J. Inst. Maths Applics (1972) **10**, 68–104

Computational Solutions of Matrix Problems Over an Integral Domain*

ERWIN H. BAREISS

Northwestern University, Evanston, Illinois

and

Argonne National Laboratory, Argonne, Illinois

[Received 7 April 1971 and in revised form 1 September 1971]

Recent methods for handling matrix problems over an integral domain are investigated from a unifying point of view. Emphasized are symbolic matrix inversion and numerically exact methods for solving $\mathbf{Ax} = \mathbf{b}$. New proofs are given for the theory of the multistep method. A proof for the existence and an algorithm for the exact solution of $\mathbf{Tx} = \mathbf{b}$, where $\mathbf{T}$ is a finite Toeplitz matrix, is given. This algorithm reduces the number of required single precision multiplications by a factor of order n over the corresponding Gaussian elimination method. The use of residue arithmetic is enhanced by a new termination process. The matrix inversion problem with elements in the ring of polynomials is reduced to operations over a Galois field. It is shown that interpolation methods are equivalent to congruence methods with linear modulus and that the Chinese remainder theorem over $GF(x-p_k)$ is the Lagrange interpolation formula.

With regard to the numerical problem of exact matrix inversion, the One- and Two-step Elimination methods are critically compared with the methods using modular or residue arithmetic. Formulas for estimating maximum requirements for storage and timing of the salient parts of the algorithms are developed. The results of a series of recent tests, using existing codes, standard matrices and matrices with random elements are reported and summarized in tabular form. The paper concludes that the two-step elimination method be used for the inversion problem of numeric matrices, and in particular when a black-box approach to the matrix inversion problem is attempted such as in commercial time sharing systems. It is recommended that the inversion problem of matrices with elements over the polynomial ring be reduced to the numeric inversion problem with subsequent interpolation. An extensive Reference list is added.

* Work performed in part under the auspices of the United States Atomic Energy Commission. The results of this paper were previously reported at the Fourth Gatlinburg Symposium on Numerical Algebra (1969) and in a conference on Linear Equations over an Integral Domain (ICM, Nice 1970).

1. Introduction

NOT INFREQUENTLY, the consulting mathematician is asked for the exact solution of a linear system over an integral domain. There are many numerical problems for which exact solutions are desired. A general advantage of the use of methods yielding an exact solution is that the error analysis of the main problem is often drastically simplified. In particular, when matrices are ill-conditioned, such methods are most helpful. They can be used to evaluate a determinant and its minors. Classical methods such as the Hurwitz-Routh criterion, as well as other methods in stability theory, become practical. Also, there is a tendency in applied mathematics to use higher order approximation methods in solving partial differential and integral equations to reduce the order of the corresponding matrix problems. Again, exact matrix inversion facilitates the work of the numerical analyst. Other mathematical problems are found in integer linear programming, in the practice of solving systems of linear diophantine equations, the calculation of the invariants of matrices and in many problems of number theory.

Instead of numeric elements, symbolic elements, such as multi-variate polynomials, may be given. One application of the exact methods to symbolic matrix inversion is the computation of generating functions of flow graphs. Such problems occur in areas of circuit theory, Markov chains, graph theory and coding theory.

In many applications in physics, statistics and algebra, the inversion of Toeplitz or Hankel matrices are required. They occur by themselves or as blocks of larger systems, and often are not well conditioned. Therefore, exact and efficient algorithms for the inversion of special matrices are very desirable.

The body of this paper is mainly concerned with the symbolic and exact numerical problem of matrix inversion. Currently, there are essentially three different methods in use: the method of multistep Gaussian Elimination, the method of using residue (or modular) arithmetic, and the method of finding the greatest common divisor applied to division-free Gaussian elimination. We shall critically review these methods, present new and hopefully simpler proofs to some theorems, develop a number of new formulas, evaluate the merit of the one- and two-step elimination method and the methods using residue arithmetic; and then discuss, in the light of our acquired theoretical insight, results of a series of tests, conducted recently at Argonne, with several existing exact matrix inversion codes.

We shall see that the different methods can be treated from a unified point of view, with the Lagrange and Newton's fundamental interpolation formulas as a common link. The reason that the multistep method will emerge as superior to the method of residue arithmetic is its simpler structure.

2. Multistep Gaussian Elimination

2.1. *The Basic Algorithms*

In this section we describe simultaneously direct methods for solving the problem of symbolic and exact numerical solution of the matrix problem

$$\mathbf{Ax} = \mathbf{c} \tag{2.1}$$
$$\mathbf{A} = [a_{ij}], \qquad \mathbf{c} = [c_i], \qquad (i, j = 0, 1, \ldots, n).$$

The elements of **A** and **c** can be elements of any commutative ring, such as Galois fields, commutative Euclidean groups, commutative ideals, integers of a commutative number field and others. Commutativity of the elements is important. For the sake of simplicity, we restrict ourselves to matrices over a multivariate polynomial domain. This restriction means no loss of generality since matrices with more general elements can be reduced to multivariate polynomials. For example, make the substitutions $x = \sin \alpha\phi$, $y = (\alpha+\phi)^{-1}$ in the first of the two matrices below to obtain a second matrix of the desired form.

$$\begin{bmatrix} \sin 3\alpha\phi & -\sin \alpha\phi \\ \sin \alpha\phi & a^2 - \dfrac{\sin \alpha\phi}{\sqrt{(\alpha+\phi)}} \end{bmatrix} \Rightarrow \begin{bmatrix} 3-4x^2 & -x \\ 1 & a^2-xy \end{bmatrix}.$$

In the first column of the second matrix a factor x had been removed.

As a general rule, (valid for all methods treated in this paper) any matrix **A** and **c** should be inspected as to whether the rows and columns of the augmented matrix $\mathbf{A}^{(0)} = [\mathbf{A}, \mathbf{c}]$ are relatively prime in order to keep the number of operations in the inversion algorithm as small as possible. In our example, this was achieved by observing $\sin 3\beta = (3\text{–}4 \sin^2 \beta) \sin \beta$. In the case when the elements are polynomials over the ring of integers $\{I\}$

$$a_{ij} = \sum_{k=0}^{d} (a_k)_{ij} x^k \qquad a_k \in \{I\} \tag{2.2}$$

the Euclidean algorithm is applied to all rows and columns. Actually, one may use algorithms developed by Brown (1968, 1971) and Collins (1971). Also for elements that are integers, $a_{ij} \in \{I\}$, Blankinship (1963) and Bradley (1970) have devised efficient new methods to obtain the g.c.d. of arrays.

In order to simplify the notation in the following algorithms we identify the elements of the column vector **c** by

$$c_i = a_{i,n+1} \tag{2.3}$$

so that the augmented matrix is, with $a_{ij} = a_{ij}^{(0)}$

$$\mathbf{A}^{(0)} = [\mathbf{A}, \mathbf{c}] = [a_{ij}^{(0)}] \qquad i = 0, 1, \ldots, n, \; j = 0, 1, \ldots, n, n+1. \tag{2.4}$$

Possible extension of **c** to a rectangular $(n+1) \times m$ matrix is evident. Also we assume that **A** is nonsingular, and pivoting is not necessary. These are weak restrictions from a practical point of view, and are resolved in Bareiss (1968). Let us concentrate on the main points under discussion.

The algorithms which we consider next in this section are the matrix triangularization algorithms:

Gaussian Elimination (ordinary)

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - (a_{ik}^{(k)}/a_{kk}^{(k)}) a_{kj}^{(k)}. \tag{2.5}$$

Division-free Elimination

$$a_{ij}^{(k+1)} = a_{kk}^{(k)}a_{ij}^{(k)} - a_{kj}^{(k)}a_{ik}^{(k)}. \tag{2.6}$$

One-step Elimination (fraction free)

$$a_{ij}^{(k+1)} = (a_{kk}^{(k)}a_{ij}^{(k)} - a_{kj}^{(k)}a_{ik}^{(k)})/a_{k-1,k-1}^{(k-1)}. \tag{2.7}$$

If we perform the elimination row-wise, the range and sequence for each of the above algorithms is given by:

$$\begin{aligned} &\text{for } k = 0, 1, 2, \ldots, n-1; \\ &\quad \text{for } i = k+1, \ldots, n; \\ &\quad\quad \text{for } j = k+1, \ldots, n+1; \\ &\quad\quad\quad \text{do algorithm.} \end{aligned} \tag{2.8}$$

In (2.7), let $a_{-1,-1}^{-1} = 1$. Implicitly, the elements below the diagonal that are not affected by the transformation are zero. After n steps, $\mathbf{A}^{(0)}$ has been transformed into $\mathbf{A}^{(n)}$:

$$\mathbf{A}^{(n)} = \begin{bmatrix} a_{00}^{(0)} & a_{01}^{(0)} & \cdots & \cdots & a_{0,n}^{(0)} & a_{0,n+1}^{(0)} \\ & a_{11}^{(1)} & \cdots & \cdots & a_{1,n}^{(1)} & a_{1,n+1}^{(1)} \\ & & & & \vdots & \vdots \\ & & a_{kk}^{(k)} & \cdots & a_{k,n}^{(k)} & a_{k,n+1}^{(k)} \\ & \bigcirc & & & \vdots & \vdots \\ & & & & a_{n,n}^{(n)} & a_{n,n+1}^{(n)} \end{bmatrix}. \tag{2.9}$$

Algorithm (2.5), with refinements and/or modifications, is generally used to solve systems of linear equations over a field. It is not suited for calculations over an integral domain.

Algorithm (2.6) can be used over an integral domain. But when used in its raw form, say over the domain of polynomials of degree d for the elements a_{ij}, elements $a_{ij}^{(k)}$ are formed of degree $2^k d$. For example, for $n = 10$ (a matrix of order 11) and entries over the ring of linear polynomials, $a_{ij} = \alpha_{ij} + \beta_{ij}x$, the algorithm produces polynomials of degree $2^{10} \times 1 = 1024$, while by Cramer's rule, one would expect $a_{nn}^{(n)}$ to be only of degree 11. Similar remarks hold when the entries of $\mathbf{A}$ are integers, $a_{ij} \in \{I\}$. To see this we remember that any integer has a "polynomial" representation (2.2),

$$a_{ij} = \pm \sum_{k=0}^{d-1} (a_k)_{ij} B^k \tag{2.10}$$

where B is the base of the number system (say 2, 8, 10, ...) and $0 \leqslant (a_k)_{ij} < B$. Thus here, d stands for the number of digits in a_{ij}. However, the division-free algorithm (2.6) in conjunction with an efficient g.c.d. algorithm as mentioned above, can be used to find the solution to (2.1). In general, it will be a rather slow process as will become apparent below. But the combination of these two algorithms, after appropriate modifications, is a tool to solve the diophantine problem

$$\mathbf{Ax} = \mathbf{c} \qquad (a_{ij}, x_i, c_i \in \{I\}).$$

The one-step algorithm (2.7) is attributed to Camille Jordan (1838–1922), but went unnoticed and was rediscovered several times since. When applied to matrices over an integral domain, it produces transformed matrices $\mathbf{A}^{(k)}$ $(k = 1, 2, \ldots, n)$ which are again over an integral domain. This means the element $a_{k-1,k-1}^{(k-1)}$ divides $a_{kk}^{(k)}a_{ij}^{(k)} - a_{kj}^{(k)}a_{ik}^{(k)}$, and therefore reduces the size of the elements considerably compared with

those obtained by division-free elimination (2.6). If the elements $a_{ij}^{(0)}$ of the matrix $\mathbf{A}^{(0)}$ are polynomials of degree d, one-step elimination (2.7) produces elements $a_{ij}^{(k)}$ of degree $\leqslant (k+1)d$ as opposed to the elements $a_{ij}^{(k)}$ of (2.6) of degree $\leqslant 2^k d$. Similar results hold for the increase in the size of the elements when they are integers. However, the analogy (2.10) must be applied with caution, since it is not obvious, for small $a_{k-1,k-1}^{(k-1)}$ that $a_{ij}^{(k+1)}$ should be small also.

If the one-step elimination is considered as an algebraic iterative expression and is applied to itself we obtain the elements $a_{ij}^{(k+2)}$ from the $a_{ij}^{(k)}$ directly, which are given by the

Two-step Elimination

$$a_{ij}^{(k+2)} = (a_{ij}^{(k)}C^{(k+1)} + a_{k,j}^{(k)}C_1^{(k+1)} + a_{k+1,j}^{(k)}C_2^{(k+1)})/a_{k-1,k-1}^{(k-1)} \tag{2.11a}$$

where

$$\begin{aligned} C^{(k+1)} &= a_{k+1,k+1}^{(k+1)} \\ C_1^{(k+1)} &= (a_{k+1,k}^{(k)}a_{i,k+1}^{(k)} - a_{k+1,k+1}^{(k)}a_{i,k}^{(k)})/a_{k-1,k-1}^{(k-1)} \\ C_2^{(k+1)} &= -a_{i,k+1}^{(k+1)}. \end{aligned} \tag{2.11b}$$

The sequence of the algorithm is given by

```
for k = 0, 2, 4, ... (k ⩽ n−2)
    for i = k+2, k+3, ..., n;
        for j = k+2, k+3, ..., n+1;
            do algorithm (2.11);
for i = k+1;
    for j = k+1, k+2, ..., n+1;
        do algorithm (2.7).
```

If n is odd, the last row is transformed again by (2.7), using $k = n-1$. This algorithm eliminates two columns simultaneously, as exemplified in Fig. 1. The matrices $\mathbf{A}^0, \mathbf{A}^2, \mathbf{A}^4, \ldots$ obtained by (2.7) and (2.11) are identical. For the two-step algorithm the number of multiplications M_2 to transform $a_{ij}^{(k)}$ into $a_{ij}^{(k+2)}$ is approximately $3(n-k)^2$, the number of divisions $D_2 \sim (n-k)^2$; for the one-step method, the corresponding numbers are $M_1 \sim 4(n-k)^2$, $D_1 \sim 2(n-k)^2$. Therefore

$$M_1 : M_2 = 4:3 \qquad D_1 : D_2 = 2:1. \tag{2.12}$$

If the divisions are counted as multiplications, then $M_1 : M_2 = 6:4 = 1{\cdot}5$, and if a division is counted as two multiplications, $M_1 : M_2 = 8:5 = 1{\cdot}6$. This means that the two-step algorithm is generally 50% faster than the one-step algorithm. More detailed information can be found in Bareiss (1966, 1968).

Lipson (1969) showed another curiosity. Let the elements of $\mathbf{A}^{(0)}$ be multivariate polynomials of degree one over the ring of integers

$$a_{ij} = (a_0)_{ij} + \sum_{v=1}^{r} (a_v)_{ij}x_v \qquad (a_v)_{ij} \in \{I\} \tag{2.13}$$

where r is the number of variables. Furthermore let N_1 and N_2 be the total number of operations (multiplications and divisions) to transform $\mathbf{A}^{(0)}$ into $\mathbf{A}^{(n)}$ by the one-step and two-step algorithms respectively. Then asymptotically

$$\lim_{r\to\infty} N_2/N_1 = \begin{cases} 0 & \text{for } n \text{ even} \\ 1 & \text{for } n \text{ odd.} \end{cases} \tag{2.14}$$

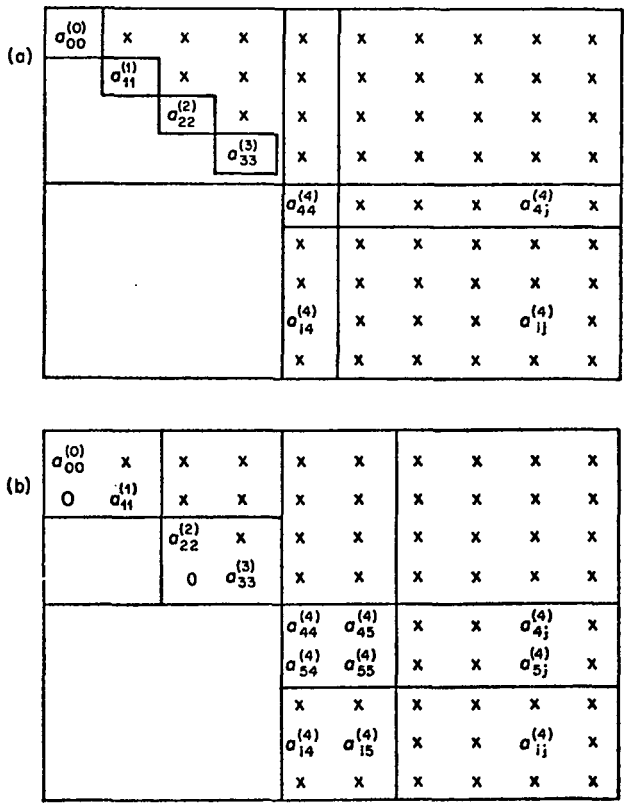

FIG. 1. Comparison of one-step (a) and two-step (b) method.

The ratios N_2/N_1 given in Table 1 were obtained (except for $r = \infty$) by an actual count of operations.

If (2.7) is substituted in (2.11), a three-step elimination algorithm is obtained. It is obvious that repetition of this process leads to any desired multi-step elimination algorithm. A code for the one-step method for $a_{ij} \in \{I\}$ was written by Boothroyd (1966); codes for the two-step methods for $a_{ij} \in \{I\}$, and a_{ij} being floating point numbers, were written by Garbow (1966) and Stallings (1970); and codes for both one-step and two-step method with symbolic entries were written by Lipson (1969, 1970).

TABLE 1

Values of N_2/N_1

	Order of matrix		
$n+1$:	3	4	5
$r = 0$	0·96	0·90	0·87
$r = 3$	0·45	0·77	0·69
$r = 6$	0·12	0·93	0·41
$r = \infty$	0	1·0	0

2.2. *Determinant Approach to Multistep Elimination*

To prove the correctness of the multistep elimination algorithms, one has three choices: (a) proof by induction as was indicated above and originally presented in Bareiss (1966), (b) generalize Sylvester's theorem and deduce the iteration formulas as was done in Bareiss (1968), or (c) define elements $a_{ij}^{(k)}$ as subdeterminants of $\det \mathbf{A}^{(0)}$, and show that they can be used to efficiently evaluate any higher order subdeterminants, including $\det \mathbf{A}$ itself. Then, by Cramer's Rule, the problem $\mathbf{Ax} = \mathbf{c}$ is solved.

We choose the third approach because (i) it is simple, (ii) leads in a natural way to a new integer-preserving algorithm for Toeplitz matrices and therefore (iii) gives a unified approach to the subject under discussion. It is then evident that the magnitude of the elements can, *a priori*, easily be bounded by the use of Hadamard's inequality. We start by defining elements $a_{kj}^{(k)}$, then show that they form an upper triangular matrix $\mathbf{A}^{(n)}$ of the form (2.9), and prove that the equations

$$\mathbf{Ax} = \mathbf{c} \Rightarrow \mathbf{A}^{(n)}\begin{bmatrix} \mathbf{x} \\ -1 \end{bmatrix} = 0 \tag{2.15a,b}$$

have identical solutions for $c_i = a_{i,n+1}$.

Define the determinant $a_{kj}^{(k)}$ with elements a_{ij} of $\mathbf{A}$ introduced in (2.1) by

$$a_{kj}^{(k)} = \begin{vmatrix} a_{00} & \cdots & a_{0,k-1} & a_{0,j} \\ \vdots & & \vdots & \vdots \\ a_{k,0} & \cdots & a_{k,k-1} & a_{k,j} \end{vmatrix} \qquad \begin{matrix} (k = 0, 1, 2, \ldots, n) \\ (j = 0, 1, 2, \ldots, n+1). \end{matrix} \tag{2.16}$$

Then

$$a_{kj}^{(k)} \equiv 0 \quad \text{for} \quad j < k \tag{2.17}$$

and

$$a_{n,n}^{(n)} = \det \mathbf{A} = D. \tag{2.18}$$

Equation (2.17) is evident because for $j < k$ the determinant in (2.16) has two columns of identical elements. Therefore the composite matrix of the elements $a_{kj}^{(k)}$ form an upper triangular matrix (2.9).

By Cramer's Rule $\mathbf{Ax} = \mathbf{c}$ has the solutions

$$x_j = y_j/D, \qquad y_j = \det [a_{\cdot,0}, \ldots, a_{\cdot,j-1}, c, a_{\cdot,j+1}, \ldots, a_{\cdot,n}]. \tag{2.19}$$

The problem (2.15a) can be reformulated as

$$\sum_{j=0}^{n} a_{k,j} y_j = c_k D \qquad (k = 0, 1, 2, \ldots, n). \tag{2.20a}$$

Similarly, the problem (2.15b) can be reformulated, and we have to prove that

$$\sum_{j=0}^{n} a_{k,j}^{(k)} y_j = a_{k,n+1}^{(k)} D \qquad (k = 0, 1, 2, \ldots, n). \tag{2.20b}$$

Proof. Let the cofactors of a_{ij} in $\mathbf{A}$ be denoted by A_{ij}. Then

$$y_j = \sum_{l=0}^{n} c_l A_{lj} \qquad (c_l = a_{l,n+1}). \tag{2.21}$$

Now, expand the determinant $a_{kj}^{(k)}$ into the last column of (2.16) and denote the cofactors of the last column elements a_{ij} by $A_i^{(k)}$. The notation $A_i^{(k)}$ instead of $A_{ij}^{(k)}$ is justified because $A_i^{(k)}$ is independent of the last column elements. We have

$$a_{k,j}^{(k)} = \sum_{i=0}^{k} a_{ij} A_i^{(k)}. \tag{2.22}$$

Substituting (2.22) and (2.21) into (2.20b) yields for the left-hand side

$$\sum_{j=0}^{n} a_{k,j}^{(k)} y_j = \sum_{j=0}^{n} \sum_{i=0}^{k} \sum_{l=0}^{n} a_{ij} A_i^{(k)} c_l^{(k)} A_{lj}$$
$$= \sum_{i=0}^{k} \sum_{l=0}^{n} c_l A_i^{(k)} \sum_{j=0}^{n} a_{ij} A_{lj}$$
$$= \sum_{i=0}^{k} \sum_{l=0}^{n} c_l A_i^{(k)} \delta_{il} D = \sum_{l=0}^{k} c_l A_l^{(k)} D$$

and for the right-hand side

$$a_{k,n+1}^{(k)} D = \sum_{l=0}^{k} a_{l,n+1} A_l^{(k)} D = \sum_{l=0}^{k} c_l A_l^{(k)} D.$$

Therefore (2.15) is true.

Our next step is to show that the determinants $a_{kj}^{(k)}$ can be obtained recursively from lower order determinants. To this end, we extend the definition of $a_{kj}^{(k)}$ to $a_{ij}^{(k)}$ by replacing the elements $a_{k,l}$ in the last row of the determinant (2.16) by the corresponding elements $a_{i,l}$. Thus,

$$a_{ij}^{(k)} = \begin{vmatrix} a_{00} & \cdots & a_{0,k-1} & a_{0,j} \\ \vdots & & \vdots & \vdots \\ a_{k-1,0} & \cdots & a_{k-1,k-1} & a_{k-1,j} \\ a_{i,0} & \cdots & a_{i,k-1} & a_{ij} \end{vmatrix} \qquad \begin{matrix} (i, k = 0, 1, \ldots, n) \\ (j = 0, 1, \ldots, n+1) \end{matrix} \tag{2.23}$$

With this notation, Sylvester's Identity has the following form for the determinant of **A**, i.e., $D = a_{nn}^{(n)}$:

$$a_{nn}^{(n)} \cdot \{a_{l-1,l-1}^{(l-1)}\}^{n-l} = \begin{vmatrix} a_{ll}^{(l)} & \cdots & a_{l,n}^{(l)} \\ \vdots & & \vdots \\ a_{n,l}^{(l)} & \cdots & a_{n,n}^{(l)} \end{vmatrix} \qquad (0 \leqslant l \leqslant n) \tag{2.24}$$

with $a_{-1,-1}^{(-1)} = 1$.

If we apply this identity to the determinant $a_{ij}^{(k)}$ of (2.23) we have

$$a_{ij}^{(k)} \{a_{l-1,l-1}^{(l-1)}\}^{k-l} = \begin{vmatrix} a_{ll}^{(l)} & \cdots & a_{l,k-1}^{(l)} & a_{lj}^{(l)} \\ \vdots & & \vdots & \vdots \\ a_{k-1,l}^{(l)} & \cdots & a_{k-1,k-1}^{(l)} & a_{k-1,j}^{(l)} \\ a_{il}^{(l)} & \cdots & a_{i,k-1}^{(l)} & a_{ij}^{(l)} \end{vmatrix} \qquad (0 \leqslant l < k). \tag{2.25}$$

This means that any expression for the hyperdeterminant in (2.25) contains a factor of the $(k-l)^{\text{th}}$ power of $a_{l-1,l-1}^{(l-1)}$. Such an expression is given by expanding the hyperdeterminant into the last column. Thus

$$a_{ij}^{(k)} \{a_{l-1,l-1}^{(l-1)}\}^{k-l} = a_{ij}^{(l)} A^{(k,l)} + \sum_{r=l}^{k-1} a_{rj}^{(l)} A_r^{(k,l)} \tag{2.26}$$

where $A_r^{(k,l)}$ is the cofactor of $a_{rj}^{(l)}$ in the last column of (2.25) and $\mathbf{A}^{(k,l)}$ is the cofactor of the corner element $a_{ij}^{(l)}$. Obviously, $A^{(k,l)}$ is independent of the elements of both the last row and the last column. In fact, letting $n = k-1$ in (2.24) yields

$$A^{(k,l)} = a_{k-1,k-1}^{(k-1)} \cdot \{a_{l-1,l-1}^{(l-1)}\}^{k-1-l}. \tag{2.27}$$

In general, the following theorem is true.

THEOREM A.

$$\{a_{l-1,l-1}^{(l-1)}\}^{k-l-1} \text{ divides } A^{(k,l)} \text{ and } A_r^{(k,l)} \quad (r = l, \ldots, k-1).$$

Proof. The theorem is true for $A^{(k,l)}$ by (2.27). To prove it for $A_r^{(k,l)}$, we derive a matrix **B** from **A** by replacing all elements of column j ($j \geqslant k$) by zero except for the element a_{rj} ($l \leqslant r < k$) which is replaced by one, i.e.,

$$a_{ij} \Rightarrow b_{ij} = \begin{Bmatrix} 0 & i \neq r \\ 1 & i = r \end{Bmatrix} = \delta_{ir} \quad \begin{matrix} (j \geqslant k) \\ (l \leqslant r < k) \\ (0 \leqslant i \leqslant n) \end{matrix} \tag{2.28a}$$

and all other elements a_{it} ($t \neq j$) remain unchanged, i.e.,

$$b_{it} = a_{it} \quad \begin{matrix} (i = 0, 1, \ldots, n) \\ (t = 0, 1, \ldots, j-1, j+1, \ldots, n+1). \end{matrix} \tag{2.28b}$$

Therefore, because $l < k$,

$$b_{l-1,l-1}^{(l-1)} = a_{l-1,l-1}^{(l-1)}, \qquad b_{it}^{(l)} = a_{it}^{(l)} \tag{2.29a}$$

and

$$b_{ij}^{(l)} = \begin{vmatrix} a_{00} & \cdots & a_{0,l-1} & 0 \\ \vdots & & \vdots & \vdots \\ a_{l-1,0} & \cdots & a_{l-1,l-1} & 0 \\ a_{i,0} & \cdots & a_{i,l-1} & \delta_{ir} \end{vmatrix} = a_{l-1,l-1}^{(l-1)} \delta_{ir}. \tag{2.29b}$$

Now, we apply (2.25) and (2.26) to **B**, observe (2.29), and obtain

$$b_{ij}^{(k)} \{a_{l-1,l-1}^{(l-1)}\}^{k-l} = \begin{vmatrix} a_{ll}^{(l)} & \cdots & a_{l,k-1}^{(l)} & b_{lj}^{(l)} \\ \vdots & & \vdots & \vdots \\ a_{k-1,l}^{(l)} & \cdots & a_{k-1,k-1}^{(l)} & b_{k-1,j}^{(l)} \\ a_{il}^{(l)} & \cdots & a_{i,k-1}^{(l)} & b_{ij}^{(l)} \end{vmatrix} = b_{rj}^{(l)} A_r^{(k,l)} \tag{2.30}$$

because $l \leqslant r < k$. Observing that $b_{rj}^{(l)} = a_{l-1,l-1}^{(l-1)}$, equation (2.30) yields

$$A_r^{(k,l)} = b_{ij}^{(k)} \{a_{l-1,l-1}^{(l-1)}\}^{k-l-1}. \qquad \text{q.e.d.}$$

The relation (2.26), when divided by the $(k-l)$th power of $a_{l-1,l-1}^{(l-1)}$ is the desired recurrence formula for calculating $a_{ij}^{(k)}$. Referring to (2.27) and the theorem, we define divided cofactors by

$$C^{(k-1)} = a_{k-1,k-1}^{(k-1)}, \qquad C_r^{(k-1)} = A_r^{(k,l)} / \{a_{l-1,l-1}^{(l-1)}\}^{k-l-1}. \tag{2.31}$$

We note also that $C_{k-1}^{(k-1)} = -a_{i,k-1}^{(k-1)}$. The recurrence formula (2.26) becomes therefore the $(k-l)$-step algorithm

$$a_{ij}^{(k)} = \left(a_{ij}^{(l)} C^{(k-1)} + \sum_{r=l}^{k-1} a_{rj}^{(l)} C_r^{(k-1)} \right) \Big/ a_{l-1,l-1}^{(l-1)}. \tag{2.32}$$

The one-step algorithm (2.7) is obtained if $l = k-1$,

$$a_{ij}^{(k)} = \frac{1}{a_{k-2,k-2}^{(k-2)}} \begin{vmatrix} a_{k-1,k-1}^{(k-1)} & a_{k-1,j}^{(k-1)} \\ a_{i,k-1}^{(k-1)} & a_{ij}^{(k-1)} \end{vmatrix}, \tag{2.33}$$

and the two-step algorithm (2.11) if $l = k-2$,

$$a_{ij}^{(k)} = \frac{1}{\left[a_{k-3,k-3}^{(k-3)}\right]^2}\begin{vmatrix} a_{k-2,k-2}^{(k-2)} & a_{k-2,k-1}^{(k-2)} & a_{k-2,j}^{(k-2)} \\ a_{k-1,k-2}^{(k-2)} & a_{k-1,k-1}^{(k-2)} & a_{k-1,j}^{(k-2)} \\ a_{i,k-2}^{(k-2)} & a_{i,k-1}^{(k-2)} & a_{ij}^{(k-2)} \end{vmatrix},$$

or

$$a_{ij}^{(k)} = \left[a_{ij}^{(k-2)}C^{(k-1)} + a_{k-1,j}^{(k-2)}C_{k-1}^{(k-1)} + a_{k-2,j}^{(k-2)}C_{k-2}^{(k-1)}\right]/a_{k-3,k-3}^{(k-3)} \tag{2.34}$$

where

$$C^{(k-1)} = a_{k-1,k-1}^{(k-1)};$$

$$C_{k-1}^{(k-1)} = \frac{-1}{a_{k-3,k-3}^{(k-3)}}\begin{vmatrix} a_{k-2,k-2}^{(k-2)} & a_{k-2,k-1}^{(k-2)} \\ a_{i,k-2}^{(k-2)} & a_{i,k-1}^{(k-2)} \end{vmatrix} = -a_{i,k-1}^{(k-1)};$$

$$C_{k-2}^{(k-1)} = \frac{1}{a_{k-3,k-3}^{(k-3)}}\begin{vmatrix} a_{k-1,k-2}^{(k-2)} & a_{k-1,k-1}^{(k-2)} \\ a_{i,k-2}^{(k-2)} & a_{i,k-1}^{(k-2)} \end{vmatrix}.$$

Therefore, the multistep algorithms are proven, and the meaning of the elements $a_{ij}^{(k)}$ is given by (2.23).

Note 1. In the case where the elements of **A** and **c** are in a commutative ring with zero divisors, it is no longer obvious that the multistep procedure (2.25) to obtain $a_{ij}^{(k)}$ implies that $a_{ij}^{(k)} = 0$ when the determinant on the right-hand side is zero. It may happen that $a_{ij}^{(k)}\{a_{l-1,l-1}^{(l-1)}\}^{k-l} = 0$ with $a_{ij}^{(k)} \neq 0$, if $\{a_{l-1,l-1}^{(l-1)}\}^{k-l}$ is a zero divisor. Of course (2.25) also holds for $a_{ij}^{(k)} = 0$. The ambiguity is avoided by stipulating

(i) $a_{ij}^{(k)} = 0$ for $j < k$,

(ii) For $j \geqslant k$, if $\{a_{l-1,l-1}^{(l-1)}\}^{k-l}$ is a zero divisor, choose $a_{ij}^{(k)} \neq 0$ such that $a_{ij}^{(k)}\{a_{l-1,l-1}^{(l-1)}\}^{k-l} = 0$. This will minimize the amount of pivoting to be done since then for $j = i$, $a_{ii}^{(k)} \neq 0$.

Note 2. The remainder of this and the entire following section are devoted to the derivation of an efficient integer preserving algorithm for finite Toeplitz matrices. Readers not interested in Toeplitz matrices, are advised to proceed directly to the section on Back Substitution.

All our efforts so far have been directed to transform $\mathbf{A}^{(0)}$ into an upper triangular matrix $\mathbf{A}^{(n)}$. For later reference we indicate how a matrix can be transformed into a lower triangular form. Assume our problem is to solve $\mathbf{Bx} = \mathbf{c}$. Then, corresponding to (2.4) the augmented matrix $\mathbf{B}^{(0)}$ has the form

$$\mathbf{B}^{(0)} = \begin{bmatrix} b_{00} & \cdots & b_{0,n+1} \\ & & \\ b_{n0} & \cdots & b_{n,n+1} \end{bmatrix}. \tag{2.35}$$

If we define, corresponding to (2.23) and (2.16),

$$b_{ij}^{(n-k)} = \begin{vmatrix} b_{ij} & b_{i,k+1} & \cdots & b_{i,n} \\ b_{k+1,j} & b_{k+1,k+1} & \cdots & b_{k+1,n} \\ \vdots & \vdots & & \vdots \\ b_{n,j} & b_{n,k+1} & \cdots & b_{n,n} \end{vmatrix} \tag{2.36}$$

for $i, k = 0, 1, 2, \ldots, n$ and $j = 0, 1, 2, \ldots, n+1$, it follows similar to (2.17) that

$$b_{kj}^{(n-k)} \equiv 0 \qquad \text{for} \quad j > k. \tag{2.37}$$

Furthermore, the problem $\mathbf{B}^{(n)}\begin{bmatrix}\mathbf{x}\\ -1\end{bmatrix} = 0$ where

$$\mathbf{B}^{(n)} = \begin{bmatrix} b_{00}^{(n)} & & & b_{0,n+1}^{(n)} \\ b_{10}^{(n-1)} & b_{11}^{(n-1)} & & \vdots \\ \vdots & \vdots & \ddots & \\ b_{n,0}^{(0)} & b_{n,1}^{(0)} & \cdots & b_{n,n+1}^{(0)} \end{bmatrix}, \tag{2.38}$$

has the same solution $\mathbf{x}$ as the original problem $\mathbf{Bx} = \mathbf{c}$, and a recurrence relations analogous to (2.32) exists for computing the elements (2.36).

2.3. *The Integer Preserving Algorithm for Toeplitz Matrices*

In the numerical solution of problems such as the inversion of convolutions, integral equations with difference kernels, least square approximations, high order approximations to difference equations, and others, the matrix $\mathbf{A}$ in the problem $\mathbf{Ax} = \mathbf{c}$ is, at least in part, a Toeplitz or a Hankel matrix. Often, these matrices are ill conditioned, so that the use of an exact elimination algorithm is desirable. In Bareiss (1968) a fraction-producing algorithm was developed which requires only n^2 multiplications as opposed to $\sim n^3$ multiplications if the ordinary Gaussian elimination is used. The algorithm which is derived here uses again only $\sim n^2$ multiplications.

We recall the definition (2.23) for $a_{ij}^{(k)}$ belonging to the matrix $\mathbf{A}$ of (2.1), and introduce the following abbreviations

$$\mathbf{A}_{k-2} = \begin{bmatrix} a_{00} & \cdots & a_{0,k-2} \\ \vdots & & \vdots \\ a_{k-2,0} & \cdots & a_{k-2,k-2} \end{bmatrix}, \quad \mathbf{l} = \begin{bmatrix} a_{0,k-1} \\ \vdots \\ a_{k-2,k-1} \end{bmatrix}, \quad \mathbf{r} = \begin{bmatrix} a_{0,j} \\ \vdots \\ a_{k-2,j} \end{bmatrix} \tag{2.39}$$

$$\mathbf{t} = [a_{k-1,0}, \ldots, a_{k-1,k-2}], \quad \alpha = a_{k-1,k-1}, \quad \beta = a_{k-1,j}$$

$$\mathbf{b} = [a_{i,0}, \ldots, a_{i,k-2}], \quad \gamma = a_{i,k-1}, \quad \delta = a_{ij}.$$

Then, (2.23) can be written as the determinant of a partitioned matrix

$$a_{ij}^{(k)} = \det \begin{bmatrix} \mathbf{A}_{k-2} & \mathbf{l} & \mathbf{r} \\ \mathbf{t} & \alpha & \beta \\ \mathbf{b} & \gamma & \delta \end{bmatrix} \tag{2.40}$$

and the one-step algorithm (2.33) as a hyperdeterminant

$$a_{ij}^{(k)} = \frac{1}{|\mathbf{A}_{k-2}|} \begin{vmatrix} \alpha^{(k-1)} & \beta^{(k-1)} \\ \gamma^{(k-1)} & \delta^{(k-1)} \end{vmatrix} \tag{2.41}$$

where the elements $\alpha^{(k-1)}, \beta^{(k-1)}, \gamma^{(k-1)}, \delta^{(k-1)}$ are the following minors of (2.40):

$$\begin{aligned} \alpha^{(k-1)} &= \det \begin{bmatrix} \mathbf{A}_{k-2} & \mathbf{l} \\ \mathbf{t} & \alpha \end{bmatrix}, & \beta^{(k-1)} &= \det \begin{bmatrix} \mathbf{A}_{k-2} & \mathbf{r} \\ \mathbf{t} & \beta \end{bmatrix} \\ \gamma^{(k-1)} &= \det \begin{bmatrix} \mathbf{A}_{k-2} & \mathbf{l} \\ \mathbf{b} & \gamma \end{bmatrix}, & \delta^{(k-1)} &= \det \begin{bmatrix} \mathbf{A}_{k-2} & \mathbf{r} \\ \mathbf{b} & \delta \end{bmatrix}. \end{aligned} \tag{2.42}$$

Clearly, by shifting in (2.40) the column of $\mathbf{l}, \alpha, \gamma$ to the left and simultaneously the row $\mathbf{t}, \alpha, \beta$ to the top, the value of the determinant is not changed, i.e.,

$$a_{ij}^{(k)} = \det \begin{vmatrix} \alpha & \mathbf{t} & \beta \\ \mathbf{l} & \mathbf{A}_{k-2} & \mathbf{r}_j \\ \gamma & \mathbf{b} & \delta \end{vmatrix} = \frac{1}{|\mathbf{A}_{k-2}|} \begin{vmatrix} \alpha^{(k-1)} & \beta^{(k-1)} \\ \gamma^{(k-1)} & \delta^{(k-1)} \end{vmatrix} \tag{2.43}$$

where $\alpha^{(k-1)}$ to $\delta^{(k-1)}$ represent now the four "corner" minors of the determinant in the middle term of (2.43) and have the numerical values of the corresponding elements (2.42). Equations (2.40) and (2.41) state Sylvester's Identity. Equation (2.43) states that the interior principal minor $\mathbf{A}_{k-2}$ of a determinant divides the hyperdeterminant of its first "corner" minors. Thus, an algorithm based on (2.43) will be integer-preserving.

Now let us consider the Toeplitz matrix

$$\mathbf{A} = \begin{bmatrix} a_0 & a_1 & \cdots & a_n \\ a_{-1} & a_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_1 \\ a_{-n} & \cdots & a_{-1} & a_0 \end{bmatrix}. \tag{2.44}$$

Then, corresponding to (2.16); or (2.23) and (2.36) with $i = k$; the elements

$$a_j^{-(k)} = \begin{vmatrix} a_0 & a_1 & \cdots & a_{k-1} & a_{j+k} \\ a_{-1} & a_0 & \cdots & a_{k-2} & a_{j+k-1} \\ \cdot & & & \cdot & \cdot \\ \cdot & & & \cdot & \cdot \\ \cdot & & & \cdot & \cdot \\ a_{-k+1} & a_{-k+2} & \cdots & a_0 & a_{j+1} \\ a_{-k} & a_{-k+1} & \cdots & a_{-1} & a_j \end{vmatrix}, \tag{2.45a}$$

$$a_j^{+(k)} = \begin{vmatrix} a_j & a_1 & \cdots & a_{k-1} & a_k \\ a_{j-1} & a_0 & \cdots & a_{k-2} & a_{k-1} \\ \cdot & & & \cdot & \cdot \\ \cdot & & & \cdot & \cdot \\ \cdot & & & \cdot & \cdot \\ a_{j-k+1} & a_{-k+2} & \cdots & a_0 & a_1 \\ a_{j-k} & a_{-k+1} & \cdots & a_{-1} & a_0 \end{vmatrix} \tag{2.45b}$$

form respectively an upper and lower triangular matrix

$$\mathbf{A}^{-(n)} = \begin{vmatrix} a_0^{(0)} & \cdots & a_n^{(0)} \\ & \ddots & \vdots \\ \bigcirc & & a_0^{(n)} \end{vmatrix}, \qquad \mathbf{A}^{+(n)} = \begin{vmatrix} a_0^{(n)} & & \bigcirc \\ \vdots & \ddots & \\ a_{-n}^{(0)} & \cdots & a_0^{(0)} \end{vmatrix}. \tag{2.46a,b}$$

Note that $a_0^{-(k)} = a_0^{+(k)} = a_0^{(k)}$.

If we apply Sylvester's Identity in the centered form (2.43) to (2.45) we observe, using an obvious notation,

$$|\mathbf{A}^{-}_{k-2}| = a_0^{-(k-2)} = a_0^{+(k-2)} = |\mathbf{A}^{+}_{k-2}| = a_0^{(k-2)} \tag{2.47}$$

$$\begin{aligned} \alpha^{-(k-1)} &= a_0^{+(k-1)} & \beta^{-(k-1)} &= -a_{j+k}^{+(k-1)} \\ \gamma^{-(k-1)} &= -a_{-k}^{-(k-1)} & \delta^{-(k-1)} &= a_j^{-(k-1)} \end{aligned} \tag{2.48a}$$

$$\begin{aligned} \alpha^{+(k-1)} &= a_j^{+(k-1)} & \beta^{+(k-1)} &= -a_k^{+(k-1)} \\ \gamma^{+(k-1)} &= -a_{j-k}^{-(k-1)} & \delta^{+(k-1)} &= a_0^{-(k-1)}. \end{aligned} \tag{2.48b}$$

Hence, the integer preserving one-step elimination algorithm for Toeplitz matrices is by (2.43)

$$a_j^{-(k)} = \frac{1}{a_0^{(k-2)}} \begin{vmatrix} a_0^{(k-1)} & a_{j+k}^{+(k-1)} \\ a_{-k}^{-(k-1)} & a_j^{-(k-1)} \end{vmatrix}, \qquad a_j^{+(k)} = \frac{1}{a_0^{(k-2)}} \begin{vmatrix} a_j^{+(k-1)} & a_k^{+(k-1)} \\ a_{j-k}^{-(k-1)} & a_0^{(k-1)} \end{vmatrix}. \tag{2.49a,b}$$

In the problem $\mathbf{Ax} = \mathbf{c}$, the augmented matrix $[\mathbf{A}, \mathbf{c}]$ cannot be assumed to be of Toeplitz form. Therefore, the elements $c_k^{-(k)}$ and $c_k^{+(k)}$ must be considered. $c_k^{-(k)}$ is defined by (2.45a) if the last column is replaced by $(c_0, c_1, \ldots, c_k)$, and similarly, $c_{n-k}^{+(k)}$ is defined by replacing the first column in (2.45b) by $(c_{n-k}, c_{n-k+1}, \ldots, c_n)$. To obtain a recursion formula for the c's, we must extend this definition, similar to the extension of (2.16) to (2.23) by

$$c_j^{-(k)} = \begin{vmatrix} a_0 & \ldots & a_{k-1} & c_{j-k} \\ \vdots & & \vdots & \vdots \\ a_{-k} & \ldots & a_{-1} & c_j \end{vmatrix}, \qquad c_j^{+(k)} = \begin{vmatrix} c_j & a_1 & \ldots & a_k \\ \vdots & \vdots & & \vdots \\ c_{j+k} & a_{-k+1} & \ldots & a_0 \end{vmatrix}. \tag{2.50a,b}$$

Then, proceeding as before, we get

$$c_j^{-(k)} = \frac{1}{a_0^{(k-2)}} \begin{vmatrix} a_0^{(k-1)} & c_{j-k}^{+(k-1)} \\ a_{-k}^{-(k-1)} & c_j^{-(k-1)} \end{vmatrix}, \qquad c_j^{+(k)} = \frac{1}{a_0^{(k-2)}} \begin{vmatrix} c_j^{+(k-1)} & a_k^{+(k-1)} \\ c_{j+k}^{-(k-1)} & a_0^{(k-1)} \end{vmatrix}. \tag{2.51a,b}$$

The sequence of the algorithm is determined by

$$\begin{aligned} &\text{for } k = 1, 2, \ldots, n-1 \\ &\quad \text{for } j = -n, \ldots, \ldots(k+1), 0, \ldots, n-k \quad \text{do (2.49a)} \\ &\quad \text{for } j = -n+k, \ldots, -1, k+1, \ldots, n \quad \text{do (2.49b)} \\ &\quad \text{for } j = k, k+1, \ldots, n \quad \text{do (2.51a)} \\ &\quad \text{for } j = 0, 1, \ldots, n-k \quad \text{do (2.51b)}, \\ &\text{for } k = n, \text{ for } j = 0 \text{ do (2.49a), (2.51b), for } j = n \text{ do (2.51}a\text{).} \end{aligned} \tag{2.52}$$

We see that the (expensive) i-loop of (2.8) is missing. This accounts for the gain in speed by a factor of order n. If a sequence of Toeplitz matrices of identical elements but of increasing order are to be evaluated, or if the matrix is symmetric, or if $a_0 = 0$, the reader is referred to Bareiss (1969) where these problems are discussed for the fraction-producing algorithm, but are equally valid here.

2.4. *Back Substitution*

All the algorithms discussed so far resulted in a transformed matrix $\mathbf{A}^{(n)}$ of the upper triangular form (2.9). Since by Cramer's rule (2.19) with $D = \det \mathbf{A} = a_{nn}^{(n)}$, (for Toeplitz matrices $D = a_0^{(n)}$)

$$y_k = x_k D \tag{2.53}$$

is an element of the integral domain, the problem $\mathbf{A}^{(n)}\begin{bmatrix}\mathbf{x}\\-1\end{bmatrix} = 0$ is equivalent to

$$\mathbf{A}^{(n)}\begin{bmatrix}\mathbf{y}\\-D\end{bmatrix} = 0. \tag{2.54}$$

Therefore

$$y_n = a^{(n)}_{n,n+1}$$
$$y_k = \left(Da^{(k)}_{k,n+1} - \sum_{j=k+1}^{n} a^{(k)}_{k,j}y_j\right)\Big/a^{(k)}_{kk} \qquad (k = n-1, n-2, \ldots, 0). \tag{2.55}$$

3. Congruence Techniques

3.1. *Elements over the Ring of Integers*

The characteristic feature of the congruence techniques is to make all computations over an integral domain modulo a prime p. This approach to the exact solution of a given set of linear equations was treated by Takahasi & Ishibashi (1961), Borosh & Fraenkel (1966), Newmann (1967), Howell & Gregory (1969, 1970), and Cabay (1970).

The theoretical advantage attributed to the congruence method is that multiple precision computations will be necessary only at the initial and final steps, with perhaps double precision computation in intermediary calculations.

The principle of the method consists of using modular or residue arithmetic for the elimination process in solving $\mathbf{Ax} = \mathbf{c}$. This process has to be repeated sufficiently often, and with moduli p_i that are relatively prime $(p_i, p_j) = 1$, such that after back substitution, the desired solution can be determined with the aid of the Chinese remainder theorem or a variation thereof. Since the moduli must also be relatively prime with respect to $\det \mathbf{A}$, which is unknown, a small probability exists that the computed answers may not be correct. To assure correctness, some codes (Newmann, 1967) substitute the solution in $\mathbf{Ax} = \mathbf{c}$, using multiple precision arithmetic to perform the check.

In the following we shall describe briefly different methods that can be used to devise specific algorithms, which in turn can be combined in a practical routine to solve $\mathbf{Ax} = \mathbf{c}$.

(*a*) *Chinese remainder theorem and mixed radix representation.* As we have seen in (2.19) the problem $\mathbf{Ax} = \mathbf{c}$, where $\mathbf{A} = [a_{ij}]$, $\mathbf{c} = [c_i]$, $a_{ij}, c_i \in \{I\}$, is equivalent to

$$\mathbf{Ay} = D\mathbf{c}, \qquad \mathbf{x} = \mathbf{y}/D \tag{3.1}$$

where $\mathbf{y} = [y_j]$, $y_j \in \{I\}$, $D = \det \mathbf{A}$. We shall write z_{p_k} for the residue of z in

$$z_{p_k} \equiv z \bmod p_k.$$

For a sequence of primes $\{p_k, (k = 1, \ldots, m)\}$ the solution $\{\mathbf{y}_{p_k}, D_{p_k}\}$ is computed such that

$$\mathbf{A}_{p_k}\mathbf{y}_{p_k} \equiv D_{p_k}\mathbf{c}_{p_k} \bmod p_k. \tag{3.2}$$

If $p = \pi_m$, and therefore m, are given such that

$$p = \pi_m = \prod_{k=1}^{m} p_k > 2 \max\left(|D|, \max_{(i)} |y_i|\right), \qquad (\pi_m, D) = 1, \tag{3.3a}$$

then by the Chinese remainder theorem with

$$\frac{\pi_m}{p_k} \cdot p_k' \equiv 1 \bmod p_k \qquad (k = 1, 2, \ldots, m) \tag{3.3b}$$

$$y_j \equiv (y_j)_{p_k} \bmod p_k \tag{3.4a}$$

we have

$$(y_i)_p \equiv \sum_{k=1}^{m} \frac{\pi_m}{p_k} p_k'(y_i) p_k \bmod \pi_m, \qquad \mathbf{y}_p = [y_i]_p = \mathbf{y}_{\pi_m} \tag{3.4b}$$

and with

$$D \equiv D_{p_k} \bmod p_k \qquad (k = 1, 2, \ldots, m) \tag{3.4c}$$

we have

$$D_p \equiv \sum_{k=1}^{m} \frac{\pi_m}{p_k} p_k' D_{p_k} \bmod \pi_m. \tag{3.4d}$$

The vector $\mathbf{y}$ and D with the correct signs are then given by

$$\mathbf{y} \equiv \mathbf{y}_p \bmod \pi_m, \qquad \max |y_j| < \pi_m/2. \tag{3.4e}$$

$$D \equiv D_p \bmod \pi_m, \qquad |D| < \pi_m/2. \tag{3.4f}$$

All present codes use "one-word" primes p_k, so that most computations can be performed in single precision arithmetic. The m inverses p_k' of π_m/p_k are usually precomputed.

Borosh & Frankel (1966) use the mixed-radix representation of the numbers z, and further let

$$-\frac{\pi_m}{2} = -\frac{1}{2}\prod_{k=1}^{m} p_k < z < \frac{1}{2}\prod_{k=1}^{m} p_k = \frac{\pi_m}{2}. \tag{3.5a}$$

This reduces the bound for the absolute magnitude for D_p and $(y_j)_p$, as well as yields the correct sign of z. It can be shown (Lindamood, 1964) that any integer z satisfying (3.5a) has a unique symmetric mixed radix representation

$$z \sim (\alpha_1, \alpha_2, \ldots, \alpha_m)$$

which stands for

$$\begin{aligned} z &= \alpha_1 + \alpha_2 p_1 + \alpha_3 p_1 p_2 + \cdots + \alpha_m p_1 p_2 \cdots p_{m-1} \\ &= \sum_{k=1}^{m} \alpha_k \pi_{k-1} \end{aligned} \tag{3.5b}$$

where, for all k,

$$|\alpha_k| < \tfrac{1}{2} p_k; \qquad \pi_k = \prod_{i=1}^{k} p_i; \qquad \pi_0 = 1. \tag{3.5c}$$

Now, given the residues z_{p_i}, the coefficients α_k as well as z can be determined recursively as follows. Let $\alpha_1 = z_{p_1} = S_1$. Then, for $k = 2, 3, \ldots, m$ do

$$\alpha_k = (z_{p_k} - S_{k-1}) \pi_{k-1}^{-1} \bmod p_k \tag{3.5d}$$

$$S_k = S_{k-1} + \alpha_k \pi_{k-1} \tag{3.5e}$$

where $S_m = z$, which is computed in multiple precision.

Applied to our problem, with $z = y_i$ or D, the corresponding coefficients α_{ik} for y and $\alpha_{n+1,k}$ for D, can be computed each time after the equation (3.2) has been

solved for a particular modulus p_k. Thus, after m runs, the y_i's have the mixed radix representation

$$\begin{aligned} y_i^{(m)} &\sim (\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{im}). \\ D^{(m)} &\sim (\alpha_{n+1,1}, \alpha_{n+1,2}, \ldots, \alpha_{n+1,m}). \end{aligned} \tag{3.6a}$$

For a concise notation, we express (3.6a) in matrix form,

$$\begin{bmatrix} \mathbf{y} \\ D \end{bmatrix}^{(m)} \sim [\alpha_{ik}], \qquad (k = 1, 2, \ldots, m), \tag{3.6b}$$

$$\begin{bmatrix} \mathbf{y} \\ D \end{bmatrix}^{(m)} = [\alpha_{ik}]\vec{\pi}^{(m)}, \qquad \vec{\pi}^{(m)} = [1, \pi_1, \ldots, \pi_{m-1}]^T. \tag{3.6c}$$

(*b*) *A New Termination Process.* Of course, we do not know the exact number m of runs for (3.2), necessary to solve $\mathbf{Ax} = \mathbf{c}$. However, we know (Howell & Gregory, 1969) that any value of π_m such that

$$\pi_m \geqslant 2 \prod_{(i)} \Big(\sum_{(j)} a_{ij}^2\Big)^{\frac{1}{2}} \sum_{(k)} |c_k| \tag{3.7}$$

satisfies (3.3a); in fact, this requirement might be unnecessarily conservative.† A conservative and simple method to determine m while the computation proceeds is suggested by the following Theorem.

THEOREM B. *Given a square matrix* $\mathbf{A}$, *vectors* $\mathbf{y}$ *and* $\mathbf{c} \neq \mathbf{0}$ *such that* $\mathbf{Ay} = \mathbf{c}D$, *where* $D = \det \mathbf{A}$, *and the mixed radix representation*

$$\begin{bmatrix} \mathbf{y} \\ D \end{bmatrix} \sim [\alpha_{jk}] \tag{3.8}$$

corresponding to a set of primes $\{p_i > 2(i = 1, 2, \ldots)\}$. *If in* $[\alpha_{jk}]$, *column m contains at least one non-zero element* α_{im}, *and is followed by at least t zero columns such that*

$$\pi_{m+t} > \tfrac{1}{2}\|\mathbf{A}, \mathbf{c}\|_\infty \sum_{k=1}^{m} \pi_k \tag{3.9}$$

then

(i) $\mathbf{Ay}^{(m)} = \mathbf{c}D^{(m)}$,

(ii) $\mathbf{Ax} = \mathbf{c}, \quad \mathbf{x} = \mathbf{y}^{(m)}/D^{(m)}$ if $D^{(m)} \neq 0$,

(iii) $D = 0$ if $D^{(m)} = 0$ and $\mathbf{y}^{(m)} \neq 0$.

Proof. Let $\begin{bmatrix} \mathbf{y} \\ D \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ D \end{bmatrix}^{(m)} + [0] + \begin{bmatrix} \mathbf{y} \\ D \end{bmatrix}'$ where $\begin{bmatrix} \mathbf{y} \\ D \end{bmatrix}' \sim [\alpha_{jk}]$ $(k > m+t)$ and $[0]$ is the $(n+1) \times t$ submatrix of zeros. Then $\mathbf{Ay} - \mathbf{c}D = \mathbf{Ay}^{(m)} - \mathbf{c}D^{(m)} + \mathbf{Ay}' - \mathbf{c}D' = \mathbf{0}$. Assume that assertion (i) is not true. Therefore by (3.6c),

$$\|\mathbf{Ay}^{(m)} - \mathbf{c}D^{(m)}\|_\infty = \|\mathbf{Ay}' - \mathbf{c}D'\|_\infty \geqslant \pi_{m+t}.$$

But, remembering (3.5c),

$$\begin{aligned} \|\mathbf{Ay}^{(m)} - \mathbf{c}D^{(m)}\|_\infty &\leqslant \max_i \sum_{k=1}^{m} \Big(\sum_{(j)} |a_{ij}|\,|\alpha_{jk}| + |c_i|\,|\alpha_{n+1,k}|\Big)\pi_{k-1} \\ &\leqslant \max_i \tfrac{1}{2} \sum_{k=1}^{m} \Big(\sum_{(j)} |a_{ij}| + |c_i|\Big)\pi_k \\ &= \tfrac{1}{2}\|\mathbf{A}, \mathbf{c}\|_\infty \sum_{k=1}^{m} \pi_k < \pi_{m+t} \end{aligned} \tag{3.10}$$

† Note however that we do not know whether $(\pi_m, D) = 1$ until D is computed and analysed.

by hypothesis. This contradicts the former inequality and hence statement (i) of the theorem is true.

Statements (ii) and (iii) follow from elementary linear algebra. Note however that $D^{(m)} \neq 0$ does not imply $D^{(m)} = D$.

Inequality (3.9) can be used to terminate the sequence of calculations for the α_{ik}. Theorem B states: As soon as a sequence of zero columns in $[\alpha_{jk}] \neq 0$ is found such that (3.9) is satisfied, then we have either an exact solution to $\mathbf{Ax} = \mathbf{c}$ or $\mathbf{A}$ is singular. Such a sequence exists because of (3.3a). The evaluation of (3.9) can be done in single precision floating point arithmetic if desired.

An Algorithm for Termination of the α_{ik}-calculation is given as example:

Initiation. $\kappa := \frac{1}{2}\|\mathbf{A}, \mathbf{c}\|_\infty$; $\pi := 1$; $\sigma := 0$, $\alpha := -1$;

MAIN. For $k = 1, 2, \ldots$ do
$p := p_k$; $\pi := p\pi$; $\sigma := \sigma + \pi$;
for all i: calculate α_{ik} and store;
if for some i, $\alpha_{ik} \neq 0$ then $\alpha := 1$; go to MAIN;
otherwise
if $\alpha = 0$ go to TEST;
otherwise
if $\alpha = -1$ go to MAIN;
otherwise $\alpha := 0$; $w := \kappa\sigma$; go to TEST,

TEST. if $\pi > w$ go to EXIT.
otherwise go to MAIN.

It should be noticed that Theorem B does not prescribe any ordering for the set $\{p_i\}$ of primes. In practical applications, the primes are usually ordered in descending order, $p_1 > p_2 > p_3 > \ldots$, for economic reasons. However, if the primes are ordered in increasing order, we have

COROLLARY. *If in Theorem B the primes are ordered such that* $p_1 < p_2 < p_3 < \ldots$ *then a sufficient lower bound for* t *is given a priori by*

$$\pi_t > \tfrac{1}{2}\|\mathbf{A}, \mathbf{c}\|_\infty. \tag{3.11}$$

Proof. The proof follows the proof of Theorem B, but we must show in addition that (3.10) is still true if π_t of (3.11) replaces $\frac{1}{2}\|\mathbf{A}, \mathbf{c}\|_\infty$ in (3.10). Under the hypothesis of the corollary

$$p_k < (p_{t+k} - p_k)(p_{k+1} + p_{k+1}p_{k+2} + \ldots + p_{k+1}p_{k+2}\cdots p_m).$$

Therefore

$$p_k + p_k p_{k+1} + \ldots + p_k p_{k+1}\cdots p_m < p_{t+k}(p_{k+1} + p_{k+1}p_{k+2} + \ldots + p_{k+1}\cdots p_m)$$

and thus recursively, starting with $k = 1$,

$$\sum_{k=1}^{m} \pi_k < p_{t+1}(p_2 + p_2 p_3 + \ldots + p_2 p_3 \cdots p_m) < \ldots < p_{t+1}p_{t+2}\cdots p_{m+t}.$$

Hence

$$\pi_t \sum_{k=1}^{m} \pi_k < \pi_{m+t} \tag{3.12}$$

which proves the inequality (3.10) after the aforementioned substitution.

The above Corollary was discovered by Cabay (1970) but the bound for t given

here is an improvement. The bound for t in Theorem B can be further sharpened if we replace (3.9) by $\|[\mathbf{A}, -\mathbf{c}][\alpha_{jk}]^{(m)}\|_\infty \pi_m < \pi_{m+t}$. However, the work involved is then comparable to the work of testing $\mathbf{A}\mathbf{y}^{(m)} - \mathbf{c}D^{(m)} = \mathbf{0}$ as soon as a zero column in $[\alpha_{ij}]$ is found.

(*c*) *Computation of a Multiplicative Inverse.* In solving $\mathbf{A}\mathbf{y} = \mathbf{c} \bmod p$ most codes need the multiplicative inverse, a^{-1} to an element a of the Galois field over p, such that

$$a^{-1}a \equiv 1 \bmod p. \tag{3.13}$$

By *Fermat's theorem*

$$a_p^{p-1} \equiv 1 \bmod p \qquad a^{-1} \equiv a_p^{p-2} \bmod p \tag{3.14}$$

which leads to an algorithm easy to code. About twice as efficient (Collins 1969) is the extended Euclidean algorithm to solve

$$ua + vp = 1 \tag{3.15}$$

for u, p being a prime. Then $u \equiv a^{-1} \bmod p$. Algorithm, flow chart, and discussion are found in Knuth (1968). The algorithm is as follows. Start with $a_0 = p$, $a_1 = a$. For $i = 0, 1, 2, \ldots$, do

$$q_i = \lfloor a_i / a_{i+1} \rfloor, \qquad a_{i+2} = a_i - q_i a_{i+1} \tag{3.16a}$$

until $a_{i+2} = 1$. Let last $i = n$. Then with $u_0 = 0$, $u_1 = 1$, for $j = 0, 1, \ldots, n$ do

$$u_{j+2} = u_j - q_j u_{j+1}, \qquad u_{n+2} = a^{-1}. \tag{3.16b}$$

Applying Lame's Theorem (1844) (Morris, 1968) to this algorithm, we can state: To complete (3.16a,b) the number of multiplications (divisions or subtractions) never exceeds ten times the number of digits in a (since $a < p$). That this bound is good is seen by taking consecutive Fibonacci numbers, such as 89 and 55, where 89 is prime, and a total of $16 < 2{\cdot}10$ iterations are required. Here, all q_i's are one. Since in general we can expect half the remainders to satisfy

$$a_i - a_{i+1} < a_{i+1}, \tag{3.16c}$$

we can start each iteration with the test (3.16c) and thus save, in the mean, $n/2$ divisions and multiplications in (3.16a). In exchange, we perform n tests and n subtractions. Similarly, in (3.16b), $n/2$ multiplications can be expected to be saved in exchange for n tests $q_i = 1$. We conclude by mentioning that the methods of least absolute remainders usually do not reduce the number of iterations appreciably.

(*d*) *Elimination In a System of Congruences.* As we mentioned before, the solution $\mathbf{A}\mathbf{x} = \mathbf{c}$ will be reduced to solving (3.2), where

$$\mathbf{A}_{p_k} \equiv \mathbf{A} \bmod p_k, \qquad \mathbf{c}_{p_k} \equiv \mathbf{c} \bmod p_k \tag{3.17}$$

has the meaning that each element of the matrices and vectors is reduced modulo p_k. Obviously the solution of (3.2) is also a solution of

$$\mathbf{A}\mathbf{y}_{p_k} = D_{p_k}\mathbf{c} \tag{3.18}$$

a fact which is often used to simplify the notation.

To transform the matrix $\mathbf{A}_{p_k}$ into a triangular matrix, we can use elementary row (and column) operations in the usual way, but all performed over the Galois field defined by p_k. In particular, the ordinary Gaussian elimination algorithm (2.5) has been used because it generates the values of the corner first principal minors, and thus of the determinant of $\mathbf{A}_{p_k}$ as the product of the transformed diagonal elements. Any

other method that leads to a convenient method of calculating D_{p_k} can be used. Newmann uses (2.5) in the Jordan-Gauss elimination, which diagonalizes $\mathbf{A}_{p_k}$. Divisions are replaced by Multiplicative inverses. Since we have no round-off error problem, partial pivoting is replaced by testing whether the pivot element is 0 mod p_k and appropriate interchanges of rows (and/or columns) are performed when necessary. After the triangularization process is completed according to (2.5) back substitution is performed as in (2.55), of course using modular arithmetic. If $D_{p_k} \neq 0$, $\mathbf{y}_{p_k}$ is a solution vector for (3.18). But if $D_{p_k} = 0$, either $D = 0$ with probability $(p_k - 1)/p_k$ or $(D, p_k) \neq 1$ with probability $1/p_k$.

If $D_{p_k} = 0$, $D \neq 0$, the solution may not have to be discarded, although it may be advisable to do so from a practical point of view. Shapiro (1968) suggests the following procedure. If the rank of $\mathbf{A}_{p_k}$ is less than $n-1$, where n is the order of $\mathbf{A}_{p_k}$, then since by Cramer's rule $\mathbf{A}_{p_k}\mathbf{x} = \mathbf{c}_{p_k}$ has only the zero solution, one takes $\mathbf{y}_{p_k} = \mathbf{0}$. If the rank of $\mathbf{A}_{p_k}$ is $n-1$, and Gaussian elimination leads to a zero pivot at column i, the elimination process belonging to a_{ii} is omitted and the remainder of the elimination is done by the Gauss-Jordan diagonalization process. Therefore we have an upper triangular part for elements to the left of column i and a diagonal part for the elements to the right of column i. The i component of $\mathbf{y}_{p_k}$ is then defined by Cramer's rule, as if $D_{p_k} = 1$, thus

$$(\mathbf{y}_{p_k})_i = a_{11}^{(1)} \cdot a_{22}^{(2)} \cdots a_{i-1,i-1}^{(i-1)} \cdot (\mathbf{c}_{p_k})_i a_{i+1,i+1}^{(i+1)} \cdots a_{n,n}^{(n)}. \tag{3.19}$$

Now, since

$$\mathbf{A}_{p_k}\mathbf{y}_{p_k} \equiv \mathbf{0} \bmod p_k, \tag{3.20}$$

the elements of $\mathbf{y}_{p_k}$ of index smaller than i can be computed by back substitution, while the remaining elements will be necessarily zero, thus

$$(y_j)_{p_k} = \begin{cases} -(a_{jj}^{(j)})_{p_k}^{-1} \sum_{l=j+1}^{i} (a_{jl}^{(j)})_{p_k} (y_l)_{p_k} & (j = i-1, \ldots, 1) \\ 0 & (j = i+1, \ldots, n). \end{cases} \tag{3.21}$$

This solution can be used to compute the coefficients in the mixed radix representation.

3.2. *Elements over a Polynomial Ring and the Equivalence of the Congruence Techniques and Polynomial Interpolation*

To solve $\mathbf{Ax} = \mathbf{c}$, where the elements a_{ij} are polynomials of order not to exceed d, we have, basically, four different methods:

(1) Multistep elimination which we described above for multivariate polynomials of degree d,

(2) Division-free elimination and row-wise simultaneous reduction by greatest common divisor,

(3) Interpolation method,

(4) Congruential method.

The multistep methods seem to be the only methods that have so far obtained extensive consideration and application. Polynomials are only a subclass of the multivariate polynomials and therefore the methods need no further explanation. Previous to the discovery of the multistep methods, division-free elimination with and without reductions was used. As we have pointed out, if the augmented matrix $\mathbf{A}$ has mutually relatively prime rows and columns, any multistep algorithm provides the reduction of each element by a natural common divisor, and leads to the actual determinant of $\mathbf{A}$,

which is often the desired end product. Thus, adding a search routine for the simultaneously greatest common divisor of arrays has not yet proven, in practical application, to add to the overall efficiency of a computer routine. There is no proof that when rows and columns of the augmented matrix are mutually prime, the resulting solution polynomials of the problem $\mathbf{Ay} = D\mathbf{c}$ are component-wise relatively prime.

If the coefficients in the polynomials $a_{ij}(x)$, $c_i(x)$ are over a number ring or field, then interpolation methods can be used. Again, the problem $\mathbf{Ax} = \mathbf{c}$ is solved for

$$\mathbf{Ay} = D\mathbf{c} \tag{3.22}$$

where $\mathbf{y}$ and D have the meaning of (2.53) except that the elements are now polynomials of degree $\leqslant d$ as in (2.2). Since by Cramer's rule, we know that for matrices of order n, the degrees of D and the components of $\mathbf{y}$ do not exceed nd, the method of solving (3.22) can be reduced to a set of numerical problems as follows. In the elements $a_{ij}(x)$, $c_i(x)$ substitute a given number x_k for the variable x. Solve the resulting numerical problem exactly by any suitable method discussed above. Repeat this process for a total of $nd+1$ different values of x_k in order to obtain a set of $nd+1$ different solutions $\mathbf{y}(x_k)$, $D(x_k)$. Use any appropriate polynomial interpolation method to represent the desired polynomial solutions $\mathbf{y}(x)$, $D(x)$. In this context, a suitable method of interpolation is to calculate exactly the coefficients of the polynomials $y_i(x)$ and $D(x)$ from the associated systems of $(nd+1)$ linear equations with the Vandermonde matrix on the left.

However, two other well-known interpolation formulas are of interest to us. They are found in any book on numerical analysis such as Hildebrand (1956). We restrict ourselves to the representation of $\mathbf{y}(x)$ for obvious reasons and let $m = nd+1$.

By Lagrange's Interpolation Formula

$$\mathbf{y}(x) = \sum_{k=1}^{m} \frac{\pi(x)}{(x-x_k)\pi'(x_k)}\mathbf{y}(x_k) \tag{3.23}$$

where

$$\pi(x) = (x-x_1)(x-x_2)\ldots(x-x_m) = \pi_m(x) \tag{3.24a}$$

$$\pi'(x_k) = (x_k-x_1)\ldots(x_k-x_{k-1})\,.\,(x_k-x_{k+1})\ldots(x_k-x_m) = \pi'_m(x_k). \tag{3.24b}$$

If we make the interpretations

$$\begin{aligned} &\mathbf{y}(x) \Rightarrow \mathbf{y}_p, \qquad \mathbf{y}(x_k) \Rightarrow \mathbf{y}_{p_k} = \mathbf{y} \bmod p_k \\ &x-x_k \Rightarrow p_k, \qquad \pi(x) \Rightarrow p = \pi_m, \qquad 1/\pi'(x_k) \Rightarrow p'_k, \end{aligned} \tag{3.25}$$

the Lagrange interpolation formula for polynomials takes the form (3.4b) of the Chinese remainder theorem. That this interpretation is justified will be explained.

By Newton's Fundamental Formula

$$\begin{aligned} \mathbf{y}(x) &= \mathbf{y}[x_1]+(x-x_1)\mathbf{y}[x_1, x_2]+\ldots+(x-x_1)(x-x_2)\ldots(x-x_{m-1})\mathbf{y}[x_1, x_2, \ldots, x_m] \\ &= \mathbf{y}[x_1]+\mathbf{y}[x_1, x_2]\pi_1(x)+\ldots+\mathbf{y}[x_1, x_2, \ldots, x_m]\pi_{m-1}(x) \end{aligned} \tag{3.26}$$

where $\pi_k(x) = \prod_{i=1}^{k}(x-x_i)$, and where the divided differences are defined iteratively by

$$\mathbf{y}[x_k] = \mathbf{y}(x_k), \qquad \mathbf{y}[x_1, \ldots, x_k] = \frac{\mathbf{y}[x_2, \ldots, x_k]-\mathbf{y}[x_1, \ldots, x_{k-1}]}{x_k-x_1}.$$

If we make the interpretations (3.25) in addition to

$$\mathbf{y}[x_1, \ldots, x_k] \Rightarrow \alpha_k, \qquad \pi_k(x) \Rightarrow \pi_k \tag{3.27}$$

Newton's fundamental formula takes the form (3.5b) of the mixed radix representation for integers.

If we denote the partial sum of the first k terms in (3.26) by $\mathbf{S}_k(x)$, then

$$\mathbf{y}(x_k) = \mathbf{S}_k(x_k) \tag{3.28}$$

$$\mathbf{S}_k(x) = \mathbf{S}_{k-1}(x) + \mathbf{y}[x_1, \ldots, x_k]\pi_{k-1}(x) \tag{3.29a}$$

$$\mathbf{y}[x_1, \ldots, x_k] = (\mathbf{y}(x_k) - \mathbf{S}_{k-1}(x_k))[\pi_{k-1}(x_k)]^{-1}. \tag{3.29b}$$

With the interpretations (3.25), (3.27), and correspondingly

$$\mathbf{S}_{k-1}(x_k) \Rightarrow \mathbf{S}_{k-1}, \qquad [\pi_{k-1}(x_k)]^{-1} \Rightarrow \pi_{k-1}^{-1} \bmod p_k \tag{3.30}$$

the recursive formulas (3.29a,b) to calculate higher order divided differences become the recursive formulas (3.5d,e) for calculating the coefficients α_k.

The relationship between polynomial interpolation and congruential representation of numbers will now be made clear. Let a polynomial $Y(x)$ of degree d_y, and a polynomial $P(x)$ of lower degree d_p be given. If we divide $P(x)$ into $Y(x)$ such that

$$Y(x) = Q(x)P(x) + R(x) \tag{3.31a}$$

where $Q(x)$ is a polynomial of degree $d_Q = d_y - d_p$, and the remainder $R(x)$ is a polynomial of degree $d_R < d_p$, then $R(x)$ is

$$R(x) \equiv Y(x) \bmod P(x). \tag{3.31b}$$

If we assign x a numerical value x_k, (3.31b) is a numerical congruence. If x_k is a zero of $Q(x)$, then

$$Y(x_k) = R(x_k) \qquad (Q(x_k) = 0). \tag{3.31c}$$

Use of this fact is made in Horner's scheme for linear and quadratic factors in evaluating polynomials at given points x_k. Let us now evaluate the polynomials of $\mathbf{y}(x)$ in (3.23) at $x = x_k$. By (3.31)

$$\mathbf{y}(x_k) \equiv \mathbf{y}(x) \bmod (x - x_k). \tag{3.32}$$

Similarly $\mathbf{y}(x)$ is congruent mod $\pi(x)$ to the right side of (3.23). Thus the interpretations (3.25) are justified except for the inverse. Let $p_k'(x)$ be defined as the multiplicative inverse in

$$\frac{\pi(x)}{x - x_k} \cdot p_k'(x) \equiv 1 \bmod (x - x_k). \tag{3.33}$$

Corresponding to (3.32) the residue of the left in (3.33) is by (3.24a,b)

$$\pi'(x_k) \cdot p_k'(x_k) = 1.$$

Thus the interpretation of the inverse is also justified. Similarly, (3.30) is shown to be true.

Therefore, the interpolation method is equivalent to a congruence method with linear modulus. If $P(x)$ is no longer linear, then it is still possible to reduce a problem of high order d_y to a lower order problem of order less than d_p. In order to obtain the multiplicative inverse to a given polynomial, the extended Euclidean algorithm for polynomials must be applied.

To illustrate the equivalence of the linear congruence method and the interpolation method, the following example is given.

Example. Solve $\mathbf{A}\mathbf{x} = \mathbf{c}$, where

$$\mathbf{A} = \begin{bmatrix} x-4 & x-3 & x-2 \\ x-5 & 4 & x-2 \\ x-2 & 3 & x \end{bmatrix}, \qquad \mathbf{c} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

We solve $\mathbf{Ay} = D\mathbf{c}$, $D = \det \mathbf{A}$. Elementary row and column operations indicate that $d_D \leqslant 2$, $d_y \leqslant 2$. Therefore, it is sufficient to use three moduli, say

$$P_1 = x-3, \qquad P_2 = x-5, \qquad P_3 = x-7.$$

The steps to obtain $\mathbf{y}$, D are, from left to right,

$$[\mathbf{A}, \mathbf{c}]_1 \equiv \begin{bmatrix} -1 & 0 & 1 & 1 \\ -2 & 4 & 1 & 0 \\ 1 & 3 & 3 & 0 \end{bmatrix} \bmod (x-3); \quad \mathbf{y}_1 \equiv \begin{bmatrix} 9 \\ 7 \\ -10 \end{bmatrix} \bmod (x-3)$$

$$D_1 \equiv -19 \bmod (x-3)$$

$$[\mathbf{A}, \mathbf{c}]_2 \equiv \begin{bmatrix} 1 & 2 & 3 & 1 \\ 0 & 4 & 3 & 0 \\ 3 & 3 & 5 & 0 \end{bmatrix} \bmod (x-5); \quad \mathbf{y}_2 \equiv \begin{bmatrix} 11 \\ 9 \\ -12 \end{bmatrix} \bmod (x-5)$$

$$D_2 \equiv -7 \bmod (x-5)$$

$$[\mathbf{A}, \mathbf{c}]_3 \equiv \begin{bmatrix} 3 & 4 & 5 & 1 \\ 2 & 4 & 5 & 0 \\ 5 & 3 & 7 & 0 \end{bmatrix} \bmod (x-7); \quad \mathbf{y}_3 \equiv \begin{bmatrix} 13 \\ 11 \\ -14 \end{bmatrix} \bmod (x-7)$$

$$D_3 \equiv 13 \bmod (x-7).$$

We apply the Chinese remainder theorem (or Lagrange interpolation) for which

$$\pi(x) = (x-3)(x-5)(x-7) \Rightarrow P$$

$$1/\pi'(3) \equiv \tfrac{1}{8} \bmod (x-3) \Rightarrow P_1'$$

$$1/\pi'(5) \equiv -\tfrac{1}{4} \bmod (x-5) \Rightarrow P_2'$$

$$1/\pi'(7) \equiv \tfrac{1}{8} \bmod (x-7) \Rightarrow P_3'$$

$$\begin{bmatrix} \mathbf{y} \\ D \end{bmatrix} \equiv \frac{(x-5)(x-7)}{8} \begin{bmatrix} 9 \\ 7 \\ -10 \\ -19 \end{bmatrix} + \frac{(x-3)(x-7)}{-4} \begin{bmatrix} 11 \\ 9 \\ -12 \\ -7 \end{bmatrix} + \frac{(x-3)(x-5)}{8} \begin{bmatrix} 13 \\ 11 \\ -14 \\ 13 \end{bmatrix}$$

$$\bmod (x-3)(x-5)(x-7) = \begin{bmatrix} x+6 \\ x+4 \\ -x-7 \\ x^2-2x-22 \end{bmatrix};$$

$$\text{Answer:} \quad \mathbf{x} = \frac{1}{D} \begin{bmatrix} x+6 \\ x+4 \\ -x-7 \end{bmatrix}, \qquad D = x^2-2x-22.$$

A check is made on $\mathbf{Ay} = \mathbf{c}D$ for $x = 0$ and yields

$$\mathbf{y}(0) = \begin{bmatrix} 6 \\ 4 \\ -7 \end{bmatrix}, \qquad D = -22.$$

We see that Lagrange interpolation at the zeros of the moduli, namely at $x_1 = 3$, $x_2 = 5$, $x_3 = 7$, leads to computations, which are stepwise identical to those using congruence techniques.

Obviously, the above considerations can be generalized and imbedded in the abstract mathematical setting of a commutative ring with a unit element and containing a sufficiently large set of prime ideals. The important conclusion is that any

symbolic inversion problem for matrices with elements *over* a unique factorization domain can be reduced to a matrix inversion problem with elements *in* the unique factorization domain and subsequent interpolation. Apparently, the equivalence of the congruence technique and polynomial interpolation has gone unnoticed in the literature. Only a short note is given in Knuth (1969: 430), but certainly not in context with our investigations. Lipson (1971) made an extensive literature search on the objectives of this section. McClellan (1971) treats the subject of this section in his forthcoming dissertation.

4. Consideration of Computational Efficiency

4.1. *Arithmetic Considerations*

We hope to obtain some clear criteria for comparing the expected optimal efficiency of the different methods of solving $\mathbf{Ax} = \mathbf{c}$ exactly. In practical applications, the question is simply to determine which computer code provides the solution to a given problem in shortest time or at least cost. As any computer scientist knows, the theoretically best algorithm may not lead to a "best" computer code because a programmer may, for the sake of convenience, introduce time or storage-consuming subroutines, or use an inappropriate computer language, and so on. "Theoretically best" means we have considered all important factors necessary to predict the efficiency of a corresponding code. Therefore, in testing the merits of different algorithms or methods by performing calculations with existing computer codes, one must make proper adjustments to programming imperfections (and insist that corresponding improvements be implemented in a revision of the code at hand).

We proceed by comparing the theoretical efficiences of the one-step method and the congruential method with ordinary Gaussian elimination and optimum auxiliary algorithms. Starting with a heuristic point of view, one usually argues along the following line. The congruence technique has the advantage of performing the bulk of the computations in single precision. The problem of solving $\mathbf{Ax} = \mathbf{c}$ is broken down into τ single precision problems and only at the end of the calculations is multiprecision arithmetic needed. As the size of the elements a_{ij} of $\mathbf{A}$ increases, τ increases linearly, while if one were to use noncongruential techniques, the computational load would increase quadratically. However, the one- and two-step methods use multiple precision indeed. The computer scientist must learn to be cautious in making heuristic predictions. The unwritten (?) law that single precision methods are superior to multiple precision methods is based on past unpleasant personal memories in dealing with high precision numbers and must be considered a prejudice. The computer does what it is instructed to do without personal feelings. What the computer scientists should demand are machines and languages that make multiple precision calculations simpler to code. It is a pity that many of the new machines still do not have an accumulator that can hold a double length word! In addition, the assertion in the above heuristic argument that the use of non-congruential techniques increases the computational load quadratically is weakened by recent developments in the art of computation. From the section "How fast can we multiply?" (Knuth 1969) one may conclude that computers can be built which multiply two n-precision numbers as fast as n single precision multiplications. For conventional computers, algorithms have been devised by Toom (1963), Cook (1966), and Schönhage (1966) which perform

this task in as few as $0(n^{1+\varepsilon})$ single precision multiplications where $\lim_{n\to\infty} \varepsilon \to 0$, while the heuristic argument assumed $\varepsilon \geqslant 1$. Schönhage (1971) conjectured an optimum $\varepsilon = 0(\log\log n/\log n)$.

In order to prepare us to discuss a multiprecision algorithm for multiplication, let us first discuss multiplication and division of two polynomials. If α is the degree of a polynomial with $\alpha+1$ terms, and therefore

$$a^{(\alpha+1)} = \sum_{i=0}^{\alpha} a_i x^i, \qquad b^{(\beta+1)} = \sum_{j=0}^{\beta} b_j x^j \tag{4.1}$$

then the product is

$$a^{(\alpha+1)} b^{(\beta+1)} = \sum_{k=0}^{\alpha+\beta} x^k \sum_{\substack{i+j=k \\ i\leqslant\alpha \\ j\leqslant\beta}} a_i b_j. \tag{4.2}$$

This product requires $(\alpha+1)(\beta+1)$ multiplications to obtain all terms $a_i b_j$, for which we write in general

$$N(a^{(\alpha)} \,.\, b^{(\beta)}) = \alpha \,.\, \beta. \tag{4.3}$$

Let us consider the problem of determining $c^{(\gamma)}$ in

$$\frac{a_1^{(\alpha_1)} b_1^{(\beta_1)} + a_1^{(\alpha_2)} b_2^{(\beta_2)}}{d^{(\delta)}} = c^{(\gamma)}, \tag{4.4}$$

for an expression where we know that the remainder is zero. This equation means that we have an identity similar to the expressions occurring in the multistep method. In other words

$$a_1^{(\alpha_1)} b_1^{(\beta_1)} + a_2^{(\alpha_2)} b_2^{(\beta_2)} \equiv 0 \bmod d^{(\delta)} = c^{(\gamma)} d^{(\delta)}.$$

It is only important to note that the numerator in (4.4) is a known polynomial $f^{(\phi)}$ of (x) where

$$\phi - 1 \leqslant \max_{i=1,2} (\alpha_i + \beta_i) - 2. \tag{4.5a}$$

Thus

$$\phi - 1 = (\delta - 1) + (\gamma - 1), \tag{4.5b}$$

and the degree of $c^{(\gamma)}$ is known. To determine the coefficients c_i $(i = 0, 1, \ldots, \gamma-1)$ we can select any γ identities resulting from $f^{(\phi)} = d^{(\delta)} c^{(\gamma)}$, such as

$$\begin{aligned} d_0 c_0 &= f_0 \\ d_1 c_0 + d_0 c_1 &= f_1 \\ \cdots\cdots\cdots \quad \cdots\cdots\cdots \quad &\cdots \\ d_{\gamma-1} c_0 + d_{\gamma-2} c_1 + \ldots + d_0 c_{\gamma-1} &= f_{\gamma-1} \end{aligned} \tag{4.5c}$$

where $d_j = 0$ for $j \geqslant \delta$.

These equations determine $c_0, c_1, \ldots, c_{\gamma-1}$ uniquely. Therefore, to find $c^{(\gamma)}$ in (4.4) we do not need to compute all $\alpha_1\beta_1 + \alpha_2\beta_2$ terms of each of the two $a^{(\alpha)} b^{(\beta)}$, but only the γ coefficients of $x^{i+j} = x^k (k < \gamma)$ which are

$$\sum_{\substack{i+j=k \\ i<\min(\alpha,\gamma) \\ j<\min(\beta,\gamma)}} a_i b_j \quad \text{for} \quad 0 \leqslant k < \gamma. \tag{4.6}$$

In (4.6) we have omitted the subscripts 1 and 2 for α, β, a_i, b_i, which would be necessary to identify the terms in (4.4). The total number of produced evaluations in (4.6) is (see Fig. 2)

$$N = \sum_{\substack{k=0 \\ i<min(\alpha,\beta) \\ j<min(\beta,\gamma)}}^{\gamma-1} \sum_{i+j=k} 1 = \begin{cases} \frac{1}{2}\gamma(\gamma+1) & \text{if } \gamma \leqslant \beta \leqslant \alpha \\ \frac{1}{2}\beta(2\gamma-\beta+1) & \text{if } \beta \leqslant \gamma \leqslant \alpha \\ \alpha\beta-\frac{1}{2}(\alpha+\beta-\gamma)(\alpha+\beta-\gamma-1) & \text{if } \alpha \leqslant \gamma \leqslant \alpha+\beta-1 \\ \alpha\beta & \text{if } \gamma \geqslant \alpha+\beta-1 \end{cases} \quad (4.7)$$

To apply to (4.4), N, α, β, must be replaced by N_i, α_i, β_i ($i = 1, 2$). If $\gamma = 1$, then only one multiplication must be performed that is later used for the division. If

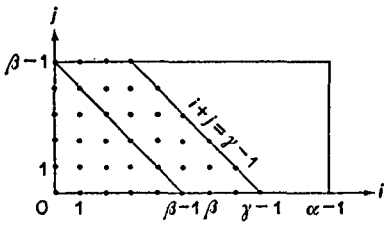

FIG. 2. Number of multiplications necessary to calculate equation (4.6).

$\alpha = \beta = \gamma$ then $N = \gamma(\gamma+1)/2$. To obtain $c_0, c_1, \ldots, c_{\gamma-1}$, the number of multiplications M and the number of divisions D are

$$M_c \leqslant \frac{\gamma(\gamma-1)}{2}, \qquad D_c = \gamma \quad (4.8)$$

respectively. Thus the total number of multiplications of the form $a \cdot b$ or $d \cdot c$ to evaluate (4.4) is

$$N_{\text{TOT}} = N_1 + N_2 + M_c. \quad (4.9c)$$

The minimum value is obtained for $\gamma = 1$, i.e., $c^{(1)}(x) = c_0$; in this case

$$N_{\text{MIN}} = 2, \qquad D_c = 1 \quad (4.9b)$$

because of the definition (4.1). If bounds are given such that $\alpha_1 = \beta_1 = \alpha_2 = \beta_2 \leqq r$, $\delta \leqq r-1$, $\gamma \leqq r+1$, then the maximum number of multiplications is

$$N_{\text{MAX}} = 2N_1 + M_c = 2\left(r^2 - \frac{(r-1)(r-2)}{2}\right) + \frac{(r+1)r}{2} = \frac{3r^2}{2} + \frac{7}{2}r - 2 \quad (4.9c)$$

and

$$D_{c\text{MAX}} = \gamma = r+1. \quad (4.9d)$$

If $\alpha_1 = \beta_1 = \alpha_2 = \beta_2 = \delta = \gamma = r$, then $N_{\text{TOT}} = \frac{1}{2}r(r+1)$, which is close to the maximum. These equations show clearly that in handling polynomial elements in the multistep methods, a special routine to perform (4.4) can be most rewarding.

If we let B be the base of a number system, such as 2, 8, 10, ..., then any given number smaller than B^α can be represented by

$$a^{(\alpha)} = a_0 + a_1 B + \ldots + a_{\alpha-1} B^{\alpha-1} = \sum_{i=0}^{\alpha-1} a_i B^i < B^\alpha \quad (4.10)$$

where α is the number of coefficients and for all i, $0 \leqslant a_i < B$. Thus the number of digits of $a^{(\alpha)}$ is $\alpha = \log_B B^\alpha$ and the length of $a^{(\alpha)}$ is defined by

$$L_1(a^{(\alpha)}) = \log_B a^{(\alpha)} < \alpha. \quad (4.11)$$

Thus, for $B = 10$, $L_1(99) \approx L_1(100) = \log_{10} 100 = 2$.

Now let $a^{(\alpha)}$ represent a multiprecision number in a computer. Usually, the word

length is a fixed multiple ω of the length of the computer base (such as 4 bytes, or 32 bits). Therefore $a^{(\alpha)}$ in (4.10) can be represented in base B^ω as

$$a^{(\alpha)} = \sum_{i=0}^{\alpha-1} a_i B^i = \sum_{j=0}^{\lfloor(\alpha-1)/\omega\rfloor} w_j B^{\omega j} < B^{\omega\cdot(\alpha/\omega)} \tag{4.12a}$$

where

$$w_j = \sum_{i=0}^{\omega-1} a_{\omega j+i} B^i < B^\omega. \tag{4.12b}$$

Similar to (4.11), the length of $a^{(\alpha)}$ in units of words is defined as

$$L_\omega(a^\alpha) = \log_{B^\omega} a^{(\alpha)} = \frac{\log_B a^{(\alpha)}}{\omega} < \frac{\alpha}{\omega} \tag{4.13}$$

Now we assume we have at our disposition a multiprecision arithmetic routine that recognizes the length of a given number $a^{(\alpha)}$ in units of word lengths. Obviously, each $a^{(\alpha)}$ is tagged by $\{\lceil\alpha/\omega\rceil, (\alpha-1) \bmod \omega\}$. Then the number of single precision multiplications to find the product of two numbers $a^{(\alpha)}$ and $b^{(\beta)}$ is in analogy to (4.1) to (4.3)

$$N_\omega(a^{(\alpha)} \cdot b^{(\beta)}) \doteq \frac{\alpha\beta}{\omega^2} \leqslant \left\lceil \frac{\alpha}{\omega} \right\rceil \left\lceil \frac{\beta}{\omega} \right\rceil. \tag{4.14}$$

Using fast multiplication, the bound (4.14) is of course lower. To solve (4.4) for multiprecision numbers $a_1^{(\alpha_1)}, \ldots, b_2^{(\beta_2)}, d^{(\delta)}$, one can treat the radix representations of $a_1^{(\alpha_1)}, \ldots, d^{(\delta)}$ as polynomials in B^ω, letting $x^k = B^{\omega k}$ in a and so on. Then the algorithm given by (4.5) and (4.6) can be used to calculate $c^{(\gamma)}$. Only at the end of the computation are adjustments made such that the radix coefficients in the representation of $c^{(\gamma)}$ are $0 \leqslant w_j < B^\omega$ (this may change $c^{(\gamma)}$ to $c^{(\gamma+1)}$, as well as have an influence on the sign of c). Since fast division is comparable in speed to fast multiplication, the above algorithm should only be considered as a first step of additional research in this area.

We have seen that in the multistep elimination, all elements $a_{ij}^{(k)}$ are determinants of order k. Therefore we are able to estimate a priori the absolute bounds for each element $a_{ij}^{(k)}$, in terms of the originally given elements a_{ij} of A by Hadamard's inequality. For simplicity's sake, we assume that

$$|a_{ij}| < B^\alpha \tag{4.15a}$$

for all elements a_{ij}. Then for any subdeterminant of order k, $D^{(k)} = |\det(a_{ij})|$, we have

$$D^{(k)} < (B^\alpha \sqrt{k})^k = B^{k\delta_k}. \tag{4.15b}$$

This is a sharp bound if we replace $<$ by $\leqslant$ in (4.15a). For example,

$$\begin{vmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{vmatrix} = 2^4, \quad D^{(4)} \leqslant (1 \cdot 4^{\frac{1}{2}})^4 = 2^4$$

is a possibility that can occur in practice. By (4.15b)

$$\delta_k = \alpha + \tfrac{1}{2} \log_B k \tag{4.16}$$

and by (4.11) the length of $D^{(k)}$ is $L_1(D^{(k)}) < k\delta_k$. In units of word lengths, by (4.13) and (4.15b)

$$L_\omega(D^{(k)}) = \log_{B^\omega} D^{(k)} < \frac{k\delta_k}{\omega}. \tag{4.17}$$

Turning now to the actual evaluation of the multistep and congruence methods, we make the following assumptions: The matrix $\mathbf{A}$ in $\mathbf{Ax} = \mathbf{c}$ is of order n. All elements of $\mathbf{A}$ and $\mathbf{c}$ are absolutely bounded by B^α.

4.2. *Maximum Storage Requirements*

We first determine the number of words needed to store the elements of the triangular matrix $\mathbf{A}^{(n)}$ obtained from $\mathbf{A}$ by multi-step methods. We have to store for

1st row: n numbers of length $L_1(a_{0j}^{(0)}) < \alpha = \delta_1$

--- ----- ----- -----

kth row: $n-k+1$ numbers of length $L_1(a_{k-1,j}^{(k-1)}) < k\delta_k$

--- ----- ----- -----

nth row: 1 number of length $L_1(a_{n-1,n-1}^{(n-1)}) < n\delta_n$.

The total length of all $n(n+1)/2$ numbers is therefore smaller than

$$\sum_{k=1}^{n} (n-k+1)k\delta_k < \tfrac{1}{6}n(n+1)(n+2)\delta_n. \tag{4.18}$$

Similarly, we can calculate a bound for the total number of words needed to store the $n(n+1)/2$ elements $a_{k,j}^{(k)}$ $(k = 0, \ldots, n-1;\ k \leqslant j < n)$, using (4.17). We conclude that asymptotically, the maximum storage requirements are

$$S_{as} \sim \frac{n^3}{6}\frac{\delta_n}{\omega} \tag{4.19}$$

words for the transformed triangular matrix $\mathbf{A}^{(n)}$ obtained from $\mathbf{A}$.

It is interesting to observe that maximum temporary storage is needed when the elimination process is half completed. Then the last $[n/2]^2$ elements have a word length $\frac{1}{2}n\delta_{n/2}/\omega$ and the maximum temporary storage required for these auxiliary elements is

$$S_{\text{temp}} \sim \frac{n^3}{8}\frac{\delta_{n/2}}{\omega}$$

words. Adding S_{temp} to $S_{n/2}$, the "permanent" storage at step $n/2$, shows that the maximum storage number is $\sim \frac{5}{24}n^3\,\delta_{n/2}/\omega$.

Using residue arithmetic the problem (3.2) is solved τ times. To estimate τ we make the assumption that we have selected the τ largest primes p_i such that $p_i < B^\omega$ $(i = 1, 2, \ldots, \tau)$, that is, the largest primes that can be accommodated in a single precision word. From (4.15b) and (3.7) follows the condition

$$\prod_{i=1}^{\tau} p_i \geqslant 2(B^\alpha\sqrt{n})^n nB^\alpha = B^{n\delta_n+\alpha+\log_B 2n} \tag{4.20}$$

for τ. Let the exact product of the p_i's be

$$\prod_{i=1}^{\tau} p_i = B^{\omega\tau-\varepsilon} \qquad (\varepsilon > 0). \tag{4.21}$$

Then equating (4.20) and (4.21) yields

$$\tau \geqslant \frac{n\delta_n}{\omega}[1+(\alpha+\varepsilon+\log_B 2n)/n\delta_n]. \tag{4.22}$$

Note that $\alpha+\frac{1}{2}\log_B n = \delta_n$. To store the transformed elements of a triangular matrix, each element requires exactly one single precision word. Thus the maximum total storage requirements for all τ triangular matrices are bounded by

$$\frac{n(n+1)}{2}\tau \approx \frac{n^3}{2}\frac{\delta_n}{\omega} \tag{4.23}$$

words.

One can conclude from (4.19) that the average length of an element transformed by the multistep method is less than $(n\delta_n)/(3\omega)$ word lengths, while for the congruential method, by (4.23) the average length, or better, the "mean space-time requirement" per element is $\tau \approx (n\delta_n)/\omega$ word lengths. It is interesting to note that the ratio of these two averages

$$\frac{n_{\delta n}/3\omega}{\tau} \approx \frac{1}{3} \tag{4.24}$$

is asymptotically independent of the original word length $\max L_\omega(a_{ij})$. The concept developed in this section will be needed below.

4.3. *The Number of Single Precision Multiplications Necessary for the Triangularization Process*

To estimate the cost of multiprecision product calculations in the multi-step methods in terms of single precision multiplication units we proceed as follows. We first assume that each element in the one-step method can be obtained by one multiplication in multiple precision and find the total equivalent for single precision. Then we adjust the result to the actual number of single precision multiplications needed by a multiplier, since this is a linear operation.

Equation (2.7) shows that the elements $a_{ij}^{(k+1)}$ are obtained from elements $a_{ij}^{(k)}$. The elements of the kth row are determinants of order k. Using the estimates (4.15) as before, the number of necessary single precision multiplications can be bounded by (4.14). Since we have equal bounds for all elements

$$N_\omega(a_{ij}^{(k-1)} \cdot a_{i',j'}^{(k-1)}) \leqslant k^2\delta_k^2/\omega \tag{4.25}$$

is the desired estimate. In the first iteration, $(n-1)^2$ terms are transformed. Therefore, $(n-1)^2 \cdot 1^2 \cdot (\delta_1/\omega)^2$ single precision operations are required. In the second iteration we have $(n-2)^2$ terms, each requiring $2^2(\delta_2/\omega)^2$ single precision multiplications, and so on, until we reach the last element. Then we add all terms and obtain

$$\begin{aligned}\sum_{k=1}^{n-1}(n-k)^2\left(\frac{k\delta_k}{\omega}\right)^2 &\leqslant \sum_{k=1}^{n-1}(n-k)^2 \cdot k^2\left(\frac{\delta_n}{\omega}\right)^2 \\ &= \left(\frac{\delta_n}{\omega}\right)^2 \tfrac{1}{30}n(n-1)[(2n-1)(8n^2-3n-1)-15n^2(n-1)] \\ &= \tfrac{1}{30}[n^5-n]\cdot\left(\frac{\delta_n}{\omega}\right)^2 .\end{aligned} \tag{4.26}$$

If we had used constant multiprecision arithmetic, the number of single precision multiplications would be $\Sigma(n-k)^2(n\delta_n/\omega)^2 \sim (\frac{1}{3})n^5(\delta_n/\omega)^2$.

Now we could adjust in (4.26) for the two terms needed to perform (2.7). Instead, we make use of the algorithm (4.5), which solves (2.7) by letting $x = \mathbf{B}^\omega$ in (4.1) and

adjusting the indices such that (4.1) corresponds to (4.10). The bounds for the $a_{ij}^{(k)}$ are such that (4.9c) applies. Letting $r = k\delta_k/\omega$, (4.9c) yields asymptotically $\frac{3}{2}(k\delta_k/\omega)^2$. Therefore, with the adjustment factor $\frac{3}{2}$, the total number of single precision multiplications for triangularization by the one-step method is

$$N_\omega(\text{one-step}) \approx \tfrac{1}{20}n^5 \cdot \left(\frac{\delta_n}{\omega}\right)^2. \tag{4.27}$$

We note that in this expression the operations "division" are included except for terms of lower order.

With regard to the two-step method, it was pointed out after (2.12) that we have a multiplicative advantage factor of $\frac{3}{4}$ to $\frac{3}{2}$. However, (2.11a) is an expression of the form (4.4) except that it has three terms in the numerator. In our case, we have equal estimates for the bounds of all terms, so that $N_1 = N_2 = N_3$ in (4.7). Extending (4.9) to three terms, $N_{\text{TOT}} = N_1+N_2+N_3+M_c$, the special value in (4.9c) becomes asymptotically $2r^2+\ldots$. One can argue that two columns are simultaneously eliminated, and therefore only about half as much work needs to be done. Then, it follows from (4.26) that asymptotically

$$N_\omega(\text{two-step}) \approx \tfrac{1}{30}n^5\left(\frac{\delta_n}{\omega}\right)^2 \tag{4.28}$$

where the number of divisions is again of lower order.

If one uses (2.7) to transform A into a diagonal form, we have for one-step elimination

$$\tfrac{3}{2}\cdot(n-1)\sum_{k=1}^{n-1}(n-k)k^2\cdot\left(\frac{\delta_n}{\omega}\right)^2 = \frac{3}{2}\left(\frac{\delta_n}{\omega}\right)^2\cdot\tfrac{1}{12}(n-1)^2n^2(n+1) \approx \tfrac{1}{8}n^5\left(\frac{\delta_n}{\omega}\right)^2 \tag{4.29}$$

multiplications to perform in single precision.

If the calculations are performed with a fast multiplication routine, then (4.25) is replaced by

$$N_\omega(a_{ij}^{(k-1)}\cdot a_{i'j'}^{(k-1)}) \approx 0\left(\left(k\frac{\delta_k}{\omega}\right)^{1+\varepsilon}\right).$$

Instead of (4.26) we have

$$\sum_{k=1}^{n-1}(n-k)^2\left(k\frac{\delta_k}{\omega}\right)^{1+\varepsilon} \approx \frac{2n^{4+\varepsilon}(\delta_n/\omega)^{1+\varepsilon}}{(2+\varepsilon)(3+\varepsilon)(4+\varepsilon)} \Rightarrow \frac{n^4}{12}\frac{\delta_n}{\omega}\bigg|_{\varepsilon\to 0}. \tag{4.26}'$$

Instead of (4.27) and (4.28), but without taking advantage of (4.5), and assuming division is comparable in speed to multiplication, we obtain for

$\mathbf{A}^{(n)}$ in triangular form and $\varepsilon \to 0$:

$$N'_\omega(\text{1-step}) \approx 0\left(\frac{n^4}{4}\frac{\delta_n}{\omega}\right), \tag{4.27}'$$

$$N'_\omega(\text{2-step}) \approx 0\left(\frac{n^4}{6}\frac{\delta_n}{\omega}\right). \tag{4.28}'$$

Similarly, we get for the diagonalization process, instead of (4.29) with fast multiplication ($\varepsilon \to 0$)

$$N'_\omega(\text{1-step}) \to 3(n-1)\sum_{k=1}^{n-1}(n-k)k\frac{\delta_k}{\omega} \approx \frac{n^4}{2}\frac{\delta_n}{\omega}. \tag{4.29}'$$

According to our remarks on page 96, formulas (4.27)′ to (4.29)′ represent the results of an ideal fast multiplication. Considering the present state of the art of computation, their value consists mainly in a challenge of the mathematical mind.

Using residue arithmetic for triangularization, the number of single precision multiplications equals the number of multiplications in ordinary Gaussian elimination, multiplied by τ (4.22),

$$N_\omega = \tfrac{1}{6}n(n-1)(2n-1)(\tau) \,.\, R \sim \frac{n^4}{3}\left(\frac{\delta_n}{\omega}\right) \,.\, R. \tag{4.30}$$

Here, we added an adjustment factor R to convert residue multiplication into ordinary single precision multiplication. Optimistically, one can take $R = 3$.

4.4. *Back Substitution*

In multiprecision, we assume in algorithm (2.55) that D and y_j are determinants of order n, while $a_{ij}^{(k)}$ has order $(k+1)$. Then, with the estimates (4.15) we have for the number of single precision multiplications necessary to carry out (2.55),

$$N_\omega(\text{Back sub}) = \frac{n\delta_n}{\omega^2}\sum_{k=1}^{n-1}(k+1)(n-k)\delta_{n-k} \leqslant \left(\frac{\delta_n}{\omega}\right)^2 \tfrac{1}{6}n^2(n-1)(n+4)$$

$$\sim \tfrac{1}{6}n^4 \,.\, \left(\frac{\delta_n}{\omega}\right)^2. \tag{4.31}$$

Using fast multiplication we have ($\varepsilon \to 0$)

$$N'_\omega(\text{Back sub}) \sim \tfrac{1}{2}n^3\frac{\delta_n}{\omega}. \tag{4.31}'$$

Using residual arithmetic requires

$$N_\omega(\text{Back sub}) \doteq \frac{n(n+1)}{2} \,.\, \tau \,.\, R \sim \frac{n^3}{2}\frac{\delta_n}{\omega}R \tag{4.32}$$

multiplications.

The results of Section 4 are summarized in Table 2. The conclusion is that the 2-step method with slow multiprecision arithmetic is faster than the congruence method for problems of order

$$n \leqslant 30\omega/\delta_n < 30\omega/\alpha. \tag{4.33}$$

An advantage of the 1-step and 2-step methods over the congruence technique is that only an efficient multiplication routine is required in addition to the basic elimination algorithm, while an efficient congruence method needs many auxiliary and specialized subroutines to insure mathematical reliability.

If the matrices are very large, Strassen (1969) showed that Gaussian elimination is not optimal, and that by appropriate partitioning the number of multiplications can be reduced from $0(n^3)$ to $0(n^{2.8})$. The same procedure can be adapted to the multistep methods. The minimum order of the matrix **A** leading to a more effective code has not yet been determined. Finally, Bradley (Yale, 1970) has an elimination routine where he uses a fast method to obtain the g.c.d. of arrays. His time estimates are quite high compared to those for the multistep elimination. The question of whether multistep and g.c.d-methods can be combined to yield higher efficiency is also still open.

TABLE 2

Summary of asymptotic estimates in units of single precision multiplication time

Method	Gauss elimination	1-step elimination		2-step elimination		Congruence technique
Multiplication method	Single precision	Slow multiplication	Ideal fast multiplication	Slow multiplication	Ideal fast multiplication	Residue arithmetic
Triangularization	$\frac{n^3}{3}$	$\frac{n^5}{20}\frac{\delta_n^2}{\omega^2}$	$0\left(\frac{n^4}{4}\frac{\delta_n}{\omega}\right)$	$\frac{n^5}{30}\frac{\delta_n^2}{\omega^2}$	$0\left(\frac{n^4}{6}\frac{\delta_n}{\omega}\right)$	$n^4\frac{\delta_R}{\omega}$
Diagonalization	$\frac{n^3}{2}$	$\frac{n^5}{8}\frac{\delta_n^2}{\omega^2}$	$0\left(\frac{n^4}{2}\frac{\delta_n}{\omega}\right)$	$\frac{n^5}{12}\frac{\delta_n^2}{\omega^2}$	$0\left(\frac{n^4}{3}\frac{\delta_n}{\omega}\right)$	$\frac{3n^4}{2}\frac{\delta_R}{\omega}$
Numerical example for triangularization	$\frac{1}{3}\times 10^6$	31×10^6	6×10^6	21×10^6	4×10^6	25×10^6

Explanations:

B: basis of number system
α: maximum number of digits in matrix element a_{ij}
a_{ij}: matrix element, an integer
$|a_{ij}| < B_\alpha$: bound of $\max_{ij} |a_{ij}|$
$|\det(a_{ij})| < (B^\alpha\sqrt{n})^n$: absolute bound for det **A**.
ω: number of digits of single precision computer word
$\delta_n = \alpha + \frac{1}{2}\log_B n > \log|\det(a_{ij})|^{1/n}$
$\delta_R = \delta_n + (\log_B 2n + \alpha)/n$

Numerical values used in Example:

$n = 100$
$\alpha \sim \delta_n \sim \delta_R \sim \omega/4$

5. Discussion of Experimental Data

In Table 2 we have summarized the more important estimates to be used in appraising the merits of the different elimination methods in pure numerical work. Since the order of the matrices is assumed to be usually less than one hundred, the asymptotic values are restricted in their reliability. However, if we want to compare different codes, Table 2 can still be used for normalization. Assume a code using residue arithmetic is tested for the time it takes to diagonalize a matrix, using Gauss-Jordan elimination. Another code using two-step elimination is tested for the same purpose. How much faster would the codes be, were they converted to yield triangular matrices with subsequent back substitution? For the code written for modular arithmetic, the asymptotic bound for the acceleration factor is $\frac{3}{2}$, while for the two-step method, the corresponding factors are $\frac{3}{1}\frac{0}{2} = 2{\cdot}5$ for slow multiplication and $\frac{6}{3} = 2$ for the idealized fast multiplication. Thus a conversion may well pay off.

At Argonne National Laboratory, several codes were tested to assess their efficiency. All use a diagonalization process. Code MATINV is an ordinary fraction-producing code in double precision, using partial pivoting, and is a standard production routine at the Laboratory. Code DIAG uses a two-step method and was written as a pilot program in double-precision; it was later translated into FORMAC to obtain high-order precision. Recently, a multiple-precision version was coded in FORTRAN, but such that the maximum word length expected from the estimate for the determinant is carried throughout. In addition, the code works always in floating point arithmetic. As we have noted, following (4.26), such a code has a disadvantage factor of at least 10, and can only be justified as a preliminary experimental code to determine whether the two-step method promises to be competitive with production codes, such as EXASØL, SØLVEX by Newman and EXACT by Howell. All codes used are listed at the end of this section.

Table 3 summarizes results obtained from inverting the ill-conditioned Pascal matrices. It came as a surprise that the two-step, double precision code DIAG, besides yielding the expected exact answer, was faster than MATINV for matrices up to order 20, the latter losing accuracy at a rate of about one digit per increase in order of the matrix. Obviously, pivoting is more expensive than the extra multiplications performed in DIAG. Table 3 also shows a speed advantage of this two-step code over the congruence technique code by a factor exceeding 100 in the mean. This can be explained by the fact that the latter codes need many more subroutines, and therefore have a costlier logical structure. These calculations were done on the CDC 3600 for economic reasons.

Table 4 summarizes the time consumption of calculations done on the IBM 360/75, which by comparison with similar problem runs on the CDC 3600 is about four times faster than the latter. The bulk of the testing was done with 4-digit random matrices $-10^4 < a_{ij} < 10^4$. This was done to obtain possibly unbiased sample matrices, as well as to test whether routine engineering problems of moderate size, say with matrices of order less than fifty, could economically be performed by exact methods. This seems to be the case, indeed. The pay-off is seen in the elimination of a good part of the error analysis for a given problem. Because the experimental code DIAG for the IBM/360 uses maximum multi-precision arithmetic throughout, the values for the

TABLE 3

(a) *Actual TIME in seconds used on CDC 3600 to invert Pascal matrices* by diagonalization. Codes MATINV and 2-STEP in floating double precision, HOWELL and NEWMAN with residue arithmetic*

Order of matrix	Condition number	Time MATINV	Time 2-STEP	Time HOWELL	Time NEWMAN
5	$1{\cdot}3\times10^{2}$	0·03	0·02	0·17	0·54
6	$4{\cdot}7\times10^{2}$	0·04	0·03	0·58	1·5
10	$9{\cdot}1\times10^{4}$	0·14	0·12	7·22	16·1
11	$3{\cdot}5\times10^{5}$	0·18	0·17	12·2	26·7
15	$7{\cdot}5\times10^{7}$	0·41	0·41	63·3	156·1
16	$2{\cdot}9\times10^{8}$	0·49	0·49	76·1	216·9
20	$6{\cdot}7\times10^{10}$	0·92	0·94	144·2	—
21	$2{\cdot}6\times10^{11}$	1·05	1·10	165·9	—
25	$6{\cdot}1\times10^{13}$	1·72	1·85	279·9	—
26	$2{\cdot}4\times10^{14}$	1·93	2·05	312·4	—

(b) *Elements of inverse of Pascal matrix with largest absolute value*

Order of matrix	Correct answer from 2-STEP and HOWELL	Results obtained from MATINV	Determinant computed with MATINV
5	60	60·	1·
6	146	146·	1·
10	22252	22252·	1·
11	82994	82994·	1·0000000000291038
15	15475205	15475204·999999399	1·0000000000290493
16	56884430	56884430·000001516	0·99999999994181096
20	11649069764	11649069752·894809	1·0000000008882119
21	45035696036	45035697082·518666	0·99999997588311781
25	9663914317396	9663723896192·8139	1·0000194577870331
26	36707034407396	36687092883725·730	1·0005756247777993

* The elements are $(i+j)!/(i!j!)(i,j=0,1,2,\ldots,n-1)$. The elements of the inverse of these Pascal matrices are integers, the determinants are 1.

2-step method were adjusted in an adjoining column by a theoretically-obtained correction factor of 10 (see remark after (4.26)), and in addition, another factor of 2 was used to normalize from diagonalization to triangularization. Adjusting for the speed difference of the two machines used, it turned out that these correction factors seem to be on the conservative side in comparison with time readings obtained for corresponding control runs from the CDC 3600 with the double precision routine DIAG. The next column adjusts for a triangularization code with an optimum fast-arithmetic routine. Based on the information of Tables 2 and 4, the adjustment factor chosen was $(n/5\cdot\delta_n/\omega)^{-1}=10/n$. The next two columns give the actual times for Howell's code EXACT, and the adjusted times if triangularization instead of diagonalization were used. Obviously, the implementation of an efficient two-step routine is desirable.

TABLE 4

TIME in seconds consumed to solve $\mathbf{Ax} = \mathbf{c}$ *with 4-digit random number elements in* $\mathbf{A}$ *and* $\mathbf{c}$, *on IBM* 360/75

Order of matrix	MATINV	2-step elimination Adjusted times Actual time multiprecision	Slow multiplication (1/2)(1/10)	Fast multiplication (10/n)	HOWELL residue Actual time	Adjusted time (2/3)
5	0·004	0·08	0·004	—	0·37	0·25
10	0·020	1·06	0·05	0·05	2·13	1·4
15	0·070	4·60	0·23	0·15	8·65	5·8
20	0·130	16·0	0·8	0·4	24·8	16·5
25	exponent overflow	38·7	1·9	0·8	56·9	38·
30		90·3	4·5	1·5	117·2	78·
35		182·5	9·1	2·6	189·4	126·
40		315·0	15·8	4·0	253·2	169·

All codes provide diagonal elimination. MATINV: double precision fraction production elimination. 2-STEP: floating point unchanged slow multiprecision arithmetic. Adjusted times give expected values if triangular reduction and slow variable multiprecision were used, or ideal fast multiplication. HOWELL: 10 digit prime residue arithmetic. For more explanations see text.

The routines and subroutines used in these tests are listed below.

MATINV

Burton S. Garbow, *Matrix Inversion with Accompanying Solution of Linear Equations*, Subroutine MATINV, ANL Subroutine Library Codes F402 (for CDC 3600) and F402S (for IBM S/360), Applied Mathematics Division, Argonne National Laboratory, July 1965, and December 1967.

BAREISS

Burton S. Garbow, *Integer-Preserving Gaussian Elimination*, Program P158, Subroutine DIAG, Applied Mathematics Division, Argonne National Laboratory, November 1966.
Burton S. Garbow, FΦRMAC translation of Subroutine DIAG, Applied Mathematics Division, Argonne National Laboratory.
Claudia R. Stallings, Multiple-precision version of Subroutine DIAG using *Multiple-precision Floating-point Arithmetic Package* for System/360, Applied Mathematics Division, Argonne National Laboratory, August 1970.
John R. Ehrman, *Multiple-precision Floating-point Arithmetic Package*, Library Program No. A1–27, Computation Group, Stanford Linear Accelerator Center, July 1968.

HOWELL

Jo Ann Howell, *Exact Solution of Linear Equations Using Residue Arithmetic*, Subroutine EXACT, Computation Center, The University of Texas at Austin.

NEWMAN

Morris Newman, Program EXASΦL, which uses primes less than 10^5, National Bureau of Standards, Washington, D.C.
Morris Newman, Program SΦLVEX, which uses primes just under 10^{10} and includes a small machine language subroutine, National Bureau of Standards, Washington, D.C.

Random Number Generators

3600

Nancy W. Clark, Subroutine RANF, ANL Subroutine Library Code G550 (for 3600),

multiplicative congruential generator using multiplier 5^{15} and modulus 2^{47}, Applied Mathematics Division, Argonne National Laboratory.

360

Nancy W. Clark, Subroutine RANF, ANL Subroutine Library Code G552S (for 360/75), multiplicative congruential generator using multiplier $2^{16}+11$ and modulus 2^{31}, Applied Mathematics Division, Argonne National Laboratory.

The programming of the experimental codes as well as the machine computations were done by Claudia Stallings and Burt Garbow. Morris Newman, Jo Ann Howell and Robert Gregory kindly supplied us with binary decks of their routines. For valuable comments either in writing or person, the author thanks G. Birkhoff, W. Cody, G. E. Forsythe, D. E. Knuth, J. D. Lipson, A. Schönhage, R. Tobey, Olga Taussky, J. Todd, and J. Wilkinson. A. S. Householder encouraged presentation of part of this paper at the 1969 Gatlinburg Conference. Don Mazukelli was most helpful in proofreading, and last, but not least, thanks to Doris Haight for her outstanding typing job.

References

BAREISS, E. H. 1966 Multistep integer-preserving Gaussian elimination. *Argonne National Lab. Rep.* No. ANL–7213.

BAREISS, E. H. 1967 The root cubing and the general root powering methods for finding the zeros of polynomials. *Argonne National Lab. Rep.*, No. ANL–7344.

BAREISS, E. H. 1968 *Math. Comp.* **22**(103), 565–578.

BAREISS, E. H. 1968 Numerical inversion of finite Toeplitz matrices and vector Toeplitz matrices. *Argonne National Lab. Rep.*, No. ANL–7740.

BAREISS, E. H. 1969 *Num. Math. 13* 404–424.

BLANKINSHIP, W. A. 1963 *Am. Math. Month.* **70,** 742–745.

BLANKINSHIP, W. A. 1966 *Communs Ass. comput. Mach.* **9,** 513.

BLANKINSHIP, W. A. 1966 *Communs Ass. comput. Mach.* **9,** 514.

BODEWIG, E. 1959 *Matrix calculus.* Amsterdam: North-Holland Publ. Co.

BOOTHROYD, J. 1966 *Communs Ass. comput. Mach.* **9,** 683–684.

BOROSH, I. & FRAENKEL, A. S. 1966 *Math. Comp.* **20,** 107–112.

BRADLEY, G. H. 1970 *Communs Ass. comput. Mach.* **13,** 433–436.

BRADLEY, G. H. 1970 *Communs Ass. comput. Mach.* **13,** 447–448.

BRADLEY, G. H. & WAHI, P. N. 1969 *An algorithm for integer linear programming: a combined algebraic and enumeration approach.* Report No. 29, Administrative Sciences Dept., Yale University.

BRADLEY, G. H. 1970 *Algorithms for Hermite and Smith normal matrices and linear Diophantine equations.* Report No. 34, Administrative Sciences Dept., Yale University.

BROWN, W. S. 1968 The complete Euclidean algorithm. *Bell Telephone Lab. Rep.*

BROWN, W. S. 1971 On Euclid's algorithm and the computation of polynomial greatest common divisors, *Proc. 2nd Symp. Symbolic Algebraic Manipulation.* Los Angeles.

CABAY, S. 1970 Exact solution of linear equations. *Proc. CSC* 2524 *Symbolic Maths Comput.* University of Toronto.

CASSELS, J. W. S. 1959 *An introduction to the geometry of numbers.* Berlin: Springer-Verlag.

COOK, S. A. & AANDERAA, S. O. 1966 *Trans. Am. math. Soc.* **142,** 291–314.

COLLINS, G. E. 1967 *J. Ass. comput. Mach.* **14,** 128–142.

COLLINS, G. E. 1969 *Math. Comp.* **25,** 197–200.

COLLINS, G. E. 1971 Calculation of multivariate polynomial resultants. *Proc. 2nd Symp. Symbolic Algebra.* Los Angeles.

COLLINS, G. E. 1971 The SAC-1 polynomial G.C.D. and resultant system,

DODGSON, G. L. 1866 *Proc. R. Soc.* (*A*), **15,** 150–155.

DURAND, E. 1961 *Solutions numériques des équations algébriques*, **II:** *Systèmes de plusieurs équations. Valeurs propres des matrices*, Paris: Masson et Cie.
FADEEVA, V. N. 1969 *Computational methods of linear algebra.* New York: Dover Publications.
FORSYTHE, G. & MOLER, C. B. 1967 *Computer solution of linear algebraic equations.* Englewood Cliffs: Prentice-Hall.
FOSDICK, L. D. & KIM, Y. J. 1970 *Test matrices I.* Report No. 403, Dept. Comp. Science, University of Illinois.
FOX, L. 1964 *An introduction to numerical linear algebra.* Oxford: Clarendon Press.
FRAZER, R. A., DUNCAN, W. J. & COLLAR, A. R. 1965 *Elementary matrices.* Cambridge University Press.
FRAME, J. S. 1949 *Bull. Am. math. Soc.* **55,** 1045.
FRÖBERG, C. E. & SUNDSTRÖM, A. 1967 *BIT* **7,** 163–169.
GANTMACHER, F. R. 1959 *The theory of matrices.* New York: Chelsea.
GOMORY, R. E. 1965 *Proc. Nat. Acad. Sci. U.S.A.* **53,** 260–265.
GREGORY, R. T. & KARNEY, D. L. 1969 *A collection of matrices for testing computational algorithms.* New York: Wiley.
HERMITE, C. 1851 *J. reine angew. Math.* **41,** 191–216.
HILDEBRAND, F. B. 1956 *Introduction to numerical analysis.* New York: McGraw-Hill.
HOWELL, J. & GREGORY, R. T. 1969 *BIT* **9,** 200–224.
HOWELL, J. & GREGORY, R. T. 1969 *BIT* **9,** 324–337.
HOWELL, J. A. & GREGORY, R. T. 1970 *BIT* **10,** 23–27.
HU, T. C. 1969 *Integer programming and network flows.* Reading, Mass.: Addison-Wesley.
ISRAEL, J. E. 1965 *Inversion of matrices* FORMAC Memorandun, Boston Programming Center.
JACOBSON, 1953 *Lectures in abstract algebra*, **II.** *Linear algebra.* Princeton: Van Nostrand.
KALLMAN, R. E., FALB, P. L. & ARBIB, M. A. 1969 *Topics in mathematical system theory.* New York: McGraw-Hill.
KNUTH, D. E. 1968 *The art of computer programming*, **1.** Fundamental algorithms. Reading, Mass.: Addison-Wesley.
KNUTH, D. E. 1969 *The art of computer programming*, **2.** *Seminumerical algorithms.* Reading, Mass.: Addison-Wesley.
KOWALEWSKI, G. 1948 *Einführung in die Determinantentheorie*, New York: Chelsea.
LIETZKE, M. H., STOUGHTON, R. W. & LIETZKE, MARGORIE P. 1964 *Math. Comp.* **18,** 449–456.
LINDAMOOD, G. E. 1964 Numerical analysis in residue number systems. *Univ. Maryland Comp. Sci. Center Report* TR–64–7.
LIPSON, J. D. 1969 In R. G. Tobey (Ed.), *Proc.* 1968 *Summer Inst. Symbolic Math. Comput.* IBM Programming Laboratory Report FSC69–0312.
LIPSON, J. D. 1970 *Flowgraphs and their generating functions.* Ph.D. Thesis, Harvard University.
LIPSON, J. D. 1970 *PL/1–FORMAC routines for computation with symbolic matrices.* University of Toronto.
LIPSON, J. D. 1971 Chinese remainder and interpolation algorithms. *Proc. 2nd Symp. Symbolic algebraic Manipulation.* Los Angeles.
LOTKIN, M. 1959 *Am. math. Mon.* **66,** 476–479.
LOTKIN, M. 1955 *Math. Comp.* **9,** 153–161.
LUTHER, H. A. & GUSEMAN, L. F. Jr. 1962 *Communs Ass. comput. Mach.* 447–448.
MACDUFFEE, C. C. 1940 *An introduction to abstract algebra.* New York: John Wiley & Sons.
MACMILLAN, R. H. 1955 *J. R. aeronaut. Soc.* **59,** 772ff.
MCCLELLAN, M. T. 1971 The exact solution of linear equations with polynomial coefficients. *Proc. 2nd Symp. Symbolic Algebraic Manipulation.* Los Angeles.
MORRIS, R. 1968 *Commun Ass. comput. Mach.* **11**(1), 38–43.
MUIR, T. *The theory of determinants in historical order of development*, four vols bound as two (I, 1693–1841; II, 1841–1860; III, 1861–1880; IV, 1880–1900). New York: Dover. *Contributions to the history of determinants* 1900–1920. London: Blackie & Sons. 1930 & 1950.

NEWMAN, M. 1967 *J. Res. natn. Bur. Stand.-B*, **71B**, 171–179.
POWER, H. M. 1967 *J. Franklin Inst.*, **283**(3), 214–234.
ROSSER, J. B. 1952 *J. Res. natn. Bur. Stand.* **49**.
SCHÖNHAGE, A. 1966 *Computing*, **1**, 182–196.
SCHÖNAGE, A. & STRASSEN, V. 1971 *Schnelle Multiplikation grosser zahlen, computing* **7**, 281–292.
STRASSEN, V. 1969 *Num. Math.* **13**, 354–356.
SZABÓ, N. S. & TANAKA, R. I. 1967 *Residue arithmetic and its applications to computer technology*. McGraw-Hill: New York.
TAKAHASI, H. & ISHIBASHI, Y. 1961 *Information Processing in Japan*, **1**, 28–42.
TODD, J. 1962 *Survey of numerical analysis*, p. 242. New York: McGraw-Hill.
TOOM, A. L. 1963 *Dokl. Akad. Nauk SSSR* **150**, 496–498. *Soviet Math.* **3**, 714–716.
WILKINSON, J. H. 1967 In M. Klerer & G. A. Korn (Eds), *Digital computer users handbook*. New York: McGraw-Hill.
WINOGRAD, S. 1967 *J. Ass. comput. Mach.* **14**, 793–802.