Vol. 45 No. 6 IPSJ Journal June 2004

Regular Paper

A Practical Implementation of Modular Algorithms for Frobenius Normal Forms of Rational Matrices

Shuichi Moritsugu[†]

Modular algorithms for computing the Frobenius normal forms of integer and rational matrices are presented and their implementation is reported. These methods compute Frobenius normal forms over $\mathbf{Z}_{p_i}$, where p_i's are distinct primes, and then construct the normal forms over $\mathbf{Z}$ or $\mathbf{Q}$ by the Chinese remainder theorem. Our implementation includes: (1) detection of unlucky primes, (2) a new formula for the efficient computation of a transformation matrix, and (3) extension of our preceding algorithm over $\mathbf{Z}$ to one over $\mathbf{Q}$. Through experiments using a number of test matrices, we confirm that our modular algorithm is more efficient in practical terms than the straightforward implementation of conventional methods.

1. Introduction

In a previous paper[19], we studied a modular algorithm for the exact computation of Frobenius normal forms of integer matrices, and substantiated its efficiency by computational experiments. In this paper, we extend the above algorithm to the case of rational matrices. We propose several algorithms and compare their efficiency and effectiveness through experiments.

The motivation for these studies is the fact that computing various types of normal forms of matrices is a very important subject in symbolic linear algebra. In particular, the Jordan normal form is theoretically fundamental, but it requires an algebraic extension field to which all the eigenvalues belong. Hence, we have to handle algebraic numbers efficiently in computer algebra systems, but this is often so difficult that computing Jordan normal forms becomes generally inefficient.

In contrast, the Frobenius normal form is computed only by rational operations of the matrix elements without algebraic extensions, and it is absolutely unique, while the Jordan normal form is unique up to the permutation of Jordan blocks. Moreover, the Frobenius normal form provides the same detailed information on the matrix as the Jordan normal form; for example[24]:

- the characteristic polynomial,
- the minimal polynomial,
- the algebraic and geometric multiplicities of the eigenvalues, and
- the structure of eigenvectors or generalized eigenvectors.

In addition, algorithms for symbolic computation of Jordan normal forms can be realized by transforming the Frobenius normal forms[9],[20]. Therefore, Frobenius normal forms are the most suitable for constructing linear algebra algorithms by means of computer algebra systems. They also have important applications such as the eigenvalue method for solving systems of algebraic equations[22],[23],[30].

Algorithms for Frobenius normal forms have been discussed in a number of papers, but these have been concerned mainly with the theoretical computational complexity. As an improvement over the classical $O(n^4)$ algorithm[16],[26], the $O(n^3)$ algorithm was proposed in Storjohann [29], and a series of algorithms by Giesbrecht[6]~[8] are known to give the best complexity. Nevertheless, implementations of such algorithms have rarely been reported, and Maple[2] is the only existing commercial computer algebra system to provide a function for Frobenius normal forms.

The purpose of this paper is to describe the implementation of computing normal forms and transformation matrices, comparing algorithms from a practical point of view. The algorithm adopted here belongs to the $O(n^3 \log n)$ class[1] and is not necessarily optimal. However, instead of precise analysis of the complexity, we focus on the practical application of modular methods for the computation of normal forms.

The basic idea of the modular algorithm was first explained by Howell[12], where only Frobenius normal forms are computed, without transformation matrices. In Mathieu and Ford[18] and Mukhopadhyay[25], p-adic methods

† Institute of Library and Information Science, University of Tsukuba

for the computation of characteristic polynomials were proposed, in which only block diagonalized forms or upper triangular forms are computed. The latest paper by Giesbrecht and Storjohann[8] also describes a modular method and an algorithm for a transformation matrix. The detection of unlucky primes is treated in both Howell[12] and Giesbrecht and Storjohann[8], but since the notions in those studies are slightly different, the procedure is elucidated in this paper from the viewpoint of actual implementation.

As another approach for Frobenius normal forms, we previously proposed a fraction-free algorithm[21] for polynomial matrices. This algorithm is also applicable to integer matrices, but it does not necessarily yield a dramatic improvement in efficiency. In this paper, therefore, we apply modular methods to integer and rational matrices, and focus particularly on the following:

- detection of unlucky primes and its implementation,
- efficient computation of transformation matrices with simple elements, and
- recovery of rational numbers and algorithms for rational matrices.

The implementation for integer matrices was reported in our previous paper[19]. Theories necessary for both integer matrices and rational matrices are summarized here in sections 2 and 3. We show an extension of the method to rational matrices in section 4, and discuss its efficiency with reference to the experimental results in section 5. Most other papers discuss only integer matrices, because it is theoretically sufficient to extract the common denominator of rational matrices. However, our results show that such an approach is quite inefficient in actual implementation.

2. Frobenius Normal Forms of Matrices

In the following sections, we assume that the elements of a given matrix are rational numbers; that is, $A = [a_{ij}]$, $a_{ij} \in \mathbf{Q}$. Refer to textbooks by Iri and Kan[13] and Kan and Iri[14] for details of the normal forms theory.

Definition 1 (Companion Matrix). The following $n \times n$ square matrix

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

is called the companion matrix associated with the polynomial $f(x) = x^n - c_{n-1}x^{n-1} - \cdots - c_1x - c_0$. In particular, the companion matrix associated with the polynomial of degree one $f(x) = x - c_0$ is the 1×1 square matrix $[c_0]$. ∎

Lemma 2. The characteristic polynomial $\varphi_C(x)$ and the minimal polynomial $\phi_C(x)$ of the companion matrix C are equal to the associated polynomial $f(x) = x^n - c_{n-1}x^{n-1} - \cdots - c_1x - c_0$. ∎

Theorem 3 (Frobenius Normal Form).

(i) Using a suitable regular matrix S, every $n \times n$ square matrix A can be transformed into a block diagonal matrix as follows:

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

This is called the Frobenius normal form (or rational normal form) of A. Each block matrix C_i $(i = 1, \cdots, t)$ is a $d_i \times d_i$ companion matrix in the form of Eq.(1), and the associated polynomial $\varphi_{i+1}(x)$ of C_{i+1} divides the associated polynomial $\varphi_i(x)$ of C_i $(i = 1, \ldots, t-1)$.

(ii) For every given matrix A, its Frobenius normal form F of Eq.(2) is uniquely determined, while a transformation matrix S is not unique. Furthermore, the minimal polynomial of A is given by $\phi_A(x) = \varphi_1(x)$, and the characteristic polynomial of A is given by $\varphi_A(x) = \varphi_1(x) \cdot \varphi_2(x) \cdots \varphi_t(x)$. ∎

For the computation of characteristic polynomials, there is a similar algorithm using matrix block diagonalization, as in Eq.(2), known as Danilevskii's method[3],[4], where the condition $\varphi_t(x) \mid \varphi_{t-1}(x) \mid \cdots \mid \varphi_1(x)$ is not required. Hence, the result given by Danilevskii's method is not unique and $\varphi_1(x)$ is not necessarily identical to the minimal polynomial of A.

The normal form in the strict sense of Theorem 3 is deduced by the constructive proof of the "cyclic decomposition theorem of a finite-dimensional vector space"[5],[11],[13]. The computational complexity is analyzed as $O(n^3 \log n)$[1], where $O(n^3)$ corresponds to basic matrix operations such as multiplication, and $O(\log n)$ is interpreted as the number of companion blocks t in the normal form of Eq.(2).

However, instead of the "constructive proof" above, this paper adopts the algorithm expressed by the composition of elementary transformations[14], which is the conversion by the authors of 13) themselves. To our knowledge, there is nothing in the English literature that gives such an algorithm explicitly, even though the formulation in Iri and Kan[13] is essentially identical to that in Hoffman and Kunze[11].

According to the formulation in Kan and Iri[14], we eliminate the elements step by step analogously to the Gaussian elimination method, and finally obtain the normal form F as well as a transformation matrix S. The number of loop counts is not stable, but depends on the arrangement of the elements in the original matrix A. However, this algorithm, whose complexity is also considered to be $O(n^3 \log n)$, is certainly deterministic. In order to compute a similar transformation $A \mapsto S^{-1}AS$, we apply the following elementary transformations successively:

Definition 4 (Elementary Transformations). The following three operations are called the elementary transformations.

$op1(k,l)$**:** Exchange the k-th row and the ℓ-th row of A, and then exchange the k-th column and the ℓ-th column of A.

$op2(k,c)$**:** Multiply the k-th row of A by c^{-1}, and then multiply the k-th column of A by c.

$op3(k,l,c)$**:** Add the ℓ-th row multiplied by c to the k-th row of A, and then subtract the k-th column multiplied by c from the ℓ-th column of A. ∎

In these transformations, the operations on rows correspond to the transformation matrix S^{-1}, and the operations on columns correspond to S. When we apply suitable transformations successively $\cdots S_3^{-1}\left(S_2^{-1}\left(S_1^{-1}AS_1\right)S_2\right)S_3\cdots$, we finally obtain F, S, S^{-1} in Eq.(2). This procedure is carried out by performing only rational operations in $\mathbf{Q}$, and can be realized by preparing two unit matrices with the same size for A, as follows:

$$\boxed{E \mid A \mid E} \longrightarrow \boxed{S^{-1} \mid F \mid S} \quad (3)$$

Remark 1. In some of the literature, including Kan and Iri[14], a companion matrix is defined as the transposed form of Eq.(1), and the Frobenius normal form as the transposed form of Eq.(2). Both definitions are related by $F^T = S^{-1}A^TS \iff F = S^TAS^{-T}$; hence, they are essentially identical. However, we adopt the form in Definition 1 for the symbolic formulation of eigenvectors. Let $S^{-1}AS = C$ in Eq.(1), and let λ be a root of the associated polynomial $f(x)$ of C. If we put

$$\boldsymbol{u} := \left[1, \lambda, \lambda^2, \ldots, \lambda^{n-1}\right]^T,$$

then we have $C\boldsymbol{u} = \lambda\boldsymbol{u}$, $A(S\boldsymbol{u}) = \lambda(S\boldsymbol{u})$. These relations give the symbolic expression of an eigenvector of C and A with the eigenvalue λ. (Hence, we do not compute S^{-1} in the actual implementation.) ∎

Remark 2. If all the elements of A are integers, then its Frobenius normal form F consists only of integers. On the other hand, we can take the transformation matrices S and S^{-1} so that only one of them is integral, even though A is rational. There exists no known algorithm that yields both S and S^{-1} as integer matrices. ∎

3. Modular Algorithm for Integer Matrices

In this section, we summarize the modular algorithm for integer matrices given in our previous paper[19]. We assume that the straightforward program based on procedure (3) over $\mathbf{Q}$ is already implemented. We propose an algorithm that consists of the following two steps:

(i) compute the normal form F s.t. $AS = SF$, using modular arithmetic, and

(ii) compute a transformation matrix S with simpler integer elements.

3.1 Chinese Remainder Theorem

The elementary transformation $op2(k,c)$ includes division by c, but this step is performed in $\mathbf{Z}_p$ as the multiplication by $s \equiv c^{-1} \pmod{p}$, where $cs + pt = 1$ from the Euclidean algorithm. Therefore, every rational operation in the elementary transformations in Definition 4 can be executed modulo a prime p, and the same procedure as over $\mathbf{Q}$ yields the Frobenius normal form over $\mathbf{Z}_p$ such that $A_pS_p = S_pF_p$.

In order to recover F over $\mathbf{Z}$ from several images $F_{p_1}, F_{p_2}, \ldots$, we apply the Chinese remainder theorem (CRT). Since the elements of S in procedure (3) generally produce much longer integers than those of F, recovery of S by the CRT is very inefficient. Hence, we abandon this procedure and do not reserve each S_{p_i}.

Let us denote by $F^{(k)}$ the normal form $F \pmod{p_1 \cdots p_k}$. Applying the Newtonian solution to the CRT (Theorem 9 in the appendix), we compute $F^{(1)} \to F^{(2)} \to \cdots$ for the moduli

$p_1, p_1p_2, \ldots$ successively.

In the lifting procedure, when we obtain $F^{(k-1)} = F^{(k)}$ for (mod $p_1 \cdots p_{k-1}$) and (mod $p_1 \cdots p_{k-1}p_k$), we check the condition $\varphi(A) = 0$ over $\mathbf{Z}$, where $\varphi(x)$ is the minimal polynomial of $F^{(k)}$. Since the minimal polynomial of A is unique, if we have the above condition, $F^{(k)}$ coincides with the normal form over $\mathbf{Z}$.

We can also estimate the bound for the elements of F and compute F_p's using a sufficient number of primes to recover the integer[8]. However, we selected the above incremental formula so that the minimum necessary number of F_p's would be computed in the framework of sequential computation. We should also note that it is difficult to extend the bound to the case of rational matrices.

3.2 Detecting and Avoiding Unlucky Primes

In this section, we discuss the criterion for detecting unlucky primes, arranging various definitions used in several previous studies. First, we introduce the following broad definition:

Definition 5 (Unlucky Prime). Let $AS = SF$ be the normal form and a transformation matrix computed over $\mathbf{Z}$, and $A_pS_p = S_pF_p$ be those computed over $\mathbf{Z}_p$. If we have $F \not\equiv F_p \pmod p$ or $S \not\equiv S_p \pmod p$, then we call the prime number p "unlucky." ▮

The following lemma shows that unlucky primes (in the sense of Definition 5) are detected by recording and comparing the history of pivoting in the elimination process.

Lemma 6 (Howell[12]). The pivoting pattern for a lucky prime p is identical to that over $\mathbf{Z}$. ▮

If an unlucky prime p divides the pivot element at some step and the pivot is reduced to 0, we have to apply $op1(k, \ell)$ to choose a nonzero pivot, but such an exchange of rows and columns does not occur in the computation over $\mathbf{Z}$. Hence, we discard the result over such modulus p, and apply the CRT to p_i's that hold the same pivoting pattern as that over $\mathbf{Z}$. Since the number of prime factors of the pivot element for each step is finite, only a finite number of unlucky primes exist for a certain matrix.

In contrast, other papers[6),8)] that also propose a modular algorithm using the CRT adopt the following narrow definition:

Definition 7 (Strictly Unlucky Prime). Let A, F, A_p, and F_p be the same as in Definition 5. If we have $F \not\equiv F_p \pmod p$, then we call the prime number p "strictly unlucky." ▮

In this definition, we do not care about transformation matrices S and S_p. It is obvious that $F \not\equiv F_p \pmod p \Rightarrow S \not\equiv S_p \pmod p$; however, if we have $F \equiv F_p \pmod p$ and $S \not\equiv S_p \pmod p$, then we do not regard p as strictly unlucky. In order to detect such strictly unlucky primes, instead of the pivoting pattern, the block structure and their sizes $(d_1, \ldots, d_t)$ are noted in the normal form $F = C_1 \oplus \cdots \oplus C_t$.

Lemma 8 (Giesbrecht[6),8)]). If and only if we have lexicographically

$$(d_1, \ldots, d_t) \succ (d_1^{(p)}, \ldots, d_{t'}^{(p)}),$$

then p is strictly unlucky. ▮

According to this lemma, we can apply the CRT using every F_p that has the same block structure as F, even if $S \not\equiv S_p \pmod p$. The existence of such p's is discussed later with reference to an example.

Nevertheless, we implemented the history of pivoting on the basis of the criterion in Lemma 6, because we also implemented the recovery of S by means of the CRT, using S_p's for the comparison of algorithms[19]. If we recover only F by means of the CRT, then we can adopt the criterion of Lemma 8, but the program has not yet been optimized.

The difficulty in detecting unlucky primes lies in the fact that neither the correct pivoting pattern over $\mathbf{Z}$ nor the block structure in F is known beforehand. In our implementation, we presume the correct pattern by a majority decision, comparing the results for first three primes p_1, p_2, and p_3. As explained in Moritsugu[19], this presumption is correct with very high probability if we use prime numbers of appropriate magnitude; for example, word size. Cases where the presumption is incorrect will rarely occur unless we purposely use very small primes; for example, those with values less than 500. Therefore, the detection of unlucky primes is practically resolved.

3.3 Construction of a Transformation Matrix

Applying similar transformations (3) successively, we can compute a transformation matrix S together with the normal form F, but the result $S = S_1S_2 \cdots$ has very long integers as its elements. If we also recover S by means of the CRT, we have to use a larger number of moduli p_i's than for the recovery of F. Experimental results[19] have shown that this approach is fairly inefficient.

However, since transformation matrices S are not unique, we have only to obtain one regular matrix S such that $AS = SF$. Hence, we first recover F by means of the CRT, and then we compute S with elements that are as simple possible, so that $AS = SF$ is satisfied. This approach is also proposed in Giesbrecht and Storjohann[8)], using another formulation.

For the sake of simplicity, we first consider the case where F has one companion block. The relation

$$AS = S \begin{bmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ c_0 & c_1 & \cdots & c_{n-1} \end{bmatrix}$$

$$\varphi(x) = x^n - c_{n-1}x^{n-1} - \cdots - c_1 x - c_0$$

is rewritten as follows by putting

$$S = \left[\, \boldsymbol{s}_1 \,\middle|\, \boldsymbol{s}_2 \,\middle|\, \cdots \,\middle|\, \boldsymbol{s}_n \,\right]:$$

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

By eliminating $\boldsymbol{s}_{n-1}, \ldots, \boldsymbol{s}_1$ in turn upward from the lowest equation, we obtain

$$\left(A^n - c_{n-1}A^{n-1} - \cdots - c_1 A - c_0 E\right)\boldsymbol{s}_n = \boldsymbol{0}; \quad \text{i.e.,} \quad \varphi(A)\boldsymbol{s}_n = \boldsymbol{0}. \qquad (4)$$

When the normal form F has more than one block, we have the following conditions for $\boldsymbol{s}_d$, where we let d be the size of the target block:

- For the first companion block, $\varphi(x)$ is the minimal polynomial of A. Then we have $\varphi(A) = O$ according to Cayley-Hamilton's theorem, and $\boldsymbol{s}_d$ can be arbitrary.
- For the second block downward, we need to solve the system of linear equations (4) actually, where $0 \le \mathrm{rank}\varphi(A) < n$. Then we let $\boldsymbol{s}_d$ be one of its solutions.

Other column vectors are computed by means of the recurrence formula

$$\boldsymbol{s}_j = A\boldsymbol{s}_{j+1} - c_j\boldsymbol{s}_d \qquad (j = d-1, \ldots, 1).$$

The complexity of the algorithm shown below is also $O(n^3 \log n)$, because we solve systems of linear equations (4) with size n, whose number is $(\#blocks - 1) \propto \log n$, and conventional methods for a system of linear equations such as Gaussian elimination have the complexity $O(n^3)$.

Algorithm 1 (Construction of a Transformation Matrix).

```
% input:  an integer matrix $A$ and its Frobenius normal form $F = C_1 \oplus C_2 \oplus \cdots \oplus C_t$
%  Let the associated polynomial of each $C_k$
       be $\varphi_k(x) = x^{d_k} - c_{k,d_k-1}x^{d_k-1} - \cdots - c_{k,1}x - c_{k,0}$.
% output:  an integer matrix
       $S = \left[\boldsymbol{s}_{1,1} \middle| \cdots \middle| \boldsymbol{s}_{1,d_1} \middle| \cdots \middle| \boldsymbol{s}_{t,1} \middle| \cdots \middle| \boldsymbol{s}_{t,d_t}\right]$
                                              s.t. $AS = SF$
(Step)  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
                $\boldsymbol{s}_{k,j} = A\boldsymbol{s}_{k,j+1} - c_{k,j}\boldsymbol{s}_{k,d_k}$;  ∎
```

If the initial vector $\boldsymbol{s}_{k,d_k}$ is inappropriately chosen, this algorithm might return a singular matrix S even when the condition $AS = SF$ is satisfied. If we obtain $\mathrm{rank}(S) < n$ by Gaussian elimination, then we try this algorithm again, choosing another initial vector $\boldsymbol{s}_{k,d_k}$. A necessary condition for such appropriate initial vectors is discussed in our previous paper[19)], which guarantees the existence of an appropriate $\boldsymbol{s}_{k,d_k}$. In our actual implementation, we set the vector $\boldsymbol{s}_{k,d_k}$ with randomly chosen simple but generic elements. Hence, Algorithm 1 is probabilistic at this step, but experiments indicate that it is practical enough.

3.4 Modular Algorithm for Integer Matrices

The whole algorithm for integer matrices is given below. For the sake of simplicity, the program will simply stop when 10 unlucky primes are detected, where the first presumption might be incorrect. Otherwise, the CRT process is terminated when $F^{(k-1)} = F^{(k)}$, and we check $\varphi(A) = O$, where $\varphi(x)$ is the minimal polynomial of $F^{(k)}$. Since the minimal polynomial is unique, $F^{(k)}$ is exactly equal to the Frobenius normal form of A when $\varphi(A) = O$.

Algorithm 2 (Modular Algorithm for Integer Matrices).

```
% input:  an $n \times n$ integer matrix $A$ and the
       list of large prime numbers $\{p_1, p_2, \ldots, p_s\}$
%  Let $F^{(k)}$ be the result (mod $p_1 \cdots p_k$) except for unlucky $p_i$. We do not reserve $S_{p_k}$.
% output:  the Frobenius normal form $F$ of
       $A$ and a transformation matrix $S$
                                              s.t. $AS = SF$
(S1)  Compute $F_{p_i}$ s.t. $A_{p_i}S_{p_i} \equiv S_{p_i}F_{p_i}$
                                 (mod $p_i$   $i = 1, 2, 3$);
(S2)  Presume the correct pivoting pattern by
       comparing the above three results;
```

```
%    To simplify the following description, we
     assume a majority decision; for example,
     the patterns for $p_1$ and $p_3$ are identical but
     $p_2$ gives a different pattern. (Refer to the
     discussion in §3.2 for details.)
(S3) Construct $F^{(3)}$ from $F_{p_1}$ and $F_{p_3}$
                                     by the CRT;
(S4) $k := 3$;  $count := 1$;
(S5) Do until $(F^{(k-1)} = F^{(k)})$
        $k := k + 1$;
        Compute $F_{p_k}$ s.t. $A_{p_k} S_{p_k} \equiv S_{p_k} F_{p_k}$
                                     (mod $p_k$);
        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 the CRT;
(S6) If $\varphi(A) \neq O$ then goto (S5);
                    % $\varphi(x)$ = min.pol. of $F^{(k)}$
(S7) Construct $S$ from $A, F^{(k)}$
                    by Algorithm 1 ( over $\mathbf{Z}$ );
(S8) While rank$(S) < n$ ( over $\mathbf{Z}$ ) do
                    reconstruct $S$ by Algorithm 1;
(S9) Return $\langle F^{(k)}, S\rangle$;                ▮
```

Figure 1 shows the result for a 10×10 matrix A with random integer elements $-10 \leq a_{ij} \leq 10$. We applied the list of prime numbers $\{13, 17, 19, 23, \ldots\}$ to test the program. In the first presumption process, $p = 13$ and $p = 19$ were supposed to be lucky primes, where no pivoting occurred in the elimination in both cases. Since $p = 17$ yielded another pivoting pattern, it was eliminated. Subsequently, since the exchange of rows and columns was executed for $p = 23$ and $p = 37$, we discarded these two primes and computed the CRT using nine other primes. We should note that the block structure was also identical:

$$d_1^{(p)} = 10 \qquad \text{for} \quad p = 17, 23, 37.$$

Hence those primes are "unlucky" in the sense of Definition 5, but not "strictly unlucky" in the sense of Definition 7, and they can be included in the CRT process for the recovery of F. In contrast, when we recovered S by means of the CRT for comparison, we needed 30 primes, excluding $p = 17, 23, 37$.

A transformation matrix S was constructed by Algorithm 1, and we obtained a regular S at the first attempt. The regularity was confirmed using modular arithmetic for efficiency, because it is sufficient to check that rank$(S) = n \pmod{p}$.

Remark 3. Let us compute the bound for F using Lemma 2.1 in 8):

$$\gamma = 2^n e^{n/2} \|A\|^n n^{n/2}, \quad \text{where } \|A\| = \max_{ij} |A_{ij}|. \tag{5}$$

Substituting $n = 10$ and $\|A\| = 10$, we obtain $\gamma \simeq 1.5 \times 10^{20}$, while the final modulus $\prod p_i \leq 10^{14}$ was used in the actual computation of the CRT. ▮

4. Algorithms for Rational Matrices

Two ways can be considered for treating rational matrices, as shown in the following subsections.

4.1 An Algorithm Executeed over the Integers

We let $\tilde{A} = kA \quad (a_{ij} \in \mathbf{Q}, \quad k, \ \tilde{a}_{ij} \in \mathbf{Z})$, where A is a rational matrix and k is the common denominator of a_{ij}'s. We can perform the computation over $\mathbf{Z}$ as far as possible by applying Algorithm 2 to $\tilde{A}$. The relation between the Frobenius normal forms of A and $\tilde{A}$ is illustrated below for the case of two blocks.

First, we compute $\tilde{F}$ and $\tilde{S}$ over $\mathbf{Z}$ by means of Algorithm 2:

$$\tilde{S}^{-1}\tilde{A}\tilde{S} = \left[\begin{array}{cccc|cc} & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ p & q & r & s & & \\ \hline & & & & & 1 \\ & & & & x & y \end{array}\right] \quad (p, q, r, s;\ x, y \ \in \ \mathbf{Z})$$
$$=: \tilde{F}. \tag{6}$$

Then, letting

$$V := \left[\begin{array}{cccc|cc} 1 & & & & & \\ & k & & & & \\ & & k^2 & & & \\ & & & k^3 & & \\ \hline & & & & 1 & \\ & & & & & k \end{array}\right], \tag{7}$$

we apply a similar transformation to

$$\tilde{F}/k = \tilde{S}^{-1}\left(\tilde{A}/k\right) S :$$

$$V^{-1}\left(\tilde{F}/k\right) V = \left[\begin{array}{cccc|cc} & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ p/k^4 & q/k^3 & r/k^2 & s/k & & \\ \hline & & & & & 1 \\ & & & & x/k^2 & y/k \end{array}\right]$$

```
matpprint A;

   [   8   -7   9  -2  -10  -10  -10    6  -5  -10  ]
   [   3   -7  -5   1   -6    7   -9    4  -6    9  ]
   [  -8   -5  -2  -7    1    1    5    9  -9   -7  ]
   [  -5    1  -6   7   -9    4   -6    9  -8   -5  ]
   [  -2   -7   1   1    4   -6    9   -1  -7    4  ]
   [  -3    1  -3  -3    1    2    2   -3  -8   -8  ]
   [  -1    4  -3   3    0   -4   -2    6  -1   -3  ]
   [  -3    0   2  -1    3    9   -7  -10  -7    1  ]
   [  -5  -10  -9   9  -10    1   -8   -2   3    6  ]
   [   7   -7  -7   1    4   -7   -2    9   7    9  ]

frobenius_nf(A,'F,'after);

p = 13  pivot = (nil nil nil nil nil nil nil nil nil)  #block = 1  size = [10]
p = 17  pivot = (nil (2 . 3) nil nil nil nil nil nil nil)  #block = 1  size = [10]
p = 19  pivot = (nil nil nil nil nil nil nil nil nil)  #block = 1  size = [10]

lucky primes = (13 19)
lucky pivots = (nil nil nil nil nil nil nil nil nil)
initial mod = 247

p = 23 is unlucky : pivot = (nil nil nil nil nil (6 . 7) nil nil nil) #block = 1  size = [10]
p = 37 is unlucky : pivot = (nil nil nil nil nil (6 . 7) nil nil nil) #block = 1  size = [10]

#block = 1   size = [10]

regular_p = t   (mod 19)
S is regular   (time for test : 0)   max length = 11 digits (time : 0)
time for construction of S : 0

product of 9 lucky primes (except 2 unlucky primes) : 14 digits

Time for Frobenius : 15
```

Fig. 1 Example of execution of Algorithm 2.

$$=: F. \tag{8}$$

Consequently, we obtain F, which is the Frobenius normal form of $\tilde{A}/k = A$. Since

$$(\tilde{S}V)^{-1}A(\tilde{S}V) = F,$$

a transformation matrix S such that $AS = SF$ is given by $S := \tilde{S}V$. In this formulation, we note that S is integral ($s_{ij} \in \mathbf{Z}$), even though A and F are rational ($a_{ij}, f_{ij} \in \mathbf{Q}$).

Remark 4. Most other papers discuss the Frobenius normal forms only for integer matrices, ignoring rational matrices. It seems that applying the above-mentioned procedure is considered to be sufficient. However, we discuss the direct recovery of rationals in the next subsection, and implement both algorithms to compare their efficiency. ▮

4.2 An Algorithm Using Integer-to-Rational Conversion

We can also recover the Frobenius normal form F (over $\mathbf{Q}$) directly from $F^{(k)}$ (mod $p_1 p_2 \cdots p_k$) by using integer-to-rational conversion (Algorithm 6 in the appendix). When we first convert A (over $\mathbf{Q}$) into $A_p \equiv A$ (mod p), we discard the unlucky prime p that divides the denominator of an element a_{ij} in A. Combining the modular arithmetic, CRT, and Algorithm 6, we compute the Frobenius normal form F (over $\mathbf{Q}$):

$$S^{-1}AS = \left[\begin{array}{cccc|cc} & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ p' & q' & r' & s' & & \\ \hline & & & & & 1 \\ & & & & x' & y' \end{array}\right] \quad (p', q', r', s';\ x', y' \in \mathbf{Q})$$

$$=: F. \tag{9}$$

We note that a transformation matrix S is unknown at this point. If we apply Algorithm 1 to this A and F, we obtain S such that $s_{ij} \in \mathbf{Q}$. However, we consider executing Algorithm 1 over the integers for efficiency in our implementation, because integer computation is much faster than rational computation in our environment (§5.1).

Therefore, we convert F into $\tilde{F}$, which is the normal form of the integer matrix $\tilde{A} = kA$, and is identical to (6). Using the same matrix V as (7), we apply the inverse of the similar transformation (8) to kF:

$$
\begin{aligned}
&V\,(kF)\,V^{-1}\\
&= \left[\begin{array}{cccc|cc}
 & 1 & & & & \\
 & & 1 & & & \\
 & & & 1 & & \\
k^4p' & k^3q' & k^2r' & ks' & & \\
\hline
 & & & & & 1\\
 & & & & k^2x' & ky'
\end{array}\right]\\
&=: \tilde{F}. \qquad (10)
\end{aligned}
$$

Thus we obtain the normal form over **Z** for $\tilde{A}$. Applying Algorithm 1 to $\tilde{A}$ and $\tilde{F}$, we construct $\tilde{S}$ such that $\tilde{A}\tilde{S} = \tilde{S}\tilde{F}$. Finally, we obtain the transformation matrix S by letting $S := \tilde{S}V$ as in the previous section. We should note again that this result S is also integral.

4.3 Practical Algorithms

We implemented the following three algorithms and experimented to compare their efficiency. For all three algorithms, the input is an $n \times n$ matrix A (over **Q**), and the output is the Frobenius normal form F (over **Q**) and a transformation matrix S (over **Q** or **Z**).

In the first algorithm, given by Kan and Iri[14], rational arithmetic is used straightforwardly.

Algorithm 3 (Rat).

(Step) According to procedure (3), compute F and S ($AS = SF$) directly over **Q**. ∎

In the second algorithm, shown in §4.1, the computation is performed over the integers as far as possible.

Algorithm 4 (Zmod).

(Z1) Extract the common denominator: $\tilde{A} := kA$.

(Z2) Compute $\tilde{F}$ (over **Z**) by using modular arithmetic and the CRT.

(Z3) Construct $\tilde{S}$ (over **Z**) from $\tilde{A}$ and $\tilde{F}$ by Algorithm 1.

(Z4) Convert $\tilde{F}$ into F (over **Q**) by Eq.(8), and let $S := \tilde{S}V$ (over **Z**).
% Steps (Z2) and (Z3) correspond to Algorithm 2. ∎

In the third algorithm, discussed in §4.2, which we propose as the most practical one, integer-to-rational conversion is used to recover rationals from modular numbers.

Algorithm 5 (Qmod).

(Q1) Compute F (over **Q**) by using modular arithmetic, the CRT, and integer-to-rational conversion.

(Q2) Convert F into $\tilde{F}$ (over **Z**) by Eq.(10), which is the normal form of $\tilde{A} := kA$.

(Q3) Construct $\tilde{S}$ (over **Z**) from $\tilde{A}$ and $\tilde{F}$ by Algorithm 1.

(Q4) Let $S := \tilde{S}V$ (over **Z**).
% This result S is essentially identical to S in the Zmod algorithm. ∎

5. Experimental Results

5.1 Environment for Implementation

We have been developing the program code for modular computation of Frobenius normal forms in the following environment:

- Computer algebra system:
  Reduce 3.7[10] (Windows version) + RLISP '88[17]
  - Computation of polynomials and rational numbers:
    algebraic mode of Reduce
  - Computation over **Z** and $\mathbf{Z}_p$:
    RLISP (symbolic mode of Reduce)
- CPU: Pentium 4 (2 GHz)
  Main memory: 1.5 GB
  (limited to 1.0 GB for the Reduce system)
- List of prime numbers less than 2^{31}:
  $\{p_1, p_2, \ldots, p_{1000}\} = \{2147483647,\ 2147483629,\ \ldots,\ 2147462143\}$
- Generation of test matrices:
  (i) Prepare a characteristic polynomial with random rational coefficients whose sizes are normalized.
  (ii) Construct the Frobenius normal form associated with the characteristic polynomial.
  (iii) Randomize the normal form by a similar transformation.
  (Maximal size of elements in the result: about 200 digits/200 digits for $n = 100$)
- Verification and comparison:
  Maple 7[2] (Windows version)

5.2 Experimental Results

We prepared two groups of test matrices and applied the three algorithms shown in §4.3. First, we note that no unlucky prime was detected in any of these examples.

Table 1 shows the result for nonderogatory matrices, whose Frobenius normal forms consist of one companion block. The number of primes used as moduli is given in the $\#p_i$ column. We set two types of matrices in each size and divided the results into two subgroups:

Upper: Non-diagonalizable (i.e., those whose minimal polynomials are not square-free),

Lower: Diagonalizable (i.e., those whose minimal polynomials are square-free).

The difference between these properties sig-

Table 1 CPU time (sec) for rational matrices I.

	Rat	Zmod			Qmod			Maple	
n	T_R	$\#p_i$	T_Z	T_Z/T_R	$\#p_i$	T_Q	T_Q/T_R	T_M	T_Q/T_M
30	19.20	171	13.34	0.695	13	2.48	0.129	108.36	0.023
35	37.20	234	29.53	0.794	15	5.36	0.144	238.91	0.022
40	66.45	285	54.08	0.814	16	9.56	0.144	481.57	0.020
45	110.61	376	106.97	0.967	18	18.33	0.166	871.97	0.021
50	168.42	465	189.12	1.123	20	31.88	0.189	1427.22	0.022
100	3845.44	>1000	(failure)		39	1170.11	0.304	51088.66	0.023
30	23.22	162	12.50	0.538	12	2.28	0.098	135.86	0.017
35	55.34	216	26.83	0.485	14	4.83	0.087	371.56	0.013
40	152.30	320	63.08	0.414	18	11.09	0.073	1152.73	0.010
45	330.14	417	123.64	0.375	20	21.22	0.064	2635.40	0.008
50	696.86	530	228.25	0.328	23	37.70	0.054	5897.08	0.006
100	42331.16	>1000	(failure)		34	1067.83	0.025	(not tried)	

cf. max length of s_{ij} (digits) for $n = 50$

	Rat	Zmod	Qmod
Upper	2113 / 2102	4318	4318
Lower	5023 / 5024	4899	4899

Table 2 CPU time (sec) for rational matrices II.

	Rat	Zmod			Qmod			Maple	
n (Structure)	T_R	$\#p_i$	T_Z	T_Z/T_R	$\#p_i$	T_Q	T_Q/T_R	T_M	T_Q/T_M
30 (15+15)	18.23	38	4.31	0.237	7	2.34	0.129	146.25	0.016
35 (20+15)	45.67	75	13.28	0.291	9	6.06	0.133	216.72	0.028
40 (25+15)	115.50	117	31.50	0.273	11	13.38	0.116	632.09	0.021
45 (30+15)	263.70	164	65.86	0.250	13	28.22	0.107	1579.33	0.018
50 (20+15+15)	128.24	75	38.92	0.304	9	19.47	0.152	866.04	0.022
55 (20+20+15)	351.95	75	63.78	0.181	9	36.27	0.103	2633.78	0.014
60 (25+20+15)	522.86	117	128.56	0.246	11	69.61	0.133	4350.91	0.016
65 (25+25+15)	1386.89	117	191.43	0.138	11	114.97	0.083	13083.23	0.009
70 (30+25+15)	1573.34	164	336.09	0.214	13	198.11	0.126	14743.82	0.013
70 (25+20+15+10)	511.84	117	292.49	0.571	11	145.27	0.284	6839.38	0.021
80 (30+25+15+10)	2635.15	164	650.16	0.247	13	375.71	0.143	34695.14	0.011
90 (30+25+20+15)	4168.56	164	900.19	0.216	13	519.23	0.125	46700.11	0.011
100 (35+30+20+15)	3640.00	210	1974.35	0.542	14	1163.09	0.320	47808.79	0.024

cf. max length of s_{ij} (digits) for $n = 100$

Rat	Zmod	Qmod
4104 / 4096	2056	2052

nificantly affected the computation time when the Rat algorithm was used. The cause of this lies mainly in the length of s_{ij} output; that is, the upper subgroup yielded much shorter s_{ij}'s than the lower subgroup. Therefore, the Rat algorithm is affected not only by the size n of matrices, but also to their properties.

In contrast, the Zmod and Qmod algorithms showed a common tendency for both the upper and lower subgroups, and seem to be entirely dominated by the matrix size. Moreover, the Qmod algorithm is always faster than Zmod, because Qmod needs much fewer primes than Zmod. Consequently, the Zmod algorithm is not useful, particularly for the upper subgroup, whereas the Qmod algorithm is superior to the Rat and Zmod algorithms for both subgroups.

Table 2 shows the result for derogatory matrices, whose Frobenius normal forms consist of more than one companion block. On the whole, the Zmod algorithm is not so inefficient as in Table 1, whereas the efficiency of the Qmod algorithm is confirmed. The values of output S produced by Zmod and Qmod are essentially identical, although they vary slightly in each computation because the initial vectors $\boldsymbol{s}_{k,d_k}$ in Algorithm 1 are randomly chosen.

All the test matrices in Table 2 are diagonalizable; that is, their minimal polynomials are square-free. We also experimented with using non-diagonalizable matrices, but the difference was not so clear as in Table 1.

To confirm the results and compare the computation time, we used the "frobenius" function in Maple 7. The results for the timing data showed a similar tendency to those of the Rat algorithm, and seem to be influenced by the length of elements in a transformation matrix P, which corresponds to S^{-T} in our formulation.

6. Concluding Remarks

In this paper, we implemented modular algorithms for the Frobenius normal forms of integer and rational matrices, and confirmed their efficiency by experiments. Modular methods for integer matrices have been proposed by several authors, but it seems that no implementation including rational matrices has yet been reported elsewhere. Finally, we itemize the results of our studies.

(1) We implemented the criterion of the history of pivoting for detecting unlucky primes, and confirmed that it works correctly. The criterion of the block structure in the normal form F is also discussed. If a transformation matrix S is not recovered by the CRT, the detection of "strictly unlucky primes" is sufficient for the recovery of F.

(2) The number of primes used as moduli is significantly reduced by the Newtonian solution to the CRT in the form of Eq.(12), in comparison with the Lagrangian solution using a theoretical bound given by Eq.(5). Moreover, since it seems difficult to extend the theoretical bound to the rational case, we consider our approach using the incremental formula to be more practical than using theoretical bounds.

(3) A transformation matrix S should be constructed after the normal form F is computed in order to avoid swelling of its elements. This procedure is probablistic, but in practical terms it returns a regular matrix almost without fail.

(4) Among commercial computer algebra systems, only Maple provides a function for Frobenius normal forms. The details of its algorithm are not open, but it tends to be affected by swelling of the elements in the transformation matrix computed simultaneously. This Maple package has been converted into the Reduce library[27], but it seems to be working less efficiently in Reduce than in Maple.

(5) For rational matrices, the Qmod algorithm should be used rather than the Zmod algorithm. When the number of primes p_i needed for each algorithm is compared, it is obvious that Qmod is superior to Zmod for matrices with longer elements. Hence, algorithms applicable only to integer matrices, which have been studied by other authors, are not sufficient for rational matrices. Our success in developing the Qmod algorithm, which is directly targeted at rational matrices, shows the benefits of our study.

(6) Divide-and-conquer algorithms for integer-to-rational conversion[28] are now being studied. Since it is not easy to implement such fast algorithms in our environment, we implemented a classical algorithm by Wang[31],[32]. Although the step of integer-to-rational conversion is not necessarily dominant in the present framework of computation, implementation of such divide-and-conquer algorithms should also be considered in the future development of practical computer algebra systems.

References

1) Augot, D. and Camion, P.: On the Computation of Minimal Polynomials, Cyclic Vectors, and Frobenius Forms, *Linear Algebra and Its Applications*, Vol.260, pp.61–94 (1997).

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.*, Vol.2, No.1, pp.169–172 (1937). (In Russian)

4) Faddeev, D. and Faddeeva, V.: *Computational Methods of Linear Algebra*, W.H. Freeman, San Francisco (1963).

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

6) Giesbrecht, M.: Fast Algorithms for Rational Forms of Integer Matrices, *ISSAC 94*, pp.305–311, ACM (1994).

7) Giesbrecht, M.: Nearly Optimal Algorithms for Canonical Matrix Forms, *SIAM J. Comput.*, Vol.24, No.5, pp.948–969 (1995).

8) Giesbrecht, M. and Storjohann, A.: Computing Rational Forms of Integer Matrices, *J. Symbolic Computation*, Vol.34, No.3, pp.157–172 (2002).

9) Gil, I.: Computation of the Jordan Canonical Form of a Square Matrix (Using the Axiom Programming Language), *ISSAC 92*, pp.138–145, ACM (1992).

10) Hearn, A.C.: *Reduce User's Manual (Ver.3.7)*, RAND Corp., Santa Monica (1999).

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

12) Howell, J.A.: An Algorithm for the Exact Reduction of a Matrix to Frobenius Form Using Modular Arithmetic, I & II, *Math. Comp.*, Vol.27, No.124, pp.887–920 (1973).

13) Iri, M. and Kan, T.: *Linear Algebra: Matrix and Its Normal Forms*, Kyoiku-Shuppan, Tokyo (1977). (In Japanese)

14) Kan, T. and Iri, M.: *Jordan Normal Forms*, Univ. of Tokyo Press, Tokyo (1982). (In Japanese)

15) Lauer, M.: Computing by Homomorphic Images, *Computer Algebra : Symbolic and Algebraic Computation*, Buchberger, B., et al. (eds.), pp.139–168, 2nd edition, Springer-Verlag, Wien (1983).

16) Lüneburg, H.: *On the Rational Normal Form of Endomorphisms: A Primer to Constructive Algebra*, Wissenschaftsverlag, Mannheim (1987).

17) Marti, J.: *RLISP '88: An Evolutionary Approach to Program Design and Reuse*, World Scientific, Singapore (1993).

18) Mathieu, M.-H. and Ford, D.: On p-adic Computation of the Rational Form of a Matrix, *J. Symbolic Computation*, Vol.10, No.5, pp.453–464 (1990).

19) Moritsugu, S.: Modular Computation of Frobenius Normal Forms of Integer Matrices, *J. Japan Soc. Symbolic and Algebraic Computation*, Vol.9, No.4, pp.52–67 (2003). (In Japanese)

20) Moritsugu, S. and Kuriyama, K.: Exact Computation of Jordan Normal Forms of Matrices by Computer Algebra, *Trans. Japan Soc. Indust. Appl. Math.*, Vol.2, No.1, pp.91–103 (1992). (In Japanese)

21) 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).

22) Moritsugu, S. and Kuriyama, K.: On Multiple Zeros of Systems of Algebraic Equations, *ISSAC 99*, pp.23–30, ACM (1999).

23) Moritsugu, S. and Kuriyama, K.: A Linear Algebra Method for Solving Systems of Algebraic Equations, *J. Japan Soc. Symbolic and Algebraic Computation*, Vol.7, No.4, pp.2–22 (2000).

24) Moritsugu, S. and Kuriyama, K.: Symbolic Computation of Eigenvalues, Eigenvectors and Generalized Eigenvectors of Matrices by Computer Algebra, *Trans. Japan Soc. Indust. Appl. Math.*, Vol.11, No.2, pp.103–120 (2001). (In Japanese)

25) Mukhopadhyay, A.: Exact Computation of the Characteristic Polynomial of an Integer Matrix, *AAECC 3*, Lect. Notes in Comp. Sci., Vol.229, pp.316–324 (1985).

26) Ozello, P.: *Calcul Exact des Formes de Jordan et de Frobenius d'une Matrice*, PhD Thesis, Université Scientifique Technologique et Médicale de Grenoble (1987).

27) Rebbeck, M.: A Linear Algebra Package for REDUCE, Technical Report, ZIB, Berlin (1994).

28) Sasaki, T., Takahashi, Y. and Sugimoto, T.: On Integer-to-Rational Conversion Algorithm for Long Integers, *Papers Res. Inst. Math. Sci., Kyoto Univ.*, Vol.1295, pp.80–86 (2002). (In Japanese)

29) Storjohann, A.: An $O(n^3)$ Algorithm for the Frobenius Normal Form, *ISSAC 98*, pp.101–104, ACM (1998).

30) Takeshima, T. and Yokoyama, K.: A Solution of Systems of Algebraic Equations: Application of Eigenvectors of Linear Maps on Residue Class Rings, *Comm. Symb. Algeb. Manip.*, Vol.6, No.4, pp.27–36 (1990). (In Japanese)

31) Wang, P.S.: A p-adic Algorithm for Univariate Partial Fractions, *SYMSAC 81*, pp.212–217, ACM (1981).

32) Wang, P.S., Guy, M.J.T. and Davenport, J.H.: P-adic Reconstruction of Rational Numbers, *ACM SIGSAM Bull.*, Vol.16, No.2, pp.2–3 (1982).

Appendix

A.1 Solutions to the CRT

Theorem 9 (CRT: the Case of Two Moduli). When m_1 and m_2 are relatively prime integers, the solution of the following system of modular equations

$$\begin{cases} x \equiv a_1 & \pmod{m_1} \\ x \equiv a_2 & \pmod{m_2} \end{cases}$$

is given by computing

$$x \equiv (a_2 - a_1) m_1 s + a_1 \pmod{m_1 m_2}, \qquad (11)$$

where $m_1 s + m_2 t = 1$; that is, $s \equiv {m_1}^{-1} \pmod{m_2}$. ∎

The number of moduli required to recover the Frobenius normal form F over $\mathbf{Q}$ is not known in advance. For integer matrices, several theoretical bounds are known, but they are often overestimated. Hence, it is recommended that the Newtonian solution to the general CRT be adopted rather than the Lagrangian solution[15]. Therefore, using prime numbers $p_1,\ p_2,\ p_3, \ldots$, in order to lift the modulus to $p_1 p_2,\ p_1 p_2 p_3, \ldots$, we repeatedly apply Theorem 9 to the following system:

$$\begin{cases} x \equiv a_{k-1} & \pmod{p_1 \cdots p_{k-1}} \\ x \equiv a_k & \pmod{p_k} \end{cases} \qquad (k = 2, 3, \ldots), \quad (12)$$

which is the Newtonian solution to the CRT with more than three moduli.

A.2 Algorithm for Integer-to-Rational Conversion

The following algorithm is well known as an extension of the classical Euclidean algorithm:

Algorithm 6 (Integer-to-Rational Conversion[31),32)]).

```
% input:  an integer $c$,  the modulus $m$
    $(0 < c < m)$,   $\tilde{m} := \sqrt{m/2}$
% output:  $a/b \equiv c \pmod{m}$
                  $-\tilde{m} < a < \tilde{m}$,   $0 < b < \tilde{m}$
(R1)  $u := (1, 0, m)$;   $v := (0, 1, c)$;
(R2)  while $v_3 \geq \tilde{m}$ do
        { $q := u_3/v_3$;   $r := u - qv$;
          $u := v$;         $v := r$        };
(R3)  if abs($v_2$) $\geq \tilde{m}$ then ERROR;
(R4)  $a :=$ sign($v_2$)$v_3$;   $b :=$ abs($v_2$);
(R5)  return $((a, b))$;                       ∎
```

This algorithm is straightforwardly used in our implementation.

(Received August 5, 2003)
(Accepted March 5, 2004)

Shuichi Moritsugu received the B.S., M.S. and Ph.D. degrees in information science from the University of Tokyo in 1984, 1986 and 1989 respectively. From 1989 to 1994, he was a research associate and then an associate professor at University of Library and Information Science, which was unified with University of Tsukuba in 2002. His research interests include algorithms and applications of computer algebra. He is a member of IPSJ, JSIAM, ACM and JSSAC (Japan Society for Symbolic and Algebraic Computation).