SIAM J. COMPUT.
Vol. 8, No. 4, November 1979

0097-5397/79/0804-0002 $01.00/0

POLYNOMIAL ALGORITHMS FOR COMPUTING THE SMITH AND HERMITE NORMAL FORMS OF AN INTEGER MATRIX*

RAVINDRAN KANNAN† AND ACHIM BACHEM‡

Abstract. Recently, Frumkin [9] pointed out that none of the well-known algorithms that transform an integer matrix into Smith [16] or Hermite [12] normal form is known to be polynomially bounded in its running time. In fact, Blankinship [3] noticed—as an empirical fact—that intermediate numbers may become quite large during standard calculations of these canonical forms. Here we present new algorithms in which both the number of algebraic operations and the number of (binary) digits of all intermediate numbers are bounded by polynomials in the length of the input data (assumed to be encoded in binary). These algorithms also find the multiplier-matrices K, U' and K' such that AK and $U'AK'$ are the Hermite and Smith normal forms of the given matrix A. This provides the first proof that multipliers with small enough entries exist.

Key words. Smith normal form, Hermite normal form, polynomial algorithm, Greatest Common Divisor, matrix-triangulation, matrix diagonalization, integer matrices, computational complexity

1. Introduction. Every nonsingular integer matrix can be transformed into a lower triangular integer matrix using elementary column operations. This was shown by Hermite ([12], Theorem 1 below). Smith ([16], Theorem 3 below) proved that any integer matrix can be diagonalized using elementary row and column operations. The Smith and Hermite normal forms play an important role in the study of rational matrices (calculating their characteristic equations), polynomial matrices (determining the latent roots), algebraic group theory (Newman [15]), system theory (Heymann and Thorpe [13]) and integer programming (Garfinkel and Nemhauser [10]).

Algorithms that compute Smith and Hermite normal forms of an integer matrix are given (among others) by Barnette and Pace [1], Bodewig [5], Bradley [7], Frumkin [9] and Hu [14]. The methods of Hu, Bodewig and Bradley are based on the explicit calculation of the greatest common divisor (GCD) and a set of multipliers whereas other algorithms ([1]) perform GCD calculations implicitly.

As Frumkin [9] pointed out, none of these algorithms is known to be polynomial. In transforming an integer matrix into Smith or Hermite normal form using known techniques, the number of digits of intermediate numbers does not appear to be bounded by a polynomial in the length of the input data as was pointed out by Blankinship [3], [4] and Frumkin [9].

To alleviate this problem, it has been suggested (Wolsey [17], Gorry, Northup and Shapiro [11], Hu [14], Frumkin [9]) that the Smith normal form of an integer matrix A can be computed modulo d (where d is the determinant of A). However this is not always valid as the following example shows. If we take

$$A = \begin{pmatrix} 5 & 26 \\ 2 & 11 \end{pmatrix}$$

* Received by the editors April 19, 1978 and in revised form September 21, 1978. This research was supported in part by N.S.F. Grant ENG-76-09936 and SFB 21 (DFG), Institut für Operations Research, Universität Bonn, Bonn, West Germany.

† Institute for Operations Research, University of Bonn and School of Operations Research, Cornell University, Ithaca, New York 14850.

‡ Institute for Operations Research, University of Bonn, Nassestrasse 2, D-5300 Bonn 1, West Germany.

then $\det(A) = d = 3$ and

$$\tilde{A} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix} \equiv A \pmod 3.$$

Comparing the GCD of all entries of A (which is 1) with the GCD of all entries of $\tilde{A}$ (which is 2) indicates that A and $\tilde{A}$ have different Smith normal forms.

Here we present polynomial algorithms for computing the Smith and Hermite normal forms. All intermediate numbers produced by these algorithms have at most a polynomial number of digits and the number of algebraic operations (additions and multiplications) performed is also bounded by a polynomial of the length of the input. Moreover the algorithms calculate the left and right multipliers (see description below) and thus prove that their entries are bounded in the number of digits by a polynomial in the length of the input. We must stress however that no exponential lower bounds have been proved on existing algorithms.

2. An algorithm for Hermite normal form. An integer square matrix with a determinant of $+1$ or -1 is called unimodular. Post-(pre-) multiplying an $(m \times n)$ matrix by a $(n \times n)((m \times m))$ unimodular matrix is equivalent to performing a series of column (row) operations consisting of (cf. Newman [15]):

1. adding an integer multiple of one column (row) to another,
2. multiplying a column (row) by -1 and
3. interchanging two columns (rows).

These column (row) operations are termed elementary column (row) operations.

THEOREM 1. (Hermite [12]). *Given a nonsingular $n \times n$ integer matrix A, there exists a $n \times n$ unimodular matrix K such that AK is lower triangular with positive diagonal elements. Further, each off-diagonal element of AK is nonpositive and strictly less in absolute value than the diagonal element in its row. AK is called the Hermite normal form of A (abbreviated HNF).*

We now give an algorithmic procedure for calculating the Hermite normal form AK and the multiplier K. All currently known algorithms build up the Hermite normal form row by row, whereas the new algorithm HNF(n, A) (see description below) successively puts the principal minors of orders $1, \cdots, n$ (the submatrices consisting of the first i rows and i columns $1 \leqq i \leqq n$) into Hermite normal form. So at the ith step we have the following pictures. "0" stands for a zero entry of the matrix and "$*$" for an entry that may not be zero.
e.g.

Bradley's algorithm [7]	HNF(n, A)
$* \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \cdot \cdot \ 0$	$* \ 0 \ 0 \ 0 \ * \ * \ * \ * \cdot \cdot \cdot \ *$
$* \ * \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \cdot \cdot \ 0$	$* \ * \ 0 \ 0 \ * \ * \ * \ * \cdot \cdot \cdot \ *$
$* \ * \ * \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \cdot \cdot \ 0$	$* \ * \ * \ 0 \ * \ * \ * \ * \cdot \cdot \cdot \ *$
$* \ * \ * \ * \ 0 \ 0 \ 0 \ 0 \ 0 \cdot \cdot \ 0$	$i \rightarrow \ * \ * \ * \ * \ * \ * \ * \ * \cdot \cdot \cdot \ *$
$i \rightarrow \ * \ * \ * \ * \ * \ * \ * \cdot \cdot \cdot \cdot \ *$	$* \ * \ * \ * \ * \ * \ * \ * \cdot \cdot \cdot \ *$
$* \ * \ * \ * \ * \ * \ * \cdot \cdot \cdot \cdot \ *$	$* \ * \ * \ * \ * \ * \ * \ * \cdot \cdot \cdot \ *$
$\vdots \qquad\qquad\qquad\qquad\quad \vdots$	$\vdots \qquad\qquad\qquad\qquad\quad \vdots$
$* \ * \ * \ * \ * \ * \ * \cdot \cdot \cdot \cdot \ *$	$* \ * \ * \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \ *$

ALGORITHM HNF(n, A): returns (HNF, K).

1. Initialize the multiplier K:

$$K = I \text{ (the } n \times n \text{ identity matrix).}$$

2. Permute the columns of A so that every principal minor of A is nonsingular; do the corresponding column operations on K:

Use a standard row reduction technique (Edmonds [8]) or see Appendix. If the matrix is found to be singular, the algorithm terminates here.

Note. This step need not be carried out, if suitable modifications are made in the rest of the algorithm. However, in the interest of simplicity, we have inserted this step here.

3. Initialize i which denotes the size of the principal minor that is already in *HNF*:

$$i = 1.$$

4. Put the $(i+1)\times(i+1)$ principal minor into HNF: (For any real matrix R, we denote by R_j the jth column of R and by $R_{i,j}$ (or R_{ij}) the element in the ith row and jth column of R).

If $i = n$ then terminate
else,
for $j = 1$ to i

4.1. *Calculate* $r = \mathrm{GCD}(A_{jj}, A_{j,(i+1)})$ and integers p and q such that $r = pA_{jj} + qA_{j,(i+1)}$ using a modified Euclidean algorithm (see Appendix)

4.2. Perform elementary column operations on A so that $A_{j,i+1}$ becomes zero:

$$D = \begin{pmatrix} p & -A_{j,(i+1)}/r \\ q & A_{j,j}/r \end{pmatrix}$$

Replace column j and $(i+1)$ of A by the two columns of the product

$$(A_j A_{i+1})D$$

Replace column j and $(i+1)$ of K by the two columns of the product

$$(K_j K_{i+1})D$$

4.3. If $j > 1$ then call REDUCE OFF DIAGONAL (j, A);

end

5. Call REDUCE OFF DIAGONAL $(i+1, A)$

6. Set $i = i+1$ and go to 4.

end HNF.

ALGORITHM REDUCE OFF DIAGONAL (k, A). (For any real real number y, $\lfloor y \rfloor$ and $\lceil y \rceil$ denote respectively the floor and ceiling of y.)

1. If $A_{kk} < 0$ set $A_k = -A_k$ and $K_k = -K_k$.

2. For $z = 1$ to $k-1$

$$\text{set } K_z = K_z - \lceil A_{kz}/A_{kk} \rceil K_k$$

$$\text{set } A_z = A_z - \lceil A_{kz}/A_{kk} \rceil A_k$$

end REDUCE OFF DIAGONAL.

We divide the proof that algorithm HNF is polynomial into two parts. First we show that intermediate numbers do not grow "too big". Using this result we prove that the number of algebraic operations (i.e. additions and multiplications) is bounded by a polynomial. The first part of the proof proceeds in 3 stages (Lemma 1–3). The simple fact stated below as a proposition is used many times in the proofs.

PROPOSITION 1. *For any real $n \times n$ matrix R,*

$$|\det R| \leqq (n \cdot \|R\|)^n$$

where $\|R\| =$ the maximum absolute value of any entry of R.

Proof. $\det R$ is the sum of $n!$ terms each of which is at most $\|R\|^n$ in absolute value. Since $n! \leqq n^n$, the proposition follows.

LEMMA 1. *For all $i = 1, \cdots, n$*

$$\|A^i\| \leqq n(n\|A^1\|)^{2n+1}, \tag{1}$$

$$\|K^i\| \leqq n(n\|A^1\|)^{2n} \tag{2}$$

$A^i(K^i)$ $(i = 1, \cdots, n)$ denote the matrix $A(K)$ after the $(i \times i)$ principal minor has been put into HNF *by the algorithm (Note that A^1* contains the original data.)

Proof. Clearly, A^i has been obtained from A^1 by elementary column operations on the first i columns of A alone. Thus if M^i and N^i denote the $(i \times i)$ principal minors of A^i and A^1 respectively, there is a unimodular $(i \times i)$ matrix $\tilde{K}^i$ such that

$$M^i = N^i \tilde{K}^i. \tag{3}$$

N^i is nonsingular, hence $\tilde{K}^i$ is unique and is given by $\tilde{K}^i = (N^i)^{-1} M^i$. Since $\|(N^i)^{-1}\|$ is at most the maximum absolute value of a cofactor of N^i we obtain (using Proposition 1):

$$\|\tilde{K}\| \leqq n\|(N^i)^{-1}\|\|M^i\| \leqq n(\|N^i\|n)^n\|M^i\|.$$

Because M^i is in Hermite normal form we obtain

$$\begin{aligned}\|M^i\| &\leqq |M^i_{11} \cdots M^i_{ii}| \\ &= |\det(M^i)| \\ &= |\det(N^i)| \\ &\leqq (i\|N^i\|)^i \leqq (i\|A^1\|)^i \leqq (n\|A^1\|)^n,\end{aligned}$$

hence

$$\|\tilde{K}^i\| \leqq n(n\|A^1\|)^{2n}.$$

Further, we have

$$A^i = A^1 \begin{bmatrix} \tilde{K}_i & 0 \cdots 0 \\ 0 & 1 \\ \vdots & \ddots \\ \vdots & \ddots \\ 0 & 1 \end{bmatrix}.$$

Thus $\|A^i\| \leqq n\|A^1\|\|\tilde{K}^i\| \leqq n(n\|A^1\|)^{2n+1}$. The proof of Lemma 2 below was inspired by a proof of Edmonds [8].

LEMMA 2. *At any stage of the execution of the algorithm,*

$$\|A\| \leqq 2^{3n} n^{(20n^3)} \|A^1\|^{12n^3}.$$

Proof. Let $A^{i,j}$ denote the matrix A after the "do loop" of Step 4 *has been executed* for the values i and j. First, we prove a bound on the entries in the $(i+1)$ st column of $A^{i,j}$ and using this, prove a bound on the rest of A at every stage. Let $d(i, j, k)$ be the determinant of the $(j+1) \times (j+1)$ submatrix of A^i consisting of columns 1 through j and column $(i+1)$ of A^i and rows 1 through j and row k of A^i. We show that there are integers r_j, $j = 1, \cdots, i$ such that

$$A^{i,j}_{k,i+1} = d(i, j, k)/r_j \qquad \text{for all } k \geqq j+1 \text{ and for all } j \leqq i. \tag{4}$$

For $j=1$ it is clear that (4) holds with $r_1=\mathrm{GCD}\,(A^i_{1,1}, A^i_{1,i+1})$. Suppose the statement is valid for $j=1,2,\cdots,p$. Let k be such that $n\geqq k\geqq p+2$. Denote $\alpha=A^{i,p}_{p+1,p+1}$, $\beta=A^{i,p}_{p+1,i+1}$, $\gamma=A^{i,p}_{k,p+1}$ and $\delta=A^{i,p}_{k,i+1}$ and let α', β', γ' and δ' be the corresponding elements of $A^{i,p+1}$.

A^i	$i+1$	$A^{i,p}$	$i+1$	$A^{i,p+1}$	$i+1$
	↓		↓		↓
	* 0 0 0 0 0 *		* 0 0 0 0 0 0		* 0 0 0 0 0 0
	* * 0 0 0 0 *		* * 0 0 0 0 0		* * 0 0 0 0 0
	* * * 0 0 0 *		* * * 0 0 0 0		* * * 0 0 0 0
$p+1\rightarrow$	* * * α 0 0 *		* * * α 0 0 β		* * * α' 0 0 0
	* * * * * 0 *		* * * * * 0 *		* * * * * 0 *
$k\rightarrow$	* * * γ * * *		* * * γ * * δ		* * * γ' * * δ'
	* * * * * * *		* * * * * * *		* * * * * * *

Then the matrices $\begin{pmatrix}\alpha & \beta\\ \gamma & \delta\end{pmatrix}$ and $\begin{pmatrix}\alpha' & 0\\ \gamma' & \delta'\end{pmatrix}$ have the same determinant and $\alpha'=\mathrm{GCD}(\alpha,\beta)$, hence using the induction hypothesis

$$\delta'=\frac{\alpha'\delta'}{\alpha'}=\frac{\alpha\delta-\beta\gamma}{\mathrm{GCD}(\alpha,\beta)}=\frac{\alpha d(i,p,k)-d(i,p,p+1)\gamma}{r_p\mathrm{GCD}(\alpha,\beta)}.$$

Since the last term above represents the expansion of $d(i,p+1,k)$ by the $(p+1)$th column of A^i we obtain

$$|\delta'|=|A^{i,p+1}_{k,i+1}|=|d(i,p+1,k)/r_{p+1}|,$$

where we set $r_{p+1}=r_p\mathrm{GCD}(\alpha,\beta)$. Thus, (4) holds. Using Lemma 1 we obtain

$$\begin{aligned}A^{i,j}_{k,i+1}&\leqq|d(i,j,k)|\\&\leqq(n\|A^i\|)^n\\&\leqq n^n(n(n\|A^1\|)^{2n+1})^n\\&\leqq n^{2n+2n^2+n}\|A^1\|^{2n^2+n}\\&\leqq n^{(5n^2)}\|A^1\|^{(3n^2)},\end{aligned}$$

which gives a bound on the $(i+1)$th column of $A^{i,j}$. For all other columns $1\leqq j\leqq i$ we conclude

$$|A^{i,j}_{k,j}|=|pA^i_{k,j}+qA^{i,j}_{k,i+1}|\leqq|p|(n^2\|A^1\|)^{2n+1}+|q|(n^{(5n^2)}\|A^1\|^{(3n^2)})$$

(cf. Step 4.2 of the algorithm). Both p and q are bounded by $\max\{|A^{i,j}_{j,j}|, |A^{i,j}_{j,i+1}|\}$ (see Appendix). Thus, $\|A\|\leqq 2n^{(10n^2)}\|A^1\|^{(6n^2)}=f$ (say). This does not still account for Step 4.3. Note that REDUCE OFF DIAGONAL (j, A) increases $\|A_z\|$ (the maximum absolute value of entry of column A_z) by at most a factor of $(1+\|A_j\|)$. Thus $\|A\|\leqq f(1+f)^n\leqq 2^n f^{2n}$. Hence Lemma 2 is proved.

LEMMA 3. *At any stage of the execution of the algorithm,*

$$\|K\|\leqq(2n\|A^1\|)^{0(n^4)}$$

Proof. We have already proved (in Lemma 1) that K^i has small enough entries. Each time the do loop of Step 4 is executed, $(-A_{j,i+1}/r)$ and $(A_{j,j}/r)$ are bounded by $2n^{(10n^2)}\|A^1\|^{(6n^2)}=d$ (cf. Lemma 2). By the modified Euclidean algorithm (see Appendix) p and q are bounded by d. Thus each execution of the do loop multiplies $\|K\|$ by at most 2d. There are at most n such multiplications before we arrive at K^{i+1} from

K^i. Thus $K \leqq (2d)^n n(n\|A^1\|)^{n^i}$ (by Lemma 1). Again to account for Step 4.3, an argument similar to the one in Lemma 2 shows that an exponent of $O(n^4)$ suffices.

THEOREM 2. *Algorithm* HNF *is polynomial.*

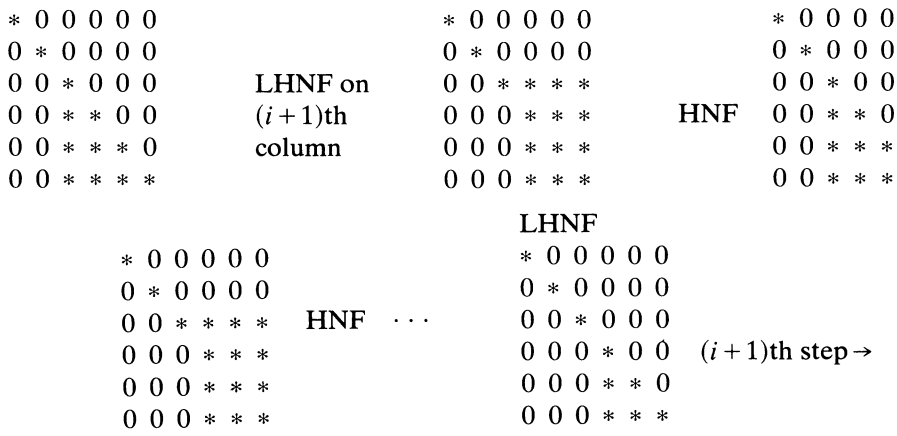
Proof of Theorem 2. Clearly the calculation in Steps 4.1, 4.2 and 4.3 of the algorithm are to be done at most n^2- times. The GCD calculation of step 4.1 is polynomial (see Appendix). Also, Step 2 of the algorithm is polynomial as shown again in the Appendix. Hence, the number of arithmetic operations as well as the number of digits of all intermediate numbers are polynomial and Theorem 2 is proved.

The algorithm HNF(n, A) is concerned with square nonsingular integer matrices. However an examination of the procedure will show that with obvious modification the algorithm works on arbitrary (m, n) integer matrix which has full row rank. In this case, the normal form is (H, 0), where 0 is a $(m \times (n-m))$ block of zeros and the $(m \times m)$ matrix H is in the form prescribed by Theorem 1. The algorithm for this (referred to later as $H(m, n, A)$) is as follows:

Use Step 2 of algorithm HNF(n, A) to permute the columns of A so that the first m principal minors are nonsingular. Call the new matrix A'. Use the steps 3–6 of the algorithm to transform the square nonsingular matrix $\begin{pmatrix} A' \\ 0I \end{pmatrix}$ into Hermite normal form (I is an $(n-m)\times(n-m)$ identity and 0 an $(n-m)\times m$ matrix of zeros). By Theorem 2 this is a polynomial algorithm. Return the first m rows of this HNF and all of K. Clearly using these algorithms we can transform any $(m \times n)$ integer matrix A with full column rank into a "left" Hermite normal form (LHNF) using row instead of column operations, i.e. A will be transformed into $\begin{pmatrix} H \\ 0 \end{pmatrix}$ where H is *upper triangular* with positive diagonal elements. Further, each off-diagonal element of H is nonpositive and strictly less in absolute value than the diagonal element in its column. Let us denote this algorithm by LHNF (m, n, A). LHNF (m, n, A) returns UA and U where U is an (m×m) unimodular matrix and UA the left Hermite normal form. Obviously LHNF (n, m, A) is still polynomial and an analogous result to Lemma 3 holds.

3. An algorithm for Smith normal form. THEOREM 3. (Smith [16]). *Given a nonsingular $(n \times n)$ integer matrix A, there exist $(n \times n)$ unimodular matrices U, K such that $S(A) = UAK$ is a diagonal matrix with positive diagonal elements $d_1, \cdots, d_n$ such that d_i divides d_{i+1} $(i = 1, \cdots, n-1)$.*

The typical step of the polynomial algorithm for the Smith normal form can be summarized in the following pictures:



Note that this algorithm puts the bottom right $(n-i)\times(n-i)$ square matrix into HNF "frequently". Just after this is done each time, the product of the diagonal entries of A equals the absolute value of the determinant of the original matrix and thus each entry of A is at most this determinant in absolute value. Thus, the numbers are prevented from cumulatively building up.

This repeated use of HNF is the crucial difference between the algorithm presented here and the standard algorithms (e.g. [7] and [1]).

In the algorithm below we use the following notation. HNF $(n-i, n-i, A)$ is the procedure which puts the bottom-right-hand minor consisting of the last $(n-i)$ rows and columns into Hermite normal form. LHNF $(n-i, i+1, A)$ is the procedure which puts the submatrix of A consisting of the last $(n-i)$ rows and the column $i+1$ into left Hermite normal form.

ALGORITHM SNF(n, A): returns $(S(A), U, K)$.

1. Set $U = K = I$ the identity matrix of order n.
2. $i = -1$.
3. $i = i+1$. If $i = n$ stop.

At this stage the top-left $(i \times i)$ matrix is already in SNF and, if $i \geqq 1$, $A_{i,i}$ divides $A_{j,k}$ $\forall\, i \leqq j \leqq n$, $i \leqq k \leqq n$.

4. Call LHNF$(n-i, i+1, A)$ (returns A and U^*)

$$U = \begin{pmatrix} I & 0 \\ 0 & U^* \end{pmatrix} U \quad (I \text{ is an } i \times i \text{ identity matrix}).$$

5. Call HNF$(n-i, n-i, A)$ (returns A and K^*)

$$K = K \cdot \begin{pmatrix} I & 0 \\ 0 & K^* \end{pmatrix} \quad (I \text{ is an } i \times i \text{ identity matrix}).$$

6. If $A_{i+1,i+1}$ is not the only nonzero element in column $(i+1)$ go to 4.
7. If $A_{i+1,i+1}$ divides every element $A_{j,k}$ $i+1 \leqq j \leqq k$, $i+1 \leqq k \leqq n$, go to 3, otherwise $A_{i+1,i+1}$ does not divide $A_{j,k}$ (say). Add column k into column $i+1$ in A and K. Go to 4.

THEOREM 4. *The algorithm* SNF *is polynomial.*

Proof. Note that for a fixed i every time Step 4 or 5 is passed $A_{i+1,i+1}$ is replaced by a proper divisor of itself except the last and possibly the first times. Thus, for a fixed i, Steps 4 and 5 are executed at most $\log \|A(i)\| + 2$ times where $A(i)$ denotes the matrix A at the beginning of the ith iteration. Clearly $\|A(i)\| \leqq \max\{|\det(A(0))|, \|A(0)\|\}$, since either $i = 0$ or $A(i)$ is in Hermite normal form. Thus we count at most $2 + \log(\|A(i)\|) \leqq n \log(n\|A(0)\|) + 2$ calls of Step 4 and 5 for each value of i. We have therefore at most $n^2(\log n\|A(0)\|) + 2n$ passes of Steps 4 and 5. But here we use Hermite normal form algorithms and using Theorem 2 this proves Theorem 4.

THEOREM 5. *The unimodular matrices U and K which the algorithm* SNF *returns have entries whose number of digits are bounded by a polynomial.*

Proof. For every U^* and K^* in Steps 4 and 5 of the algorithm we have $\|U^*\|$, $\|K^*\| \leqq (c \cdot n \cdot \|A(0)\|)^{p(n)}$ for some polynomial $p(n)$ and constant c (cf. Lemma 3). By previous arguments, U and K at any stage are the product of at most $n \log((n\|A(0)\|)^n)$ of these matrices. Thus the theorem is proved.

Clearly analogous to LHNF, we can modify the algorithm SNF(n, A) in such a way that it works for arbitrary (n, m) integer matrices and remains polynomial. The details are rather elementary and are left to the reader.

We must remark that the algorithms in this paper are not meant to be efficient. The main concern has been simplicity of presentation. A computer code that includes several efficiency improvements is available from the authors.

Appendix.

LEMMA A.1. *If A is a nonsingular $n \times n$ matrix, then its columns can be permuted so that in the resulting matrix, all principal minors are nonsingular.*

Proof. The first row of A must have at least one nonzero element. Thus after a permutation of columns, if necessary, we can assume that A_{11} is nonzero. Assume for induction that the columns of A have been permuted to give a matrix A' in which the first i principal minors are nonsingular for some i, $1 \leqq i \leqq n-1$. Let A'' be the matrix consisting of all columns of A' and only rows 1 through $(i+1)$. A' is nonsingular implies that rank $(A'') = i+1$. Thus at least one of the columns say k, among $(i+1)$, $(i+2) \cdots, n$ of A'' cannot be expressed as a linear combination of the first i columns of A''. Swapping columns k and $(i+1)$ in A' leads to a matrix whose first $(i+1)$ principal minors are nonsingular. This completes the inductive proof.

The algorithm below uses essentially the idea of the proof.

Step 2 of Algorithm HNF.

for i = 1 to n
det = determinant of the $i \times i$ principal minor of A
$j = 1$
do while $(j \leqq n)$ **and** (det = 0)
$j = j+1$
det = determinant of the $i \times i$ submatrix of A consisting of the first i rows of A and columns 1 through $(i-1)$ and column j of A
end
if $j > n$, **terminate** /* A is singular */
Interchange columns j and i of A and of K.
end.

As remarked earlier, this subroutine is wasteful from the point of view of efficiency. However, it is polynomial, since determinant calculations can be done in polynomial time and at most n^2 determinants are to be computed.

GCD ALGORITHM. We use any standard algorithm (e.g. [2] or [6]) that for any two given numbers a and b, not both zeros, find p, q and r such that $r = GCD(a, b) = pa + qb$. We then execute the following step:

We assume $|a| \geqq |b|$, else swap a and b.

If $|q| > |a|$, then do: $\quad p = p + \left\lfloor \frac{q}{a} \right\rfloor \cdot b$

$$q = q - \left\lfloor \frac{q}{a} \right\rfloor \cdot a$$

end.

Note that we still have $r = pa + qb$. But now, $|q| < |a|$. Thus

$$|qb| < |ab|,$$

$$pa + qb = r \Rightarrow |pa| < |ab| + |r|$$

$$\Rightarrow |p| < |b| + \left|\frac{r}{a}\right| \leqq |b| + 1.$$

Thus $|p|, |q| \leqq |a|$.

The Euclidean algorithm is well-known to be polynomial and certainly the step above is also polynomial.

Acknowledgment. The authors wish to thank Professor Les Trotter for reading the manuscript and suggesting several improvements. Thanks are due to the referee for several helpful suggestions.

REFERENCES

[1] S. BARNETTE AND I. S. PACE, *Efficient algorithms for linear system calculations; Part I—Smith form and common divisor of polynomial matrices*, Internat. J. of Systems Sci., 5 (1974), 403–411.

[2] W. A. BLANKINSHIP, *A new version of the Euclidean algorithm*, Amer. Math. Monthly, 70 (1963), 742–745.

[3] ———. Algorithm 287, *Matrix triangulation with integer arithmetic I* [F1], Comm. ACM, 9 (1966), p. 513.

[4] ———. *Algorithm* 288, *Solution of simultaneous linear diophantine equations* [F4], Comm. ACM, 9 (1966), p. 514.

[5] E. BODEWIG, *Matrix Calculus*, North Holland, Amsterdam, 1956.

[6] G. H. BRADLEY, *Algorithm and bound for the greatest common divisor of n* integers, Comm. ACM, 13 (1970), 433–436.

[7] ———. *Algorithms for Hermite and Smith normal matrices and linear diophantine equations*, Math. Comput., 25 (1971), pp. 897–907.

[8] J. EDMONDS, *Systems of distinct representatives and linear algebra*, J. Res. Nat. Bur. Standards, 71B (1967), pp. 241–245.

[9] M. A. FRUMKIN, *Polynomial time algorithms in the theory of linear diophantine equations*, M. Karpinski, ed., Fundamentals of Computation Theory (Springer, Lecture Notes in Computer Science 56, New York, 1977) pp. 386–392.

[10] R. GARFINKEL AND G. L. NEMHAUSER, *Integer Programming*, J. Wiley & Sons, New York, 1972.

[11] G. A. GORRY, W. D. NORTHUP AND J. F. SHAPIRO, *Computational experience with a group theoretic integer programming algorithm*, Math. Programming, 4 (1973), pp. 171–192.

[12] C. HERMITE, *Sur l'introduction des variables continues dans la théorie des nombres*, J. R. Angew. Math., 41 (1851), pp. 191–216.

[13] M. HEYMANN AND J. A. THORPE, *Transfer equivalence of linear dynamical systems*, SIAM J. Control Optimization, 8 (1970), pp. 19–40.

[14] T. C. HU, *Integer Programming and Network Flows*, Addison-Wesley, Reading, MA, 1969.

[15] M. NEWMAN, *Integral Matrices*, Academic Press, New York, 1972.

[16] H. J. S. SMITH, *On systems of indeterminate equations and congruences*, Philos. Trans., 151 (1861), pp. 293–326.

[17] L. A. WOLSEY, *Group representational theory in integer programming*, Technical Report No. 41, Massachusetts Institute of Technology, Cambridge, MA, 1969.