記号処理 11－3
（1980. 3. 21）

記号行列式に対する効率的なガウス消去法

佐々木建昭（理研） 村尾裕一（東大・理）

概要：多変数多項式を要素とする行列式の効率的なガウス消去法を提案する。提案される方法は、Bareiss の分数なしのガウス消去法をさらに改善したものであり、中間表式膨張（intermediate expression swell）を押え、容量・時間ともに効率的なアルゴリズムである。この新しいアルゴリズムをREDUCE上にインプリメントして、対称テープリッツ行列およびヴァンデアモンデ行列を例題として、REDUCE組込みのBareiss法および小行列法と比較したところ、我々の方法はBareiss法より4倍程度高速で、小行列式法と同程度であることが判明した。

1．はじめに

数式を要素とする行列式、いわゆる記号行列式の計算は、数式処理システムで最もよく利用される計算である。行列の要素が数値である場合と異なり、記号行列式を計算すると、数式が爆発的に膨張するのが普通である。そのため、数値計算では最も高速な方法であるガウス消去法は、記号行列式の計算に対しては最も遅い方法になってしまう。記号行列式の計算は単純ではあるけれども、数値計算と数式処理の差がはっきりとみえて興味深い。

本稿では、多項式 a_{ij} を要素とする行列をMとし、Mの行列式をDと表わす。

$$M=\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & & & \vdots \\ a_{n,1} & & & a_{n,n} \end{pmatrix} \qquad D=|M|$$

Mの (i,j) 要素のことを M_{ij} とも表わす。

現在までに提案されている記号行列式の代表的な計算法を列記しよう。

i) Bareissのガウス消去法（fraction-free method）[1]。 第2節で詳述

ii) 数値計算－補間法[2] 行列式の答は多変数多項式になるはず。それを $P(x_1,\cdots,x_N)$ とすれば、行列式の $x_1,\cdots,x_N$ にそれぞれ $b_1,\cdots,b_N$ を代入して、数値行列を計算した結果は $P(b_1,\cdots,b_N)$ である。この数値計算を十分多数個の $(b_1^{(i)},\cdots,b_N^{(i)})$ について行ない、その結果から $P(x_1,\cdots,x_N)$ を構成する。

iii) 小行列式法[3] まず、第1,2行から構成できるすべての零でない2×2行列式を計算し、ストアしておく。次に、第1,2,3行から構成できるすべての零でない3×3行列式を、ストアされた2×2行列式を使って計算する。同様の操作をくり返すことにより、重複計算を組織的に避ける。

例

$$\begin{vmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{vmatrix} = \begin{array}{l} +a_1(b_2(c_3d_4-d_3c_4)-c_2(b_3d_4-d_3b_4)+d_2(b_3c_4-c_3b_4)) \\ -b_1(a_2(c_3d_4-d_3c_4)-c_2(a_3d_4-d_3a_4)+d_2(a_3c_4-c_3a_4)) \\ +c_1(a_2(b_3d_4-d_3b_4)-b_2(a_3d_4-d_3a_4)+d_2(a_3b_4-b_3a_4)) \\ -d_1(a_2(b_3c_4-c_3b_4)-b_2(a_3c_4-c_3a_4)+c_2(a_3b_4-b_3a_4)) \end{array}$$

iv) ラプラス展開法[4]。 ラプラスの展開公式（行あるいは列による展開）を利用
v) 直接展開法[5]。 行列式のもともとの定義による計算法

これらの方法は、普遍的にどれかがよくどれかが悪いということはなく、場合によって良し悪しがかわる。総じて言えることは[6]、i)とiii)は密な行列式（零の要素の数が少ない）に対して有効で、特に変数の数が1の場合に適している。iii)は行列式が密な場合にも疎（零要素の数が多い）な場合にも、平均的に高速である。特に多変数多項式を要素とする行列式に対しては、i)やiii)よりも高速である。唯一の欠点はメモリ的に効率が悪いことである。iv)とv)は密な行列式に対しては効率が悪い。疎な行列式に関しては、iii)より低速だがi)やiii)よりは高速である。しかし、メモリの観点から見れば効率がよい。

Bareissの方法の出番は密な1変数の行列式の場合だけということだが、しかしメモリの観点からみると、iii)の小行列式法より優れている。そこで、我々はあえてガウス法の改善に取組む。我々の改善策によると、ガウス法は多変数多項式を要素とする密な行列式に対しても、小行列式法と同程度に高速になる。したがって、メモリ効率のよい分だけ、小行列式法に勝ることになる。しかしながら、我々の方法は、1変数の行列式や疎な行列式に対しては、余り役にはたたない。

2. オリジナルなガウス消去法

数値計算で用いられているガウスの消去法の問題点をまず明らかにしよう。例として4×4行列式を計算してみる：

$$(2\text{-}1)\quad \begin{vmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{vmatrix} = \begin{vmatrix} a_1 & a_2 & a_3 & a_4 \\ & (a_1b_2-a_2b_1)/a_1 & (a_1b_3-a_3b_1)/a_1 & (a_1b_4-a_4b_1)/a_1 \\ & (a_1c_2-a_2c_1)/a_1 & (a_1c_3-a_3c_1)/a_1 & (a_1c_4-a_4c_1)/a_1 \\ & (a_1d_2-a_2d_1)/a_1 & (a_1d_3-a_3d_1)/a_1 & (a_1d_4-a_4d_1)/a_1 \end{vmatrix}$$

$$(2\text{-}2)\quad = \begin{vmatrix} \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} & \begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix} & \begin{vmatrix} a_1 & a_4 \\ b_1 & b_4 \end{vmatrix} \\ \begin{vmatrix} a_1 & a_2 \\ c_1 & c_2 \end{vmatrix} & \begin{vmatrix} a_1 & a_3 \\ c_1 & c_3 \end{vmatrix} & \begin{vmatrix} a_1 & a_4 \\ c_1 & c_4 \end{vmatrix} \\ \begin{vmatrix} a_1 & a_2 \\ d_1 & d_2 \end{vmatrix} & \begin{vmatrix} a_1 & a_3 \\ d_1 & d_3 \end{vmatrix} & \begin{vmatrix} a_1 & a_4 \\ d_1 & d_4 \end{vmatrix} \end{vmatrix} / a_1^2$$

(2-3)

$$= \begin{vmatrix} \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} & \begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix} & \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} & \begin{vmatrix} a_1 & a_4 \\ b_1 & b_4 \end{vmatrix} \\ \begin{vmatrix} a_1 & a_2 \\ c_1 & c_2 \end{vmatrix} & \begin{vmatrix} a_1 & a_3 \\ c_1 & c_3 \end{vmatrix} & \begin{vmatrix} a_1 & a_2 \\ c_1 & c_2 \end{vmatrix} & \begin{vmatrix} a_1 & a_4 \\ c_1 & c_4 \end{vmatrix} \\ \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} & \begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix} & \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} & \begin{vmatrix} a_1 & a_4 \\ b_1 & b_4 \end{vmatrix} \\ \begin{vmatrix} a_1 & a_2 \\ d_1 & d_2 \end{vmatrix} & \begin{vmatrix} a_1 & a_3 \\ d_1 & d_3 \end{vmatrix} & \begin{vmatrix} a_1 & a_2 \\ d_1 & d_2 \end{vmatrix} & \begin{vmatrix} a_1 & a_4 \\ d_1 & d_4 \end{vmatrix} \end{vmatrix} \Big/ \{a_1^2(a_1b_2 - a_2b_1)\}$$

すなわち、消去のたびに分母に因子がかかる。もとの行列式は有理式ではなく多項式のはずゆえ，分母の因子は最後には分子と完全に打ち消し合うはずである。ということは、その分だけ分子の表式が長大化していることを意味する。実際，行列式は計算すると4次の多項式になるはずだが，(2-3)の分子行列式の各要素が既に、4次の多項式である。非常に無駄な計算をしていることが知れよう。

3．Bareissの方法

上述のガウス法の無駄を省いて効率をあげたのがBareissで，1968年のことである。(2-3)の分母をみると、二つの多項式の積である。Bareissは分母の因子が消去の途中で分子行列式の各要素を割り切るのではないかと予想した。(2-1)では、分母の因子は分子のどの要素も割り切らない。(2-3)では分子は2行2列の行列式で、分母はa_1を2個含むから、ちょうど割り切れそう。計算してみると，

$$\begin{vmatrix} \begin{vmatrix} A_1 & A_2 \\ B_1 & B_2 \end{vmatrix} & \begin{vmatrix} A_1 & A_3 \\ B_1 & B_3 \end{vmatrix} \\ \begin{vmatrix} A_1 & A_2 \\ C_1 & C_2 \end{vmatrix} & \begin{vmatrix} A_1 & A_3 \\ C_1 & C_3 \end{vmatrix} \end{vmatrix} = A_1(A_1B_2C_3 + A_2B_3C_1 + A_3B_1C_2 - A_1B_3C_2 - A_2B_1C_3 - A_3B_2C_1)$$

$$= A_1 \begin{vmatrix} A_1 & A_2 & A_3 \\ B_1 & B_2 & B_3 \\ C_1 & C_2 & C_3 \end{vmatrix}$$

となり、予想は正しい。

この計算で、$A_1, A_2, A_3, \cdots, C_3$は何であってもよく，もちろん行列式でもよい。一方、オリジナルなガウス消去法をみると，k回目の消去後の分子行列式の要素を$A_1, A_2, \cdots, B_1, B_2, \cdots, \cdots\cdots$とすると、(k+2)回目の消去後の分子行列式の要素は、ちょうど上式と同じ形をしている。したがって、オリジナルなガウス消去法において、k回目の消去で生じる分母の因子は，(k+2)回目の消去後の分子行列式の各々の要素を割り切ることが知れる。そこで，2回目以後の消去のあと、各要素に割り算を施せば、分母を常に1にしままで計算できる。

M を $M^{(1)}$ と書き，k 回目の消去をしたあとの行列を $M^{(k+1)}$ と書くと，Bareiss のアルゴリズムは次のように書ける。ここで $k \leq 0$ に対しては $M^{(k)}_{i,j} = 1$ とする。

Bareiss Fraction-free Gaussian Elimination

Input : an $n \times n$ matrix $M = (M^{(1)}_{ij})$;
Output : $D = |M|$;

```
for k ← 1 to n−1 do
    pivotting so that M^(k)_{k,k} ≠ 0;
    for i ← k+1 to n do
        for j ← k+1 to n do
            M^(k+1)_{i,j} ← (M^(k)_{k,k} M^(k)_{i,j} − M^(k)_{i,k} M^(k)_{k,j}) / M^(k−1)_{k−1,k−1}    (3)
        end;
    end;
end;
return D ← M^(n)_{n,n};
```

このアルゴリズムでは，$M^{(1)}, M^{(2)}, \cdots$ がすべてメモリに残るが，k 回目の消去で必要な式は，$M^{(k-1)}_{k-1,k-1}$ と $M^{(k)}$ の要素だけゆえ，実際の計算では不必要な式は常にメモリからクリヤする。

オリジナルなガウス消去法に比べて，Bareiss のアルゴリズムでは，計算途上の式の量が大中に少なくなる。すなわち，中間表式膨張は大中に押さえられた。しかし，完全になくなったわけではない。よく調べると，$M^{(k)}_{i,j}$ は，M の要素から作られる k 次の小行列式になっていることがわかる。したがって，$M^{(k+1)}$ の要素を計算するには，k 次の小行列式を2個づつかけて差をとり，それを $(k-1)$ 次の小行列式で割ることになる。よって，中間表式は最後の答の表式より一般に大きくなる。たとえば，k 次の小行列式が次数 k の多項式ならば，k 次の多項式を二つづつかけて，それを $(k-1)$ 次の多項式で割り，答として $(k+1)$ 次の多項式を得ていることになる。

Bareissの方法における中間表式膨張は、Mが数値行列があるいは一変数の行列であるならば大したことはない。実際、整数あるいは一変数の行列の場合は、Bareissの方法は非常に高速である。問題は、要素が多変数多項式の場合である。P_1 と P_2 が項数 t_1 と t_2 の多変数多項式とするとき、多項式 $P_1 \cdot P_2$ は、項数が最大 $t_1 \cdot t_2$ の多項式になる。中間表式が非常に膨張するのみならず、(3)の右辺の計算に多大の時間がかかることになる。Bareissの方法をここまで詳しく分析すると何とかこの中間表式膨張なしに計算できないかと誰もが考えるであろう。次節でそのことが可能であることを示す。

4. Bareiss の方法の改善

Bareiss の 2-step 消去公式（前頁の(3)式）を考えよう。今、Mの主対角要素はすべて零でなく、他の要素とは独立な変数である場合を考えよう：

(4-1) $\quad a_{i,i} = X_i \neq 0, \quad i=1,\cdots,n$

前節で、$M^{(k)}_{i,j}$ はMの要素から作られる k次の小行列式であることを述べたが、具体的には次の形になる（文献(1)を参照）：

(4-2) $$M^{(k)}_{i,j} = \begin{vmatrix} X_1 & a_{1,2} & \cdots & a_{1,k-1} & a_{1,j} \\ a_{2,1} & X_2 & \cdots & a_{2,k-1} & a_{2,j} \\ \vdots & & & \vdots & \vdots \\ a_{k-1,1} & a_{k-1,2} & \cdots & X_{k-1} & a_{k-1,j} \\ a_{i,1} & a_{i,2} & \cdots & a_{i,k-1} & a_{i,j} \end{vmatrix}$$

特に、$M^{(k-1)}_{k-1,k-1}$ は、対角線上に $X_1,\cdots,X_{k-1}$ が並ぶ $(k-1)$次の主小行列式である。したがって、(3)式の除多項式 $M^{(k-1)}_{k-1,k-1}$ は、変数 $X_1,\cdots,X_{k-1}$ に関していえば、最高次数項が $X_1X_2\cdots X_{k-1}$ であり、各々の変数について一次の多項式である。したがって、(3)式の分子 $(M^{(k)}_{k,k}\ M^{(k)}_{i,j} - M^{(k)}_{i,k}\ M^{(k)}_{k,j})$ のうち、$X_1X_2\cdots X_{k-1}$ に比例する項をすべてとり出すと、その係数多項式は $M^{(k+1)}_{i,j}$ の項をすべて含んでいるはずである。我々のアイディアは、(3)式の分子の計算の際、$X_1X_2\cdots X_{k-1}$ に比例しない項を計算しないですませようというものである。

$M^{(k-1)}_{k-1,k-1}$ は $X_1X_2\cdots X_{k-1}$ の項をもち、その係数は1であるから、$X_1,\cdots,X_{k-1}$ を主変数とみて $(X_1X_2\cdots X_{k-1})^2$ を $M^{(k-1)}_{k-1,k-1}$ で割ることができる：

(4-3) $$(X_1X_2\cdots X_{k-1})^2 = Q^{(k-1)}\,M^{(k-1)}_{k-1,k-1} + R^{(k-1)}.$$

ここで、$Q^{(k-1)}$ は商、$R^{(k-1)}$ は余りである。またここで、$R^{(k-1)}$ は $X_1X_2\cdots X_{k-1}$ に比例する項はもたないことに注意されたい（もしそんな項があれば、$R^{(k-1)}$ は $M^{(k-1)}_{k-1,k-1}$ でさらに割れる）。(3)式と(4-3)とを辺々かけ合わせると

(4-4) $$M^{(k+1)}_{i,j}\cdot(X_1X_2\cdots X_{k-1})^2 = (M^{(k)}_{k,k}\ M^{(k)}_{i,j} - M^{(k)}_{i,k}\ M^{(k)}_{k,j})\,Q^{(k-1)} + M^{(k+1)}_{i,j}\,R^{(k-1)}$$

を得る。ところで、$M^{(k+1)}_{i,j}$ は $X_1,\cdots,X_{k-1}$ の各々の変数に関して一次の多項式であり、$R^{(k-1)}$ は $X_1X_2\cdots X_{k-1}$ に比例する項を含まない。よって、上式の右辺第二項は $(X_1X_2\cdots X_{k-1})^2$ に比例する項は含まない。即ち、右辺第一項の中から $(X_1X_2\cdots X_{k-1})^2$ に比例する項のみとり出せば、その係数多項式が $M^{(k+1)}_{i,j}$ になる。

一般の問題においては、対角要素が他とは独立な変数とは限らない。そこで、計算に先立って、対角要素を別の独立な変数でおきかえておき、計算したあとで元にもどす。そうすれば、上記の計算法が使えることになる。

5．アルゴリズム

以下、k回目の消去においては、$X_1, \cdots, X_{k-1}$ を主変数として扱うものとする。（$X_k, \cdots, X_n$ はk回目の消去では主変数とはみなさない。） 主変数を特別視する特別な乗算を導入しよう。その乗算記号を$\otimes$と表現することにする。

主変数とその他の変数から成る多項式 P_1 と P_2 をかけ、その結果を $X_1 \cdots X_{k-1}$ に比例する部分とそうでない部分にわけるとする：

$$P_1 \cdot P_2 = Q \cdot X_1 X_2 \cdots X_{k-1} + R.$$

このとき，我々の特殊な乗算は、その答としてQを返すものとする：

$$P_1 \otimes P_2 = Q.$$

対角要素のおきかえと主変数の扱いに注意し、アルゴリズムを構成する：

Efficient Gaussian Elimination Algorithm

Input : An $n \times n$ matrix $M \equiv M^{(1)}$ independent of X_i or Y_i
Output : $D = |M|$

Step 1. [Replace $M^{(1)}_{i,i}$ by a new variable Y_i.]
　　for $i \leftarrow 1$ to $n-1$ replace $M^{(1)}_{i,i}$ by Y_i;
Step 2. [Eliminate columns.]
　　for $k \leftarrow 1$ to $n-1$ do
　　$Q^{(k-1)} \leftarrow$ { if $k=1$ then 1 else $(X_1 X_2 \cdots X_k)^2 / M^{(k-1)}_{k-1,k-1}$ の商(注) };
　　for $i \leftarrow k+1$ to n do
　　for $j \leftarrow k+1$ to n do
　　$M^{(k+1)}_{i,j} \leftarrow$ { if $k=1$ then $M^{(1)}_{1,1} M^{(1)}_{i,j} - M^{(1)}_{i,1} M^{(1)}_{1,j}$
　　else $(M^{(k)}_{k,k} \otimes M^{(k)}_{i,j} - M^{(k)}_{i,k} \otimes M^{(k)}_{k,j}) \otimes Q^{(k-1)}$ };
　　end;
　　end;
　　replace Y_k by X_k;
　　end;
Step 3 [Recover original polynomial.]
　　for $i \leftarrow 1$ to $n-1$ substitute $M^{(i)}_{i,i}$ for X_i in $M^{(n)}_{n,n}$;
Step 4 [Return.]
　　return $D \leftarrow M^{(n)}_{n,n}$;
(注)：次節参照

Y_i は、k回目の消去の際，行列の要素多項式には $X_1, \cdots, X_{k-1}$ のみが主変数として現われるようにするための，中間変数である。

6. 乗算⊗の一実現法と $Q^{(k-1)}$ の計算法

多項式の内部表現として，多くの数式処理システムで採用されている帰納的表現を使えば，乗算⊗は容易に実現できる。ここで，多項式 $P(X_1,\cdots,X_k)$ の帰納的表現とは，P を X_1 の多項式とみて

$$P(X_1,\cdots,X_k)=P_\ell(X_2,\cdots,X_{k-1})X_1^\ell+\cdots+P_1(X_2,\cdots,X_{k-1})X_1+P_0(X_2,\cdots,X_k)$$

と表現し、$P_\ell,\cdots,P_0$ も同様に表現する方法のことである。

$M_{ij}^{(k)}$ は $X_1,\cdots,X_{k-1}$ の各々の変数について1次であるから，これを $X_1,\cdots,X_{k-1}$ の順に主変数とみなして帰納的に表現すると

$$\begin{aligned}M_{ij}^{(k)}&=P(X_1,\cdots,X_{k-1})=P_1^{(1)}(X_2,\cdots,X_{k-1})X_1+P_0^{(1)}(X_2,\cdots,X_{k-1})\\&=\{P_{1,1}^{(2)}(X_3,\cdots,X_{k-1})X_2+P_{1,0}^{(2)}(X_3,\cdots,X_{k-1})\}X_1\\&\quad+\{P_{0,1}^{(2)}(X_3,\cdots,X_{k-1})X_2+P_{0,0}^{(2)}(X_3,\cdots,X_{k-1})\}\\&=\cdots\cdots\end{aligned}$$

とかける。そこで，二つの多項式 $P(X_1,\cdots,X_{k-1})$ と $\tilde{P}(X_1,\cdots,X_{k-1})$ が上のように表現されているとき，

$$\begin{aligned}P\otimes\tilde{P}&=P_1^{(1)}\otimes\tilde{P}_1^{(1)}X_1+(P_1^{(1)}\otimes\tilde{P}_0^{(1)}+P_0^{(1)}\otimes\tilde{P}_1^{(1)})\\&=\cdots\cdots\end{aligned}$$

ただし，⊗は主変数以外の変数については普通の乗算と同じ

と計算すれば，$P\cdot\tilde{P}$ のうち $(X_1X_2\cdots X_{k-1})$ に比例する項の係数多項式が計算できる。この計算法では、$P\cdot\tilde{P}$ のうち $(X_1X_2\cdots X_{k-1})$ に比例しないものは全く計算されないから高速で，かつ余分なメモリを必要としない。すなわち，Bareissの方法の欠点である中間表式膨張が解消されたことになる。

上述の計算法を用いると，(4-3)式の $Q^{(k-1)}$ も効率的に計算することができる。

$$M_{k-1,k-1}^{(k-1)}=X_1\cdots X_{k-1}+R(X_1,\cdots,X_{k-1}),\quad R\text{は主変数に関して}(k-2)\text{次以下.}$$

とかけることに注意すると

$$(X_1X_2\cdots X_{k-1})^2/M_{k-1,k-1}^{(k-1)}=X_1X_2\cdots X_{k-1}-R+R^2/(X_1X_2\cdots X_{k-1}+R)$$

となる。R^2 のうち $(X_1X_2\cdots X_{k-1})$ に比例する項だけが $Q^{(k-1)}$ に寄与するが，それらの項はすべて $R\otimes R$ で計算される。よって，上式は

$$X_1X_2\cdots X_{k-1}-R+R\otimes R-(R\otimes R)\cdot R/(X_1X_2\cdots X_{k-1}+R)$$

となる。これをくり返すと，$Q^{(k-1)}$ として次の表式を得る。

$$Q^{(k-1)}=X_1X_2\cdots X_{k-1}-R+R\otimes R-(R\otimes R)\otimes R+\cdots$$

この式の右辺は，$R\otimes R\otimes\cdots\otimes R$ が0になるまで計算をする。

7. アルゴリズムのテスト

我々のアルゴリズムをREDUCE上にインプリメントし，REDUCEに既に組込まれている，Bareissの方法および小行列式法と比較した。実験に用いた例題は，文献6で用いられた例題のうち，要素が多変数多項式であるVandermonde行列と，対称Toeplitz行列，および一変数多項式を要素とする行列の行列式である。その結果は次の通りであった。

表．実験結果（単位：秒）

問題	n	小行列式法	我々の方法	Bareiss法
Vandermonde	3	0.422	0.460	0.429
	4	1.135	1.985	5.563
	5	5.375	11.684	43.092
	6	33.114	78.590	
対称 Toeplitz	3	0.615	0.890	0.876
	4	3.706	3.639	7.472
	5	19.579	16.696	46.606
	6	118.312	93.917	471.206
	7	822.449	579.006	
一変数 行列式	3	2.047	2.255	1.037
	4	14.000	27.768	3.287
	5	137.988	278.286	8.193

(注) Vandermonde行列： $a_{ij} = (x_j)^{i-1}$

対称Toeplitz行列： $a_{ij} = x_{|i-j|}$

一変数多項式の行列： $a_{ij} = \sum_{k=0}^{i+j-2} x^k$

上の結果をみると，一変数行列式に対しては予想通りBareissの方法が我々の方法を上回るが，多変数多項式行列に対しては，我々の方法はBareissの方法を改善して，小行列式法と同程度の効率を示すことがわかる。しかし，一変数行列式でBareissの方法に負けるということは，我々にの方法における変数のおきかえく特にStep 3の X_i を $M_{ii}^{(i)}$ でおきかえるところ）がかなりの時間を要しているかも知れない。ハッシングでREDUCEのSUBコマンドを高速にすると、我々の方法はさらに速くなる可能性をもっている。

本研究にあたっては、津田塾大学の渡辺隼郎氏および名大プラズマ研の金田康正氏から有益なコメントを頂き、またいろいろ討論して頂いた。両氏に謝意を表したい。

参考文献

1) E. H. Bareiss, "Sylvester's identity and multistep integer-preserving Gaussian elimination," Math. Comp. 22 (1968), pp. 565-578.

2) M. T. McClellan, "The exact solution of systems of linear equations with polynomial coefficients," J. ACM 20 (1973), pp. 563-588.

3) W. M. Gentleman and S. C. Johnson, "Analysis of algorithms, a case study: Determinants of polynomials," Proc. 5th ACM Symp. on Theory of Computing, Austin, Texas (1973), pp. 135-142.

4) J. Smit, "The efficient calculation of symbolic determinants," Proc. 1976 ACM Symp. on Symbolic and Algebraic Computation, Yorktown Heights, New York (1976), pp. 105-113.

5) 小田泰元，今福幸春，"数式処理言語ALの改良と応用 — AL-1Eによる記号行列式計算 —，" 情報処理学会記号処理研究会，資料3（1978年，2月）

6) E. Horowitz and S. Sahni, "On computing the exact determinant of matrices with polynomial entries," J. ACM 22 (1975), pp. 38-50