Exact Solutions and Canonical Forms
- An Application of Modular Arithmetic

Asim Kumar Roy

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

November 1982

ABSTRACT

EXACT SOLUTIONS AND CANONICAL FORMS

- AN APPLICATION OF MODULAR ARITHMETIC

Asim Kumar Roy

Modular methods for obtaining an exact solution of a linear system and for obtaining canonical forms of integral matrices are studied. When an integral matrix is singular the generalized inverse of the matrix is obtained exactly and hence a general solution to a singular system is given exactly. Methods to obtain Hermite Normal Form and Smith Normal Form are given. A heuristic modular method to obtain the Smith Normal Form of an integral matrix is proposed and its performance commented.

To my

Parents

ACKNOWLEDGEMENTS

I greatly acknowledge the invaluable assistance of my thesis supervisor Dr.V.S.Alagar, in all respect of the preparation of this fruitful thesis. In particular, I express my sincere gratitude to Dr.Alagar for proposing me this problem and constant discussion thereafter. Prof.Alagar also read and criticized many versions of this thesis. The final version owes much to him ; the mistakes are my own.

I also thank Dr.Alagar and the FCAC group (Dr.Mckay and others) for the financial assistance given to me during my stay at Concordia University.

Last, but not least, I wish to thank my parents for their patience and understanding during my stay away from them for more than two years.

TABLE OF CONTENTS

Notations Used

SNF	:	Smith Normal Form.
HNF	:	Hermite Normal Form.
CRT	:	Chinese Remainder Theorem.
GCD	:	Greatest Common Divisor.
LCM	:	Least Common Multiple.
mod p	:	Modulo a prime p.
H_n	:	Hilbert Matrix of order n, $h_{ij} = \frac{1}{i+j-1}$.
O(n)	:	Order of n.
(m,n)	:	Greast Common Divisor of m and n.
[m,n]	:	Least Common Multiple of m and n.
$\lceil x \rceil$	:	Smallest integer greater than x.
A_p	:	$(a_{ij} \bmod p)$, where $A=(a_{ij})$.
Y_p	:	$(y_i \bmod p)$, where $Y=(y_i)$.
A^t	:	Transpose of matrix A.

CHAPTER ONE

Introduction.

In the study of controls and system theory, in operations research as well as in the study and characterisation of finitely generated abelian groups exact solutions of linear equations and canonical forms of integer matrices play an important role. Many of the problems that are of practical significance involve integral matrices of large order and require either an exact solution or the transformation of the given matrix to a canonical form. In general the transformed matrix accurately reflect the structure of the original problem.

Some instances where an exact solution is desirable are as follows: In electrical engineering, an exact solution of a linear system of equations are required in designing non-recursive digital filters. In biomedical engineering extremely narrow band filters are needed which accordingly require the component values to be ascertained as exactly as possible. In the solution of structural engineering problems a large number of parameters are usually involved. A solution of the parameters with absolutely no error is required for the safe design of structures. It is highly desirable in such instances to attempt and obtain exact solutions of the systems.

A general advantage of a method yielding an exact solution is that the error analysis is completely eliminated. Moreover

when the matrix is extremely ill-conditioned exact methods are most helpful. In practice one has to deal with a large number of variables. Hence one needs not only exact solutions but an efficient algorithm for obtaining exact solutions.

Two important canonical forms arise in most of the applications. These are known as Hermite Normal Form (HNF) and Smith Normal Form (SNF). Several methods are known to obtain such normal forms. However, there is a great amount of computational difficulty in such procedures.

Solving a system of linear equations, computing a particular solution of a diophantine system of equations and transforming a matrix to HNF or SNF are all related; for they require the determination of a set of linearly independent vectors from the given system of equations. Because of this underlying uniformity and their significance we discuss them more or less as equivalent problems in this thesis.

After explaining the well understood congruence method for obtaining exact solutions we propose how the actual primes may be chosen to assure the uniqueness of an exact solution (see chapter 3). We briefly analyze the proposed congruence method and compare its efficiency with other methods on several examples. This is done in chapter 4.

We also investigate the computation of an exact solution through the HNF of the given matrix. Moreover we also show HNF can repeatedly be used to obtain SNF of a given matrix. Finally we investigate the known methods for obtaining the SNF of a

matrix and comment on their inadequacy (see chapter 5). Several counter examples are given to show how some of the methods that have been proposed in the past cannot work in general. We propose a heuristic congruence method in chapter 6 and illustrate with examples where this method succeeds and where it fails. Although we are not successful in formulating a congruence method for obtaining SNF, the examples on which our algorithm succeeds suggest that there are primes (to be carefully chosen) for which the congruence method will work.

CHAPTER TWO

Review of Previous Research.

In this chapter we give a brief review of the various methods proposed in the past to find the exact solution of a linear system of equations with rational coefficients. There are many numerical problems for which exact solutions are desirable. An important advantage of a method that gives an exact solution is that the error analysis inherent in a numerical method is simplified most of the time or completely eliminated. In particular, when the matrices are ill-conditioned, an exact method gives the true answer whereas a numerical method gives only an approximate answer which may be far from the true answer.

When the exact rational solution to a nonsingular system $Ax=b$ of linear equations with integer coefficients is required, Moenck[M&C79] describes an iterative method based on solutions of reduced systems modulo powers of a single prime p. There are three steps in such a method. The first step is to compute an inverse (mod p) to A. That is, we find an integer matrix C with entries in [0,p-1] such that $AC \equiv I \bmod p$. The second step is to compute an approximate solution y to the exact solution x of $Ax=b$. The final step is to obtain x from the approximate solution y.

It is remarked in [M&C79] that this is p-adic method.

However the final step of recovering the exact rational solution x from y is not discussed in [M&C79].

Below we give a brief and clear description of the iterative algorithm [M&C79] that computes a p-adic approximation y of the exact solution x of the linear system Ax=b.

Algorithm 2.1.

Let p be a prime such that p does not divide the determinant of the matrix A. This algorithm finds an approximation y to the solution x of Ax=b in the sense that $Ay=b \bmod p^k$ holds for suitable k and p and x may be recovered from y.

step1 Set $b_0 \leftarrow b$. Find C whose entries are in [0,p-1] so that $AC \equiv I \bmod p$.

step2 $i \leftarrow -1$

Repeat

$i \leftarrow i+1$;

$x_i \leftarrow Cb_i \bmod p$;

$b_{i+1} \leftarrow p^{-1}(b_i - Ax_i)$

until $(b_{i+1}=b_i)$.

step3 (now we have constructed $x_0, x_1, \ldots x_{k-1}$ for some $k \geq 1$.)

Set $y \leftarrow \sum_{j=0}^{k-1} p^j x_j$.

(Note that $Ay = b_0 - p^k b_k \equiv b_0 \bmod p^k$ and hence y is the p-adic approximation to the exact solution.)

Note that in step2, the entries of vector x_i will be in [0,p-1] and $b_i - Ax_i = 0 \bmod p$ will hold (and hence b_{i+1} will have integer entries). Here we have made an effort to simplify the computations. Note that in [M&C79] the step3 computation is done iteratively.

The following examples illustrate the working of the above algorithm.

Example 2.1

Let $Ax = b$,

where

$$A = \begin{bmatrix} 6 & 7 & 8 \\ 11 & 6 & 7 \\ 5 & 6 & 12 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 13 \\ 16 \end{bmatrix}, \quad \text{and } p = 5.$$

We shall determine y and an integer $k \geq 1$ so that $Ay = b \bmod p^k$ holds.

$$A_1 = A \bmod 5 = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix}.$$

$$A_1^{-1} = \begin{bmatrix} 0 & 1 & -1 \\ 2 & -2 & -1 \\ -1 & 1 & 1 \end{bmatrix}.$$

$$C = A_1^{-1} \bmod 5 = \begin{bmatrix} 0 & 1 & 4 \\ 2 & 3 & 4 \\ 4 & 1 & 1 \end{bmatrix}.$$

$$x_0 = \begin{bmatrix} 0 & 1 & 4 \\ 2 & 3 & 4 \\ 4 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 13 \\ 16 \end{bmatrix} \bmod 5 = \begin{bmatrix} 2 \\ 0 \\ 3 \end{bmatrix}.$$

$$5b_1 = (b_0 - Ax_0) = \begin{bmatrix} 1 \\ 13 \\ 16 \end{bmatrix} - \begin{bmatrix} 6 & 7 & 8 \\ 11 & 6 & 7 \\ 5 & 6 & 12 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 3 \end{bmatrix} = \begin{bmatrix} -35 \\ -30 \\ -30 \end{bmatrix},$$

Therefore

$$b_1 = \begin{bmatrix} -7 \\ -6 \\ -6 \end{bmatrix}.$$

$$x_1 = Cb_1 \bmod 5 = \begin{bmatrix} 0 \\ 4 \\ 0 \end{bmatrix}.$$

$$5b_2 = (b_1 - Ax_1) = \begin{bmatrix} -7 \\ -6 \\ -6 \end{bmatrix} - \begin{bmatrix} 6 & 7 & 8 \\ 11 & 6 & 7 \\ 5 & 6 & 12 \end{bmatrix} \begin{bmatrix} 0 \\ 4 \\ 0 \end{bmatrix} = \begin{bmatrix} -35 \\ -30 \\ -30 \end{bmatrix},$$

Therefore

$$b_2 = \begin{bmatrix} -7 \\ -6 \\ -6 \end{bmatrix}.$$

Since $b_2 = b_1$, we terminate the iteration. Now

$$y = \sum_{j=0}^{1} p^j x_j = (x_o + px)$$

$$= \begin{bmatrix} 2 \\ 0 \\ 3 \end{bmatrix} + 5 \begin{bmatrix} 0 \\ 4 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 20 \\ 3 \end{bmatrix}$$

and $Ay = b \bmod 25$ holds.

The actual solution in this case is

$$\begin{bmatrix} 2 \\ -5 \\ 3 \end{bmatrix}$$

and this must be constructed from y by a p-adic method. We shall not address ourselves to this question in this thesis. See [K&R75] for the application of p-adic number system to exact computations.

The following example shows that we may have k>n, (the order of the matrix) for convergence in the iterative algorithm.

Example 2.2.

Let $A = \begin{bmatrix} 6 & 3 & 2 \\ 6 & 4 & 3 \\ 20 & 15 & 12 \end{bmatrix}$, $b = \begin{bmatrix} 261 \\ 5 \\ 301 \end{bmatrix}$, $p = 5$.

C = inverse of A mod 5 = $\begin{bmatrix} 4 & 2 & 3 \\ 4 & 1 & 2 \\ 0 & 0 & 3 \end{bmatrix}$.

Applying step2 of the algorithm 2.1 we get

$$b_o = \begin{bmatrix} 261 \\ 5 \\ 301 \end{bmatrix}, \quad x_o = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}.$$

$$b_1 = \begin{bmatrix} 48 \\ -4 \\ 42 \end{bmatrix}, \quad x_1 = \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix}.$$

$$b_2 = \begin{bmatrix} 8 \\ -3 \\ 0 \end{bmatrix}, \quad x_2 = \begin{bmatrix} 1 \\ 4 \\ 0 \end{bmatrix}.$$

$$b_3 = \begin{bmatrix} -2 \\ -5 \\ -16 \end{bmatrix}, \quad x_3 = \begin{bmatrix} 4 \\ 0 \\ 2 \end{bmatrix}.$$

$$b_4 = \begin{bmatrix} -6 \\ -7 \\ -24 \end{bmatrix}, \quad x_4 = \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix}.$$

$$b_5 = \begin{bmatrix} -3 \\ -4 \\ -15 \end{bmatrix}, \quad x_5 = \begin{bmatrix} 0 \\ 4 \\ 0 \end{bmatrix}.$$

$$b_6 = \begin{bmatrix} -3 \\ -4 \\ -15 \end{bmatrix}.$$

Since $b_6 = b_5$, we terminate step2 and we have $k=6$.

Moenck [M&C79] has developed the iterative algorithm to obtain exact solutions of $Ax=b$ where A and b have general polynomial entries. Moreover the cost analysis is not given in detail in [M&C79] and we believe that the cost analysis of this

approach needs a careful study. We are not addressing to this approach in this thesis and simply mention this as another approach to finding exact solutions of linear systems.

Gaussian elimination is an accepted basic technique for the solution of a system of equations regardless of the type of matrix A. When the entries of matrix A are rational numbers then by suitable scaling we can transform the entries to integers. When the matrix entries are polynomials with rational coefficients then once again the entries can be reduced to polynomials with integer coefficients. An exact method using Gaussian elimination in its naive form will normally produce rational numbers with large numerators and denominators during the intermediate stages when the coefficients are originally integers. If the algorithm is used over the domain of polynomials, say all linear, then the algorithm will produce polynomials of degree nearly 1000 during the intermediate stage for a matrix of order 11, whereas by Cramer's rule one expect no polynomials of degree greater than 11. Hence multiple precision operation would become inevitable when the raw form of Gaussian elimination is applied and consequently it becomes a slow process.

Rosser [Ros52] develops a method to control the growth of intermediate results but which introduces many new multiprecision computations. The fraction free variation of Gaussian elimination also cannot guarantee to limit the size of intermediate results.

Bareiss [Bar72] discusses a fraction free elimination

method (one-step elimination) which eliminates one variable at a time and also discusses an improved fraction free method (two-step elimination) that eliminates two variables at a time instead of one.

One-step elimination algorithm is

for k=0,1,...,n-1 do

for i=k+1,k+2,...,n do

for j=k+1,k+2,...,n+1 do

$$a_{ij}^{(k+1)} = a_{kk}^{(k)} a_{ij}^{(k)} - a_{kj}^{(k)} a_{ik}^{(k)} \; / \; a_{k-1,k-1}^{(k-1)}$$

and the two-step elimination algorithm is

for k=0,2,4,...,n-2 do

for i=k+2,k+3,...,n do

for j=k+2,k+3,...,n+1 do

$$a_{ij}^{(k+2)} = \{a_{ij}^{(k)} c^{(k+1)} + a_{kj}^{(k)} c_1^{(k+1)} + a_{k+1,j}^{(k)} c_2^{(k+1)}\} \; / \; a_{k-1,k-1}^{(k-1)}$$

where

$$c^{k+1} = a_{k+1,k+1}^{k+1},$$

$$c_1^{k+1} = (a_{k+1,k}^{(k)} a_{i,k+1}^{(k+1)} - a_{k+1,k+1}^{(k+1)} a_{ik}^{k}) \; / \; a_{kk}^{(k)},$$

$$c_2^{(k+1)} = -a_{i,k+1}^{(k+1)}.$$

If n is odd, the last row is transformed by one-step algorithm using $k = n-1$.

However multiprecision arithmetic is still not ruled out in this method, although the two-step fraction free algorithm has been reported to be 50% faster than one step fraction free method. More detailed information and test results can be found in Bareiss [Bar72].

Congruence techniques are noted for avoiding the problem of intermediate expression swelling. Moreover in residue arithmetic one works only with integers modulo a prime and cosequently the computations are exact, there are no rounding errors and hence even ill-conditioned problems can be handled easily. The basic idea of the approach is to (1) replace the original system of equations by a system of congruences modulo several primes p_i (it is enough that the moduli p_i are coprimes to each other) (2) solve each reduced system mod p_i using Gauss-Jordan diagonalization method and (3) combine the solutions by the Chinese Remainder Theorem (CRT) to obtain the solution to the original system.

Several authors notably Borosh [B&F66], Newman [New67], Howell [H&G69], Cabay [C&L77], McClellan [Mcl77] and Frumkin [Fru76] have discussed modular methods for obtaining exact solutions of linear equations. Once a modulus is chosen the rest of the method is essentially similar to Gauss Jordan elimination. Therefore the discussed methods differ only in the choice of primes. A closer look of the previous research reveals that multiple precision arithmetic would become necessary only when the final solution is put togather using the Chinese Remainder Theorem.

Newman [New67] uses 10 carefully chosen primes and shows by test results that they are sufficient to solve the system exactly when the coefficient matrix is a Hilbert matrix of order n, $3 \leq n \leq 12$. Cabay [C&L77] considers a carefully chosen set of 100 primes, say $p_1, p_2, \ldots, p_{100}$ and calculates

$$P_i = \prod_{j=1}^{100} p_j, \quad i \neq j$$

and q_i where $q_i P_i \equiv 1 \bmod p_i$ and stores them in a large table. These and the inverses are used to solve any system in a particular machine. The main conclusion of their test results is that for matrices of high order ($10 \leq$ order ≤ 26) the congruencial method as implemented by them outperforms the two-step method of Bareiss [Bar72]. Fraenkel [F&D71] discusses an implementation for the exact solution of a system with 120 variables with integer coefficients in the range [-2180,2568] and reports that his choice of 10 primes (14 digits long) successfully solves the system in 19 minutes in CDC 1604 computer.

The principal disadvantage of the congruence method is that it is limited to the system with integral elements and the whole process can become very time consuming unless the sequence of solutions modulo the various primes converge quickly to the true solution. Although one set of primes may be sufficient to solve several different systems, for some of these systems fewer number of primes would have been sufficient. In this sense it is desirable to determine the exact number of primes and the actual set of primes for a given system of linear equations.

Moreover which one of the two methods - direct computation using multiprecision arithmetic or modular technique - is superior in terms of speed and storage requirment has not been entirely determined.

The congruence techniques can immediately be carried over to matrices with polynomial coefficients, the general idea of modular method is certainly applicable. McClellan [Mcl77] discusses such a method.

In this thesis we are mainly concerned with matrices with integer entries and our main contribution to exact solutions (refer chapter 3 and 4) is the determination of the number of primes and the actual choice of primes depending on the order of the matrix and the maximum entry of the augmented matrix. We give a brief complexity analysis and give test results.

CHAPTER THREE

Description of Congruence Method.

In this chapter we give the congruential method for obtaining the exact solution of a system of linear equations with integral coefficients. A new method is proposed for choosing primes in a certain range. This range is determined from the order and the maximum absolute value of the entries in the augmented matrix.

3.1 Basic Notation and Results.

Let A be an (nxn) nonsingular integral matrix and b an (nx1) integral vector. Let

$A = (a_{ij})$, $1 \leq i,j \leq n$,

d = determinant of the matrix A,

A^{adj} = adjoint of A,

M_A = maximum absolute value of the elements in A,

M_b = maximum absolute value of the elements in b.

For any nonsingular matrix A we know

$$AA^{adj} = A^{adj}A = dI \qquad (3.1.1)$$

where I is an identity matrix of the same order as that of A. Now the equation

$$Ax = b \qquad (3.1.2)$$

can be written as

$$x = A^{-1}b = \frac{A^{adj}}{d} b = \frac{y}{d}$$

where $y = A^{adj} b$.

Therefore the solution of (3.1.2) is

$$x = \frac{y}{d}.$$

For any prime p_k and integer z let z_k denote the least positive residue of z mod p_k. We write

$$z_k \equiv z \bmod p_k. \qquad (3.1.4)$$

For any vector $y=(y_1,y_2,\ldots,y_n)$ we also write $y_p=(y_1 \bmod p, y_2 \bmod p, \ldots, y_n \bmod p)$. We need the following basic lemma in the development of an exact solution to a system of linear equations with integer coefficients.

Lemma 3.1

Let M_y be the maximum element in vector y where $y=A^{adj}b$ and d be the determinant of the matrix A. Let P be an integer satisfying the conditions

i) $(d,P) = 1$,

ii) $P > 2 \max(|d|, M_y)$.

Then any pair of solutions (d_p, y_p) of the congruences

iii) $d \equiv d_p \bmod P$,

iv) $Ay_p \equiv db \bmod P$,

satisfying the conditions

v) $|d_p| < \frac{P}{2}$,

vi) $M_{y_p} < \frac{P}{2}$,

imply $d = d_p$ and $y = y_p$.

Proof See Newman [New67].

If the modulus is chosen as a product of primes (or mutually coprime integers) then subject to the conditions of lemma 3.1 and the Chinese Remainder Theorem, the solution of the equation (3.1.2) can be written down exactly. This we illustrate below.

Now we choose several moduli $p_1,\ldots,p_s$ such that $(p_i,p_j)=1$ and $P = p_1p_2\ldots p_s$. Define p_i' as

$$\frac{P}{p_i} p_i' \equiv 1 \bmod p_i, \quad 0 \leq p_i' < p_i, \ 1 \leq i \leq s.$$

Then the solution of the system

$$z \equiv a_i \bmod p_i, \qquad 1 \leq i \leq s$$

is given by

$$z = \sum_{i=1}^{s} \frac{P}{p_i} p_i' a_i \bmod P. \tag{3.1.5}$$

Define d_{p_i} and y_{p_i}, $1 \leq i \leq s$ as

$$d \equiv d_{p_i} \bmod p_i \quad \text{and}$$

$$A y_{p_i} \equiv db \bmod p_i .$$

Then we write d_p using (3.1.5) as

$$d_p = \sum_{i=1}^{s} \frac{P}{p_i} p_i' d_{p_i} \bmod P.$$

By applying the Chinese Remainder Theorem to corresponding components of the vectors y_{p_i}, we write

$$y_P = \sum_{i=1}^{s} \frac{P}{p_i} p_i' y_{p_i} \bmod P.$$

Having found d_p and y_p we can find d and y such that

$$d \equiv d_p \bmod P,$$

$$y \equiv y_p \bmod P.$$

The practical utility of an ideal program that implements the above method rests primarily on choosing the modulus P and hence the primes. This may be done by the use of Hadamard inequality

$$|d| \leq \prod_{i=1}^{n} \left(\sum_{j=1}^{n} a_{ij}^2 \right)^{\frac{1}{2}}. \qquad (3.1.6)$$

This inequality can be written down in several equivalent forms. For example,

We know $|a_{ij}| \leq M_A$.

Therefore $|d| \leq \prod_{i=1}^{n} \{nM_A^2\}^{\frac{1}{2}} = (n^{\frac{1}{2}} M_A)^n$,

$$|d| \leq n^{\frac{n}{2}} (M_A)^n . \qquad (3.1.7)$$

We know $y = A^{adj}\, b$

where $A^{adj} = (c_{ij})$ (say). Let $y = (y_i)$, $i = 1, \ldots, n$.

Therefore $y_i = \sum_{j=1}^{n} c_{ij}\, b_j$.

Since c_{ij} is the determinant of an $(n-1)\times(n-1)$ submatrix of A, we have

$$|y_i| \leq \sum_{j=1}^{n} |c_{ij}|\,|b_j|$$

$$\leq \sum_{j=1}^{n} (n-1)^{\frac{n-1}{2}} M_A^{n-1} M_b .$$

Hence we have

$$M_{y_i} \leq n(n-1)^{\frac{n-1}{2}} M_A^{n-1} M_b . \qquad (3.1.8)$$

Now combining (3.1.7) and (3.1.8) condition (ii) of lemma 3.1 becomes

$$P = \prod_{i=1}^{s} p_i > 2 \max\{ n^{\frac{n}{2}} M_A^n ,\ n(n-1)^{\frac{n-1}{2}} (M_A)^{n-1} M_b \}. \qquad (3.1.9)$$

Another formulation of Hadamard bound can be stated as follows :

$$|d| \geq \prod_{j=1}^{n} \left(\sum_{i=1}^{n} a_{ij}^2 \right)^{-\frac{1}{2}} \qquad \text{if } \bar{b} \leq \bar{a}$$

$$\geq \prod_{j=1}^{n} \left(\sum_{i=1}^{n} a_{ij}^2 \right)^{\frac{1}{2}} \bar{b}/\bar{a} \qquad \text{if } \bar{b} > \bar{a}$$

where $\bar{a} = \min \{ \left(\sum_{i=1}^{n} a_{ij}^2 \right)^{\frac{1}{2}} \}$,

$$\bar{b} = \sum_{i=1}^{n} (b_i^2)^{\frac{1}{2}}.$$

See [C&L77].

From Hadamard bound (3.1.9) we can obtain a bound for the number of primes. Let $p_1,\ldots,p_s$ be primes such that $(d,p_i)=1$. The algorithm that we describe later makes sure that $(d,p_i)=1$ holds for every prime chosen. Then we have

$$P = \prod_{i=1}^{s} p_i > 2 \max\{ n^{\frac{n}{2}} M_A^n , n(n-1)^{\frac{n-1}{2}} M_A^{n-1} M_b \}.$$

Let the right hand side of the above inequality be W and

$$U = \min \{ p_i \}, \qquad 1 \leq i \leq s$$

$$V = \max \{ p_i \}. \qquad 1 \leq i \leq s$$

Therefore $V^s > W$.

If $W = 2n^{\frac{n}{2}} M_A^n$ then

$$V > 2^{\frac{1}{s}} n M_A^2. \qquad (3.1.10)$$

$$s < (\log_2 W - 1) / (\log_2 n + 2\log_2(M_A))$$

assuming $\frac{n}{2s} > 1.$ (3.1.11)

If $W = 2n(n-1)^{\frac{n-1}{2}} (M_A)^{n-1} M_b$ then

$$V > 2^{\frac{1}{s}} n^{\frac{1}{s}} (n-1) M_A^2 (M_b)^{\frac{1}{s}}, \quad (3.1.12)$$

$$s < \{ \log_2 W - (1+\log_2 n + \log_2 M_b) \} / \{\log_2(n-1) + 2\log_2 M_A\}$$

assuming $\frac{n-1}{2s} > 1.$ (3.1.13)

We need to choose s primes whose product P satisfies lemma 3.1. The bound for s given by (3.1.11) or (3.1.13) has been obtained from the Hadamard determinantal bound (3.1.7).

However in practice we decided to choose $s=\lceil \frac{n}{2} \rceil$ primes greater than or equal to V. Although this choice of s violates the condition (3.1.11) or (3.1.13), we remark the following: In most of the practical problems we have found Hadamard determinantal bound to be an overestimate of the actual value of the determinant. By lemma 3.1 we only need

$$P = \prod_{i=1}^{s} p_i > 2\max(|d|, M_y),$$

where $|d|$ and M_y are usually far less than the bound given in (3.1.7) and (3.1.8). The assumption $s=\lceil \frac{n}{2} \rceil$ has the effect

of lowering the true value of V. Recall that V is the maximum of the p_i's to be chosen. Since we always choose primes larger than this maximum, the error introduced in lowering the true V does not seem to affect our choice.

When n is moderately large, say $n \leq 20$, our bound is realistic; however the actual primes indeed depend on W. The version of Hadamard inequality given by Cabay [C&L77] is preferable to get the actual primes. We write

$$P \geq \prod_{j=1}^{n} \left(\sum_{i=1}^{n} a_{ij}^2 \right)^{\frac{1}{2}} \quad \text{if} \quad \bar{b} < \bar{a} \qquad (3.1.14)$$

and $$P \geq \prod_{j=1}^{n} \left(\sum_{i=1}^{n} a_{ij}^2 \right)^{\frac{1}{2}} \bar{b}/\bar{a} \quad \text{if} \quad \bar{b} \geq \bar{a} \qquad (3.1.15)$$

where $$\bar{a} = \min \left\{ \left(\sum_{i=1}^{n} a_{ij}^2 \right)^{\frac{1}{2}} \right\},$$

$$\bar{b} = \sum_{i=1}^{n} (b_i^2)^{\frac{1}{2}}.$$

Therefore we have

$$V^{\left\lceil \frac{n}{2} \right\rceil} > P.$$

This gives V which we call LBND.

If the order of the coefficient matrix is large and elements of augmented matrix are large then naturally one need

very large primes. Since such large primes are not readily available and since we are forced to do extended precision calculation we try to find an upper bound (UBND) of the primes such that all required primes lie between [LBND,UBND]. This can be obtained from the well known Prime Number Theorem [Knu69] which states " for large X the number of primes $\leq$ X is asymptotically equal to $\frac{X}{\log X}$".

We have already found LBND. Our aim is to find UBND such that

$$\frac{UBND}{\log(UBND)} - \frac{LBND}{\log(LBND)} = s;$$

that is $$\frac{UBND}{\log(UBND)} = c,$$

where $$c = \frac{LBND}{\log(LBND)} + s;$$

which implies $$UBND = e^{\frac{UBND}{c}}. \qquad (3.1.16)$$

The equation (3.1.15) can be solved by Newton's root finding method [J&R77].

Let $$f(UBND) = UBND - e^{\frac{UBND}{c}}$$

then $$f'(UBND) = 1 - \frac{1}{c} e^{\frac{UBND}{c}}.$$

From Newton's root finding method we know

$$UBND_{m+1} = UBND_m - \frac{f(UBND_m)}{f'(UBND_m)}.$$

So with suitable initial guess and with suitable ϵ one can get UBND when

$$\text{abs}\ (UBND_{m+1} - UBND_m) < \epsilon .$$

Of course initial guess should be at least LBND because we are looking for an integer greater LBND. The following algorithm first finds a rough UBND and then tries to lower the UBND as much as possible by successive trials.

Algorithm FINDUBND

step1 Input s, LBND (LBND is the initial guess).

step2 $r \leftarrow \frac{LBND}{\log(LBND)}$.

step3 UBND <-- LBND; c <-- s+r.

step4 $Q \leftarrow \frac{UBND}{c}$.

step5 T <-- UBND - exp(Q).

step6 $d \leftarrow 1 - \frac{1}{c}\exp(Q)$.

step7 $L \leftarrow UBND - \frac{T}{d}$.

step8 If abs(UBND - L) < 1E-6 then print UBND and terminate.

step9 UBND <-- L, return to step4.

Lowering UBND by Successive Trials

Let V = smallest prime > LBND, and T denote the Hadamard bound.

step1) Find all primes in [V,UBND] that is

$V = p_1 < p_2 <, \ldots, < p_s < UBND.$

step2) $R \leftarrow\!- \prod_{i=1}^{s} p_i$; $R' \leftarrow\!- R$.

step3) Find the next prime smaller than p_1 and call this newprime.

step4) $R' \leftarrow\!- (\frac{R'}{p_s})$ (newprime).

step5) If $R' < T$ then terminate and accept $p_1, \ldots, p_s$.

step6) For i = s-1 down to 1 do $p_{i+1} \leftarrow\!- p_i$.

step7) $p_1 \leftarrow\!-$ newprime , return to step3.

3.2 Description of Proposed Method.

In this section we describe the modular algorithm in detail. The algorithm is presented first and then we comment on important steps of the algorithm and we give a simple analysis.

Algorithm MODULAR.

step1 Input matrix A, vector b and order of the matrix.

step2 Find s (number of primes required to solve).

step3 Find lowerbound(LBND) and upperbound(UBND) of primes.

step4 Generate s primes $p_1,\ldots,p_s$ in the range (LBND,UBND).

step5 Find the product $P = p_1p_2\ldots p_s$.

step6 i = 0.

step7 i = i+1.

step8 8.1 Reduce the augmented matrix modulo prime (p_i).

8.2 Solve the system of equations by Gauss Jordan method and find d_{p_i} and y_{p_i} such that

$$d_{p_i} \equiv d \bmod p_i,$$

$$Ay_{p_i} \equiv db \bmod p_i.$$

step9 For $Q_i=\frac{P}{p_i}$ determine Q_i' such that $Q_iQ_i' \equiv 1 \bmod p_i$.

step10 (accumulate the previous results with the current result obtained at step8)

$$d = d + \frac{P}{p_i} Q_i' d_{p_i},$$

$$y = y + \frac{P}{p_i} Q_i' y_{p_i}.$$

step11 If $i < s$ then return to step7.

step12 Reduce d , y modulo P such that

$$| d | < \frac{1}{2} P,$$

$$M_y < \frac{1}{2} P.$$

step13 Print the results d and y.

A substantial amount of computation takes place at step 8.2 in MODULAR algorithm. We give below a detailed description of step 8.2 with respect to one modulus p_i .

Let $C=(c_{ij})$ denote the augmented matrix (A,b).

Step 8.2 of MODULAR Algorithm.

step 8.2.1 k <-- 1.

step 8.2.2 det <-- 1 (Initialize the value of the determinant mod p_i).

step 8.2.3 Find r in the range $k \leq r \leq n$ such that $(c_{rk},p_i) = 1$. If such an r is not found then do the following:

a) Discard the prime p_i, b) Choose another prime $p_t > p_s$ and set $p_j <-- p_{j+1}$, $j=i,\ldots,s-1$; $p_s <-- p_t$, $P <-- \frac{P}{p_i} p_t$, c) Return to step 8.2.1. (Note that this step occurs only when the system is singular modulo one of the chosen primes. By initially choosing large primes, the probability of such an event has been minimized).

step 8.2.4 If r = k then do step 8.2.6 else do step 8.2.5.

step 8.2.5 Interchange rows r and k. Retain the matrix as C. Replace det by -det.

step 8.2.6 Find g such that $gc_{kk} \equiv 1 \bmod p_i$. (The inverse is computed using Euclidean algorithm.)

step 8.2.7 Replace det by c_{kk}det and replace c_{kt} by gc_{kt}, $k \leq t \leq (n+1)$.
(All computations must be performed modulo p_i. Retain the resulting matrix as C).

step 8.2.8 For $1 \leq s \leq n$, $s \neq k$ replace c_{st} by $c_{st} - c_{sk}\, c_{kt}$, $k \leq t \leq (n+1)$.
(Here also the computation must be performed modulo p_i).

step 8.2.9 If $k < n$ then $k = k+1$, return to step 8.2.3 else do the following:

step 8.2.10 d_{p_i} <-- det.

step 8.2.11 $y_{p_i}^{(i)}$ <-- $(\det)c_{i,n+1} \bmod p_i$, $1 \leq i \leq n$.

After successful completion of the steps from 8.2.1 to 8.2.9 the augmented matrix C in step 8.2.11 becomes

$$\begin{bmatrix} 1 & 0 & 0 \ldots\ldots\ldots 0 & * \\ 0 & 1 & 0 \ldots\ldots\ldots 0 & * \\ 0 & 0 & 1 \ldots\ldots\ldots 0 & * \\ \cdot & \cdot & \cdots\cdots\cdots & \cdot \\ \cdot & \cdot & \cdots\cdots\cdots & \cdot \\ 0 & 0 & 0 \ldots\ldots\ldots 1 & * \end{bmatrix}$$

where * implies that the entry may not be zero.

We comment on the computation of step10 of MODULAR algorithm : We can compute $Q_i{}^{d}p_i$ and $Q_i{}^{y}p_i$ modulo p_i before multiplying each by $\frac{P}{p_i}$. This will simplify the calculations in step12.

The algorithm can be improved further by computing the GCD using a series of shifts rather than using the conventional method. The details on this improved GCD calculations can be found in Knuth [Knu69].

CHAPTER FOUR

Analysis of Different Methods.

In this chapter we analyze the iterative method, two-step elimination method (see chapter 2 for a brief discussion) and MODULAR method. The cost of an algorithm is measured by the number of single precision arithmetic operations required in the algorithm to solve a system of equations.

4.1 Analysis of Iterative Method.

Let

$$Ax = b \tag{4.1.1}$$

be the equation with A an nxn integer matrix and b an nx1 integer vector. For simplicity we assume

$$d = \det(A) \neq 0.$$

Let

$$c_j = \left(\sum_{i=1}^{n} a_{ij}^2 \right)^{\frac{1}{2}}, \quad j=1,\ldots,n, \tag{4.1.2}$$

$$c_o = \left(\sum_{i=1}^{n} b_i^2 \right)^{\frac{1}{2}}, \tag{4.1.3}$$

$$M_A = \max \; | \; a_{ij} \; |, \qquad 1 \leq i,j \leq n \tag{4.1.4}$$

$$M_b = \max \; | \; b_i \; |, \qquad 1 \leq i \leq n. \tag{4.1.5}$$

Let

$$c_k = \min (c_i), \qquad 0 \leq i \leq n \qquad \text{and}$$

$$r = \prod_{i=0}^{n} c_i, \qquad i \neq k \qquad (4.1.6)$$

From Cramer's rule it is clear that in the solution vector each rational component has both its numerator and denominator less than or equal to r. Let

$$M = \max (M_A , M_b). \qquad (4.1.7)$$

Then $$r = \prod_{i=0}^{n} c_i, \qquad i \neq k$$

$$\leq (nM^2)^{\frac{n}{2}}.$$

Therefore $$r \leq M^n n^{\frac{n}{2}}. \qquad (4.1.8)$$

Let p be a prime such that p does not divide d and $p \leq M$. Assume that M and hence p are single precision integers. The first step in the iterative solution is to form A_1 such that $A_1 = A \mod p$. The cost of forming A_1 measured as the number of multiplications and divisions is $O(n^2)$. Now each entry of A_1 lies in [0,p-1]. Next to compute $C = A_1^{-1} \mod p$ that is $CA_1 \equiv I \mod p$ using any well known algorithm, the cost of this step can be accounted as $O(n^3)$.

In step2 we can compute x_i in $O(n^2)$ single precision multiplications. However the entries of b_{i+1} may become large and may require multiple precision arithmetic. When this

happens to compute Ax_i we need n^2 multiplications each involving a single precision entry in A (at most of length log(M)) and a single precision integer in x_i (at most of length $\log(p) \leq \log(M)$). Assume that this product can be done in O(logMlog(logM)). Hence The total cost of step2 is $O(kn^2 \log M \log(\log(M))$ arithmetic operations. Similarly we can show that the maximum cost of step3 is

$$\sum_{j=0}^{k-1} O(jn \log M \log(j \log(M)).$$

Therefore the total cost of the algorithm cannot exceed

$$O(kn^2 \log M \log\log M) + O\left(\sum_{j=0}^{k-1} jn \log M \log(j \log M)\right) \qquad (4.1.9)$$

After simplifying we write the righthand side of (4.1.9) as

$$O(kn^2 \log M \log\log M) + O(k^2 n \log M \log\log M) + O(kn^2 \log k \log M).$$

Hence fixing the maximum absolute value of the entries of input matrices, the total cost of the iterative algorithm cannot exceed

$$O(n^3) + O(kn^2) + O(k^2 n) + O(nk^2 \log k).$$

The exact value of k is not known to us; we believe that finding k may still be an open problem. Note that in example 2.1 we had k=2 and in example 2.2 we had k=6 and in both cases we hade n=3. It seems from these examples and others we have tried that it is safe to assume that k must be greater than or equal to n. So we conclude that the cost of finding an approximate solution by the iterative algorithm 2.1 is at least $O(n^3 \log n)$.

4.2 Analysis of MODULAR Method.

Here we give a detailed description of the cost analysis of our MODULAR method. The cost of steps 8 to 13 in MODULAR algorithm is independent of any specific primes as long as a prime can fit in one computer word. We itemize the cost below:

step 8.1 1) To reduce the augmented matrix modulo a prime we need n^2 divisions. So the cost of doing this step is $O(n^2)$.

step 8.2 1) Each GCD call i.e., to find (x,y) we need the following operations :

Let $z = \max(x,y)$. NI = number of iteration $\sim 2\log(z) = 2\log(M)$. Then total number of divisions/multiplications is 3NI and total number of additions/subtractions is 2NI.

The total number of GCD calls is $\frac{n(n+1)}{2}$. Therefore the total cost is at most $O(n^2\log(M))$.

2) In solving a system of linear equation using Gauss Jordan elimination process we need

$\frac{n^3}{2} + n^2 - \frac{n}{2}$ multiplications/divisions

$\frac{n^3}{2} - \frac{n}{2}$ additions/subtractions.

Therefore to solve a reduced system the cost is $O(n^3)$.

Since we need $s=\lceil\frac{n}{2}\rceil$ primes to solve the system using

our MODULAR algorithm, the cost of MODULAR algorithm is

$$O(sn^3) = O(n^4).$$

We observe that in many problem instances for which $n \leq 20$ and whose entries are initially single precision integers, algorithm MODULAR successfully finds $s = \lceil \frac{n}{2} \rceil$ primes to obtain the exact solution.

However there are some exceptions e.g., when we transform a Hilbert matrix of order 14 by suitable scaling we have a matrix with single precision integer entries. According to our hypothesis $s = \lceil \frac{n}{2} \rceil = 7$ primes are sufficient; the primes found according to MODULAR algorithm are 20 digits long and hence our analysis should be modified to reflect multiprecision arithmetic in step 8.2 of the algorithm MODULAR. Analysing similar to section 4.1 for multiprecision arithmetic operations, the cost in such instances is of $O(n^4 \log(M_p))$ where M_p is the maximum of primes.

For large values of n say $n>20$ our estimate $s = \lceil \frac{n}{2} \rceil$ can still be valid provided sufficiently large primes are made available. Recently Fraenkel [F&A71] reports 10 primes each 14 digits long that successfuly solve a system with 120 variables. In view of this it seems to us that the total number of arithmetic operation in a congruence method is at most $O(n^4)$.

Finally we comment how our MODULAR algorithm can be successfully carried over to find a particular solution of a system of linear diophantine equations. For example, if A is an

integral matrix of order mxn of rank m then the system of linear equations $Ax = b$ can be solved as follows : Define $B=AA^t$, $By=b$ and $x=A^ty$. Then, x is the solution of $Ax=b$. Using MODULAR algorithm described earlier we obtain the solution vector y and then compute $x = A^ty$.

Clearly to compute B requires $O(mn^2)$ arithmetic operations. The cost of solving $By=b$ using MODULAR algorithm is $O(m^4)$. Now we need to find x given by $x=A^ty$. This has a cost $O(nm^2)$. Hence a particular solution of a linear diophantine system of m equations in n variables can be obtained with a cost $O(m^4+mn^2+nm^2)$.

Since $m \leq n$ and in many problem instances n is large compared to m, the cost $O(m^4+mn^2+nm^2)$ is quite reasonable to obtain a particular solution of a system of m diophantine equations in n variables.

Alternatively when a square matrix is singular or we have a rectangular (mxn) matrix of rank $r \leq m$ then we can use a generalized inverse to obtain a particular solution or a general solution of the resulting diophantine system.

Let A be an (mxn) matrix. Then A^g is said to be a generalized inverse of A [H&W70] provided

$$AA^gA = A \quad \text{and} \quad A^gAA^g = A.$$

It is shown in [Bur50] that the standard way to give the general solution of $Ax = b$ is to use a generalized inverse.

For a matrix A of rank $r \leq m$, the invariant factor theorem

[Bur50] , [M&M70] provides an algorithm for constructing two unimodular matrices X and Y such that XAY = S,

where $S = s_{ij}$

with $s_{ii} \neq 0$ for $i \leq r$

$s_{ij} = 0$ otherwise.

Moreover $s_{ii} | s_{i+1,i+1}$ for $i = 1,\ldots,r-1$.

Let S^g be the (mxn) matrix satisfying

$$s^g_{ii} = s^{-1}_{ii}, \quad \text{for } i \leq r$$

$$s_{ij} = 0 \quad \text{otherwise.}$$

Now, let $A^g = YS^gX$ then

$$XAY = S = SS^gS = XAYS^gXAY = XAA^gAY \tag{4.2.1}$$

which shows that

$$A = AA^gA.$$

Similarly it can also be shown that $A = A^gAA^g$.

It is also shown in [H&W70] that, if A and b are integral and the vector equation

$$Ax = b$$

is consistent, then Ax = b has an integral solution if and only if the vector A^gb is integral. Moreover in this case the general integral solution of Ax = b is

$$x = A^gb + (I - A^gA)y \tag{4.2.2}$$

where y is any arbitrary integral vector of order m.

The matrix S is known as the Smith Normal Form of A. So we note that the Smith Normal Form of a matrix is closely related to solving a linear system of equations exactly. We discuss in chapter 6 how to obtain the Smith Normal Form of an integer matrix.

Below we give examples to show two different ways of obtaining an exact solution of a singular system of equations.

Example 4.2.1.

Let $A = \begin{bmatrix} -9 & -8 & -5 \\ 6 & 5 & 2 \\ 3 & 2 & -1 \end{bmatrix}$ and $b = \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}$.

The value of the determinant of matrix A is zero and the rank of A is 2. Using a standard procedure (section 5.3), we get

$$X = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 2 \\ 1 & 2 & -1 \end{bmatrix}, \qquad Y = \begin{bmatrix} 0 & 1 & -3 \\ 0 & -1 & 4 \\ -1 & 1 & -1 \end{bmatrix}$$

and

$$S = XAY = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Now

$$A^g = YS^gX = \frac{1}{3}\begin{bmatrix} 0 & 1 & 2 \\ 0 & -1 & -2 \\ 0 & 1 & -1 \end{bmatrix}.$$

A particular solution is

$$A^g b = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}.$$

Now the general solution is given by (4.2.3). If we take $y=[-1,0,1]^t$, then another solution of the same system is $[4,-5,1]^t$.

Example 4.2.2 (using modular arithmetic).

We choose the prime p = 127. We compute

$(A^g)_p = F^tXE$ and A^g as in [A&K77].

We get

$$E = \begin{bmatrix} 14 & 0 & 0 \\ 125 & 124 & 0 \\ 126 & 125 & 1 \end{bmatrix}, \quad F = \begin{bmatrix} 1 & 0 & 0 \\ 112 & 1 & 0 \\ 3 & 123 & 1 \end{bmatrix}, \quad X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

and

$$(A^g)_p = \begin{bmatrix} 44 & 45 & 0 \\ 125 & 124 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad A^g = \begin{bmatrix} 5 & 8 & 0 \\ -6 & -9 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

See [A&K77] for a detailed calculation. Therefore we have

$$A^g b = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}, \qquad (I - A^g A)y = \begin{bmatrix} 3 \\ -4 \\ 0 \end{bmatrix}.$$

Therefore a particular solution is

$$x = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} + \begin{bmatrix} 3 \\ -4 \\ 0 \end{bmatrix} = \begin{bmatrix} 4 \\ -5 \\ 1 \end{bmatrix}$$

where $y = [-1,0,1]^t$.

4.3 Analysis of Two Step Multiprecision Elimination Process.

Bariess [Bar72] describes one-step and two-step elimination method in the following way:

One-step method:

$$a_{ij}^{k+1} = \{ a_{kk}^{k} a_{ij}^{k} - a_{kj}^{k} a_{ik}^{k} \} / a_{k-1,k-1}^{k-1} \qquad (4.3.1)$$

Two-step method:

In the (k+2) iteration (4.3.1) can be written as

$$a_{ij}^{(k+2)} = \{ a_{kk}^{(k)} a_{ij}^{(k)} - a_{kj}^{(k)} a_{ik}^{(k)} \} / a_{k-1,k-1}^{(k+1)} \qquad (4.3.2)$$

Substituting for $a_{ij}^{(k+1)}$ and $a_{k+1,j}^{(k+1)}$ from (4.3.1) into (4.3.2) and simplifying we get

$$a_{ij}^{(k+2)} = \{ a_{ij}^{(k)} c^{(k+1)} + a_{k,j}^{(k)} c_1^{(k+1)} + a_{k+1,j}^{(k)} c_2^{(k+1)} \} / a_{k-1,k-1}^{(k+1)}$$

where

$$c^{(k+1)} = a_{k+1,k+1}^{(k+1)}$$

$$c_1^{(k+1)} = \{ a_{k+1,k}^{(k)} a_{i,k+1}^{(k+1)} - a_{k+1,k+1}^{(k+1)} a_{i,k}^{(k)} \} / a_{kk}^{(k)}$$

$$c_2^{(k+1)} = -a_{i+k+1}^{(k+1)}$$

Bariess [Bar72] estimates the cost of multiprecision two-step elimination method in terms of single precision multiplication units. He shows that the cost of applying the two-step elimination method to triangularize a matrix of order n is equal to single precision multiplication where

$$c(\varepsilon) = 2n^{4+\varepsilon} (\delta_n/\omega)^{1+\varepsilon} / (2+\varepsilon)(3+\varepsilon)(4+\varepsilon)$$

ω = the number of digits in the single precision computer word

δ_n = the number of digits in the largest component of the original matrix

ε is such that $M(\varepsilon)$ is the cost (interms of single precision multiplication) of the multiplying multiprecision integer of length n.

When classical multiple precision arithmetic is used, $\varepsilon = 1$ so that

$$c(1) = \frac{2n^5}{60} \left(\frac{\delta_n}{\omega} \right)^2$$

For multiplying two m precision integers for large m Schonhage and Strassen [Knu69] gave an asymptotically superior algorithm which requires

$$O(m \log m \log\log m)$$

single precision operations. Hence the two-step elimination method using the fast multiprecision multiplication method would have a cost of at least $O(mn^4 \log m)$. Moreover when the size of the matrix is not that large and the entries are also only

moderately large, even the application of fast multiprecision multiplication method in this two-step method will slow down the process due to considerable overhead.

Thus it seems that the congruential method is superior to two-step method as well as iterative method. Moreover the congruential method computes the adjoint matrix as well as the determinant whereas the iterative method does not compute the determinant or the adjoint.

Although our analysis shows that the congruence method and the iterative method can have the same cost for small n, the implementation of congruence method is more straightforward and simple since almost all the arithmetic operations can be performed in single precision arithmetic. We have an implementation of the congruence method that tested several systems for exact solutions. In the next section we give a number of test results based on our congruence method.

4.4 Test Results.

This section gives a brief summary selected from a number of test problems. In particular we emphasize our results that test Hilbert matrices and random matrices whose entries are uniform integers. There are at least two reasons why we choose to test on Hilbert matrices. These are 1) The integer entries in a Hilbert matrix after suitable scaling is large whereas even when the order of the matrices are 14 or 15 the actual value of the determinant is only a single precision integer. In other words the intermediate results are extremely large compared to actual determinant and hence Hadamard bound is very loose i.e., large primes are invariably selected. However the final determinant value indicates that small single precession primes would have been sufficient and 2) If the system $Ax = b$ where A is Hilbert matrix is solved numerically then the final solution is invariably inexact due to illconditioned nature of the matrix A.

There are two important reasons why we tested our method on random matrices. These are 1) Hadamard bound seems to be close. As a consequence when a matrix is large, say of order 30 while its entries are small, say each 3 digits, " a predetermined set of primes " may not solve the system exactly and 2) It is expected that random matrices are more likely to reflect the goodness of our method of choosing sufficient number of primes.

We compare our MODULAR algorithm with that of Newman [New67]. We recall that Newman selected 10 primes and gave results establishing the ability of these primes to solve

Hilbert matrices of order upto 12. However Newman did not test his primes on random matrices. Hence our comparison is done on two fronts: 1) The ability of the chosen set of primes by our method to solve a system with the ability of the 10 primes chosen by Newman to solve the same system and 2) The actual computing time (in seconds).

We felt that an ideal computing environment i,e, dedicated computing is hard to achieve in multitask batch system. So we decided to use LSI11. For all results below the primes chosen by Newman [New67] are

9999889, 9999901, 9999907, 9999929, 9999931,
9999937, 9999943, 9999971, 9999973, 9999991.

Test set 1. Hilbert matrix

A Hilbert matrix of order n is defined by

$$H_n = \frac{1}{i+j-1}, \qquad 1 \leq i,j \leq n.$$

By suitable scaling we transform the matrix H_n to have integer entries. Since H_n is symmetric we simplify the representation of H_n to its one dimensional representation, e.g,

H_3 after scaling is

$$\begin{bmatrix} 60 & 30 & 20 \\ 30 & 20 & 15 \\ 20 & 15 & 12 \end{bmatrix},$$

which we write

$$H_3 : \begin{bmatrix} 60 \\ 30 \\ 20 \\ 15 \\ 12 \end{bmatrix}.$$

In our result below we do not show H_n explicitly but it can be obtained easily.

1. Solution of $H_9 x = b$; $b = \begin{bmatrix} 27353 \\ 15902 \\ 129549 \\ 104901 \\ 1156475 \\ 1009002 \\ 897570 \\ 1619672 \\ 25114365 \end{bmatrix}$

Value of $\det(H_9) = 1960$.

Primes chosen = { 120671039891, 120671039911, 120671039999,
120671040053, 120671040079 }.

Solution = { 9, -4, 6, -3, 7, -3, 6; 5, 2 }.

Computing time 52.21 (MODULAR) 97.13 (Newman).

2. Solution of $H_{10}\bar{x} = b$; $b = \begin{bmatrix} 9951 \\ 66662 \\ 50064 \\ 528133 \\ 446887 \\ 388446 \\ 688242 \\ 10513609 \\ 9552995 \\ 166404042 \end{bmatrix}$

Value of $\det(H_{10})$ = 49392.

Primes chosen = { 455557513207387, 455557513207393 ,
455557513207411, 455557513207421 ,
45555751320469 }.

Solution = { 3, -4, 6, 9, -7, -4, 6, -5, 3, 2 }.

Computing time 70.06 (MODULAR) 127.22 (Newman).

3. Solution of $H_{11}x = b$, $b = \begin{bmatrix} 48560 \\ 14887 \\ 114060 \\ 90600 \\ 81373 \\ 153338 \\ 2502362 \\ 2419953 \\ 44595098 \\ 43302709 \\ 42067754 \end{bmatrix}$

Value of $\det(H_{11})$ = 1481760.

Primes chosen = { 54855582202943 , 54855582202991 ,
54855582202997 , 54855582203029 ,
54855582203047 , 54855582203131 }.

Solution = { 5, -9, 4, -7, 8, -6, 3, 2, -5, 4, 6 }.

Computing time 106.85 (MODULAR) 162.27 (Newman).

4. Solution of $H_{12}x = b$, $b =$

$$\begin{bmatrix} -243872 \\ -1304379 \\ -709704 \\ -432244 \\ -556973 \\ -6266692 \\ -4172371 \\ -52009476 \\ -32673682 \\ -18597175 \\ -8138652 \\ -5597520 \end{bmatrix}$$

Value of $\det(H_{12}) =$ 5821200.

Primes chosen = { 154527338109037693, 154527338109037783,
154527338109037807, 154527338109037871,
154527338109037909, 154527338109037921 }.

Solution = { -10, -5, 5, 4, -8, 4, 3, 6, 8, -5, -2, 7 }.

Computing time 144.83 (MODULAR) (204.82) (Newman).

5. Solution of $H_{13}x = b$, $b = -$

$$\begin{bmatrix} 3012190 \\ 1740633 \\ 1255182 \\ 1984963 \\ 28079255 \\ 24136174 \\ 403145747 \\ 360294850 \\ 326084458 \\ 298078961 \\ 6317888238 \\ 5861285142 \\ 27342217543 \end{bmatrix}$$

Value of $\det(H_{13})$ = 164656800.

Primes chosen = { 175893435098187127, 175893435098187179,
175893435098187181, 175893435098187217,
175893435098187221, 175893435098187223,
175893435098187227 }.

Solution = { 5, 7, -3, 5, -8, -2, 3, 6, 8, -9, 8, -7, 3 }.

Computing time 209.11 (MODULAR) 253.09 (Newman).

6. Solution of $H_{14}x = b$, $b =$

$$\begin{bmatrix} 1955691 \\ 1058921 \\ 1482711 \\ 19702805 \\ 16371844 \\ 268529666 \\ 237828275 \\ 214493943 \\ 196056125 \\ 4163985429 \\ 3876059150 \\ 18156744780 \\ 17100497644 \\ 48530484434 \end{bmatrix}$$

Value of $\det(H_{14})$ = 336370320.

Primes chosen = { 30372380655898606511, 30372380655898606517,
30372380655898606573, 30372380655898606667,
30372380655898606727, 30372380655898606747,
30372380655898606783 }.

Solution = { 4, 3, 6, -8, -6, 4, -3, -5, -3, 3, 6, 8, 9, -5 },

Computing time 257.15 (MODULAR) 307.72 (Newman).

Test set 2. Random Matrices

We genereted random matrices with entries uniformly distributed integers in a certain range.

1.	Order	5.
	Maximum element	297.
	Determinant value	103847423380.
	Primes chosen	{ 52361, 52363, 5369 }.
	Computing time	8.01 (MODULAR) 29.55 (Newman).
2.	Order	5.
	Maximum element	487.
	Determinant value	2486259045724.
	Primes chosen	{ 297371, 297377, 297391 }.
	Computing time	8.83 (MODULAR) 30.43 (Newman).

3. Order 5.

Maximum element 493.

Determinant value 460363300215.

Primes chosen { 289657, 289669, 289717 }.

Computing time 8.13 (MODULAR) 28.96 (Newman).

4. Order 11.

Maximum element 345.

Determinant value 603958122166041708964399996.

Primes chosen { 274723, 274739, 274751,
274777, 274783, 274787 }.

Computing time 97.91 (MODULAR) 164.63 (Newman).

5. Order 11.

Maximum element 449.

Determinant value 24242954865397543556625395O7.

Primes chosen { 469613, 469627, 469649,
469649, 469657, 469673 }.

Computing time 98.53 (MODULAR) 165.08 (Newman).

6. Order 30.

Maximum element 499.

Determinant value 54542557949188774733464175793562l
384837166912162384394542316032470
85698860037739.

Primes chosen { 2631449, 2631457, 2631467,
2631469, 2631487, 2631493,
2631509, 2631511, 2631523,
2631527, 2631529, 2631533,
2631539, 2631553, 2631581 }.

Computing time 2612.98 (MODULAR) **** (Newman).

Newman [New67] mentions that a number of test problems were run with uniform success with the ten chosen primes. We point out that these ten primes have failed to solve the random system shown in example 6 of Test set 2. It is clear therefore that a fixed set of primes although sufficient to solve systems of equations arising within a specific environment, may fail to guarantee the convergence in some problems. Hence it becomes important to have a set of primes chosen specifically to solye a given system of equations.

CHAPTER FIVE

Normal Forms.

In this chapter we consider the exact solution of a linear system of equations through different normal forms of the integer matrix of the system. Such normal forms are of interest in integer linear programming problems as well as in computational group theory. We shall briefly examine some computational procedures for obtaining such normal forms and suggest two methods for obtaining a normal form of an integer matrix. We shall compare the computational cost of the modular approach discussed in the last chapter with the cost of exact solutions through normal forms.

5.1 Review of Different Normal Forms.

In integer programming problems as well as in computational group theory the coefficient matrix can be transformed into an equivalent matrix by postmultiplying it with a unimodular matrix (whose determinant value is ± 1). Such postmultiplication by a unimodular matrix is equivalent to performing a series of column operations over the ring of integers i.e., 1) adding an integer multiple of one column to another, 2) multiplying a column by -1 , 3) interchanging two columns. If the coefficient matrix is premultiplied by a unimodular matrix then it is equivalent to doing elementary row operations.

Hermite [Her51] showed that every non singular square

matrix can be transformed into a lower triangular matrix using elementary column operations over the ring of integers. More precisely we have

Theorem1 [Her51]

Given an (nxn) integer matrix C of full rank, there exists a (nxn) unimodular matrix K such that CK is a lower triangular matrix with positive diagonal elements. Further, each off diagonal element of CK is nonpositive and strictly less in absolute value than the diagonal element in its row.

The matrix CK is called Hermite Normal Form (HNF). For a given C, Hermite normal form is unique. If elementary row operations are used Smith [Smi61] showed that an integer matrix can be diagonalized.

Two square matrices A and B of order n over a ring R of integers are said to be equivalent if there exist invertible matrices X and Y such that B = XAY. Smith [Smi61] showed that X and Y can be properly chosen so that B is a diagonal matrix.

Theorem2 [Smi61]

Given an (nxn) integer matrix C with full rank, there exist (nxn) unimodular matrices X and Y such that D = XCY is a diagonal matrix with positive diagonal elements such that $d_1|d_2|\ldots|d_n$, where x|y means x divides y .

The matrix S is called Smith Normal Form (SNF) of A. Recently much attention has been given to computing SNF because of its significance in integer programming [Bra71], [Hu70], [G&N72], [K&B79] and in studying the structure of finite abelian groups, which is again related to integer programming problems [Wol69], [H&S79]. The two normal forms are related to one another and Kannan [K&B79] has given computational procedures to obtain SNF through Hermite Normal Form (HNF) and Left Hermite Normal Form (LHNF). We show in section 5.5 how we can get SNF through repeated application of HNF without making use of LHNF. We feel our approach is more uniform and hence can be implemented efficiently.

When HNF is computed according to Kannan's [K&B79] algorithm, large intermediate results do not occur for several cases of input matrix. Hence our interest has been mainly to obtain HNF and hence an exact solution of a system without using modular arithmetic.

However for several random matrices of small order such as (5x5) (e.g., see the matrix in Conclusion) Kannan's [K&B79] method of obtaining HNF causes overflow in intermediate expressions. This is so even when the entries of the (5x5) coefficient matrix are small (≤ 567) i.e., the claim made by Kannan [K&B79] that his proposed method suppresses such overflow does not seem to be true. In this context it is important to investigate a modular method for obtaining HNF as well as SNF. As these are related problems we discuss the feasibility of modular methods for normal forms at the end of the section 5.5.

5.2 Exact Solutions Of Linear System Through HNF.

We will discuss algorithms to compute HNF of an integer matrix in section 5.4. The standard classical algorithm that applies successive column operations produces intermediate numbers that do not appear to be bounded by a polynomial in the length of the input data. These algorithms build up the HNF row by row. Kannan [K&B79] gave a method to obtain HNF of a non-singular square matrix that successively puts the submatrices consisting of first i rows and i columns, $1 \leq i \leq n$, into HNF. In other words at the i-th iteration the principal (ixi) submatrix is in HNF. It was also shown in [K&B79] that this algorithm has intermediate numbers whose number of digits is bounded by a polynomial. We shall review this method later in section 5.5.

Assuming that the HNF of a matrix has been found using this algorithm (see section 5.4) we can solve a system Ax = b where A is (nxn) nonsingular integral matrix and b is (nx1) integral column vector, exactly. Given

$$Ax = b \tag{5.2.1}$$

where A has full rank, we put A^t in HNF.

Let $$C = A^tU \tag{5.2.2}$$

where C is a lower triangular and U is a unimodular matrix. Therefore we have

$$C^tx = U^tAx = U^tb$$

i.e., $C^t x = b'$, where $b' = U^t b$ (5.2.3)

Since C^t is an upper triangular matrix, we obtain x using backward substitution. More formally we do the following.

Algorithm HEXACT

step1 Form HNF(n, A^t).
this produces a lower triangular matrix C and a unimodular matrix U.

step2 Form $b' = U^t b$.

step3 Using backward substitution solve for x from (5.2.3).

The first step of the algorithm HEXACT is done over integers and hence it is exact. The 2nd step computes b' exactly. In order to obtain the solution vector x exactly we will have to use rational arithmetic.

The HNF in step1 can be obtained in $O(n^3+n^2 \log M)$ operations, where M is the maximum absolute value that might arise in the matrix during the execution of the algorithm. In step2 of HEXACT we do n^2 multiplication and in step3 we do $O(n^2)$ rational arithmetic operations. Therefore the exact solution can be obtained in $O(n^3+n^2 \log M)$ operations. Note that many of the operations involve GCD calculations. It is shown in Kannan [K&B79] that

$$M \leq 2^{3n}\, n^{20n^3}\, M_1^{12n^3}$$

where M_1 is the maximum absolute value in the original matrix

A. Hence the cost of the algorithm is $O(n^5 \log n)$.

Thus it seems that our modular method for exact solutions is superior to obtaining HNF and then solving the system exactly. However we remark that such comparisons based on worst case bounds do not completely reflect the performance of the algorithms in practice.

The practicality of the method depends to a large extent on the size of the intermediate numbers as well as the value of determinant. We tested the same set of systems using HEXACT. Our results given below reveal that HEXACT is superior to MODULAR for all the examples tested (here also timings are given in seconds). In all examples tested the size of the intermediate numbers become large when HEXACT is applied; in fact much larger than the size of the intermediate numbers produced by MODULAR when large primes are used. Hence we implemented our program using ALGEB language (designed and implemented by David Ford to support extended precision integer arithmetic) in LSI11.

Solution of $H_9x = b$.

Computing time 24.75 (HEXACT) 52.21 (MODULAR).

Solution of $H_{10}x = b$.

Computing time 34.7 (HEXACT) 70.06 (MODULAR).

Solution of $H_{11}x = b$.

Computing time 7.3 (HEXACT) 106.85 (MODULAR).

Solution of $H_{12}x = b$.

Computing time 63.31 (HEXACT) 144.83 (MODULAR).

Solution of $H_{13}x = b$.

Computing time 83.45 (HEXACT) 209.11 (MODULAR).

Solution of $H_{14}x = b$.

Computing time 107.28 (HEXACT) 257.15 (MODULAR).

Test results of random matrices are given below. Ra(i) denotes system i of random matrix used in Test set 2.

Solution of Ra(1).

Computing time 4.58 (HEXACT) 8.01 (MODULAR).

Solution of Ra(2).

Computing time 4.58 (HEXACT) 8.83 (MODULAR).

Solution of Ra(3).

Computing time 4.41 (HEXACT) 8.13 (MODULAR).

Solution of Ra(4).

Computing time 51.75 (HEXACT) 97.91 (MODULAR).

Solution of Ra(5).

Computing time 52.01 (HEXACT) 98.53 (MODULAR).

5.3 Standard Procedure Of Computing Smith Normal Form (SNF).

Let A be a non-singular matrix which is to be transformed into Smith Normal Form S. Let S = XAY, where X and Y are two unimodular matrices. Let $A_{k(c)}$ denote the k-th column of A and $A_{k(r)}$ denote k-th row of A. The following steps will illustrate how Smith Normal Form can be obtained using elementary row and column operations.

Algorithm STANDARD.

step1 t <-- 0. ((n-t)x(n-t) matrix must be put in SNF)

step2 t <-- t+1; if t=n terminate.

step3 Find the smallest non-zero element in absolute value in the submatrix (n-t+1)x(n-t+1) of A. Let a_{ij} be the smallest value. Then interchange i-th row with t-th row of A and X and j-th column with t-th column of A and Y (now a_{tt} is the smallest non-zero element in the matrix A).

step4 If $a_{tt} | a_{tj}$, j=t+1,...,n then do step5; otherwise for some j=k (say) a_{tt} does not divide a_{tk}. Let $a_{tk} = qa_{tt} + r$ where q and r are the quotient and the remainder after division of a_{tk} by a_{tt}, $0 < r < a_{tt}$. Replace $A_{k(c)}$ and $Y_{k(c)}$ by $A_{k(c)} - qA_{t(c)}$ and $Y_{k(c)} - qY_{t(c)}$.
Return to step3.

step5 If $a_{tt} | a_{it}$, $i=t+1,\ldots,n$ then do step6; otherwise for some $i=k$ (say) a_{tt} does not divide a_{kt}. Let $a_{kt} = qa_{tt} + r$, where q and r are the quotient and the remainder after division of a_{kt} by a_{tt}, $0 < r < a_{tt}$. Replace $A_{k(r)}$ and $X_{k(r)}$ by $A_{k(r)} - qA_{t(r)}$ and $X_{k(r)} - qX_{t(r)}$.
Repeat from step3.

step6 At this stage $a_{tt} | a_{tj}$, $j=t+1,\ldots,n$, $a_{tt} | a_{it}$, $i=t+1,\ldots,n$. Let $a_{tj} = q_j a_{tt}$ for $j=t+1,\ldots,n$ and $a_{it} = p_i a_{tt}$ for $i=t+1,\ldots,n$ then replace $A_{j(c)}$, $A_{i(r)}$, $Y_{j(c)}$ and $X_{i(r)}$ as follows

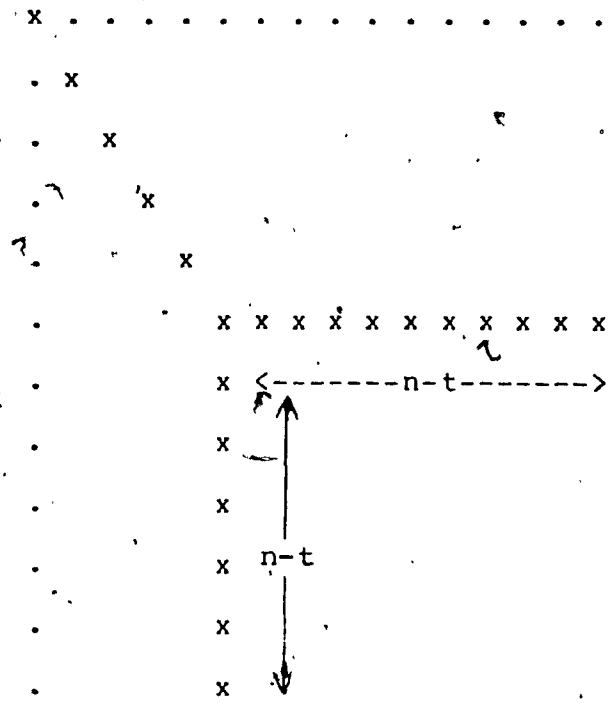
$$A_{j(c)} \leftarrow A_{j(c)} - q_j A_{t(c)} \qquad j=t+1,\ldots,n,$$

$$A_{i(r)} \leftarrow A_{i(r)} - s_i A_{t(r)} \qquad i=t+1,\ldots,n,$$

$$Y_{j(c)} \leftarrow Y_{j(c)} - q_j Y_{t(c)} \qquad j=t+1,\ldots,n,$$

$$X_{i(r)} \leftarrow X_{i(r)} - s_i X_{t(r)} \qquad i=t+1,\ldots,n.$$

Now the matrix is of the form



step7 If $a_{tt} | a_{ij}$, $i,j=t+1,\ldots,n$ then return to step2.

step8 If a_{tt} does not divide a_{ij} for some i,j then let $a_{ij} = qa_{tt} + r$, $0<r<a_{tt}$ and interchange i-th row and j-th column as follows:

$$A_{i(r)} = A_{i(r)} + qA_{t(r)},$$

$$A_{j(c)} = A_{j(c)} - A_{t(c)},$$

$$X_{i(r)} = X_{i(r)} + qX_{t(r)},$$

$$Y_{j(c)} = Y_{j(c)} - Y_{t(c)}.$$

Return to step3.

Complexity of STANDARD Algorithm.

Let us analyze the standard algorithm by assuming M as the largest absolute value in the matrix during intermediate computations. It is not known whether the number of digits in M is exponential or bounded by a polynomial.

Let c denote the operation of comparison of two integer numbers, e denote the operation of checking divisibility, s denote the operation of subtraction of a multiple of one number from another number and f denote the operation of interchanging the position of two numbers. Clearly these are the basic operations involved in algorithm STANDARD.

Then for a matrix of size i we need $i^2c + 2if$ operations in step3, $i(e+s+f)$ operations in both step4 and step5, $2i(i-1)s$ operations in step6, $(i-1)^2e + 2i(s+f)$ operations in step7 and step8 combined during one iteration of the loop. After at most logM iterations of step3 through step8, a_{tt} where $t=n-i+1$ will divide a_{kj}, $k,j=i+1,\ldots,n$ or a_{tt} will be reduced to 1. Hence to reduce a matrix in SNF we need

$$\log M\{\sum_{i=1}^{n-1} \{(i^2c+2if)+2i(e+s+f) + 2i(i-1)s + (i-1)^2e + 2i(s+f)\}.$$

For large n we retain only the leading terms of the above expression and obtain

$$\log M(n^3\frac{c}{3} + 3n^2f + n^3\frac{e}{3} + 2n^3\frac{s}{3})$$

as an estimate on the number of operations.

Note that s includes a multiplication and a subtraction, e is a division. Hence the cost is $O(n^3 \log M)$ where multiplication/division cost is only accounted.

As remarked earlier M can become very large even for small n. In algorithm STANDARD we repeatedly do step3 to step8 until the next diagonal element is either reduced to 1 or it divides every other element in the submatrix i.e., for submatrix of size i in the worst case i^2 divisions might be necessary. Hu [Hu70] first proposed a different procedure which first diagonalizes an integer matrix and then checks for divisibility among the diagonal elements. The formal description of the modified algorithm is given below.

Hu's Algorithm For Computing Smith Normal Form.

The following algorithm reduces every entry modulo the value of the determinant d during the computations. We shall remark later that such a reduction may not give the Smith Normal Form in all cases.

Because of reduction mod d, the maximum entry in the matrix at any time during the computation is d-1. Hence at most $\log_2 d$ iterations need to be done in the following algorithm.

Algorithm Hu.

step1 Find the smallest element in absolute value within 1st row and 1st column. If a_{1j} is the smallest element then interchange j-th column with 1st column.

If a_{j1} is the smallest element then interchange j-th row with 1st row. (now a_{11} is the smallest non-zero element in the matrix A).

step2 Same as step4 of standard procedure.

step3 Same as step5 of standard procedure.

step4 Same as step6 of standard procedure.

step5 Repeat steps 1 through 4 on the submatrix of order one less i.e.,

$$\begin{bmatrix} a_{22} \cdots\cdots\cdots\cdots a_{2n} \\ \cdots\cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots\cdots \\ a_{n2} \cdots\cdots\cdots\cdots a_{nn} \end{bmatrix}$$

and this process continues until the matrix becomes diagonal.

step6 At this stage the matrix is diagonal. Let $d_1, d_2, \ldots, d_n$ be the diagonal elements.
If $d_1 | d_i$, $i=2,\ldots,n$ then check $d_2 | d_i$, $i=3,\ldots,n$ and so on until $d_{n-1} | d_n$; then terminate the algorithm.
If for some j and k, $j<k$, d_j does not divide d_k then replace j-th row by

$$A_{j(r)} = A_{j(r)} + A_{k(r)}.$$

Repeat diagonalization process (similar to steps 1 to 5) for the submatrix consisting of the last n-j+1 row and n-j+1 columns. Then repeat from step6.

Complexity Of Hu's Algorithm.

For the first iteration the algorithm needs the following operations assuming the size of the matrix is n.

step1 needs	2(n-1)c + nf	operations.	
step2 needs	(n-1)e + ns + nf	operations.	(5.3.1)
step3 needs	(n-1)e + ns + nf	operations.	
step4 needs	2n(n-1)s	operations.	

Hence the number of operations needed to diagonalize the first time is

$$T = n(n-1)c + \frac{3}{2}n(n-1)f + \frac{1}{3}n(n+1)(2n+1)s + n(n-1)e$$

Step6 of algorithm Hu can be applied by considering 2x2 matrix

$$\begin{bmatrix} d_i & 0 \\ 0 & d_j \end{bmatrix}$$

where d_i does not divide d_j, $i<j$.

Hence the number of operations of one iteration of step1 to step4 within one iteration to be done in step6 is (with n=2 in 5.3.1)

$$R = 2c+2f+d+2s+2f+d+2s+2f+4s.$$

This is in addition to a maximum of $\frac{n(n-1)}{2}$ divisions that finds i,j such that d_i does not divide d_j. Hence for one iteration of step6 the total number of operations is

$$n\{ R + \frac{n(n-1)}{2} \}.$$

However the total number of iterations is at most $\log_2 d$. Therefore the total number of operations is at most

$$T + n\log_2 d \{ R + \frac{n(n-1)}{2} \}.$$

5.4 HNF Algorithm.

In transforming an integer matrix to SNF using any of the algorithms described earlier, the number of digits in the maximum intermediate values produced does not appear to be bounded by a polynomial in the input data. This was first pointed out by Frumkin [Fru77]. In order to eliminate intermediate swelling of numbers, Wolsey [Wol69], Hu [Hu70] and Frumkin [Fru77] have suggested that the Smith Normal Form of an integer matrix A be computed modulo d where d is the absolute value of the determinant of the matrix A. However this is not always valid as the following two examples illustrate.

Example 5.4.1. (This example works with modulo d, d=64)

$$\begin{bmatrix} 2 & 4 & 6 \\ 4 & 12 & 20 \\ 6 & 20 & 42 \end{bmatrix}.$$

Having done all the computations modulo d we get the SNF as

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 8 \end{bmatrix}$$

and the actual Smith Normal Form is also

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 8 \end{bmatrix}.$$

Example 5.4.2.

(This example does not work with modulo d, d=127)

$$\begin{bmatrix} 3 & 4 & 5 \\ 12 & 13 & 19 \\ 17 & 1 & 9 \end{bmatrix}.$$

Having done all the calculation modulo d we get the SNF as

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

but the actual Smith Normal Form of the example is

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 127 \end{bmatrix}.$$

Kannan [K&B79] gives an algorithm to compute the Smith Normal Form (SNF) using Hermite Normal Form (HNF) and Left Hermite Normal Form (LHNF). It is shown in Kannan [K&B79] that all intermediate numbers produced by his algorithm remain bounded by a polynomial, in the length of input data. The algorithm given below is adopted from Kannan [K&B79] and it computes the Hermite Normal Form of successive (ixi) submatrices as opposed to building the Hermite Normal Form row by row. As before we denote $A_{k(r)}$ and $A_{k(c)}$ the k-th row and k-th column of A respectively.

Algorithm HNF.

step1 i<--0.

step2 i<--i+1 ; if i=n terminate.

step3 For j=1,...,i do the following:

3.1) r <-- $(a_{jj}, a_{j,i+1})$; Find p and q such that

$$r = pa_{jj} + qa_{j,i+1}.$$

3.2) Form D as

$$D \leftarrow \begin{bmatrix} p & -a_{j,i+1}/r \\ q & a_{jj}/r \end{bmatrix}$$

3.3) Form E with j-th and (i+1)-th column of A i.e.,

$$E \leftarrow (A_{j(c)}, A_{i+1(c)}).$$

3.4) F <-- ED.

3.5) Replace j-th column of A by first column of F and replace (i+1)-th column of A by second column of F i.e., $A_{j(c)} \leftarrow F_{1(c)}$, $A_{i+1(c)} \leftarrow F_{2(c)}$.

3.6) If j>1 then Call Reduceoffdiagonal(j,A).

step4 Call Reduceoffdiagonal(i+1,A).

step5 Repeat from step2.

Algorithm Reduceoffdiagonal(k,A).

step1 If $a_{kk}<0$ then $A_{k(c)} \leftarrow\!- (-A_{k(c)})$.

step2 For $z=1,\ldots,k-1$ do the following:

$$\text{factor} \leftarrow\!- \lceil a_{kz}/a_{kk} \rceil,$$

$$A_{z(c)} \leftarrow\!- A_{z(c)} - \text{factor } A_{k(c)}.$$

The algorithm HNF can be modified to work on any (mxn) matrix with full row rank m. It will then produce the normal form (H,0) where H is (mxm) lower triangular matrix and the 0 is mx(n-m) zero matrix. If we use the modified algorithm on any (mxm) integer matrix A with full column rank and perform row instead of column operations then A will be transformed into a Left Hermite Normal Form (LHNF) $\binom{H}{0}$ where H is upper triangular with positive diagonal elements. Both the algorithms HNF and LHNF can be combined repeatedly to produce the Smith Normal Form of an integer matrix. In such an algorithm the bottom right (n-i)x(n-i) square matrix is transformed into HNF frequently. For details see in Kannan [K&B79].

5.5 Algorithm For Computing Smith Normal Form.

We propose a method for computing the SNF of an integer matrix which is based on repeated computation of HNF of principal submatrices of the original matrix. We need the following basic result for the development of the algorithm. Let A be a nonsingular square matrix. For some i, $0 \leq i \leq n$ we can write the Hermite Normal Form of A as

$$A_i = \begin{bmatrix} D_i & X \\ Y & E_{n-} \end{bmatrix},$$

where

D_i is (ixi) diagonal matrix,

E_{n-i} is (n-i)x(n-i) HNF of A,

X is ix(n-i) zero matrix and

Y is (n-i)xi zero matrix.

Note that if the HNF of A has no principal diagonal submatrix then we can take i=0 in the above representation and our definition yields $A_o = E_n = HNF(A)$. It is not difficult to show that

$$HNF(A_i^t) = \begin{bmatrix} D_i & 0 \\ 0 & HNF[(E_{n-i})^t] \end{bmatrix}$$

For, in applying algorithm HNF to A_i^t we do steps 4 to 6 repeadly (see SNF algorithm below).

Steps 3.1 to 3.5 of algorithm HNF do not change the matrix D_i as well as X. When step3.6 is called, propagation of nonzero elements in lower part of D_i and in X could happen if $a_{kz}/a_{kk} > 0$ i.e., if $a_{kz} > 0$. But $z \leq k-1$ and so a_{kz} always remains zero. Hence there will be no propagation of non zero values i.e., the step

$$A_{z(c)} \leftarrow\!- \; A_{z(c)} - \left\lceil a_{kz}/a_{kk} \right\rceil A_{k(c)}$$

will not change $A_{z(c)}$. Hence it is sufficient to apply the algorithm HNF to $(E_{n-i})^t$ in order to obtain the HNF of A_i^t.

If after we compute HNF$[(E_{n-i})^t]$ the result has a nontrivial principal diagonal submatrix then the HNF of the original matrix has a diagonal submatrix of order at least i+1. This implies that one can obtain a diagonal matrix of full rank by repeatedly doing HNF on the lower right non-diagonal submatrix of the original matrix. We make use of this result in our algorithm below.

Algorithm SNF.

step1 Do HNF on A and call the resulting lower triangular matrix A_o (assuming A_o has no diagonal submatrix).
i <-- 0. { (ixi) diagonal submatrix }

step2 If i=n terminate. Let

$$A_i = \begin{bmatrix} D_i & 0 \\ 0 & E_{n-i} \end{bmatrix}$$

step3 Set B <-- E_{n-i}.

step4 Do HNF(B^t) and call the result C.

step5 If $c_{11} | c_{j1}$, j=2,...,n-i then write

$$D_{i+1} = \begin{bmatrix} D_i & 0 \\ 0 & c_{11} \end{bmatrix}$$

and E_{n-i-1} is the principal (n-i-1)x(n-i-1) submatrix of C. Now set i <-- i+1, return to step2.

step6 Now c_{11} does not divide c_{j1} for some j, $2 \leq j \leq n-i$ set B <-- C and return to step4.

Analysis Of SNF Algorithm.

Let w_i be the amount of work required to obtain an (i+1)x(i+1) principal diagonal from an (ixi) principal diagonal i.e., in the above algorithm we do steps 2,3,4 and 5 w_i times to obtain an (i+1)x(i+1) principal diagonal from an (ixi) principal diagonal.

In the worst case we can have all the diagonal elements of

D_i equal to 1. Now the maximum absolute value of any entry in E_{n-i} is d, where d is the absolute value of the determinant of the matrix A. Hence at most logd iterations of step4 and step6 are to be done before we obtain D_{i+1} from D_i. Since step4 cost dominates step6 cost, we have

$$w_i = \text{logd(cost of doing HNF in step4)}.$$

Therefore the total cost is

$$\sum_{i=0}^{n-2} w_i = \sum_{i=0}^{n-2} \text{logd(cost of HNF of } (n-i)\text{x}(n-i)).$$

We itemize the cost in step4 of doing HNF. Here we do HNF of an (n-i)x(n-i) matrix. Each call of HNF requires the following operations:

1)	(n-i)(n-i-1)/2	GCD calls.
2)	(n-i)(n-i-1)/2	2x2 matrix multiplications.
3)	(n-i)(n-i-1)/2	Reduceoffdiagonal calls.

Each matrix multiplication requires 8 multiplications; each GCD call requires a maximum of logd division/multiplications operations and each call to Reduceoffdiagonal requires n multiplications and 1 division. Hence the cost of doing HNF of an (n-i)x(n-i) matrix is at most

$$\frac{1}{2}(n-i)(n-i-1)[\text{logd}+8+n+1].$$

Therefore the total cost of doing SNF algorithm is at most

$$\sum_{i=0}^{n-2} \frac{1}{2}(n-i)(n-i-1)[\text{logd}+8+n+1]\text{logd} = O(n^4\text{logd}).$$

CHAPTER SIX

Smith Normal Form Using Modular Arithmetic.

In this chapter we give a method to obtain the SNF of an integral matrix using modular arithmetic. Hu [Hu70] first proposed obtaining SNF of an integral matrix using modulo d (absolute value of the determinant) arithmetic. Unfortunately such a method does not work in all cases. Borosh [B&F66], Cabay [C&L77] and the modified congruence method suggested by us in this thesis avoid overflow in the intermediate computations in finding an exact solution of an integral system of linear equations. Havas [H&S79] has commented that it does not seem possible to compute SNF using an analogous congruence method. He has given a method to compute SNF for several prime power modulii. However his method demands the calculation of the determinant of the given matrix. A brief outline of his method follows.

6.1 Study Of Different SNF Algorithms.

Algorithm HAV. [Havas]

step1. Calculate the determinant of the matrix d.

step2. Factorize $d = \prod_{p|d} p^{e(p)}$.

step3. For each prime p in the factorization of d, find the

p-primary invariants by computing the Smith Normal Form modulo $p^{e(p)+1}$.

step4. Construct the SNF using the Chinese Remainder Theorem.

We have the following comments on this algorithm: 1) Computing a determinant directly is equivalent to solving a system directly and hence all computational difficulties inherent in the exact solution of a system of linear equations is inherent in computing a determinant. 2) When the value of determinant is large one has to factorize, which is a nontrivial procedure and then for each prime power in such a factorization a complete reduction of original matrix has to be done. 3) This method does not seem to work in all cases. See example 6.2.2.

Rayward-Smith [Ray79] has given a method that avoids the factorization of the determinant and hence avoids the repeated reduction of the original matrix modulo prime powers. Instead he has suggested finding the prime factors in the factorization of the diagonal elements i.e., his proposal is to use Hu's method to obtain a diagonal matrix and then get the prime power factors of each diagonal element. From these factors the SNF can be built up. More formally his algorithm is as follows:

Algorithm RS. [Rayward-Smith]

step1 Produce a diagonal matrix for a given matrix A – Let the diagonal matrix be $(x_1,\ldots,x_n)$.

step2 The invariant factors can be calculated directly from the diagonal entries $(x_1,\ldots,x_n)$ as follows:

Let $p_1<p_2\ldots<p_m$ be all prime factors of elements of diagonal matrix, so p_m is the largest prime dividing any of $x_1,x_2,\ldots,x_n$. Since $d = x_1x_2\ldots x_n$, it follows that $p_1,\ldots,p_m$ are distinct prime factors of d.

For each p_j, $j=1,\ldots,m$, let b_{jk} be the largest power to which p_j can be raised such that the result will be a factor of x_k, $k=1,\ldots,n$. This associates with each p_j a vector

$$b_j = \begin{bmatrix} b_{j1} \\ b_{j2} \\ . \\ . \\ . \\ b_{jn} \end{bmatrix}$$

Hence $x_k = \prod_{k=1}^{m} (p_j)^{b_{jk}}$, $k=1,\ldots,n$.

If the entries in b_j are ordered in ascending numerical order a new vector

$$c_j = \begin{bmatrix} c_{j1} \\ c_{j2} \\ \cdot \\ \cdot \\ \cdot \\ c_{jn} \end{bmatrix}$$

is formed, where $c_{j1} \leq c_{j2} \leq \ldots \leq c_{jn}$. The i-th invariant factor is given by

$$d_i = \prod_{j=1}^{m} (p_j)^{c_{ji}}.$$

Example 6.1.1.

Assume the first step has been done and we have the following diagonal matrix

$$\begin{bmatrix} 5 & 0 & 0 \\ 0 & 12 & 0 \\ 0 & 0 & 10 \end{bmatrix}$$

Suppose $m=3$ and $p_1=2$, $p_2=3$ and $p_3=5$, so three vectors are produced

$$b_1 = \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix}, \quad b_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad b_3 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

This yields

$$c_1 = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}, \quad c_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad c_3 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}.$$

Hence $d_1 = 2^0 3^0 5^0 = 1$, $d_2 = 2^1 3^0 5^1 = 10$, $d_3 = 2^2 3^1 5^1 = 60$.

The method does not always produce a correct form (see example 5.4.2) when diagonalization is done through step1 to step5 according to Hu's algorithm. If modulo d is not used then overflow problem must be tackled. Hence our conclusion is that none of these methods will perform satisfactorily in general.

Example 6.2.2.

$$A = \begin{bmatrix} 109 & 481 & 480 \\ 423 & 1866 & 2361 \\ 536 & 2363 & 2361 \end{bmatrix},$$

where determinant of matrix A is 18.

But the correct SNF of A is $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 6 \end{bmatrix}$.

Using algorithm HAV., Smith Normal Form of A with respect to (mod 2^2) and (mod 3^3) are

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{bmatrix}.$$

Applying the Chinese Remainder Theorem on the components of above two diagonal matrix we get

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 57 & 0 \\ 0 & 0 & 30 \end{bmatrix},$$

which is not the correct Smith Normal Form of A.

6.2 Smith Normal Form: A Heuristic Approach.

Our approach consists of two stages. In the first stage we obtain several diagonal forms of the given matrix modulo several suitably chosen primes. We can construct a diagonal form of the original matrix using the Chinese Remainder Theorem. If this diagonal matrix is denoted by $(d_1,\ldots,d_n)$ then we can replace any two diagonal elements d_i, d_j, $i\neq j$ by their GCD and LCM respectively. By such successive replacements we obtain a diagonal matrix in which $d_i | d_{i+1}$ i.e., we have a matrix in SNF.

Let A be an nxn nonsingular matrix and p be a prime. The following algorithm is a slight modification of the diagonalization procedure given in chapter 3.

Algorithm H

step1 a_{ij} <-- a_{ij} mod p , $1 \leq i,j \leq n$.

step2 k <-- 1 ; Temp <-- 1

step3 Find r, $k \leq r \leq n$, such that $(a_{rk},p) = 1$.

step4 If r=k then repeat from step6 otherwise from step5.

step5 Interchange rows r and k and replace Temp <-- (-Temp).

step6 Find the inverse a'_{kk} of a_{kk} such that $a_{kk}a'_{kk} \equiv 1 \bmod p$.

step7 Temp <-- a_{kk} mod p.

a_{kt} <-- $a_{kt}a_{kk}$ mod p, $k \leq t \leq n+1$.

step8 For $1 \leq s \leq n$, $s \neq k$ replace a_{st} by

a_{st} <-- $a_{st} - a_{sk}a_{kt}$ mod p, $k \leq t \leq n+1$.

step9 a_{kk} <-- Temp.

step10 If $k = n$ then terminate otherwise repeat from step3.

If we choose t primes $p_1,\ldots,p_t$ then we can have t sets of diagonal matrices. Let B be an nxt matrix such that the j-th column of B is the diagonal matrix obtained by the above algorithm with prime p_j. Below we show how to get one diagonal matrix using the Chinese Remainder Theorem.

step1 P <-- $p_1p_2\ldots p_t$.

step2 Find the inverse Q_i of r_i with respect to p_i where $r_i = \frac{P}{p_i}$, $i=1,\ldots,t$.

step3 For $i=1,\ldots,n$ do the following:

$$Diag_i \text{<--} \sum_{j=1}^{t} r_jQ_jB_{ij} \text{ mod } P.$$

Next step is to compute the SNF from the diagonal matrix which we have obtained. We use the following lemma.

Lemma 6.2.1.

The matrix diag{...,m,...,n,...} is equivalent to diag{...,(m,n),...,[m,n],...} i.e., any two non zero elements of a diagonal matrix may be replaced by their greatest common divisor and least common multiple.

Proof

(See [Smi66]).

Ideally we expect to get the SNF by applying Lemma 6.2.1 on the diagonal matrix. But it does not work in general.

Example 6.2.1.

Let us consider an 4x4 Hilbert matrix with suitable scaling

$$\begin{bmatrix} 12 & 6 & 4 & 3 \\ 30 & 20 & 15 & 12 \\ 20 & 15 & 12 & 10 \\ 105 & 84 & 70 & 60 \end{bmatrix}.$$

We get the correct SNF using algorithm H with primes 37 and 47. After diagonalization of the above matrix using the heuristic approach with respect to prime 37 the diagonal form is [12,5,25,2] and with respect to prime 47 the diagonal form is [12,5,16,26]. Now reconstructing the new diagonal matrix of the above two using the CRT we have [12,5,580,261]. Now applying lemma 6.2.1 on each of the resultant diagonal elements one can

get [1,1,1,3] which in fact is the correct SNF of the 4x4 Hilbert matrix.

However algorithm H fails to obtain the correct SNF with primes 47 and 79. After diagonalization of the above matrix using the heuristic approach with respect to prime 47 the diagonal form is [12,5,16,26] and with respect to prime 79 the diagonal form is [12,5,53,12]. Now reconstructing the new diagonal matrix of the above two diagonal forms using the CRT we get [12,5,1238,1671]. Now applying lemma 6.2.1 on each of the resultant diagonal elements one gets [1,1,2,1858] which is not a correct SNF of the 4x4 Hilbert matrix.

We applied algorithm H and then lemma 6.2.1 on several random matrices. In several cases we were able to obtain the correct SNF; however for several other examples we were unable to obtain the correct SNF. Moreover our observation reveals that for a selected set of primes the SNF can be obtained using modular arithmetic (as it is seen in example 6.2.1). We give several examples below and for each example we give a set of primes for which algorithm H gives the correct SNF.

We give below the Hilbert matrix (of order 3 to 12) and the corresponding prime(s) for which correct SNF are obtained.

matrix	prime(s)	SNF
H_3	7	[1,1,2]
H_4	11	[1,1,1,3]
H_5	13,17	[1,1,1,1,12]
H_6	17,19,23	[1,1,1,1,1,20]
H_7	17,19,23	[1,1,1,1,1,1,300]
H_8	17,19,23	[1,1,1,1,1,1,1,525]
H_9	23,29,31	[1,1,1,1,1,1,1,1,1960]
H_{10}	59,61,67	[1,1,1,1,1,1,1,1,1,49392]
H_{11}	37,41,47,53	[1,1,1,1,1,1,1,1,1,1,1481760]
H_{12}	89,97,101,103	[1,1,1,1,1,1,1,1,1,1,1,5821200]

Here we give several random matrices and the corresponding primes for which we can obtain correct SNF. We denote RMi(j) as the i-th order and j-th kind random matrix and those can be seen in Appendix 2.

RMi(j)	primes	SNF
RM3(1)	5,7,11	[2,4,8]
RM4(1)	5,7,11,13	[1,3,6,8]
RM5(1)	7,11	[1,1,1,1,23]
RM6(1)	11,13	[1,1,1,1,3,15]
RM7(1)	17,19,23	[1,1,2,4,8,16,32]
RM8(1)	7,11,13	[1,1,1,1,1,5,5,10]

Here we give some other random matrices and the corresponding primes for which we are not getting correct SNF whereas the correct SNF should be the same as before.

RMi(j)	primes	SNF
RM3(2)	5,7,11	[1,4,226]
RM4(2)	5,7,11,13	[1,1,1,324]
RM5(2)	7,11	[1,1,1,1,65]
RM6(2)	11,13	[1,1,1,1,14,84]
RM7(2)	17,19,23	[1,1,1,2,1,1,1526]
RM8(2)	7,11,13	[1,1,1,1,1,1,1,107]

Thus we conclude that finding a proper set of primes to obtain the SNF of a matrix using modular method is still an open problem.

Conclusion

During the last ten to twenty years there has been a growing interest in computational aspects of algorithms to transform a matrix to a canonical form. A closely related problem of interest is that of finding an exact particular solution of a system of linear diophantine equations in integers. Many of the known algorithms for finding both general and particular solutions of a system of linear diophantine equations have been based on Hermite Normal Form. However in computing by Hermite's method the absolute value of the intermediate results can reach $(2)^{2m}$ where m is the number of equations.

In order to avoid swell in the intermediate expression Gauss elimination algorithm may be done over a finite field (arithmetic being modulo a prime). In chapters three and four we have shown how as few primes as are necessary may be chosen to obtain an exact solution of a system. The main conclusion is that just sufficient primes can be chosen depending upon the following: size of the matrix, the norm of the matrix as well as the wordsize of the particular machine on which the problem is to be solved.

Since the problem of finding an exact solution of linear system is related to obtaining Hermite Normal Form, we adapted a method due to Kannan [K&B77] to obtain the HNF of a matrix and then solve the given system exactly by back substitution. It seems from the examples tested by us that the problem of overflow is still inherent and not completely eliminated. To

quote one example, the algorithm [K&B79] produces very large intermediate numbers when applied to the matrix

$$\begin{bmatrix} 34 & 543 & 245 & 239 & 65 \\ 23 & 56 & 567 & 54 & 32 \\ 123 & 234 & 345 & 456 & 567 \\ 43 & 54 & 65 & 457 & 89 \\ 432 & 321 & 213 & 87 & 98 \end{bmatrix}.$$

Hence an attempt was made to obtain the canonical forms (in particular Smith Normal Form) using modular arithmetic. We remark that the method suggested by Hu [Hu70], Frumkin [Fru77] and Havas [H&S79] fail on several practical problems. Our results are not conclusive either. We have given examples where such congruence methods will indeed find Smith Normal Form correctly. However more extensive and indepth study into congruence methods for obtaining Hermite and Smith Normal Forms are necessary. Through extensive testing and search we have been able to find the set of primes to be used in congruence method related to finding the SNF of some special matrices.

Appendix 1

Test of Primality

We compute the prime numbers using probabilistic algorithm of Rabin [Rab76]. To test whether a number x is a prime, choose m random numbers $1 \leq b_1 \leq \ldots \leq b_m < x$. For any x and e>0 if $\log(\frac{1}{e}) \leq m$ then x would be a prime number with probability greater than (1-e).

Algorithm FINDPRIME. [Rabin]

step1 i <-- 0.

step2 i <-- i+1 ; if i = m+1 terminate.

step3 If $b^{n-1} \neq 1 \bmod x$ then FLAG1=true.

step4 If { $\frac{x-1}{2^i} = m$ is integral,

$1 < (b^m-1,x) < x$} then FLAG2=true.

step5 If FLAG1 or FLAG2 are true then (x is a composite number) terminate otherwise repeat from step2.

In this algorithm the probability of error that x be prime is smaller than $1/2^m$. So if one uses m > 25 there is a small chance for an error in the decision that x is a prime. In our test we chose m=100.

The main difficulty is in step3 i.e., to find b^{x-1} when x is large. Now if b and x fit in one computer word then step3 can be done very fast using binary exponentiation.

Now in order to do step4 we do the following:

We use the following method to find $GCD(b^m-1,x)$:

a) If $mod(b^m,x) = 1$ then $GCD(b_m,x) = n$.

b) If $mod(b^m,x) = 0$ then $mod(b^m-1,x) = x-1$.

Now $GCD(b^m-1,x) = GCD(x-1,x) = 1$.

c) If a) and b) do not happen

Compute $y = mod(b^m,x)$.

Now $GCD(b^m,x) = GCD(y-1,x)$.

Appendix 2

Random Matrices

RM3[1]:

$$\begin{bmatrix} 2 & 6 & 6 \\ 6 & 22 & 30 \\ 4 & 24 & 56 \end{bmatrix}$$

RM4[1]:

$$\begin{bmatrix} 1 & 3 & 3 & 4 \\ 3 & 12 & 18 & 33 \\ 2 & 15 & 39 & 95 \\ 8 & 36 & 78 & 206 \end{bmatrix}$$

RM5[1]:

$$\begin{bmatrix} 1 & 3 & 3 & 4 & 5 \\ 3 & 10 & 12 & 19 & 23 \\ 2 & 9 & 16 & 33 & 38 \\ 8 & 28 & 39 & 73 & 87 \\ 5 & 21 & 36 & 76 & 114 \end{bmatrix}$$

RM6[1]:

$$\begin{bmatrix} 1 & 3 & 3 & 4 & 5 & 6 \\ 3 & 10 & 12 & 19 & 23 & 22 \\ 2 & 9 & 16 & 33 & 38 & 29 \\ 8 & 28 & 39 & 73 & 87 & 83 \\ 5 & 21 & 36 & 76 & 94 & 95 \\ 9 & 30 & 43 & 90 & 121 & 190 \end{bmatrix}$$

RM7[1]:

1	3	3	4	5	6	7
3	10	12	19	23	22	24
2	9	17	37	42	34	39
8	28	42	88	108	110	140
5	21	39	94	129	164	165
9	30	50	133	209	376	416
5	19	45	148	245	584	1031

RM8[1]:

1	3	3	4	5	6	7	8
3	10	12	19	23	22	24	26
2	9	16	33	38	29	31	25
8	28	39	73	87	83	98	83
5	21	36	76	192	83	91	69
9	30	43	90	115	144	184	151
5	19	36	91	118	188	354	400
5	21	36	78	102	145	266	454

RM3[2]:

146	644	642
566	2496	2490
716	3156	3152

RM4[2]:

613	2785	3216	4422
2817	12810	14859	20547
4694	21371	24933	34725
10544	47984	55860	77586

RM5[2]:

625	2985	3698	5036	9446
2861	13670	16955	23114	43307
4701	22473	27913	38103	71289
10775	51502	63938	87234	163315
11339	54217	67364	91964	172087

RM6[2]:

935	4185	4658	6786	13176	12237
3531	15830	17675	25704	49797	46321
4816	21633	24253	35218	68104	63440
13280	59542	66518	96819	187450	174587
13344	59857	66944	97449	188522	175784
21539	96412	107413	156877	304269	283435

RM7[2]:

2333	11299	15154	24380	43289	48906	66122
8603	41526	55243	88219	156966	175862	237597
11760	56665	75023	119087	211967	236192	318143
33610	162704	217840	349234	619776	698596	941502
32727	158017	210332	335844	597195	669430	903326
54659	265174	357069	576151	1021398	1158574	1564681
62519	304751	414610	673274	1189213	1361518	1832996

RM8[2]:

1000	4935	6558	9691	18881	21792	28618	18289
3481	17106	22595	33398	64855	74331	97886	62381
4101	19989	26163	38947	74717	84555	112147	70944
12370	60636	80068	119345	229722	262617	347134	220378
11369	55461	72774	108750	208429	236499	314172	198702
19609	96102	127123	190607	365789	418775	555271	352026
23280	114747	153136	229708	441545	510330	672926	428078
19938	98515	131270	194413	378340	438090	574800	367463

REFERENCES

[A&K77] E.O.Adegbeyeni and E.V.Krisnamurthy: "Finite field computation technique for exact solution of systems of linear equations and interval linear programming problems"; Int.J.System Sci. vol 8, no.10, 1977, 1181-1192.

[Bar72] E.H.Bareiss: "Computational solution of matrix problems over an integral domain"; J.Inst.Math.Applic, 10,1972,68-104.

[B&f66] I.Borosh and A.S.Fraenkel: "Exact solution of linear equations with rational coefficients by congruence techniques"; Math. Compu. 23, 1969, 197-200.

[Bra71] G.H.Bradley: "Equivelent integer programs and canonical problems"; Management sciences, vol.17, no.5, 1971, 354-366.

[Bur50] W.J.Burton: "The arithmetic theory of quadratic forms, The Carus Mathematical graph"; Mathematical Asso. of America, no. 10, DC, 1950, 61-61.

[C&L71] S.Cabay and T.P.L.Lam: "Congruence techniques for the exact solution of integer system of linear equations"; ACM Trans. Math. Soft, Vol.3, No.4, 1977, 386-397.

[F&L71] A.S.Fraenkel and D.Loewenthal": Exact solution of linear equation with rational coefficient"; J. Nat. Bur. Standard, 75B, 1&2, 1971.

[Fru77] M.A.Frumkin: "Polynomial time algorithm in the theory of linear diophantine equation"; (proc. 1977 Int. FCT-conference, Poland, 1975) Lecture Notes of computer science, No.56, 386-392.

[G&N72] R.Garfinkel and G.L.Nemhauser: "Integer Programming"; J.Wiley & sons, New York, 1972.

[H&S79] G.Havas and S.Sterling: "Integer matrices and abelian group"; EUROSAM 79, Int. symp. on sym. and algeb. manipulation, France, 1979, 431-451.

[Her51] C.Hermite: "Sur l'introduction des variables continues dans la theorie des nombres"; J. R. Angew. Math, 41, 1851, 191-216.

[H&G69] J.A.Howell and R.T.Gregory: "An algorithm for solving linear algebraic equations using residue arithmetic I"; BIT, 9, 1969, 200-204.

[H&G69] J.A.Howell and R.T.Gregory: "An algorithm for solving linear algebraic equations using residue arithmetic II"; BIT, 9, 1969, 324-337.

[Hu 70] T.C.Hu: "Integer Programming and Networks Flows"; Addison Wesley, Reading, MA, 1969.

[H&W70] M.F.Hurt and C.Waid: "A generalized inverse which gives all the integral solutions to a system of linear equations"; SIAM J. Appl. Math. vol 19, no.3, 1970, 547-550.

[J&R77] L.W.Johnson and R.D.Riess: "Numerical Analysis"; Addison Wesley, Reading, MA, 1977, p-120.

[K&B79] R.Kannan and A.Bachem: "Polynomial algorithms for computing the Smith and Hermite normal form of an integer matrix"; SIAM J Compu, vol.8, No.4, 1979, 499-507.

[Knu69] D.E.Knuth: "The Art of Computer Programming" Seminumerical algorithms; vol.2, Addison Wesley, 1969.

[K&R75] E.V.Krishnamurthy, T.M.Rao, K.Subramanian: "Finite segment p-adic number systems with application to exact computation", Proc. Indian Acad. Sci, 81A, 58-79, 1975.

[M&M64] M.Marcus and H.Minc: "A Survey of Matrix Theory and Matrix Inequalities"; Allyn and Bacon, Boston, 1964.

[Mcl77] M.T.McClellan: "The exact solution of linear equations with rational function coefficient"; ACM Trans Math Soft, vol 3, no 1, 1977, 1-25.

[M&C79] R.T.Moenk and J.H.Carter: "Approximate algorithm to derive exact solutions to system of linear equations"; EUROSAM 79, Int. symp. on symb and algeb. manipulation, France, June 1979, 65-73.

[New67] M. Newman: "Solving equations exactly"; J. Res. Nat. Bur.Stan. 71B, 1967, 171-179.

[Rab76] M.O.Rabin: "Probabilistic Algorithm"; In Algorithm and Complexity. Ed. J.F.Traub, Academic Press, New York, 1976.

[Ray79] V.J.Rayward-Smith: "On computing the Smith Normal Form of an integer matrix"; ACM Trans Math. Software, vol 5, no 4, 1979, 451-456.

[Ros52] J.B.Rosser: "A method of computing exact inverses of matrices with integer coeficients"; J.Res.Nat.Bur. Stand, 49, 1952, 349-358.

[Smi66] D.A.Smith: "A basis algorithm for finitely generated abelian groups"; Math. Algorithm, 1, 1966, 13-16.

[Smi61] H.J.S.Smith: "On systems of intermediate equations and

congruences"; Philos. Trans, 151, 1861, 293-326.

[Ste67] J.Stein: "Computational Problems associated with Racah Algebra"; J.Comp.Phy, 1, 1967, 397-405.

[Wol69] L.A.Wolsey: "Group representational theory in integer programming"; Tech.Report no. 41, MIT Oper. Res, Cambridge, Mass, 1969.