数式処理 *J.JSSAC (2003)*
Vol. 9, No. 4, pp. 52 - 67

論文

整数行列のFrobenius標準形のモジュラー計算法

森継 修一*

筑波大学 図書館情報学系

(Received 2002/3/14 Revised 2002/11/11)

Abstract

A modular method for computing the Frobenius normal forms of integer matrices is presented. This method computes Frobenius normal forms over $\mathbf{Z}_{p_i}$, where p_i's are distinct primes, then it constructs the normal form over $\mathbf{Z}$ by Chinese remainder theorem. In this paper, we describe mainly:

- the detection of unlucky primes;
- new formulation for the efficient computation of a transformation matrix.

The experimental results show that our new algorithm is more efficient than straightforward implementation of conventional methods.

1 はじめに

一般に線形代数の教科書では、行列の代表的な標準形として Jordan 標準形が示されていることが多い。ただし、Jordan 標準形の計算には全固有値が属するまで体の代数拡大が必要であり、これを数式処理で実行する際には、固有値を代数的数として扱うため拡大体上の計算の効率化に難点がある。

これに対し、Frobenius 標準形（有理標準形）は、体の拡大を必要とせず行列要素間の四則演算（有理演算）のみで計算が可能であり、その結果はブロックの並び順まで含めて完全に一意的である。さらに Frobenius 標準形は、もとの行列の特性多項式・最小多項式、固有値の代数的・幾何学的重複度や対応する（一般）固有ベクトルの構成などの完全な情報を Jordan 標準形と同等に含んでおり [17][19][21]、数式処理による行列計算のアルゴリズムを考えるには Jordan 標準形よりも Frobenius 標準形を基礎とする方が適している。

本研究の直接の動機は、固有値法による連立代数方程式の解法 [26][20] において、実験の結果、全体の計算時間の中で有理数行列の Frobenius 標準形の計算が支配的だったことである。Gröbner Basis の計算が飛躍的に進歩しているのに対して、固有値法全体を効率化するためには、行列計算の部分の改良が不可欠である。

*moritsug@slis.tsukuba.ac.jp

Frobenius 標準形の計算量については、$O(n^4)$ の古典的算法 [14][24] が確率的算法 [6][7] によって $O(n^3 \log n)$ に改良され、現在は、変換行列まで含めて $O(n^3)$ で求められるかどうか [25] が議論になっている。本研究で採用した基本アルゴリズムは、$O(n^3 \log n)$ のクラスに属すると考えられる（第 2 章）が、本稿では、標準形および変換行列の計算法の実装まで含めて、実用上の効率を検討することを目的とする。

本研究は、上述の基本アルゴリズムにモジュラー法を導入して高速化を図ろうとするものであり、基本的な発想は Howell[10]（ただし、標準形のみを求め、変換行列は計算しない）と共通している。また、Giesbrecht[6] もモジュラー法を採り上げているが、unlucky な素数の判定が不完全と考えられる（第 4 章）。その他、p-adic 計算法を取り入れた研究 [16][23] があるが、いずれも「標準形」までは求めずに「ブロック対角化（あるいは上三角化）」に留め、特性多項式を求めることを主目的としているので、それ以外の応用には不十分と考えられる。したがって本稿では、有理標準形と変換行列のモジュラー法を用いた計算法において、特に

- unlucky な素数の発見と回避の方法およびその実装
- 変換行列の計算法の改良による計算の高速化の実験的評価

について議論する。

2 行列の Frobenius 標準形

以下の議論は任意の体の元を要素とする行列に対して成り立つが、具体的には、厳密な数値を表現する有理数にとるものとする。すなわち、$A = [a_{ij}]$, $a_{ij} \in \mathbf{Q}$ とおく。

定義 1 (コンパニオン行列)
次の $n \times n$ 正方行列

$$C = \begin{bmatrix} 0 & 1 & & & \\ 0 & 0 & \ddots & & O \\ \vdots & \vdots & \ddots & \ddots & \\ 0 & 0 & \cdots & 0 & 1 \\ c_0 & c_1 & \cdots & c_{n-2} & c_{n-1} \end{bmatrix} \tag{1}$$

は、多項式 $f(x) = x^n - c_{n-1}x^{n-1} - \cdots - c_1 x - c_0$ に随伴するコンパニオン行列と呼ばれる。特に、1 次多項式 $f(x) = x - c_0$ のコンパニオン行列は 1×1 行列 $[c_0]$ とする。

補題 2
行列 C (1) の特性多項式 $\varphi_C(x)$ と最小多項式 $\phi_C(x)$ は、随伴多項式 $f(x) = x^n - c_{n-1}x^{n-1} - \cdots - c_1 x - c_0$ に一致する。

定理 3 (Frobenius 標準形)
(i) 任意の $n \times n$ 正方行列 A は、適当な正則行列 S により次の形のブロック対角行列に相似変換され、これを A の Frobenius（または有理）標準形という。

$$F = S^{-1}AS = C_1 \oplus C_2 \oplus \cdots \oplus C_t. \tag{2}$$

各ブロック行列 $C_i\ (i=1,\cdots,t)$ は、$d_i \times d_i$ コンパニオン行列 (1) であり、C_{i+1} の随伴多項式 $\varphi_{i+1}(x)$ は、C_i の随伴多項式 $\varphi_i(x)$ を割り切る $(i=1,\ldots,t-1)$。

(ii) 与えられた行列 A に対して、(2) の形の分解は常に存在し、かつ Frobenius 標準形 F は一意的に決まる。(相似変換行列 S は一意ではない。) さらに、A の最小多項式 $\phi_A(x)$ は $\varphi_1(x)$ に一致し、A の特性多項式 $\varphi_A(x)$ は $\varphi_1(x)\cdot\varphi_2(x)\cdots\varphi_t(x)$ で与えられる。

行列を (2) のようなブロック対角形に変換して特性多項式を求める方法が Danilevskii 法 [3][4] として古くから知られているが、そこでは $\varphi_t(x) \mid \varphi_{t-1}(x) \mid \cdots \mid \varphi_1(x)$ という整除条件を満たすことを要求していないので、結果が一意とは限らず、得られる $\varphi_1(x)$ は必ずしも A の最小多項式に一致しない。定理 3 の厳密な意味での有理標準形は、「ベクトル空間の巡回分解定理」の構成的証明 [5][9][11] によって与えられ、その計算量は $O(n^3 \log n)$ と解析されている。これは、$O(n^3)$ が行列積などの基本演算に対応し、標準形 (2) におけるブロックの個数 t が $O(\log n)$ に相当すると解釈される [1]。定理に直接に基づく算法は、関係式 $F = S^{-1}AS$ における変換行列 S を先に求めていく形で表現され、[6][7] では、S を構成するための初期ベクトルをランダムにとる確率的算法を与えている。

これに対して、本研究では、[11] を同著者が行列の基本変形の表現に翻訳した形のアルゴリズム [12] を採用する。この場合は、標準形 F を求めることに付随して変換行列 S も求められる。もとの行列 A の要素の配置によってループの反復回数が一定ではないが、完全に確定的な算法である。この場合でも本質的な計算量は上記と共通であり、$O(n^3 \log n)$ のクラスに属すると考えられる。

この算法では、次の基本変形操作を組み合わせて、Gauss 消去法に類似の消去計算を実行することにより、与えられた行列 A を標準形 F に変換していく。

定義 4 (基本変形)
任意の $n \times n$ 正方行列 A に対する次の 3 つの操作を基本変形と呼ぶ。

$op1(k,\ell)$ ：A の第 k 行と第 ℓ 行を入れ換え、続いて第 k 列と第 ℓ 列を入れ換える。
$op2(k,c)$ ：A の第 k 行を c^{-1} 倍し、続いて第 k 列を c 倍する。
$op3(k,\ell,c)$ ：A の第 k 行に第 ℓ 行の c 倍を加え、続いて第 ℓ 列から第 k 列の c 倍を引く。

これらの基本変形は行列の相似変換（$A \mapsto S^{-1}AS$）として表現され、行に対する操作が左の変換行列 S^{-1} に、列に対する操作が右の変換行列 S に対応している。(ちなみに Gauss 消去法は、行に対する基本変形のみで実現されるので、$A \mapsto GA$ と表される。)

消去に必要な相似変換を $\cdots S_3^{-1}\left(S_2^{-1}\left(S_1^{-1}AS_1\right)S_2\right)S_3\cdots$ のように順次適用していくと、最終的に式 (2) における F, S, S^{-1} が得られるが、基本変形操作の定義より、この過程で必要となるのは有理数同士の四則演算のみであることに注意する。

注意 1
コンパニオン行列として (1) を転置させた形で定義し、Frobenius 標準形を (2) の転置で定義する流儀もある。実際、本稿におけるプログラムの本体も [12] に従って F^T を求めている。その場合は、$F^T = S^{-1}A^TS \iff F = S^TAS^{-T}$ という関係を利用し、A

の代わりに A^T から計算を始めて、その結果を再度転置すればよい。この際、変換行列は、転置されるだけでなく、左右が入れ換わることに注意する。

注意 2
行列 A の要素をすべて整数にとった場合、その Frobenius 標準形の各要素も整数となる。A の特性多項式 $\varphi_A(x) = \det(xE - A)$ の係数は、行列式の計算規則から必ず整数になり、各コンパニオンブロック (1) の最下行の要素は、$\varphi_A(x)$ の因子の係数に対応するからである。ただし、変換行列 S, S^{-1} については、いずれか一方を整数行列にとることはできるが、両者を同時に整数行列にとるようなアルゴリズムは知られていない。

3 モジュラー計算アルゴリズム

本稿では整数行列 A のみを対象とし、固有ベクトルの計算への応用 [21] を想定して、$AS = SF$ をみたす F, S を求めることとし、S の各要素が整数になるようアルゴリズムを構成する。

この設定においても、基本変形 $op2(k, c)$ には「c^{-1} 倍する」という除算が含まれているため、途中の段階では行列要素として有理数が現れることになる。本研究ではここにモジュラー計算法の適用を試みる。基本変形における有理演算はすべて素数 p を法として実行可能なので、$\mathbf{Z}_p$ での Frobenius 標準形 $A_pS_p = S_pF_p$ が同じプログラムで求められることになる。(Euclid 互除法により、$cs + pt = 1$ をみたす s, t を求めれば、$s \equiv c^{-1} \pmod{p}$ となり、$op2$ の除算は乗算の形で実現される。)

複数の素数を法として計算した結果 $(F_{p_1}, S_{p_1}), (F_{p_2}, S_{p_2}), \ldots$ から $\mathbf{Z}$ 上での F, S を復元するのに、本稿では Chinese Remainder Theorem の利用を考える。

定理 5 (CRT：法が 2 つの場合)
m_1, m_2 が互いに素な整数のとき、連立合同式の解は、$m_1s + m_2t = 1$ を満たす 1 組の整数 s, t を用いて、次の式で与えられる。

$$\begin{cases} x \equiv a_1 & \pmod{m_1} \\ x \equiv a_2 & \pmod{m_2} \end{cases} \quad \Longrightarrow \quad x \equiv a_1m_2t + a_2m_1s \pmod{m_1m_2}$$

したがって、素数 $p_1,\ p_2,\ p_3, \ldots$ に対して、法を $p_1,\ p_1p_2,\ p_1p_2p_3, \ldots$ と上げていくには、

$$\begin{cases} x \equiv a_{k-1} & \pmod{p_1 \cdots p_{k-1}} \\ x \equiv a_k & \pmod{p_k} \end{cases} \qquad (k = 2,\ 3, \ldots)$$

に対して、定理 5 を繰り返し適用すればよい（Newton の補間公式)。行列 F, S に対しては、それぞれの要素 f_{ij}, s_{ij} について個別に（計 $2n^2$ 回）計算することになる。

一般に、変換行列 S の要素は、与えられた行列 A やその Frobenius 標準形 F の要素に比べて遥かに巨大な整数となり、その上限の見積もりは困難であり、法をどこまで上げたらよいか予想が不可能なので、mod $p_1 \cdots p_{k-1}$ での値 $S^{(k-1)}$ と mod $p_1 \cdots p_{k-1}p_k$ での値 $S^{(k)}$ とが一致したら、整数上で $AS^{(k)} = S^{(k)}F^{(k)}$ を満たしているかどうかを終了判定とする。

以上を用いて、Frobenius 標準形のモジュラー計算をアルゴリズムとしてまとめると、以下のようになる。ここで、行列 A に対して $AS = SF$ なる F, S を求めるサブプログラムは、すでに出来ているものとする。

アルゴリズム 1 (Frobenius 標準形のモジュラー計算：不完全版)

% 入力：整数要素の $n \times n$ 行列 A　　ある程度大きな素数のリスト $\{p_1, p_2, \ldots, p_s\}$
% 　計算過程での $F^{(k)}, S^{(k)}$ は $\bmod p_1 \cdots p_k$ での値を表す。
% 出力：A の Frobenius 標準形 F と相似変換行列 S　（$AS = SF$）
Compute F_{p_1}, S_{p_1} s.t. $A_{p_1} S_{p_1} \equiv S_{p_1} F_{p_1} \pmod{p_1}$;
$k := 1$; $F^{(1)} := F_{p_1}$; $S^{(1)} := S_{p_1}$;
loop : Do until $(S^{(k-1)} = S^{(k)})$
$k := k + 1$;
Compute F_{p_k}, S_{p_k} s.t. $A_{p_k} S_{p_k} \equiv S_{p_k} F_{p_k} \pmod{p_k}$;
Construct $F^{(k)}$ from $F^{(k-1)}$ and F_{p_k} by CRT;
Construct $S^{(k)}$ from $S^{(k-1)}$ and S_{p_k} by CRT;
If $(AS^{(k)} = S^{(k)} F^{(k)}$ (over $\mathbf{Z}$)) then return $\langle F^{(k)}, S^{(k)} \rangle$ else goto loop;

このアルゴリズムには、次節で述べる「unlucky な素数」への対応が含まれていないので、停止性が保証されない。もし停止すれば正しい F, S を出力するが、ここでは「不完全版」としておく。

4 unlucky な素数の発見と回避

定義 6 (unlucky な素数)
$\mathbf{Z}$ 上で求めた標準形 $AS = SF$ および、$\mathbf{Z}_p$ での標準形 $A_p S_p = S_p F_p$ とにおいて、$F \not\equiv F_p \pmod{p}$ または $S \not\equiv S_p \pmod{p}$ となる場合、素数 p は unlucky であるという。

モジュラー計算においては、unlucky な素数を見つけて排除することが必須である。上述のアルゴリズム 1 において、１組でも unlucky な F_p, S_p が含まれていると、いくら法 p_i の個数を増やしても CRT によって F, S を復元することができず、ループが停まらなくなる。

unlucky な素数に関しては、消去計算の過程における pivot 選択の履歴を記録し比較することにより識別できるという事実が、次の補題により示されている。

補題 7 (Howell[10])
lucky な素数 p を法とする場合の pivot 選択のパターンは、$\mathbf{Z}$ 上での計算で起きる pivot 選択のパターンと一致する。

これは、unlucky な素数 p が偶然 pivot 要素を割り切って軸位置に 0 がくると、本来起きないはずの行交換・列交換 $op1(k, \ell)$ が起きることを意味する。したがって、そのような mod p での計算結果は捨てて、$\mathbf{Z}$ 上と同じ pivot 選択パターンをたどった p_i だけを CRT に用いる、というアイデアが得られる。各ステップにおける pivot の素因数は有限個なので、特定の行列に対する unlucky な素数は全体として有限個しか存在しない。

なお、同様にモジュラー算法と CRT の適用を扱った Giesbrecht[6] は、unlucky な素数の判定基準として標準形 $F = C_1 \oplus \cdots \oplus C_t$ における各ブロックのサイズ $(d_1, \ldots, d_t)$ に着目し、

辞書的順序として $(d_1, \ldots, d_t) \succ (d_1^{(p)}, \ldots, d_s^{(p)})$ であれば、p は unlucky である。

という条件を用いている。ただし、これは逆が必ずしも成り立たないことに注意を要する。unlucky な素数であっても、同一のブロックの構造 $(d_1, \ldots, d_t) = (d_1^{(p)}, \ldots, d_t^{(p)})$ を与える例がすでに Howell[10] によって示されており、Giesbrecht[6] の与えたモジュラー算法は（実装報告もないため）不完全と判断せざるを得ない。

したがって本研究では、上記の補題 7 を判定基準として採用する。ただし、$\mathbf{Z}$ 上での計算過程における「正しい pivot 選択のパターン」は未知なので、現在の実装では、最初の 3 つの素数 p_1, p_2, p_3 を法として F_{p_i}, S_{p_i} を計算し、それらのパターンを照合して多数決を行なった結果を「正しいパターン」と仮定して計算を始めることとする。（必要に応じて p_4 以下を追加して照合を行なえば、より確実な推定が可能となるが、実用上は「2 つまたは 3 つの素数に対してパターンが一致すること」を確かめれば十分であろう。）以後の流れは、以下の 2 つのケースがありうる。

1. 仮定したパターンが正しかった場合

   $p_4, p_5, \ldots$ について F_{p_i}, S_{p_i} を求め、pivot 選択のパターンを先の仮定と照合し、一致しない p_i は unlucky と判定して除外する。結果的に lucky/unlucky の判定が正確に行なわれるため、lucky な素数だけを用いて CRT を適用するので、正しい F, S が復元されてアルゴリズムは必ず停止する。結果の確認は、関係式 $AS = SF$ を用いる。

2. 仮定したパターンが誤っていた場合

   $p_4, p_5, \ldots$ について F_{p_i}, S_{p_i} を求めると、pivot 選択のパターンが先の仮定と一致せず、（本当は lucky であるにもかかわらず）unlucky と判定される p_i が続出することになる。このとき lucky と判定される p_i は本当は unlucky なので、CRT によって正しい F, S が復元できず、アルゴリズムは停止しなくなる。したがってこの場合は、途中で処理を打ち切り、最初に戻って「正しいパターン」の推定からやり直すこととする。ただし、打ち切りの判断については、用いる素数の大きさと与えられた行列の成分に依存するので、一律な基準を設けることは不可能であり、経験則によらざるを得ない。

アルゴリズム 1 に対して、以上の方針で unlucky な素数を発見・回避する機能を追加すると、次の形でまとめられる。（ただし、上記 2. のケースについては、「unlucky という判定が 10 回現れたら処理を打ち切る」という単純な形で記述してある。）

アルゴリズム 2 (Frobenius 標準形のモジュラー計算：unlucky 素数対応版)

% 入力：整数要素の $n \times n$ 行列 A　　ある程度大きな素数のリスト $\{p_1, p_2, \ldots, p_s\}$
% 　　計算過程での $F^{(k)}, S^{(k)}$ は $\bmod p_1 \cdots p_k$（ただし unlucky と判定された p_i を
% 　　除く）での値を表す。
% 出力：A の Frobenius 標準形 F と相似変換行列 S　　（$AS = SF$）
Compute F_{p_i}, S_{p_i} s.t. $A_{p_i} S_{p_i} \equiv S_{p_i} F_{p_i}$ $(\bmod p_i \quad i = 1, 2, 3)$;

Presume the correct pivoting pattern using the above 3 results;
% ここでは、p_1, p_3 のパターンが一致したとして、これを正しいパターンと仮定し、
% p_2 は異なるパターンをたどったので unlucky と判定したとする。
Construct $F^{(3)}$ from F_{p_1} and F_{p_3} by CRT;
Construct $S^{(3)}$ from S_{p_1} and S_{p_3} by CRT;
$k := 3$; $count := 1$;
loop : Do until $(S^{(k-1)} = S^{(k)})$
$k := k + 1$;
Compute F_{p_k}, S_{p_k} s.t. $A_{p_k} S_{p_k} \equiv S_{p_k} F_{p_k} \pmod{p_k}$;
If p_k does not yield the presumed pivoting pattern
then { $count := count + 1$; If $count = 10$ then STOP; }
else { Construct $F^{(k)}$ from $F^{(k-1)}$ and F_{p_k} by CRT;
Construct $S^{(k)}$ from $S^{(k-1)}$ and S_{p_k} by CRT } ;
If $(AS^{(k)} = S^{(k)} F^{(k)}$ (over $\mathbf{Z}$)) then return $\langle F^{(k)}, S^{(k)} \rangle$ else goto loop;

このアルゴリズムは、正しい F, S を出力して停止するか、途中で強制終了させられるかのいずれかである。後者、すなわち最初の推定が誤っていた場合は、素数のリストを取り換えて再スタートすることとする（さらに、打ち切りのタイミングを遅らせてみる必要があるかもしれない）。

しかしながら、実際の計算例では、プログラムのテストのため意図的に 2〜3 桁の素数を用いた場合には unlucky な素数が頻出するが、10 桁程度の一連の素数を法にとった場合には、unlucky という判定自体が一度も現れていない。したがって、ある程度大きな素数を用いることにすれば、最初の「正しい pivot 選択パターン」の推定を誤ることはほとんど起こり得ず、実用上、再試行の繰り返しに陥る可能性はほとんどなくなると考えられる。

図 1 は、$-10 \le a_{ij} \le 10$ のランダムな要素の 10×10 行列 A に対するプログラムの出力である。この例では、23,29 を法とすると行交換が起きなかったのでこれを正しいパターンと仮定、その後、59,67,131,149 を法としたときに行交換が起きたのでこれらを unlucky な素数として除外し、他の 27 個の素数による結果から CRT で復元すると、$\mathbf{Z}$ 上の F, S が求められた。なお、上記 4 個の unlucky な素数に対しても、$\mathbf{Z}$ 上と同じくブロック 1 個からなる標準形 $(d_1^{(p)} = 10)$ が得られたので、Giesbrecht[6] の判定基準では不完全であることが実験的にも確かめられた。

5 変換行列の構成法

アルゴリズム 2 を含めて、これまでの「相似変換を順次適用する」という算法を図式的に表せば、

E	A	E	$\longrightarrow$	S^{-1}	F	S

となる。ここで、左側の単位行列には行操作を、右側の単位行列には列操作を順次適用する。

図 1: unlucky な素数の発見・回避の実行例

```
a:=
mat((-2,1,-2,-10,5,-3,-7,-2,0,-5),(5,-1,0,-5,-8,0,-3,5,-5,-9),
    (5,4,4,-3,-1,5,2,8,7,-7),(-5,-8,8,-9,-4,-10,-7,9,7,3),
    (-10,-6,3,-4,4,-1,3,7,8,-8),(2,8,1,-1,9,-4,3,-3,4,5),
    (-6,-6,1,7,-5,1,6,-6,1,3),(7,0,7,-7,-1,7,-7,-5,-2,-7),
    (-8,-5,-6,-8,9,-9,1,-9,6,5),(-6,9,-2,-6,6,0,-3,0,7,7))$
frobenius_nf(A,'F,'int);
lucky primes = (23 . 29)
lucky pivots = (nil nil nil nil nil nil nil nil nil nil)
initial mod = 667
p = 59 unlucky : pivot = (nil (2 . 3) nil nil nil nil nil nil nil)
p = 67 unlucky : pivot = (nil nil (3 . 4) nil nil nil nil nil nil)
p = 131 unlucky : pivot = (nil nil nil nil nil nil nil (8 . 9) nil)
p = 149 unlucky : pivot = (nil nil nil nil nil nil nil nil (0 . 9)
(1 . 9) (2 . 9) (3 . 9) (4 . 9) (5 . 9) (6 . 9) (7 . 9) (8 . 9) nil)
check of S & F over Z : t
product of 27 lucky primes (except 4 unlucky primes) : 52 digits
total time for computation : 2689 ms
```

本稿における定式化では、右側の S だけを求めることにし、その要素が整数になるよう列操作の表現行列 S_k を定めている。しかし、上記の過程 $S = S_1 S_2 \cdots$ において、相似変換のための列操作が累積していくと、一般に S の要素は巨大な整数に成長してしまう。アルゴリズム 2 をそのままインプリメントし、従来の「分数なし計算法」[13][18] に基づくプログラムと比較したところ、必ずしも計算速度の向上が明らかではなかった [22]。これは、S の要素の成長に伴い、モジュラー算法では法 p_i の個数を増やさなければならないためである。

ここで、Frobenius 標準形の定義に立ち帰って考えれば、相似変換行列 S は一意ではないため、$AS = SF$ をみたす正則なものが 1 つ見つかれば十分である。したがって本節では、標準形 F だけを先に求め、その後で、$AS = SF$ をみたす正則行列 S（しかもできるだけ簡潔な要素を持つもの）を構成することを考える。F の要素は S ほどには成長しないので、モジュラー法を適用する場合においても、法 p_i の個数がごく少数ですむことが期待できる。

簡単のため、F が 1 つのブロックからなる場合を考える。$AS = SF$ の関係

$$AS = S \begin{bmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ c_0 & c_1 & \cdots & c_{n-1} \end{bmatrix} \qquad \varphi(x) = x^n - c_{n-1}x^{n-1} - \cdots - c_1 x - c_0$$

において、$S = \left[\, \boldsymbol{s}_1 \,\middle|\, \boldsymbol{s}_2 \,\middle|\, \cdots \,\middle|\, \boldsymbol{s}_n \,\right]$ として両辺を列毎に比較すると

$$\begin{cases} A\boldsymbol{s}_1 &= c_0\boldsymbol{s}_n \\ A\boldsymbol{s}_2 &= \boldsymbol{s}_1 + c_1\boldsymbol{s}_n \\ & \cdots \\ A\boldsymbol{s}_{n-1} &= \boldsymbol{s}_{n-2} + c_{n-2}\boldsymbol{s}_n \\ A\boldsymbol{s}_n &= \boldsymbol{s}_{n-1} + c_{n-1}\boldsymbol{s}_n \end{cases}$$

となる。下の行から順に「A を左から乗じて代入」を繰り返して $\boldsymbol{s}_{n-1}, \ldots, \boldsymbol{s}_1$ を消去すると

$$\left(A^n - c_{n-1}A^{n-1} - \cdots - c_1A - c_0E\right)\boldsymbol{s}_n = \boldsymbol{0} \qquad \text{すなわち} \qquad \varphi(A)\boldsymbol{s}_n = \boldsymbol{0} \tag{3}$$

を得る。これを複数のブロックからなる一般の場合 (ブロックのサイズ : d) で考えると、

- 最初のコンパニオンブロックに対しては、$\varphi(x)$ は最小多項式なので、Cayley-Hamilton の定理により $\varphi(A) = O$ が成り立つから、$\boldsymbol{s}_d$ は任意のベクトルにとることができる。
- ２番目以降のブロックに対しては、斉次方程式 (3) を実際に解く必要があるが、解ベクトルのひとつを $\boldsymbol{s}_d$ にとればよい。

残りの列ベクトルは、漸化式　$\boldsymbol{s}_j = A\boldsymbol{s}_{j+1} - c_j\boldsymbol{s}_d \quad (j = d-1,\ \ldots\ ,1)$　より求めることができる。以上の過程を、一般のブロック対角な標準形に対して示すと、次のようになる。計算量としては、 $n \times n$ の連立１次方程式を、(ブロックの個数 -1) 回だけ解く必要があるので、この部分も $O(n^3 \log n)$ である。

アルゴリズム 3 (変換行列の再構成の算法)

% 入力：整数要素の $n \times n$ 行列 A とその Frobenius 標準形 $F = C_1 \oplus C_2 \oplus \cdots \oplus C_t$
% 　　各 C_k の随伴多項式を $\varphi_k(x) = x^{d_k} - c_{k,d_k-1}x^{d_k-1} - \cdots - c_{k,1}x - c_{k,0}$ とする。
% 出力：$AS = SF$ をみたす行列 $S = \left[\boldsymbol{s}_{1,1} \middle| \cdots \middle| \boldsymbol{s}_{1,d_1} \middle| \cdots \middle| \boldsymbol{s}_{t,1} \middle| \cdots \middle| \boldsymbol{s}_{t,d_t}\right]$
for $k = 1$ to t do
　Compute $\boldsymbol{s}_{k,d_k}$ by solving $\varphi_k(A)\boldsymbol{s}_{k,d_k} = \boldsymbol{0}$;
　for $j = d_k - 1$ downto 1 do $\boldsymbol{s}_{k,j} = A\boldsymbol{s}_{k,j+1} - c_{k,j}\boldsymbol{s}_{k,d_k}$;

このアルゴリズムから得られる S は、$AS = SF$ を満たしているものの、正則であるという保証がない。漸化式における初期ベクトル $\boldsymbol{s}_{k,d_k}$ として不適切なものが選ばれると、行列 S が非正則になってしまう可能性がある。正則な変換行列が無数に存在することは定理 3 によって保証されており、$\mathrm{rank}(S) = n$ かどうかを計算してみれば、出力 S が正則であることの確認は容易なので、もし、非正則な S が得られていた場合は、初期ベクトル $\boldsymbol{s}_{k,d_k}$ を選び直して、再度 S を構成してみるものとする。

以下では、斉次方程式 $\varphi_k(A)\boldsymbol{s}_{k,d_k} = \boldsymbol{0}$ の解としての初期ベクトル $\boldsymbol{s}_{k,d_k}$ がみたすべき条件について検討する。まず、次の概念を用いる。

定義 8 (最小消去多項式)
$n \times n$ 行列 A と n 次元列ベクトル$\boldsymbol{v}$ について、$f(A)\boldsymbol{v} = \mathbf{0}$ をみたす多項式 $f(x)$ は $\boldsymbol{v}$ を消去するといい、$\boldsymbol{v}$ を消去する最小次数のモニックな多項式を$\boldsymbol{v}$ の最小消去多項式という。

再び、標準形 F が 1 つのブロックからなる場合を考える。漸化式から求められた $\boldsymbol{s}_{n-1}, \ldots, \boldsymbol{s}_1$ と $\boldsymbol{s}_n$ の関係は、次のように表される。

$$\begin{cases} \boldsymbol{s}_{n-1} &= (A - c_{n-1}E)\boldsymbol{s}_n \\ \boldsymbol{s}_{n-2} &= (A^2 - c_{n-1}A - c_{n-2}E)\boldsymbol{s}_n \\ & \cdots \\ \boldsymbol{s}_2 &= (A^{n-2} - c_{n-1}A^{n-3} - \cdots - c_3A - c_2E)\boldsymbol{s}_n \\ \boldsymbol{s}_1 &= (A^{n-1} - c_{n-1}A^{n-2} - \cdots - c_3A^2 - c_2A - c_1E)\boldsymbol{s}_n \end{cases} \tag{4}$$

$\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_n$ の 1 次独立性を調べるため、 $t_1\boldsymbol{s}_1 + t_2\boldsymbol{s}_2 + \cdots + t_{n-1}\boldsymbol{s}_{n-1} + t_n\boldsymbol{s}_n = \mathbf{0}$ とおいて、(4) を代入して整理すると、

$$\begin{aligned} \{ & t_1A^{n-1} + (t_2 - c_{n-1}t_1)\,A^{n-2} + \cdots \\ & + (t_{n-1} - c_{n-1}t_{n-2} - \cdots - c_3t_2 - c_2t_1)\,A \\ & + (t_n - c_{n-1}t_{n-1} - c_{n-2}t_{n-2} - \cdots - c_2t_2 - c_1t_1)\,E\}\,\boldsymbol{s}_n = \mathbf{0} \end{aligned} \tag{5}$$

を得る。(3) の解ベクトル $\boldsymbol{s}_n$ $(\neq \mathbf{0})$ の最小消去多項式が真に n 次ならば、上式 (5) において $\{\quad\}$ 内 $= O$ でなければならない。ところがそのためには、行列 A の最小多項式は n 次なので、A の $n-1$ 次式である $\{\quad\}$ 内においては $A^{n-1}, A^{n-2}, \ldots, A, E$ の各係数が 0 でなければならず、$t_1 = 0,\ t_2 = 0,\ \ldots,\ t_{n-1} = 0,\ t_n = 0$ が順次定まることになる。したがって、$\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_n$ が 1 次独立であるための条件は、以下のようにまとめられる。

命題 9
$n \times n$ の Frobenius 標準形 F が 1 つのブロックからなる場合、初期ベクトル $\boldsymbol{s}_n$ の最小消去多項式が真に n 次であれば、アルゴリズム 3 の出力である変換行列 S は正則となる。

不適切な $\boldsymbol{s}_n$ の例として、行列 A のある固有ベクトルに一致してしまった場合があげられる。このとき、$A\boldsymbol{s}_n = \lambda\boldsymbol{s}_n$ より $\boldsymbol{s}_n$ は 1 次の消去多項式 $f(x) = x - \lambda$ を持ち、漸化式から $\boldsymbol{s}_{n-1} = A\boldsymbol{s}_n - c_{n-1}\boldsymbol{s}_n = (\lambda - c_{n-1})\,\boldsymbol{s}_n$ となり、$\boldsymbol{s}_n$ と $\boldsymbol{s}_{n-1}$ は 1 次従属となる。

上記の議論を複数のブロックからなる標準形 F の場合に一般化する。サイズ $d \times d$ のブロックに対応する S の列ベクトル $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_d$ は、$\varphi(A)\boldsymbol{s}_d = \mathbf{0}$ (3) の解 $\boldsymbol{s}_d$ を初期ベクトルとして計算する。ここで、$\varphi(x)$ は A の特性多項式の因子である d 次式なので、簡単のため、相異なる d 個の固有値を根にもつとする（重複固有値がある場合、必要に応じて一般固有ベクトルに拡張して考える）と、(3) は

$$\{(A - \lambda_1E)\cdots(A - \lambda_dE)\}\,\boldsymbol{s}_d = \mathbf{0} \tag{6}$$

と表され、その解としての $\boldsymbol{s}_d$ は、λ_i (A 全体の固有値としての幾何学的重複度を k_i とする）に対応する固有ベクトルを $\boldsymbol{v}_{ij}\quad (j = 1, \ldots, k_i)$ として、

$$\boldsymbol{s}_d = (b_{11}\boldsymbol{v}_{11} + \cdots + b_{1k_1}\boldsymbol{v}_{1k_1}) + \cdots + (b_{d1}\boldsymbol{v}_{d1} + \cdots + b_{dk_d}\boldsymbol{v}_{dk_d}) \qquad \boldsymbol{s}_d \in \mathbf{Z}^n, \quad \boldsymbol{v}_{ij} \in (\mathbf{Z}[\lambda_i])^n \tag{7}$$

と表される。このとき、$\boldsymbol{s}_d$ の最小消去多項式が真に d 次となるための条件から次を得る。

命題 10
Frobenius 標準形 F におけるサイズ $d \times d$ のブロックでは、初期ベクトル $\boldsymbol{s}_d$ を (7) のように表したとき、

すべての添字 i について、「$b_{i1}, \ldots, b_{ik_i}$ の中に非零のものが 1 個以上存在する」

ならば、アルゴリズム 3 から得られた S の列ベクトル $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_d$ は、1 次独立となる。

すなわち、$\boldsymbol{s}_d$ が A の固有値 $\lambda_1, \ldots, \lambda_d$ に対応する固有ベクトルの成分を各 1 個ずつ以上「満遍なく」含むようにとれれば、適切な初期ベクトルとなる。具体的に固有ベクトル $\boldsymbol{v}_{ij}$ を求めるのは現実的でないので、上記の命題 9,10 は $\boldsymbol{s}_n, \boldsymbol{s}_d$ の直接的な構成法を与えるものではないが、一定の条件下でアルゴリズム 3 が正則な S を出力することが示された。

実際に (3) を解く際には、整数上で分数なしガウスの消去法により計算を進めて、ある 1 次独立な基底ベクトルの組を用いたパラメータ表示により、

$$\boldsymbol{s}_d = u_1 \boldsymbol{w}_1 + \cdots + u_r \boldsymbol{w}_r \qquad \boldsymbol{s}_d, \boldsymbol{w}_i \in \mathbf{Z}^n \tag{8}$$

の形で解が求められる。((7) と (8) は、$r = \sum_{i=1}^{d} k_i$ で関係付けられる。) このとき、$\boldsymbol{s}_d$ の表現 (8) におけるパラメータ u_i をランダムな整数で与えた場合、表現 (7) において、$b_{i1} = \cdots = b_{ik_i} = 0$ なる i が発生する確率は ($\lambda_i \in \mathbf{Z}$ の場合を除いて) 極めて低いと考えられる。したがって、現在の実装では、パラメータ u_i に対して $\{+1, -1\}$ をランダムに割り振ることにして、このブロックに対応する固有ベクトルの成分を「満遍なく」含む初期ベクトル $\boldsymbol{s}_d$ を選ぶと同時に S の要素の成長を抑えることを試みている。

アルゴリズム 3 の出力 S に対しては、ガウスの消去法で rank を計算することにより S の正則性を確認し、もし非正則ならば、上述の u_i の値の割り振りからやり直すこととする。この部分 (下記のアルゴリズム 4 の中の while ループ) で試行錯誤が繰り返される可能性がある点では確率的アルゴリズムであるが、実験的には、この過程で非正則な変換行列が生成された失敗例は見つかっていない。

モジュラー計算による標準形の計算と、変換行列の再構成を組み合わせて、全体のアルゴリズムとしては、以下のような形にまとめられる。unlucky な素数の扱いと停止性については、アルゴリズム 2 と同様である。また、CRT の終了判定は、$F^{(k-1)} = F^{(k)}$ となった時点で、この $F^{(k)}$ から導かれる $F^{(k)}$ および A の最小多項式 $\varphi(x)$ を用いて、$\varphi(A) = O$ を確かめるものとする。したがって、CRT の繰り返しが終了すれば、正しい Frobenius 標準形 F が得られていることになる。

アルゴリズム 4 (Frobenius 標準形のモジュラー計算＋変換行列の再構成)

% 入力：整数要素の $n \times n$ 行列 A　　ある程度大きな素数のリスト $\{p_1, p_2, \ldots, p_s\}$
% 　計算過程での $F^{(k)}$ は mod $p_1 \cdots p_k$ (ただし unlucky と判定された p_i を除く)
% 　での値を表す。S_{p_k} は保存しない。
% 出力：A の Frobenius 標準形 F と相似変換行列 S　　($AS = SF$)

Compute F_{p_i} s.t. $A_{p_i}S_{p_i} \equiv S_{p_i}F_{p_i} \pmod{p_i} \quad i = 1, 2, 3$);
Presume the correct pivoting pattern using the above 3 results;
% ここでは、p_1, p_3 のパターンが一致したとして、これを正しいパターンと仮定し、
% p_2 は異なるパターンをたどったので unlucky と判定したとする。
Construct $F^{(3)}$ from F_{p_1} and F_{p_3} by CRT;
$k := 3$; $count := 1$;
loop : Do until $(F^{(k-1)} = F^{(k)})$
$k := k + 1$;
Compute F_{p_k} s.t. $A_{p_k}S_{p_k} \equiv S_{p_k}F_{p_k} \pmod{p_k}$;
If p_k does not yield the presumed pivoting pattern
then { $count := count + 1$; If $count = 10$ then STOP; }
else Construct $F^{(k)}$ from $F^{(k-1)}$ and F_{p_k} by CRT;
If $\varphi(A) \neq O$ then goto loop; % $\varphi(x)$ = min.pol. of $F^{(k)}$
Construct S from $A, F^{(k)}$ by Algorithm-3 (over $\mathbf{Z}$);
While rank$(S) < n$ (over $\mathbf{Z}$) do reconstruct S by Algorithm-3;
Return $\langle F^{(k)}, S \rangle$

6 実験的評価

以上に示したアルゴリズム 2,4 を、数式処理システム Reduce3.6[8] および RLISP '88[15] を用いてインプリメントし、両者の比較を行なった。機種は H9000VK270 (CPU : PA-8000, 160MHz)、メモリは 128MB で制限し、HP-UX 版 Reduce3.6 を使用した。

なお、素数のリストとしては、2^{31} 未満の

$$\{p_1, p_2, \ldots, p_{1000}\} = \{2147483647,\ 2147483629,\ \ldots,\ 2147462143\}$$

を用意し、すべての行列に対して、この順に適用した。

表 1 の行列は、12×12 から 30×30 までの 7 個について、$-10000 \sim 10000$ の範囲のランダムな整数を要素として与えた。このようにランダムに行列要素を定めると、ほとんどの場合、コンパニオンブロック 1 つからなる Frobenius 標準形を持つ。

表 2 の行列は標準形がブロックに分かれる場合で、ランダムな係数を与えられた多項式に随伴するコンパニオンブロックから有理標準形を想定し、逆に相似変換を施して作成した。各行列の標準形は、以下のような構造を持ち、2 個のブロックに分かれる 12×12 から 5 個のブロックに分かれる 42×42 まで、9 個の行列を用意した。

$12(8,4)$ $14(10,4)$ $16(9,5,2)$ $18(10,6,2)$ $20(11,6,3)$ $25(10,8,5,2)$
$30(12,9,6,3)$ $30(10,8,6,4,2)$ $42(12,10,8,6,4,2)$

計算時間の測定結果を表 1,2 に示す。$\#p_i$ は法として用いた素数の個数、len$|s_{ij}|$ は変換行列 S の要素の最大桁数である。

表 1: CPU-Time(sec) for integer matrices I

	Algorithm-2			Algorithm-4				Maple	
n	$\#p_i$	len$\|s_{ij}\|$	T_2	$\#p_i$	len$\|s_{ij}\|$	T_4	T_4/T_2	T_M	T_4/T_M
12	31	279	16.23	7	47	2.05	0.126	13.20	0.155
14	43	384	44.02	8	55	4.25	0.097	39.17	0.109
16	56	507	103.42	9	63	8.30	0.080	100.53	0.083
18	71	651	227.38	10	73	16.09	0.071	229.86	0.070
20	89	818	485.72	11	81	29.28	0.060	497.32	0.059
25	141	1301	2391.33	13	103	112.31	0.047	2496.91	0.045
30	206	1911	9333.34	15	126	361.43	0.039	9003.52	0.040

表 2: CPU-Time(sec) for integer matrices II

	Algorithm-2			Algorithm-4				Maple	
n	$\#p_i$	len$\|s_{ij}\|$	T_2	$\#p_i$	len$\|s_{ij}\|$	T_4	T_4/T_2	T_M	T_4/T_M
12	10	75	3.06	3	25	1.22	0.399	2.47	0.494
14	15	123	7.83	3	38	2.11	0.269	6.21	0.340
16	16	133	11.46	3	36	3.73	0.326	13.38	0.279
18	18	157	18.84	3	43	6.10	0.324	27.88	0.219
20	21	181	31.09	3	44	9.82	0.316	33.29	0.295
25	32	288	123.80	3	78	40.16	0.324	175.13	0.229
30	38	338	265.84	3	100	104.94	0.395	360.71	0.291
30	45	406	366.10	4	102	124.98	0.341	427.50	0.292
42	93	851	3546.36	4	146	997.12	0.281	1544.37	0.646

表 1 の実験例ではすべて、コンパニオン行列 1 つからなる標準形が得られたため、変換行列 S を構成する際に、斉次方程式 (3) を解く必要がない。結果的に n が大きくなるほど効果が増大していることが分かる。

表 2 の実験例では、標準形がブロック構造を持つため、変換行列 S を構成する際に、斉次方程式 (3) を実際に解いている。そのため、速度向上の効果は表 1 の例ほどではないが、それでも約 28〜40%となっている。

両タイプの行列に対して、アルゴリズム 4 では、アルゴリズム 2 に比べて S の要素の成長を相当に抑えることができたため、法の個数 k が非常に小さくて済み、結果として計算時間の改善が見られた。

他の商業的数式処理システムでは、Maple[2] のみが Frobenius 標準形を計算する組込関数を持っていて、アルゴリズムの詳細は不明であるが、参考のため、同じテスト行列に対する

計算時間を表 1,2 に示してある。これらの実験例については、Maple とアルゴリズム 2 が同程度の速度であり、アルゴリズム 4 は Maple よりも計算速度が向上していることが分かる。

7 まとめと今後の課題

本研究では、モジュラー法を用いて整数行列の Frobenius 標準形の計算アルゴリズムを改良し、実装によりその効果を確認した。モジュラー法の適用そのものは以前から複数提案されてきたが、標準形および変換行列の計算まできちんと実装して議論した研究は、過去には見られない。

(1) unlucky な素数の発見・回避については、ほぼ実現し、失敗の確率はかなり低く抑えられたと考えられる。

(2) 変換行列 S を A, F から再構成するアルゴリズムを実装したところ、実験結果から有望と思われる。特に、変換行列の成分の成長を抑えるための計算法は、従来の研究には含まれない、全く新しい視点である。今後は確率的部分の評価・検証を行ない、より要素の成長を抑える計算法を考察したい。

(3) Maple の組込関数 frobenius については、$\left(S^T\right)^{-1}$ を求める仕様になっているので、本研究と同じ用途に適用するためには、さらに逆行列の計算が必要である。その際、S の要素の成長を考慮していない点が短所になりうるので、独自のプログラム開発を続けることに意義がある。

(4) 最終的には、プログラムの対象を有理数要素の行列への拡張する必要がある。その際、最初に共通分母をくくり出して $\mathbf{Z}$ 上で計算して、

$$\frac{1}{k}\begin{bmatrix} & & & p \\ 1 & & & q \\ & 1 & & r \\ & & 1 & s \end{bmatrix} \approx \begin{bmatrix} & & & p/k^4 \\ 1 & & & q/k^3 \\ & 1 & & r/k^2 \\ & & 1 & s/k \end{bmatrix}$$

の関係を利用して $\mathbf{Q}$ 上に戻すことが可能である。一方で、有理数そのものをモジュラー計算から復元することもできるので、どの段階で $\mathbf{Z}_p$ 上から $\mathbf{Q}$ 上へ戻すのがよいか、比較検討する必要がある。

参 考 文 献

[1] Augot, D. and Camion, P.: On the Computation of Minimal Polynomials, Cyclic Vectors, and Frobenius Forms, *Linear Algebra and its Applications*, **260**, 1997, 61–94.

[2] Char, B. W., et al.: *Maple V Library Reference Manual*, Springer, N.Y., 1991.

[3] Danilevskii, A. M.: On the numerical solution of the secular equation, *Mat.Sb.*, **2**(1), 1937, 169–172. (in Russian).

[4] ファジェーエフ, ファジェーエバ: 線型代数の計算法 (上), 産業図書, 東京, 1970.

[5] Gantmacher, F. R.: *The Theory of Matrices (2nd ed.)*, **1**, Chelsea, N.Y., 1959.

[6] Giesbrecht, M.: Fast Algorithms for Rational Forms of Integer Matrices, *ISSAC 94*, ACM, N.Y., 1994, 305–311.

[7] Giesbrecht, M.: Nearly Optimal Algorithms for Canonical Matrix Forms, *SIAM J. Comput.*, **24**(5), 1995, 948–969.

[8] Hearn, A. C.: *Reduce User's Manual (Ver. 3.6)*, RAND Corp., Santa Monica, 1995.

[9] Hoffman, K. and Kunze, R.: *Linear Algebra (2nd ed.)*, Prentice-Hall, New Jersey, 1971.

[10] Howell, J. A.: An Algorithm for the Exact Reduction of a Matrix to Frobenius Form Using Modular Arithmetic. I & II, *Math.Comp.*, **27**(124), 1973, 887–920.

[11] 伊理正夫, 韓太舜: 線形代数 - 行列とその標準形, 教育出版, 東京, 1977.

[12] 韓太舜, 伊理正夫: ジョルダン標準形, 東京大学出版会, 東京, 1982.

[13] 栗山和子, 森継修一: 行列の有理標準形の分数なし計算法, 日本応用数理学会論文誌, **6**(3), 1996, 253–264.

[14] Lüneburg, H.: *On the Rational Normal Form of Endomorphisms: A Primer to Constructive Algebra*, Wissenschaftsverlag, Mannheim, 1987.

[15] Marti, J.: *RLISP '88: An Evolutionary Approach to Program Design and Reuse*, World Scientific, Singapore, 1993.

[16] Mathieu, M.-H. and Ford, D.: On p-adic Computation of the Rational Form of a Matrix, *J.Symbolic Computation*, **10**(5), 1990, 453–464.

[17] 森継修一, 栗山和子: 行列の Jordan 標準形の数式処理による厳密計算法, 日本応用数理学会論文誌, **2**(1), 1992, 91–103.

[18] Moritsugu, S. and Kuriyama, K.: Fraction-free Method for Computing Rational Normal Forms of Polynomial Matrices, RISC-Linz Report Series 97-18, RISC-Linz, 1997.

[19] Moritsugu, S. and Kuriyama, K.: On Multiple Zeros of Systems of Algebraic Equations, *ISSAC 99*, ACM, N.Y., 1999, 23–30.

[20] Moritsugu, S. and Kuriyama, K.: A Linear Algebra Method for Solving Systems of Algebraic Equations, *J.JSSAC*(日本数式処理学会誌), **7**(4), 2000, 2–22.

[21] 森継修一, 栗山和子: 行列の固有値・固有ベクトル・一般固有ベクトルの数式処理による記号的計算法, 日本応用数理学会論文誌, **11**(2), 2001, 103–120.

[22] 森継修一, 栗山和子: 整数行列の Frobenius 標準形のモジュラー計算法, 京都大学数理解析研究所講究録, **1199**, 2001, 220–227.

[23] Mukhopadhyay, A.: Exact Computation of the Characteristic Polynomial of an Integer Matrix, *AAECC 3*, *Lect. Notes in Comp. Sci.*, **229**, 1985, 316–324.

[24] Ozello, P.: *Calcul Exact des Formes de Jordan et de Frobenius d'une Matrice*, PhD thesis, Université Scientifique Technologique et Medicale de Grenoble, 1987.

[25] Storjohann, A.: An $O(n^3)$ Algorithm for the Frobenius Normal Form, *ISSAC 98*, ACM, N.Y., 1998, 101–104.

[26] 竹島卓, 横山和弘: 連立代数方程式の一解法 - 剰余環上の線形写像の固有ベクトルの利用, 数式処理通信, **6**(4), 1990, 27–36.