss elimination method is well known. But the method takes much time and causes significant errors, because the calculation deals with rational or decimal fraction numbers.

Our paper presents Sylvester's Identity and Chinese remainder theorem as the methods to calculate determinants which avoid computation of rational numbers, and the close analysis of the methods.

7.1 はじめに

整数に関する線形計算を行う際, 数式処理システムにおいて, 任意精度でかつ高速な計算は不可欠である. また科学技術計算では, 線形計算の解を求めるのに殆ど実行時間を費やしている現状である.

そこで本報告では, 線形計算でよく使われている計算の中から, 行列式と連立1次方程式を扱った. またその行列式計算を応用し, 終結式を用いた, 互いに素である1変数多項式の extended gcd [1] も取

[1] f, g が互いに素な多項式のとき, $f \cdot X + g \cdot Y = 1$ となる多項式 X, Y をこのように呼ぶこととする.

り扱った. 特に我々は, これらの基礎である整数要素の行列式計算を主に取り上げ, 効率よく求める方法を調査し, 評価したので報告する.

7.2 行列式の計算法

一般的な行列式の求め方に掃き出し法がある. しかしこの方法では, 整数要素の演算において結果が整数であることは自明であるにもかかわらず, 計算時間のかかる有理数演算を行う. さらに, 有理数を実数として処理するシステム上では, 誤差を生じる問題も発生する. 従って, 有理数や実数を用いた掃き出し法をそのまま利用するのは, 数式処理システム上において適した方法でない.

そこで有理数演算を用いずに, 整数要素や整数係数の多項式要素の行列式を整数演算によって求めるアルゴリズムとして, 次のような方法がある.

(I) 掃き出し法において分母を別に計算する (Sylvester's Identity).

(II) 互いに素である複数の法を掃き出し法で計算し, 最後にまとめて中国剰余定理と Hadamard の不等式により行列式を求める.

7.2.1 計算法 (I) Sylvester's Identity による方法

ここでは, 以下に示す Sylvester の恒等式 (Sylvester's Identity) を用いて, 行列式の値を求める.

■ Sylvester's Identity

$A = [a_{i,j}]$ を n 次正方行列とする.

各 $k(= 0, 1, \ldots, n-1)$ に対し, $A^{(k)} = \left[a_{i,j}^{(k)}\right]$ $(k+1 \le i \le n, k+1 \le j \le n)$ を次のように定める.

$$a_{0,0}^{(-1)} = 1 \quad , \qquad \left[a_{i,j}^{(0)}\right] = A \tag{1}$$

とし,

$$a_{i,j}^{(k)} = \det \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} & a_{1,j} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k} & a_{2,j} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{k,1} & a_{k,2} & \cdots & a_{k,k} & a_{k,j} \\ a_{i,1} & a_{i,2} & \cdots & a_{i,k} & a_{i,j} \end{bmatrix} \tag{2}$$

と定義したとき,

$$\det(A) \left[a_{\ell,\ell}^{(\ell-1)}\right]^{n-\ell-1} = \det \begin{bmatrix} a_{\ell+1,\ell+1}^{(\ell)} & \cdots & a_{\ell+1,n}^{(\ell)} \\ \vdots & \ddots & \vdots \\ a_{n,\ell+1}^{(\ell)} & \cdots & a_{n,n}^{(\ell)} \end{bmatrix} \tag{3}$$

となる ($1 \leq \ell \leq n-1$). 特に式 (2) の右辺にあらわれる行列を改めて A とおいたとき, A は $(k+1)$ 次正方行列であるので, ここで式 (3) を適用すると以下のように応用できる.

• $\ell = k-1$ としたとき single-step fraction-free と呼ばれ, 次のように表される.

$$a_{i,j}^{(k)} = \frac{1}{a_{k-1,k-1}^{(k-2)}} \begin{vmatrix} a_{k,k}^{(k-1)} & a_{k,j}^{(k-1)} \\ a_{i,k}^{(k-1)} & a_{i,j}^{(k-1)} \end{vmatrix} \tag{4}$$

つまり

$$a_{i,j}^{(k)} = (a_{k,k}^{(k-1)} a_{i,j}^{(k-1)} - a_{k,j}^{(k-1)} a_{i,k}^{(k-1)}) / a_{k-1,k-1}^{(k-2)},$$

$$(k+1 \leq i \leq n,\ k+1 \leq j \leq n;\ k = 1, 2, \ldots, n-1)$$

となる. またこのとき $A^{(k-1)} \to A^{(k)}$ の計算量は以下の通りである.

$$\text{乗算} \quad : \quad 2(n-k)^2, \qquad \text{加減算} : \quad (n-k)^2, \qquad \text{除算} \quad : \quad (n-k)^2 \tag{5}$$

• $\ell = k-2$ としたとき two-step fraction-free と呼ばれ, 次のように表される.

$$a_{i,j}^{(k)} = \frac{1}{\left[a_{k-2,k-2}^{(k-3)}\right]^2} \begin{vmatrix} a_{k-1,k-1}^{(k-2)} & a_{k-1,k}^{(k-2)} & a_{k-1,j}^{(k-2)} \\ a_{k,k-1}^{(k-2)} & a_{k,k}^{(k-2)} & a_{k,j}^{(k-2)} \\ a_{i,k-1}^{(k-2)} & a_{i,k}^{(k-2)} & a_{i,j}^{(k-2)} \end{vmatrix} \tag{6}$$

つまり

$$\begin{cases} c_0^{(k-2)} &= (a_{k-1,k-1}^{(k-2)} a_{k,k}^{(k-2)} - a_{k-1,k}^{(k-2)} a_{k,k-1}^{(k-2)}) / a_{k-2,k-2}^{(k-3)}, \\ c_{i1}^{(k-2)} &= (a_{k-1,k}^{(k-2)} a_{i,k-1}^{(k-2)} - a_{k-1,k-1}^{(k-2)} a_{i,k}^{(k-2)}) / a_{k-2,k-2}^{(k-3)}, \\ c_{i2}^{(k-2)} &= (a_{k,k-1}^{(k-2)} a_{i,k}^{(k-2)} - a_{k,k}^{(k-2)} a_{i,k-1}^{(k-2)}) / a_{k-2,k-2}^{(k-3)}, \\ a_{i,j}^{(k)} &= (c_0^{(k-2)} a_{i,j}^{(k-2)} + c_{i1}^{(k-2)} a_{k,j}^{(k-2)} + c_{i2}^{(k-2)} a_{k-1,j}^{(k-2)}) / a_{k-2,k-2}^{(k-3)}, \\ a_{k,k}^{(k-1)} &= c_0^{(k-2)}, \\ a_{k,m}^{(k-1)} &= a_{k,m}^{(k)} = (a_{k-1,k-1}^{(k-2)} a_{k,m}^{(k-2)} - a_{k-1,m}^{(k-2)} a_{k,k-1}^{(k-2)}) / a_{k-2,k-2}^{(k-3)}, \end{cases}$$

$$(k+1 \leq i \leq n,\ k+1 \leq j \leq n,\ k+1 \leq m \leq n;\ k = 2, 4, \ldots, 2\lfloor \tfrac{n-1}{2} \rfloor)$$

となる. またこのとき $A^{(k-2)} \to A^{(k)}$ の計算量は以下の通りである.

$$\begin{cases} \text{乗算} \quad : \quad 3(n-k)^2 + 4(n-k) + 2(n-k) + 2 \\ \text{加減算} : \quad 2(n-k)^2 + 2(n-k) + (n-k) + 1 \\ \text{除算} \quad : \quad (n-k)^2 + 2(n-k) + (n-k) + 1 \end{cases} \tag{7}$$

• $\ell = k-3$ としたとき three-step fraction-free と呼ばれ, 次のように表される.

$$a_{i,j}^{(k)} = \frac{1}{\left[a_{k-3,k-3}^{(k-4)}\right]^3} \begin{vmatrix} a_{k-2,k-2}^{(k-3)} & a_{k-2,k-1}^{(k-3)} & a_{k-2,k}^{(k-3)} & a_{k-2,j}^{(k-3)} \\ a_{k-1,k-2}^{(k-3)} & a_{k-1,k-1}^{(k-3)} & a_{k-1,k}^{(k-3)} & a_{k-1,j}^{(k-3)} \\ a_{k,k-2}^{(k-3)} & a_{k,k-1}^{(k-3)} & a_{k,k}^{(k-3)} & a_{k,j}^{(k-3)} \\ a_{i,k-2}^{(k-3)} & a_{i,k-1}^{(k-3)} & a_{i,k}^{(k-3)} & a_{i,j}^{(k-3)} \end{vmatrix} \tag{8}$$

つまり

$$Det3(a,b,c,d,e,f,g,h,i,j) = \{a(ei-fh)+b(fg-di)+c(dh-eg)\}/j$$

としたとき

$$\begin{cases} c_0^{(k-3)} &= Det3(a_{k-2,k-2}^{(k-3)}, a_{k-2,k-1}^{(k-3)}, a_{k-2,k}^{(k-3)}, a_{k-1,k-2}^{(k-3)}, a_{k-1,k-1}^{(k-3)}, a_{k,k-1}^{(k-3)}, \\ & \quad a_{k,k-2}^{(k-3)}, a_{k,k-1}^{(k-3)}, a_{k,k}^{(k-3)}, a_{k-3,k-3}^{(k-4)})/a_{k-3,k-3}^{(k-4)}, \\ c_{i3}^{(k-3)} &= Det3(a_{k-2,k-2}^{(k-3)}, a_{k-2,k-1}^{(k-3)}, a_{k-2,k}^{(k-3)}, a_{k-1,k-2}^{(k-3)}, a_{k-1,k-1}^{(k-3)}, a_{k,k-1}^{(k-3)}, \\ & \quad a_{i,k-2}^{(k-3)}, a_{i,k-1}^{(k-3)}, a_{i,k}^{(k-3)}, a_{k-3,k-3}^{(k-4)})/a_{k-3,k-3}^{(k-4)}, \\ c_{i2}^{(k-3)} &= Det3(a_{k-2,k-2}^{(k-3)}, a_{k-2,k-1}^{(k-3)}, a_{k-2,k}^{(k-3)}, a_{k,k-2}^{(k-3)}, a_{k,k-1}^{(k-3)}, a_{k,k}^{(k-3)}, \\ & \quad a_{i,k-2}^{(k-3)}, a_{i,k-1}^{(k-3)}, a_{i,k}^{(k-3)}, a_{k-3,k-3}^{(k-4)})/a_{k-3,k-3}^{(k-4)}, \\ c_{i1}^{(k-3)} &= Det3(a_{k-1,k-2}^{(k-3)}, a_{k-1,k-1}^{(k-3)}, a_{k-1,k}^{(k-3)}, a_{k,k-2}^{(k-3)}, a_{k,k-1}^{(k-3)}, a_{k,k}^{(k-3)}, \\ & \quad a_{i,k-2}^{(k-3)}, a_{i,k-1}^{(k-3)}, a_{i,k}^{(k-3)}, a_{k-3,k-3}^{(k-4)})/a_{k-3,k-3}^{(k-4)}, \\ a_{i,j}^{(k)} &= (c_0^{(k-3)}a_{i,j}^{(k-3)} - c_{i3}^{(k-3)}a_{k,j}^{(k-3)} + c_{i2}^{(k-3)}a_{k-1,j}^{(k-3)} - c_{i1}^{(k-3)}a_{k-2,j}^{(k-3)})/a_{k-3,k-3}^{(k-4)}, \\ a_{k,k}^{(k-1)} &= c_0^{(k-3)}, \\ a_{k-1,h}^{(k-2)} &= a_{k-1,h}^{(k)} = (a_{k-2,k-2}^{(k-3)}a_{k-1,h}^{(k-3)} - a_{k-1,k-2}^{(k-3)}a_{k-2,h}^{(k-3)})/a_{k-3,k-3}^{(k-4)}, \\ a_{k,m}^{(k-1)} &= a_{k,m}^{(k)} = Det3(a_{k-2,k-2}^{(k-3)}, a_{k-2,k-1}^{(k-3)}, a_{k-2,m}^{(k-3)}, a_{k-1,k-2}^{(k-3)}, a_{k-2,k-1}^{(k-3)}, a_{k-1,m}^{(k-3)}, \\ & \quad a_{k,k-2}^{(k-3)}, a_{k,k-1}^{(k-3)}, a_{k,m}^{(k-3)}, a_{k-3,k-3}^{(k-4)})/a_{k-3,k-3}^{(k-4)}, \end{cases}$$

$$(\ k+1 \le i \le n,\ k+1 \le j \le n,\ k+1 \le m \le n,\ k-1 \le h \le n;\ k = 3,6,\ldots,3\lfloor \tfrac{n-1}{3} \rfloor\ )$$

となる. ただし, $n \equiv 2 \pmod 3$ のときは最後に以下の手順を行う.

$$a_{n,n}^{(n-1)} = (a_{n-2,n-2}^{(n-3)}a_{n-1,n-1}^{(n-3)} - a_{n-2,n-1}^{(n-3)}a_{n-1,n-2}^{(n-3)})/a_{n-3,n-3}^{(n-4)}$$

またこのとき $A^{(k-3)} \to A^{(k)}$ の計算量は以下の通りである.

$$\begin{cases} \text{乗算} & : \ 4(n-k)^2 + 27(n-k) + 9(n-k) + 2(n-k+2) + 9 \\ \text{加減算} & : \ 3(n-k)^2 + 15(n-k) + 5(n-k) + (n-k+2) + 5 \\ \text{除算} & : \ (n-k)^2 + 6(n-k) + 2(n-k) + (n-k+2) + 2 \end{cases} \tag{9}$$

となる. ただし, いずれの step の fraction-free においても

$$a_{k,j}^{(k-1)} = 0 \quad , \quad j < k \quad , \tag{10}$$

である. 式 (1), (10) の条件に基づき, 式 (4), (6), (8) のいずれかの掃き出し法による漸化式を利用して計算していくと, 最終的に元の行列は, 式 (11) に示すような上三角行列に変形される（ただし, 空白は 0）. なお本プログラムでは掃き出し法において, 整数要素を扱うことを前提にしているので, 数

要素のときのみ掃き出しの軸成分を絶対値最小になるように選んで掃き出しを行っている.

$$A^{(n-1)} = \begin{bmatrix} a_{1,1}^{(0)} & a_{1,2}^{(0)} & \cdots & \cdots & a_{1,n}^{(0)} \\ & a_{2,2}^{(1)} & \cdots & \cdots & a_{2,n}^{(1)} \\ & & \cdots & \cdots & \cdots \\ & & a_{k,k}^{(k-1)} & \cdots & a_{k,n}^{(k-1)} \\ & & & \cdots & \cdots \\ & & & & a_{n,n}^{(n-1)} \end{bmatrix} \tag{11}$$

このとき $a_{n,n}^{(n-1)}$ が元の行列の行列式になるアルゴリズムである. すなわち

$$a_{n,n}^{(n-1)} = \det(A).$$

つまり本アルゴリズムは, 式 (1), (10) に基づき, 式 (4), (6), (8) のいずれかの掃き出し法による漸化式を利用し, 元の行列 A を順に帰納的に変形させて, $a_{n,n}^{(n-1)}$ である行列式を計算する方法である.

最後に式 (5), (7), (9) により, $A^{(0)} \rightarrow A^{(n-1)}$ の総計算量をまとめると表 1 となる.

	single-step	two-step		three-step		
		n : odd	n : even	$n \equiv 1 \pmod 3$	$n \equiv 2 \pmod 3$	$n \equiv 0 \pmod 3$
乗算	$\frac{2n^3-3n^2+n}{3}$	$\frac{n^3-2n+1}{2}$	$\frac{n^3-2n}{2}$	$\frac{4n^3+39n^2-114n+71}{9}$	$\frac{4n^3+39n^2-114n+58}{9}$	$\frac{4n^3+39n^2-114n}{9}$
加減算	$\frac{2n^3-3n^2+n}{6}$	$\frac{4n^3-3n^2-4n+3}{12}$	$\frac{4n^3-3n^2-4n}{12}$	$\frac{2n^3+11n^2-39n+26}{6}$	$\frac{n^3+6n^2-20n+11}{3}$	$\frac{n^3+6n^2-20n}{3}$
除算	$\frac{2n^3-3n^2+n}{6}$	$\frac{2n^3+3n^2-8n+3}{12}$	$\frac{2n^3+3n^2-8n}{12}$	$\frac{n^3+9n^2-24n+14}{9}$	$\frac{n^3+9n^2-24n+13}{9}$	$\frac{n^3+9n^2-24n}{9}$

表 1　各 step の四則演算の総計算量

7.2.2 計算法 (II) 中国剰余定理による方法

■中国剰余定理

$m_1, m_2, \ldots, m_t$ のどの 2 つも互いに素である連立合同式

$$x \equiv \begin{cases} r_1 \pmod{m_1} \\ r_2 \pmod{m_2} \\ \quad \vdots \\ r_t \pmod{m_t} \end{cases} \tag{12}$$

に解は存在し, その 1 つの解を x_t とすると一般解は

$$x \equiv x_t \pmod{m_1 m_2 \ldots m_t} \tag{13}$$

である. ■

ここでは上記の定理を利用して, 以下のように行列式を計算する.

加減乗算はある整数を法として計算することによって, 小さな桁数の数で計算ができ計算量を減らせる. さらにそのような法のもとで, 逆元が存在する場合は, 除算も有理数演算を利用せずに乗算によって同様に計算できる. すなわち行列式は, ある整数を法とした掃き出し法で計算できる場合がある. しかし法の値が小さい時, そのような法のもとで条件を満たす行列式は, 法を周期とした比較的に短周期で複数個存在するので, 一意に定まらない.

そこで, 互いに素なる t 個の比較的小さな整数 $m_1, m_2, \ldots, m_t$ を法として, 各法について掃き出し法で行列式 $r_1, r_2, \ldots, r_t$ を計算する. それからこれらを式 (12) に当てはめて, 連立合同式を解けば, 式 (13) のように解の周期を長くすることができる. さらに与えられた行列式の限界値が既知のとき, その範囲を超えるように式 (13) の法 $m_1 m_2 \ldots m_t$ を定めれば, 行列式の値は一意に求まる.

従って本アルゴリズム (は, まず行列式の限界値を計算し, この値を超えるように t 個の法を定める. ここで定めた各法における行列式を掃き出し法によって計算し, これらの連立合同式を解いて行列式を求める方法である.

その行列式の値の限界を求める方法として, 次に示す Hadamard の不等式がある. また, 連立合同式の解法は, 次に示す Gauss の方法, Garner の方法の 2 通りある.

さらに我々は, 大きな次数や桁数要素の行列を扱うことを前提にしている. 従ってそのような場合においても, 行列式を計算できるような法を抽出せねばならない. そこでそのような法の抽出方法として, 次の 3 通りの方法を利用する.

1) システム内の素数テーブルを最大の数から利用する方法. (本開発システム上では 10 進 8 桁の素数 999 個 : 99,981,793 ~ 99,999,989)

2) 円分数 $M(35, x), M(39, x)$; $2 \leq x \leq 1{,}000$ により 50 桁以上の素数 508 個を予め (手作業で) 抽出した素数テーブルを読み込んで計算する.

3) n 番目の一般フィボナッチ数 f_n の値の相関性

$\gcd(p, q) = 1$, $f_0 = 0$, $f_1 = 1$ とし, $f_n = p \cdot f_{n-1} + q \cdot f_{n-2}$ と定めたとき

$$\gcd(m, n) = 1 \Longrightarrow \gcd(f_m, f_n) = 1$$

を利用して法 f_n を抽出する. ただし計算の簡略化のため m, n は素数, $p = q = 1$ とした.

■ Hadamard の不等式

n 次正方行列 $A=[a_{i,j}]$ に対し　　$(\ 1\leq i\leq n\ ,\ 1\leq j\leq n\ )$

$$|\det(A)|\leq\prod_{i=1}^{n}\sqrt{a_{i,1}^2+a_{i,2}^2+\ldots+a_{i,n}^2}$$

が成立する. この式により, $|\det(A)|$ の上限値がわかる.

■ Gauss の方法

$$M=m_1m_2\ldots m_t\quad,\qquad M_k=M/m_k\quad(1\leq k\leq t)$$

とする. このとき $\gcd(M_k,m_k)=1$ である. そして

$$\ell_kM_k\equiv 1\qquad(\mathrm{mod}\ m_k)$$

なる ℓ_k をとったとき, 式 (12) の解は以下の通りである.

$$x\equiv r_1\ell_1M_1+r_2\ell_2M_2+\ldots+r_t\ell_tM_t\qquad(\mathrm{mod}\ M)$$

■ Garner の方法

式 (12) の解は以下のように定められる.

$$v_0\equiv r_1\quad(\mathrm{mod}\ m_1),\qquad L_k=\prod_{i=1}^{k}m_i\qquad(1\leq k\leq t)$$

とする. このとき $\gcd(L_{k-1},m_k)=1$ である. そして

$$v_{k-1}L_{k-1}\equiv r_k-(v_0+v_1L_1+\ldots+v_{k-2}L_{k-2})\quad(\mathrm{mod}\ m_k)$$

によって $v_1,v_2,\ldots,v_{t-1}$ ($0\leq v_k<m_{k+1}$) を順に帰納的に定めたとき（このとき途中の計算は法 m_k の最大値内で行われる）, 解は

$$x\equiv v_0+v_1L_1+\ldots+v_{t-1}L_{t-1}\qquad(\mathrm{mod}\ L_t)$$

である. この 1 回の計算でのみ大きな値を扱うため, 計算が高速化される利点がある.

7.3 応用プログラム

7.2節で述べた行列式計算アルゴリズムを応用して, 整数または整数多項式の係数の連立 1 次方程式, 1 変数多項式の extended gcd 解法プログラムも開発した.

連立 1 次方程式は, アルゴリズムとして Sylvester の fraction-free を応用した Gauss の消去法を用

いた. この方法で不定解, 不能解も求められ, また文字係数の場合も解けるようにした.

1変数多項式の extended gcd は, 整数係数の多項式演算を扱わねばならないので, Sylvester の fraction-free を応用した. 入力した2つの多項式が互いに素 (Resolutant $\neq 0$) であるときに extended gcd を求めるようにした.

なおいずれのプログラムにおいても, 入力に対して正しい出力が得られることを確認した.

7.4 実験内容ならびに結果

はじめに開発環境は以下の通りである.

ソフトウェア　富士通情報社会科学研で開発中の数式処理処理システム/ ライブラリ Risa の計算エンジンの言語インターフェース asir

ハードウェア　HP9000/735 (OS:HP-UX, メモリ 80MB)

この環境下で 7.2 節で述べた3種のアルゴリズムによる行列式の演算効率の比較を下記の要領で行った.

比較に用いた行列式の演算プログラムは (I) の single, two, three-step fraction-free, (II) の Gauss の方法, Garner の方法, 有理数演算を用いる掃き出し法, asir 内の行列式を求める組み込み関数 det, そして asir のライブラリ内の行列式を求める関数 deter の8種である. なお, 関数 det, deter は single-step fraction-free に基づいて作成されている. これらの関数を用いて以下の実験, 計測を行った.

A)　次数は 9 ～ 15 次で, 要素は 100 ～ 1,000 桁までの整数要素の正方行列を用いて行列式を求め, 計算時間の計測を行う.

B)　Sylvester の fraction-free アルゴリズムについて, 次数は 10 ～ 300 次で, 要素は 5 または 10 桁までの整数要素の正方行列を用いて行列式を求め, 計算時間の計測をする.

C)　Sylvester の fraction-free アルゴリズムについて, 多項式演算ができることを確認し, 1次多項式要素（係数は 5 桁までの整数）の正方行列を用いて行列式を求め, 計算時間の計測を行う.

D)　中国剰余定理のアルゴリズムついて, 7.2.2 節で述べた法により, 8 ～ 11 次の正方行列の行列式を求め, 計算時間の計測を行う.

上記の方法により, 以下のような行列式の演算効率がわかる. なおこれらの実験結果の一部を 7.7 節にまとめておいた.

7.4.1 行列式の演算効率

まず, 上記 8 種の関数の計算時間の優劣を調査するため, A) の実験を行った. 実験結果より次のことがわかった. なお 12 次の測定結果のみ表 2 に示しておいた.

- 少ない桁数要素の時, Sylvester の fraction-free は中国剰余より計算時間を要さない. しかし,
  - ◇9 次正方行列で 1,000 桁以上の要素において,

◇10 次正方行列で 700 桁以上の要素において,

◇11 次正方行列で 500 桁以上の要素において,

◇12 次正方行列で 400 桁以上の要素において,

◇13 次正方行列で 350 桁以上の要素において,

◇14 次正方行列で 300 桁以上の要素において,

◇15 次正方行列で 250 桁以上の要素において

いかなる step の Sylvester の fraction-free より中国剰余の方が, 逆に速くなる.

•有理数演算は, Sylvester や中国剰余定理による整数演算アルゴリズムより約 10 倍以上計算時間を要する.

•Sylvester の fraction-free に着目すると, 9 ～ 15 次といった低次において, two-step が最速であり, 次いで single-step, three-step の順である. このとき two-step は, single-step より 2 割程度計算時間が短い. ただし次数が 3 の倍数の時, single-step と three-step の形成が逆転する.

•組み込み関数やライブラリ関数は, 我々の single-step より若干計算時間を要する. とくに, 要素の桁数が大きくなるにつれて, これらの計算時間の差も大きくなる.

実験 A) では, three-step の効果が発揮されなかった. しかし, 次数を増やすにつれて, three-step が効果を発揮することが考察できる. そこで, 効果を発揮する次数を調べるため, 実験 B) を行った. 10 桁の整数要素における 100 次までの結果を表 3 に示し, この結果から次のようなことがいえる.

•先に述べたように, 10 次程度の低次において two-step が最速であり, 次いで single-step, three-step の順の計算スピードである. ところが, 20 次前後を超えると, three-step と single-step の計算時間が逆転する. さらに次数を増やし, 70 次前後以上に至ると two-step より three-step が速くなり, three-step が最速となる.

•このことから, 次数を増すにつれて, 高 step の fraction-free が効果を発揮し, 有用となることが理解できる.

これまで整数要素について調査してきたが, C) の実験により, 多項式要素についての調査も行った. 結果は表 4 であり, これより以下のようなことが理解できる.

•Sylvester の fraction-free を用いたプログラムでは, 多項式の演算もできる.

•整数要素と同様に, single-step より two-step は, 低次においても有効なアルゴリズムである. さらに, single-step と three-step の計算時間の逆転する次数も, 整数要素の時とほぼ同じ 20 次前後である.

•従って, 次数が大きくなるにつれて, 整数要素のとき同様に高 step の fraction-free アルゴリズムが有効であることが推測できる.

•またこのとき, single-step は組み込み関数とほぼ同じ計算時間であり, GC time の分だけ僅かながら我々の single-step の方が速い. 従って整数要素の時には, 掃き出しの軸成分を絶対値最小になるように選んで掃き出しを行うと, 計算時間を短縮できることが考察できる.

これまで Sylvester の fraction-free アルゴリズムについてのみ着目してきたが, ここで中国剰余定

理によるアルゴリズムに着目する. 実験 A) では, 法を 2) によって抽出したが, 他の抽出方法について調査, 比較するため D) の実験を行った. 8 次の結果のみ表 5 に示しておいた. 実験結果より次のようなことがいえる.

•1) の方法, つまり組み込み関数 lprime() によって抽出した素数を法として利用する方法の実験結果について考察する.

◇抽出された法は 1 word 整数であるので, 1 つの法についての行列式計算の時間は短い.

◇しかし利用する法は, 2), 3) の方法に比べて多く必要とする. 従って, 他の 2 方法に比べて計算時間がかかる.

◇lprime() の参照できる素数の数も 999 個と少ないため, 約 8,000 桁未満の行列式しか計算できない.

•次に 2) の方法の実験結果について考察する.

◇法は 50 桁以上の整数を用いているため, 1 つの法についての計算時間は, 1 word 整数による計算よりかなり要する.

◇使用する法は 1) よりかなり少ないため, 2 割程度計算時間が短くなっている.

◇しかしながら, 素数テーブルを手作業で作らねばならないといった大変な手間を要するため, 数式処理システム上において適した方法でない.

◇さらにテーブルの大きさに応じた行列式計算の限界（本素数テーブルでは約 30,000 桁程度までしか計算できない）がある.

•最後に 3) の方法の実験結果について考察する. この方法は上記 2 方法の弱点を克服している.

◇計算時間は, 2) の方法より多少要するが, 比較的短時間で計算できる.

◇法の計算方法も簡単であり, 数式処理システムを有効に利用できる.

◇計算できる行列式の桁数も, 500 番目までのフィボナッチ数の法を用いた場合で 172,176 桁であり, 相当な桁数まで計算できる. 本プログラムにおいて, 1,229 番目の素数 (9,997) 番目のフィボナッチ数まで, つまり $f_2 \sim f_{9997}$ の 1,229 個の法をもてる.

•従って, 中国剰余定理による行列式計算に用いる法は, フィボナッチ数を用いるのがよいと考えられる.

ことが確認できた.

7.5 評価

本報告では, 整数要素による行列式演算の効率について述べてきた. 実験結果より次のようなことが確認できた.

•Sylvester の fraction-free アルゴリズムは多項式要素の演算もでき, 次数を増やすにつれて高 step のアルゴリズムが有用となる. また two-step は, 低次でも single-step より計算時間を要さな

いため, かなり有効なアルゴリズムといえる. しかし, 要素の桁数が大きくなると不適である. また整数要素の行列式計算は, 絶対値最小になるように軸成分を抽出すれば, 計算時間を短縮できる.

•逆に中国剰余定理の方法は, 要素の桁数が大きいときに有効な手段である. また法としてフィボナッチ数を用いると, 計算時間も比較的短時間で計算でき, さらにかなり大きい桁となる行列式の計算ができる. しかしながら, 本プログラムでは, 多項式の演算ができない.

•今回は整数, 整数係数の多項式要素について扱ったが, 要素の lcm を乗ずることによって有理数のみならず, Sylvester の fraction-free では, 多項式も扱えるので整数以外の要素のときにも応用できる.

•京都の発表以後, 富士通情報社会研の野呂正行氏に HP の asir へ karatsuba による多倍長乗算を impliment していただきました. この karatsuba 法を用いると, 今回検討した中でかなり有用な fraction-free アルゴリズムである Sylvester の two-step において, 約 2 ～ 3 割の計算時間の改善がはかれました. 他の step の計算時間も改善されています（表 2 参照）.

7.6 参考文献

[1] Erwin H. Bareiss, "Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination", Math. Comp., 22(103) pp.565-578(1968).

[2] Erwin H. Bareiss, "Computational Solutions of Matrix Problems Over an Integral Domain", J.Inst.Maths Applics, 10, pp68-104(1972).

[3] K.Geddes, S.Czapor and G.Labahn: "Algorithms for Computer Algebra", Kluwer Academic Publishers, 1992.

[4] 木田祐司, 牧野潔夫: "UBASIC によるコンピュータ整数論", 日本評論社.

[5] 森本光生, 木田祐司, 小林美千代: "円分数の素因数分解（その 3 ）", 上智大学数学講究録.

[6] 長嶋秀世: "数値計算法", 槙書店.

7.7 参考

以下に実験結果のデータを一部示す. なお各データとも表内の数字は, 特に断りのないかぎり計算時間であり, "[CPU time (sec)] + [GC time (sec)]" である. また表 5 の f_{p1} とは, 1 番目の素数番目のフィボナッチ数, つまり f_2 のことである.

アルゴリズム	100 桁	200 桁	300 桁	400 桁	500 桁	600 桁
中国剰余 (Garner)	4.42 + 1.04	9.18 + 1.05	13.62 + 1.89	20.49 + 2.2	26.65 + 2.93	33.76 + 2.87
single-step	2.12 + 0.1	7.75 + 0.08	16.9 + 0.12	30.72 + 0.16	46.32 + 0.14	67.01 + 0.18
two-step	1.56 + 0.03	5.71 + 0.08	12.36 + 0.06	22.83 + 0.1	34.44 + 0.13	49.78 + 0.12
two-step (karatsuba)	1.5 + 0.15	5 + 0.44	10.08 + 0.64	17.91 + 0.64	26.09 + 1.42	36.68 + 1.55
three-step	2.08 + 0.05	7.67 + 0.06	16.69 + 0.15	30.65 + 0.1	46.33 + 0.17	67.29 + 0.14
有理数演算	57.75 + 14.32	213.98 + 26.06				
組み込み関数 det	2.18 + 0.05	7.88 + 0.07	17.28 + 0.09	31.03 + 0.11	48.13 + 0.13	68.97 + 0.15
ライブラリ関数 deter	2.18 + 0.06	7.94 + 0.05	17.33 + 0.08	31.09 + 0.08	47.98 + 0.17	69.02 + 0.13

表 2　実験 A による 12 次正方行列の計測結果

アルゴリズム	10 次	20 次	30 次	60 次	70 次	100 次
中国剰余 (Garner)	0.34 + 0.11	2.3 + 0.63	9.37 + 2.1	123.21 + 13.65		
single-step	0.04 + 0.02	0.61 + 0.11	3.6 + 0.34	85.59 + 3.01		
two-step	0.03 + 0.01	0.4 + 0.06	2.33 + 0.14	54.66 + 1.48	114.18 + 4.68	632.11 + 11.72
three-step	0.05 + 0.01	0.49 + 0.11	2.77 + 0.11	55.95 + 1.41	113.81 + 2.35	596.64 + 8.26

表 3　実験 B による 10 桁の整数要素の行列式における計測結果

アルゴリズム	10 次	15 次	20 次	25 次	30 次
single-step	0.39 + 0.1	3.22 + 0.99	16.42 + 4.12	63.02 + 10.92	193.28 + 33.58
two-step	0.28 + 0.11	2.43 + 0.79	12.31 + 2.83	46.36 + 7.58	141.34 + 21.69
three-step	0.38 + 0.19	3.36 + 1.08	16.72 + 3.65	60.52 + 10.26	177.31 + 27.99
組み込み関数 det	0.36 + 0.15	3.26 + 1.31	16.38 + 6.8	63.14 + 19.01	193.76 + 58.35

表 4　実験 C による 1 次多項式要素の行列式における計測結果

法の摘出方法	法の使用数	計算時間	法の摘出方法	法の使用数	計算時間
組み込み関数 lprime()	895	27.47 + 4.32	f_{p21} を始点とした Fibonacci 数	98	23.66 + 1.38
円分数の素因数分解	129	22.26 + 1.98	f_{p41} を始点とした Fibonacci 数	82	23.74 + 1.36
f_{p1} を始点とした Fibonacci 数	117	23.82 + 2.02	f_{p81} を始点とした Fibonacci 数	68	23.8 + 1.27

表 5　実験 D による 8 次正方行列における行列式の計測結果