Computer Sciences Department
The University of Wisconsin
1210 W. Dayton St.
Madison, Wisconsin 53706

THE EXACT SOLUTION OF SYSTEMS OF LINEAR EQUATIONS WITH POLYNOMIAL COEFFICIENTS*

by

Michael T. McClellan†

Technical Report #136 Δ

September 1971

*Research supported by the National Science Foundation (GJ-239) and the National Aeronautics and Space Administration (NGL-21-002-008).

†Present affiliation: Computer Sciences Center, University of Maryland.

ΔAlso the author's Ph.D. thesis at the University of Wisconsin, Madison (Computer Science Department).

ACKNOWLEDGEMENT

My gratitude to Professor George E. Collins for his guidance and continued interest throughout this research. Also, a special thanks to my wife Bonnie.

The support of the National Science Foundation (GJ-239) and the National Aeronautics and Space Administration (NGL-21-002-008) is gratefully acknowledged.

ABSTRACT

Modular algorithms for linear equations solution, matrix inversion, determinant calculation, null space basis generation, and matrix multplication are developed, all for matrices with polynomial entries. Theoretical computing times are obtained for all algorithms. The algorithms are programmed in Fortran IV, forming a module of the SAC-1 system for symbolic and algebraic calculation, and empirical computing times are given for representative cases.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 Problem Description and Background

Let $\mathcal{J}$ be an arbitrary integral domain and let A and B be matrices over $\mathcal{J}$, where A is m by n, nonzero, and of rank r and B is m by q. Consider the matrix equation AX = B, representing q systems of linear equations with the same coefficient matrix, one for each column of B. For simplicity, this equation will be referred to as a system of linear equations. Assuming the system is consistent, it has a <u>particular solution</u> X', an n by q matrix over $Q(\mathcal{J})$, the quotient field of $\mathcal{J}$, such that AX' = B. Moreover, if $r < n$, the null space of A contains nonzero elements. Suppose X" is an n by n-r matrix over $Q(\mathcal{J})$ with linearly independent columns, which solves the homogeneous equation AX = 0. Since the columns of X" then form a basis for the null space of A, such a matrix X" will be referred to as a <u>null space basis</u> for A.

From the basic theory we know that every solution of the system AX = B can be written as the sum of X' and some n by q matrix, each of whose columns is a linear combination of the columns of X". Hence, each such pair (X',X") will be said to constitute a <u>general solution</u> over $Q(\mathcal{J})$ of the system AX = B. Of course, when n = m = r = q and B is the identity

matrix, then X' is the inverse of A and X" is not defined.

In this thesis we will be concerned primarily with integral domains $\mathcal{J}$ which are not fields (i.e., the integers and various polynomial domains). This does not exclude pertinence to systems with coefficients in $Q(\mathcal{J})$, however, for such systems can be transformed to systems with coefficients in $\mathcal{J}$ by initially multiplying the rows of A and B by suitable elements of $\mathcal{J}$. This thesis will be further restricted in that only the problem of computing <u>exact</u> solutions to linear equations will be considered. However, a brief remark concerning approximate solutions can be made at this point. When $\mathcal{J}$ is the ring of integers I or the field of rational numbers, such systems are usually solved by approximate numerical methods such as Gaussian or Gauss-Jordan elimination using floating-point arithmetic on a digital computer. The effect of ill-conditioning on the accuracy of these methods is well-known.

Exact or symbolic methods in the field of rational numbers have offered a solution to the problem of error but at the expense of efficiency. In particular, the computation of greatest common divisors (GCD's) and the phenomenon of integer coefficient growth restrict the applicability of exact Gaussian elimination methods in the rational number field considerably. Moreover, when $\mathcal{J}$ is taken to be $I[x_1,\ldots,x_s]$,

the domain of polynomials in s variables over the integers, or its quotient field, the field of rational functions, then the problem of efficiency is much more serious and is further aggravated by the related phenomenon of degree growth. The occurrence of these phenomena (i.e., integer coefficient growth and degree growth) in the problems of polynomial GCD and resultant calculations has been detailed and major advances in their control have been reported in [COG69b], [BRO68], [COG71a], and [BRO71].

A first step in solving the problem of the computing time efficiency of exact methods for solving linear equations has been to restrict $\mathcal{J}$ to be I or $I[x_1,\ldots,x_s]$ and to perform the elimination using only arithmetic operations in $\mathcal{J}$. That is, if C = (A,B) is the augmented matrix of the linear system, then C is transformed only by elementary row operations in $\mathcal{J}$. One early method was devised by Rosser to control integer coefficient growth in inverting an integral matrix (see [ROS52]). In this method the elimination involves only subtractions of multiples of rows from other rows, where multiplication by small integers is used almost exclusively; no divisions are performed.

Another method, known as the exact division method or exact division algorithm does apply divisions in an attempt to restrict coefficient growth. As the name implies, this

algorithm assures that the divisions are exact, producing quotients in $\mathcal{J}$. This method can be applied in any integral domain, for example, $\mathcal{J} = I$ and $\mathcal{J} = I[x_1,\ldots,x_s]$. In the latter case, the divisions also tend to restrict degree growth as well. See [LIP69], [BAR68], and [LUG62] for recent treatments of this algorithm; an earlier description may be found in [BOD59], pp. 101-109, and a similar description in [FOX64], pp. 82-87. A more general treatment than the above of the exact division algorithm for transforming matrices is given in Chapter 2, where the determinant form of the entries of the transformed matrix are obtained, while allowing row interchanges. In general, no GCD's are computed during the elimination in exact division algorithms; but coefficient and (in the polynomial case) degree growth are still considerable. In solving linear equations, GCD computations are necessary only in the final stage, if the rational form of the solution components is desired.

The approach of obtaining a solution to a linear system $AX = B$ using only arithmetic in $\mathcal{J}$ suggests an alternative form for a general solution. Let $d \in \mathcal{J}$ and Y be an n by q matrix over $\mathcal{J}$, Let Z be an n by $n-r$ matrix over $\mathcal{J}$, if $r < n$, or $Z = 0 \in \mathcal{J}$, if $r = n$. Then (d,Y,Z) will be said to constitute a <u>general solution over $\mathcal{J}$</u> of the linear system $AX = B$ if $AY = d \cdot B$ and Z is a null space basis for A when $r < n$.

The exact division algorithm obtains such a general solution, as will be explained in Chapter 4. When m = n = r = q and B is the identity matrix, then (d,Y) can be considered to represent a matrix inverse for A, since A[(1/d)Y] is the identity matrix. Such a representation of an inverse of a matrix is, of course, not unique, as is also true of a general solution (d,Y,Z) of the equation AX = B. Unless otherwise specified, all matrices and general solutions will be considered in the remainder of this thesis to be over an integral domain $\mathcal{J}$.

During the past decade methods have been developed for solving systems of linear equations, inverting matrices, and performing numerous other algebraic computations using modular arithmetic. Some of the earliest reported applications to these problems may be found in [TAI61], where arithmetic in the finite field GF(p) (i.e., integers modulo p) was employed to invert an integral matrix. Its applications to the solution of linear equations with integer coefficients has been described in [BOF66], [NEW67], and [HOG69]. The former describes the more general case in which rank(A) $\leq$ m $\leq$ n. Some empirical computing time studies were performed; but no computing time analysis was done.

The breakthrough for applications of modular methods involving polynomials came with advances in the exact calcu-

lation of polynomial GCD's. In [COG69b] G. E. Collins outlined an algorithm for doing multivariate polynomial GCD calculation involving the interpolation of polynomials over GF(p). This algorithm has proved to be superior to all others, under theoretical and empirical analysis (see [COG71a], [BRO71]). Complete modular algorithms of this sort apply reductions modulo a prime and Garner's variant of the Chinese Remainder Algorithm to integers and polynomials over the integers. Also, evaluations over GF(p) and interpolation are required for polynomials. These concepts are discussed in Chapter 2.

1.2 A Preview of the Remaining Chapters

Up to this time, the application of modular methods to the solution of linear equations (including matrix inversion) has been limited to the case of integer coefficients and, except for [BOF66], to single systems of linear equations with unique solutions. The algorithms to be presented in this thesis include: 1) a complete modular algorithm for computing a general solution for any consistent system of linear equations with integer or polynomial coefficients, 2) a complete modular algorithm for computing determinants of matrices with integer or polynomial entries, and 3) a complete modular algorithm for computing products of matrices with integer or polynomial entries. Many other algorithms

are given, whose functions will be summarized below.

In Chapter 2, theoretical results are obtained which are applied later in the design of the algorithms. In particular, basic results concerning the exact division algorithm are established, showing that this algorithm computes a canonical form Γ of a matrix -- in the sense that it is computable and uniquely definable by closed formulas, for every matrix. It is shown in the last section of Chapter 2 that this canonical form is also computable over a field by a Gaussian elimination algorithm. This latter algorithm will appear in a different form in Chapter 4 as the nucleus of the modular algorithm for solving linear equations. Sections 2.3-2.5 obtain basic definitions and results concerning the relations of homomorphisms and the mappings Γ. In particular, mod-p and evaluation homomorphisms, and the Chinese Remainder Theorem and interpolation are discussed.

In Chapter 3, algorithms for performing matrix arithmetic are presented: addition, subtraction, and multiplication. Both the classical and the modular algorithms are given for matrix multiplication. Algorithms for applying mod-p and evaluation homomorphisms, Garner's Method and incremental interpolation elementwise to matrices are included and applied immediately in the modular algorithm for matrix multiplication. The inclusion of this algorithm is necessary to

accomplish with the least computing time possible the substitution tests in the modular algorithm for solving linear equations.

In Chapter 4 the modular algorithm for computing general solutions for systems of linear equations is given. The actual algorithm is presented in Sections 4.3-4.5. Basic theoretical results are established in the first section and auxiliary algorithms given in the second section. Following the linear equation solution algorithm, a modular algorithm for computing determinants is presented. Finally, two simple algorithms for accomplishing matrix inversion and null space basis generation are given. Each of these employ the linear equation solution algorithm to accomplish the major portion of the computation.

All the algorithms of Chapters 3 and 4 assume that most of the mathematical objects involved are implemented as list structures in the SAC-1 Mathematical Symbol Manipulation System (see [COG67], [COG68a], [COG68b], [COG68c] [COG69a]). More will be said concerning data structures in the first section of Chapter 3. Also given in this section are the basic concepts of theoretical (a priori) computing time analysis, which will provide the framework for obtaining the computing times of all algorithms in Chapters 3 and 4.

To complement the theoretical computing time analysis of Chapters 3 and 4 we present some empirical computing time studies in Chapter 5. The algorithms considered are those for determinant calculation, solution of linear equations, and matrix inversion.

The algorithms of Chapters 3 and 4 have been implemented as a module of the SAC-1 System (see [COG71c]). All programs have been written in ASA Fortran which is true of all SAC-1 modules (see [ASA64]). The programs are listed in the appendix.

CHAPTER II

THE EXACT COMPUTATION OF A REDUCED ROW ECHELON FORM OF A MATRIX

In this chapter theoretical results are presented for the exact computation of a particular reduced row echelon form of a matrix, that is, the determinantal reduced row echelon form. Section 2.1 presents some preliminary definitions and results concerning the determinantal reduced row echelon form of a matrix over an arbitrary integral domain $\mathcal{J}$. In Section 2.2 an effective mapping is defined for computing exactly this reduced form for a matrix over $\mathcal{J}$. It is shown in these two sections that the elements of this reduced form are defined by explicit closed formulas whose arguments are obtained from inherent properties of the original matrix.

The last four sections of this chapter develop the theoretical framework for alternate algorithms to compute the determinantal reduced row echelon form for particular integral domains. In Section 2.3 the relation between mappings defined on matrices over $\mathcal{J}$ induced by homomorphisms and the effective mapping of Section 2.2 is investigated. One outcome of this investigation is the discovery

of a sufficient condition for commutativity. In Section 2.4 the computation of this reduced form for matrices over the integral domains I and $I[x_1,\ldots,x_s]$ by the method of mod-p mappings is presented. Then in Section 2.5 the computation by the method of evaluation mappings of this reduced form is investigated for matrices over the integral domain $\mathcal{J}[x]$, with special pertinence to $GF(p)[x_1,\ldots,x_s]$. Finally, an effective mapping for computing this reduced row echelon form for matrices over a field is presented in Section 2.6, with potential application to the finite field GF(p).

Section 2.1. Matrices Over An Arbitrary Integral Domain

2.1.1. Preliminary Definitions

Let $\mathcal{J}$ be an arbitrary integral domain and $\mathbb{m}(\mathcal{J})$ denote the set of all matrices over $\mathcal{J}$. Let A be an m by n matrix in $\mathbb{m}(\mathcal{J})$. The rows of A are denoted by $A_1, A_2, \ldots, A_m$. Let $i_1, \ldots, i_s$ and $j_1, \ldots, j_t$, $1 \leq s \leq m$, and $1 \leq t \leq n$, be sequences of integers such that $1 \leq i_k \leq m$ and $1 \leq j_h \leq n$, for $1 \leq k \leq s$ and $1 \leq h \leq t$. The matrix consisting of the elements of A common to rows $i_1, \ldots, i_s$ and columns $j_1, \ldots, j_t$, in that order is denoted by $A\begin{bmatrix} i_1, \ldots, i_s \\ j_1, \ldots, j_t \end{bmatrix}$. If $s = t$, its determinant is denoted by $A\begin{pmatrix} i_1, \ldots, i_s \\ j_1, \ldots, j_s \end{pmatrix}$. Thus $A\begin{bmatrix} i_1, i_2, \ldots, i_s \\ j_1, j_2, \ldots, j_t \end{bmatrix}$ is the s by t matrix $A' \in \mathbb{m}(\mathcal{J})$ with elements $A'(k,h) = A(i_k, j_h)$, $1 \leq k \leq s$ and $1 \leq h \leq t$, and if $s = t$,

$$A\begin{pmatrix} i_1, i_2, \ldots, i_s \\ j_1, j_2, \ldots, j_s \end{pmatrix} = \det\left(A\begin{bmatrix} i_1, i_2, \ldots, i_s \\ j_1, j_2, \ldots, j_s \end{bmatrix}\right) = A'\begin{pmatrix} 1, 2, \ldots, s \\ 1, 2, \ldots, s \end{pmatrix}$$

This notation is a generalization of that used in [GAF59], Chapter 1, where it is required in addition that $i_1 < i_2 < \ldots < i_s$ and $j_1 < j_2 < \ldots < j_s$.

Let Ω be the set of all finite sequences of positive integers, including the null sequence $\emptyset$ of length 0. The lexicographical ordering (>) of Ω is defined as follows.

If one of K and L in Ω is $\emptyset$, then $K > L$ if and only if $K \neq \emptyset$. For two nonnull elements $K = (k_1,\ldots,k_s)$ and $L = (h_1,\ldots,h_t)$ of Ω, we have $K > L$ if and only if (1) either there exists a least integer i: $1 \leq i \leq \min(s,t)$ such that $k_i \neq h_i$, where in fact $k_i > h_i$, or (2) $s > t$ and $k_i = h_i$, $1 \leq i \leq t$. ($K < L$ means $L > K$.) Of course, $K = L$ if and only if $s = t$ and $k_i = h_i$, $1 \leq i \leq s$. We note that Ω is linearly ordered by $>$ and so every finite nonempty subset of Ω has a least element (see [KND68]).

Two subsets of Ω will be of particular importance in this discussion and are defined here. Let $\mathcal{J}$ denote the set of all sequences J of positive integers in ascending order, including $\emptyset$:

$$\mathcal{J} = \{(j_1,j_2,\ldots,j_s): s \geq 0 \text{ and } 1 \leq j_1 < j_2 \ldots < j_s\}.$$

Moreover, let $\mathcal{P}_m$, $m > 0$, denote the set of all permutations on the first m positive integers $1,2,\ldots,m$. Each $\pi \in \mathcal{P}_m$ can be represented uniquely as the sequence $(\pi(1),\pi(2),\ldots\pi(m))$. For μ, $\nu \in \mathcal{P}_m$, the product $\mu\nu$ is defined in the standard way: $\mu\nu = (\mu(\nu(1)), \mu(\nu(2)),\ldots,\mu(\nu(m)))$. Clearly, $\mathcal{P}_m \subset \Omega$. Let $\mathcal{P} = \bigcup_{m=1}^{\infty} \mathcal{P}_m$, the set of all permutations defined in terms of an initial segment of the positive integers. Thus, $\mathcal{J} \subset \Omega$ and $\mathcal{P} \subset \Omega$ and so $>$ is a linear ordering of $\mathcal{J}$ and $\mathcal{P}$. We can extend $>$ to a linear ordering of $\mathcal{J} \times \mathcal{P}$ in the following way.

For any two elements (J, I) and (K, L) of $\mathcal{J} \times \mathcal{P}$, we have (K,L) > (J,I) if and only if either K > J, or K = J and L > I.

We now define for a matrix $A \in \mathfrak{m}(\mathcal{J})$ elements $J_A \in \mathcal{J}$ and $I_A \in \mathcal{P}$, which are uniquely determined by A. We will then define two matrices in $\mathfrak{m}(\mathcal{J})$ corresponding to A which play an essential role in the solution of linear equations.

Let A be an m by n nonzero matrix in $\mathfrak{m}(\mathcal{J})$ of rank r and let $M_j = A\begin{bmatrix}1,2,\ldots,m\\1,2,\ldots,j\end{bmatrix}$, $1 \le j \le n$, the matrix consisting of the first j columns of A. For $1 \le h \le r$, let j_h be the least integer j: $1 \le j \le n$ such that $\operatorname{rank}(M_j) = h$. Since $0 \le \operatorname{rank}(M_k) \le \operatorname{rank}(M_h) \le r$, for $1 \le k \le h \le n$, clearly $1 \le j_1 < j_2 < \ldots < j_r$. Hence, defining $J_A = (j_1, j_2, \ldots, j_r)$, we see that J_A is unique for A and $J_A \in \mathcal{J}$. For A a zero matrix we take $J_A = \emptyset$.

Let A be nonzero as above. Then j_1 exists and there is an integer h_1: $1 \le h_1 \le m$ such that $A(h_1, j_1) \neq 0$ and, hence, such that $A\begin{pmatrix}h_1\\j_1\end{pmatrix} = A(h_1, j_1) \neq 0$. If $r > 1$, suppose inductively that $h_1, \ldots, h_{k-1}$, $1 < k \le r$, distinct integers, $1 \le h_t \le m$, $1 \le t < k$, have been found such that $A\begin{pmatrix}h_1,\ldots,h_s\\j_1,\ldots,j_s\end{pmatrix} \neq 0$, for $1 \le s < k$. Columns $j_1, \ldots, j_k$ of A are linearly independent and so $\operatorname{rank}\left(A\begin{bmatrix}1,\ldots,m\\j_1,\ldots,j_k\end{bmatrix}\right) = k$.

Since the row rank equals the column rank of a matrix, there is a row, say row h_k, of $A\begin{bmatrix}1,\ldots,m\\ j_1,\ldots,j_k\end{bmatrix}$ linearly independent of rows $h_1,\ldots,h_{k-1}$. Rows $h_1,\ldots,h_{k-1}$ of $A\begin{bmatrix}1,\ldots,m\\ j_1,\ldots,j_k\end{bmatrix}$ are linearly independent, since $A\begin{pmatrix}h_1,\ldots,h_{k-1}\\ j_1,\ldots,j_{k-1}\end{pmatrix} \neq 0$, and so $\operatorname{rank}\left(A\begin{bmatrix}h_1,\ldots,h_k\\ j_1,\ldots,j_k\end{bmatrix}\right) = k$. Thus, $A\begin{pmatrix}h_1,\ldots,h_k\\ j_1,\ldots,j_k\end{pmatrix} \neq 0$ and we have defined a sequence $h_1,\ldots,h_r$ of distinct integers, $1 \leq h_t \leq m$, $1 \leq t \leq r$, such that $A\begin{pmatrix}h_1,\ldots,h_s\\ j_1,\ldots,j_s\end{pmatrix} \neq 0$, for $1 \leq s \leq r$. If we let $h_{r+1},\ldots,h_m$ be the remaining integers, in any order, such that $\{h_1,\ldots,h_m\} = \{1,\ldots,m\}$, then $L = (h_1,\ldots,h_m) \in \wp_m$. Define $\wp_A$ as the set of all such sequences L:

$$\wp_A = \left\{(h_1,\ldots,h_m) \in \wp_m : A\begin{pmatrix}h_1,\ldots,h_s\\ j_1,\ldots,j_s\end{pmatrix} \neq 0,\ 1 \leq s \leq r\right\}.$$

Then $\wp_A$ has a minimal element, $I_A = (i_1,\ldots,i_m)$. If A is a zero matrix, we let $I_A = (1,2,\ldots,m)$ and $\wp_A = \wp_m$.

Given the lexicographical ordering $>$ of $\wp$, if A is nonzero, then I_A will be minimal only if i_1 is the least integer i: $1 \leq i \leq m$ such that $A\begin{pmatrix}i_1\\ j_1\end{pmatrix} \neq 0$ and, if $r > 1$, for $s = 2,\ldots,r$, only if i_s is the least integer i: $1 \leq i \leq m$ and $i \neq i_t$, $1 \leq t < s$, such that $A\begin{pmatrix}i_1,..,i_{s-1},i\\ j_1,\ldots\ldots,j_s\end{pmatrix} \neq 0$. Moreover, if $r < m$, I_A will be minimal only if the remaining components satisfy: $1 \leq i_{r+1} < i_{r+2} < \ldots < i_m \leq m$.

We define two special matrices in $\mathfrak{m}(\mathfrak{I})$ corresponding to any given matrix $A \in \mathfrak{m}(\mathfrak{I})$. If A is m by n and nonzero and if $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$, define the m by n matrix $\bar{A}$ in $\mathfrak{m}(\mathfrak{I})$ by:

$$\bar{A}(k,j) = \begin{cases} A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_{k-1},j \end{pmatrix}, & 1 \le k \le r \text{ and } 1 \le j \le n \\ 0, & r < k \le m \text{ and } 1 \le j \le n \end{cases}$$

Similarly, we define the m by n matrix $\bar{\bar{A}}$ in $\mathfrak{m}(\mathfrak{I})$ by:

$$\bar{\bar{A}}(k,j) = \begin{cases} A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_{k-1},j,j_{k+1},\ldots,j_r \end{pmatrix}, & 1 \le k \le r \text{ and } 1 \le j \le n \\ 0, & r < k \le m \text{ and } 1 \le j \le n. \end{cases}$$

If A is a zero matrix, then we take $\bar{A} = \bar{\bar{A}} = A$. Hereafter, for every matrix $A \in \mathfrak{m}(\mathfrak{I})$, $\bar{A}$ and $\bar{\bar{A}}$ will be the matrices so defined.

2.1.2. Row Echelon Matrices and Row Equivalence. In Chapter 7 of [BIG65] row echelon matrices and row equivalence are discussed for matrices over a field. In what follows these concepts will be treated for matrices over an integral domain and some basic results derived.

An m by n nonzero matrix $B \in \mathbb{M}(\mathcal{I})$ is a row echelon (RE) matrix (is in row echelon (RE) form) if there is a sequence of integers $1 \leq k_1 < k_2 < \ldots < k_s \leq n$, for some s: $1 \leq s \leq m$, such that, for $1 \leq i \leq s$, the i^{th} row B_i of B satisfies: $B(i,k_i) \neq 0$ and $B(i,j) = 0$, $1 \leq j < k_i$. If $s < m$, the rows $B_{s+1},\ldots,B_m$ are zero. Trivially, we say that a zero matrix in $\mathbb{M}(\mathcal{I})$ is the case $s = 0$ and is in RE form. The sequence $K = (k_1,\ldots,k_s)$, for $s > 0$, or $K = \emptyset$, for $s = 0$, is called the row echelon (RE) sequence for the RE matrix B. Furthermore, an m by n nonzero matrix $C \in \mathbb{M}(\mathcal{I})$ is called a reduced row echelon (RRE) matrix (is in reduced row echelon (RRE) form) if C is an RE matrix and, in addition, satisfies the condition that the only nonzero element in column k_i, $1 \leq i \leq s$, is $C(i,k_i) \neq 0$, where $K = (k_1,\ldots,k_s)$ is the RE sequence for C. The elements $C(i,k_i)$, $1 \leq i \leq s$, constitute the diagonal of C. Trivially, a zero matrix is also in RRE form.

The following theorem shows that the matrices $\bar{A}$ and $\bar{\bar{A}}$ for $A \in \mathbb{M}(\mathcal{I})$ are, in fact, RE and RRE matrices, respectively,

in $\mathbb{M}(\mathcal{J})$.

Theorem 2.1.2.1. For every matrix $A \in \mathbb{M}(\mathcal{J})$, $\bar{A}$ is an RE matrix and $\bar{\bar{A}}$ is an RRE matrix, both with RE sequence J_A.

Proof: If A is a zero matrix, then $\bar{\bar{A}} = \bar{A} = A$ and $J_A = \emptyset$ is their common RE sequence. Suppose A is m by n and non-zero, with $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$. Of course, for $1 \le k \le r$, $\bar{A}(k,j_k) = A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix} \neq 0$ and $\bar{\bar{A}}(k,j_k) = A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_r \end{pmatrix} \neq 0$, by the definitions of J_A and I_A. Consider rows k: $1 \le k \le r$ of $\bar{A}$ and $\bar{\bar{A}}$. For $1 \le h < j_k$, $\operatorname{rank}\left(A\begin{bmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_{k-1},h \end{bmatrix}\right) \le \operatorname{rank}\left(A\begin{bmatrix} 1,\ldots,m \\ j_1,\ldots,j_{k-1},h \end{bmatrix}\right) \le k-1$ and so $\bar{A}(k,h) = A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_{k-1},h \end{pmatrix} = 0$. Similarly, for $1 \le h < j_k$,

$$\operatorname{rank}\left(A\begin{bmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_{k-1},h,j_{k+1},\ldots,j_r \end{bmatrix}\right) \le \operatorname{rank}\left(A\begin{bmatrix} 1,2,\ldots,m \\ 1,2,\ldots,j_{k-1},h,j_{k+1},\ldots,j_r \end{bmatrix}\right) \le \operatorname{rank}\left(A\begin{bmatrix} 1,\ldots,m \\ 1,\ldots,j_{k-1},h \end{bmatrix}\right) + (r-k),$$

for by adjoining r - k columns to a matrix we increase its rank by at most r - k. Thus, $\operatorname{rank}\left(A\begin{bmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_{k-1},h,j_{k+1},\ldots,j_r \end{bmatrix}\right) \le (k-1) + (r-k) = r-1$ and so $\bar{\bar{A}}(k,h) = A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_{k-1},h,j_{k+1},\ldots,j_r \end{pmatrix} = 0$. Moreover, for $1 \le k \le r$ and $1 \le h \le r$, $h \neq k$, $\bar{\bar{A}}(k,j_h) =$

$A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_{k-1},j_h,j_{k+1},\ldots,j_r \end{pmatrix} = 0$. Since rows r+1,...,m of both $\bar{A}$ and $\bar{\bar{A}}$ are zero by definition, it has been shown that $\bar{A}$ and $\bar{\bar{A}}$ satisfy the definitions for RE and RRE matrices, respectively.//

For A nonzero with $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$ we define $\delta(A) = A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_r \end{pmatrix}$ and for A a zero matrix we define $\delta(A) = 0$. Thus, if A is nonzero, the diagonal elements $\bar{\bar{A}}(i,j_i)$, $1 \le i \le r$, of $\bar{\bar{A}}$ all equal $\delta(A)$.

The following definitions for matrices over an integral domain are based on the usual definitions for matrices over a field. If A, B $\in \mathbb{m}(\mathcal{J})$, $\mathcal{J}$ an arbitrary integral domain, then B is <u>row-equivalent</u> to A in $\mathbb{m}(\mathcal{J})$ if A can be transformed to B by <u>elementary row operations</u> in $\mathbb{m}(\mathcal{J})$. These operations are: interchanging two rows, multiplying or dividing a row by a nonzero element $c \in \mathcal{J}$, and adding to a row another row multiplied or divided by an element $c \in \mathcal{J}$, which is nonzero for division. We note that a row A_i of an m by n matrix $A \in \mathbb{m}(\mathcal{J})$ may be divided by a nonzero $c \in \mathcal{J}$ only if each element satisfies: $A(i,j) = c \cdot b_j$, $b_j \in \mathcal{J}$, for $1 \le j \le n$.

The term "row echelon" actually pertains to all matrices in $\mathbb{m}(\mathcal{J})$. We say that a matrix $B \in \mathbb{m}(\mathcal{J})$ is <u>a row echelon</u>

(RE) form for a matrix $A \in \mathfrak{m}(\mathcal{J})$ if B is a RE matrix row-equivalent to A. A sequence $K \in \mathcal{J}$ is called a row echelon (RE) sequence for a matrix $A \in \mathfrak{m}(\mathcal{J})$, if A has an RE form $B \in \mathfrak{m}(\mathcal{J})$ with RE sequence K. Moreover, a matrix $C \in \mathfrak{m}(\mathcal{J})$ is called a reduced row echelon (RRE) form for $A \in \mathfrak{m}(\mathcal{J})$ if C is an RRE matrix row-equivalent to A. Section 2.2 will be devoted to showing that $\bar{A}$ is an RE form in $\mathfrak{m}(\mathcal{J})$ for A and then that $\bar{\bar{A}}$ is an RRE form in $\mathfrak{m}(\mathcal{J})$ for A.

The following theorem shows that, if a matrix in $\mathfrak{m}(\mathcal{J})$ has an RE sequence, then it is unique.

Theorem 2.1.2.2. If A is a matrix in $\mathfrak{m}(\mathcal{J})$ with an RE sequence K, then $K = J_A$.

Proof: Case 1: A is an RE matrix. If A is a zero matrix, then $K = \emptyset = J_A$. Suppose A is an m by n nonzero matrix and let $K = (k_1,\ldots,k_s)$ and $J_A = (j_1,\ldots,j_r)$, for $r > 0$ and $s > 0$. For each $j = 1,2,\ldots,n$ let $M_j = A\begin{bmatrix}1,\ldots,m\\1,\ldots,j\end{bmatrix}$. Then j_t is the least integer j: $1 \le j \le n$ for which $\text{rank}(M_j) = t$, for $1 \le t \le r$, where $r = \text{rank}(A)$. If $k_1 > 1$, $\text{rank}(M_j) = 0$, for $1 \le j < k_1$, since each M_j is a zero matrix. Let $k_{s+1} = n+1$. Then, for each $t = 1,2,\ldots,s$, $\text{rank}(M_j) = t$, where $k_t \le j < k_{t+1}$. The reason for this is as follows. Only the first t rows of M_j are nonzero,

for $k_t \leq j < k_{t+1}$, and so $\text{rank}(M_j) \leq t$. However, columns $k_1,\ldots,k_t$ of M_j are linearly independent and so $\text{rank}(M_j) \geq t$, $k_t \leq j < k_{t+1}$. Thus, $\text{rank}(M_j) = t$. Hence, for each t: $1 \leq t \leq s$, k_t is the least integer j: $1 \leq j \leq n$ such that $\text{rank}(M_j) = t$ and so $k_t = j_t$ for $1 \leq t \leq \min(r,s)$. Therefore, since $r = \text{rank}(A) = \text{rank}(M_n) = s$, we have $K = J_A$, when A is an RE matrix.

Case 2: A is not an RE matrix. Of course, A is non-zero. Let A be m by n and let $B \in \mathcal{M}(\mathcal{I})$ be an RE form for A with RE sequence $K = (k_1,\ldots,k_r)$. For each $j = 1,2,\ldots,n$, let $M_j = A\begin{bmatrix}1,\ldots,m\\1,\ldots,j\end{bmatrix}$ and $N_j = B\begin{bmatrix}1,\ldots,m\\1,\ldots,j\end{bmatrix}$. We know $K = J_B$ by Case 1 and so k_t is the least integer j: $1 \leq j \leq n$ such that $\text{rank}(N_j) = t$, for $1 \leq t \leq r$. Since the rank of a matrix is invariant under elementary row transformations and since the same sequence of elementary row operations which transformed A to B will transform M_j to N_j, $1 \leq j \leq n$, then $\text{rank}(M_j) = \text{rank}(N_j)$. Note that $\text{rank}(A) = \text{rank}(B) = r$. Thus, k_t is the least integer j: $1 \leq j \leq n$ such that $\text{rank}(M_j) = t$, for $1 \leq t \leq r$, and so $J_A = (k_1,\ldots,k_r) = K$.//

Let E be an m by n matrix in $\mathcal{M}(\mathcal{I})$. Let $\epsilon = \epsilon_{(i)(j)}$, $1 \leq i, j \leq m$, represent the operation of interchanging rows i and j of E, giving $F = \epsilon(E) \in \mathcal{M}(\mathcal{I})$. The mapping ϵ produces

a permutation $\pi = \pi_{(i)(j)} \in \mathcal{P}_m$ of the rows of E. That is, $F_{\pi(h)} = E_h$, for $1 \leq h \leq m$, where $\pi(h) = h$, for $1 \leq h \leq m$, $h \neq i$, $h \neq j$, and also $\pi(i) = j$ and $\pi(j) = i$. π will be called a row-interchange permutation. The following result explicitly defines the permutation of rows which results from applying a sequence of row interchanges to a matrix.

Theorem 2.1.2.3. Let E be an m by n matrix in $\mathcal{M}(\mathcal{I})$. For each t: $1 \leq t \leq s$, $s \geq 1$, suppose $\epsilon_t = \epsilon_{(i_t)(j_t)}$, a row-interchange mapping, and $\pi_t = \pi_{(i_t)(j_t)} \in \mathcal{P}_m$, where $1 \leq i_t$, $j_t \leq m$. Let $\gamma = \epsilon_s \cdots \epsilon_1$ and $\tau = \pi_s \cdots \pi_1$. If $F = \gamma(E)$, then $F_{\tau(h)} = E_h$, for $1 \leq h \leq m$.

Proof: For each $t = 1,2,\ldots,s$ define $G^{(t)} = \epsilon_t(G^{(t-1)})$, where $G^{(0)} = E$. For $t = 1$, $G^{(1)} = \epsilon_1(G^{(0)}) = \epsilon_1(E)$ and hence $G^{(1)}_{\pi_1(h)} = E_h$, $1 \leq h \leq m$. If $s > 1$, suppose inductively for $1 < t \leq s$ that $G^{(t-1)}_{\pi_{t-1} \cdots \pi_1(h)} = E_h$, $1 \leq h \leq m$. By definition $G^{(t)} = \epsilon_t(G^{(t-1)})$ and so $G^{(t)}_{\pi_t(h)} = G^{(t-1)}_h$, $1 \leq h \leq m$. Since each of $\pi_{t-1},\ldots,\pi_1$ is a one-to-one mapping of $\{1,2,\ldots,m\}$ onto $\{1,2,\ldots,m\}$, so is the product $\pi_{t-1} \cdots \pi_1$; hence, $\{\pi_{t-1} \cdots \pi_1(h) : 1 \leq h \leq m\} = \{1,2,\ldots,m\}$. It follows that $G^{(t)}_{\pi_t \cdot \pi_{t-1} \cdots \pi_1(h)} =$

$G^{(t)}_{\pi_t(\pi_{t-1} \cdots \pi_1(h))} = G^{(t-1)}_{\pi_{t-1} \cdots \pi_1(h)} = E_h$, for $1 \leq h \leq m$,

by applying the inductive hypothesis, completing the inductive step.//

The permutation τ of the rows of E produced by the mapping γ is called a row permutation of E.

Section 2.2. The Exact Computation of a Reduced Row Echelon Form of a Matrix Over an Arbitrary Integral Domain

In this section an effective mapping Γ of $\mathbb{M}(\mathcal{I})$ into $\mathbb{M}(\mathcal{I})$, for $\mathcal{I}$ an arbitrary integral domain, is constructed such that $\Gamma(A) = \bar{\bar{A}}$, for every $A \in \mathbb{M}(\mathcal{I})$. Γ will apply only elementary row operations in $\mathbb{M}(\mathcal{I})$ in the transformation of A to $\bar{\bar{A}}$, showing that $\bar{\bar{A}}$ is row equivalent to A in $\mathbb{M}(\mathcal{I})$. It will then follow that $\bar{\bar{A}}$ is an RRE form in $\mathbb{M}(\mathcal{I})$ for A.

In subsection 2.2.1 an effective mapping Γ_1 of $\mathbb{M}(\mathcal{I})$ into $\mathbb{M}(\mathcal{I}) \times \mathcal{J} \times \mathcal{P}$ is constructed such that $\Gamma_1(A) = (\bar{A}, J_A, I_A)$, for every $A \in \mathbb{M}(\mathcal{I})$. Then in subsection 2.2.2 Γ_1 is employed to construct the mapping Γ. This mapping is a general version of the exact division method for triangularizing and then diagonalizing a matrix over an integral domain. See [LIP69], [LUG62] and [BAR68] for recent alternative treatments of this method. Earlier descriptions may be found in [BOD59], pp. 101-109, and in [FOX64], pp. 82-87.

2.2.1. The Computation of a Row Echelon Form by Exact Division

The effective mapping Γ_1 such that $\Gamma_1(A) = (\bar{A}, J_A, I_A)$, for every $A \in \mathbb{M}(\mathcal{I})$, will be defined by the following algorithm.

That this algorithm is, in fact, an algorithm or effective mapping will be indicated in the subsequent discussion. In the remainder of this subsection it will be shown that the output of the algorithm is actually $(\bar{A}, J_A, I_A)$ and that $\bar{A}$ is an RE form for A.

Algorithm 2.2.1.1. Transforming a Matrix Over an Integral Domain $\mathcal{J}$ to Row Echelon Form by Exact Division

Input: An m by n matrix $A \in \mathbb{M}(\mathcal{J})$.

Output: The triple $(B, J, I) \in \mathbb{M}(\mathcal{J}) \times \mathcal{J} \times \mathcal{P}$, where

$$B = \bar{A},\ J = J_A,\ \text{and}\ I = I_A.$$

(1) [Initialize.] $B \leftarrow A$; $J \leftarrow \emptyset$; for $1 \le h \le m$, $I[h] \leftarrow h$; $k \leftarrow 1$; $s \leftarrow 0$; $D \leftarrow 1$, the identity of $\mathcal{J}$.

(2) [Locate a nonzero pivot, if any.] $s \leftarrow s+1$; if $s > n$, go to (6); for $t = k, k+1, \ldots, m$, search for $B(t,s) \neq 0$; if not found, go to (2); suffix s to the sequence J; $h \leftarrow k$.

(3) [Change the row order, if necessary.] If $t = k$, go to (4); $E \leftarrow B_t$; $f \leftarrow I[t]$; for $u = t, t-1, \ldots, k+1$, set $B_u \leftarrow B_{u-1}$ and $I[u] \leftarrow I[u-1]$; then set $B_k \leftarrow E$ and $I[k] \leftarrow f$.

(4) [Recompute rows $B_{k+1}, \ldots, B_m$, if any.] $h \leftarrow h+1$; if $h > m$, go to (5); $G \leftarrow B(h,s)$; compute $B_h \leftarrow [B(k,s) \cdot B_h - G \cdot B_k]/D$; go to (4).

(5) [Initialize next pivot operation, if necessary.] If $k = m$, go to (6); $D \leftarrow B(k,s)$; $k \leftarrow k+1$; go to (2).

(6) [Return] Return B, J, and I.

It will not be proven rigorously that this is an algorithm in a formal sense. Suffice it to observe in the following discussion that Algorithm 2.2.1.1 will terminate in a finite number of finite, well defined, and unambiguous steps, for every $A \in \mathbb{m}(\mathcal{J})$. Three sequences are generated: $(B^{(0)}, B^{(1)},\ldots,B^{(q)})$, $(J_0, J_1,\ldots,J_q)$, and $(I_0, I_1,\ldots,I_q)$, $q \geq 0$, where $B^{(0)} = A$, $J_0 = \emptyset$, and $I_0 = (i_{0,1},\ldots,i_{0,m}) = (1,2,..,m)$, for A m by n. We will show that, for $0 \leq k \leq q$, $J_k \in \mathcal{J}$, $I_k \in \wp_m$, and all matrices, including $B^{(k)}$, are m by n matrices in $\mathbb{m}(\mathcal{J})$. The algorithm sets $B = B^{(q)}$, $J = J_q$, and $I = I_q$. We will see that B is row equivalent to A. Finally, after establishing some lemmas, it will be proven in Theorem 2.2.1.4 that $J = J_A$, $I = I_A$, and $B = B_A$, a RE form for A in $\mathbb{m}(\mathcal{J})$, where $\mathcal{J}$ is an arbitrary integral domain.

There are q pivot operations performed, the k^{th} of which is accomplished in steps (2)-(5) and produces $B^{(k)}$, J_k, and I_k, $1 \leq k \leq q$. We note that, for $k = 0$, $B^{(o)} = A$ is an m by n matrix in $\mathbb{m}(\mathcal{J})$, $J_o = \emptyset \in \mathcal{J}$, and $I_o \in \wp_m$. Assume inductively for $0 < k \leq q+1$ that $B^{(k-1)}$, an m by n matrix in $\mathbb{m}(\mathcal{J})$, $J_{k-1} = (j'_1,\ldots,j'_{k-1}) \in \mathcal{J}$, and $I_{k-1} =$

$(i_{k-1,1},\ldots,i_{k-1,m}) \in \wp_m$ have been computed.

The k^{th} pivot operation begins with the pivot search in step (2), with initialization from step (1) or step (5). If either 1) $j'_{k-1} = n$, or 2) $k > m$, or 3) all $B^{(k-1)}(i,j) = 0$, for $k \le i \le m$ and $j'_{k-1} < j \le n$, then the k^{th} pivot operation cannot be performed and the algorithm terminates with $B = B^{(q)}$, $J = J_q$, and $I = I_q$, where $q = k-1$. The case $q = 0$ corresponds to the case of A a zero matrix, whence $B = B^{(o)} = A$, $J = J_o = \emptyset$, and $I = I_o = (1,2,\ldots,m)$. Suppose A is nonzero and the k^{th} pivot $B^{(k-1)}(t,s) \neq 0$ does exist. Then $j'_k = s$ is the least integer j: $j'_{k-1} < j \le n$ such that $B^{(k-1)}(h,j) \neq 0$, for some h: $k \le h \le m$, and t is the least such h. Thus, $J_k = (j'_1,\ldots,j'_{k-1},j'_k) \in \mathcal{J}$.

In step (3) I_k and an m by n matrix $\tilde{B}^{(k)} \in \mathbb{M}(\mathcal{J})$ are computed, which computation is described as follows. Let $\epsilon_{k,o} = \epsilon_{(k)(k)}$ and $\pi_{k,o} = \pi_{(k)(k)} \in \wp_m$, and let $s_k = t-k \ge 0$. If $s_k > 0$, let $\epsilon_{k,u} = \epsilon_{(k+u-1)(k+u)}$ and $\pi_{k,u} = \pi_{(k+u-1)(k+u)} \in \wp_m$, for $1 \le u \le s_k$. If we let $\gamma_k = \epsilon_{k,o} \cdot \epsilon_{k,1} \cdots \epsilon_{k,s_k}$ and $\tau_k = \pi_{k,o}\, \pi_{k,1} \cdots \pi_{k,s_k}$, then step (3) computes $\tilde{B}^{(k)} = \gamma_k(B^{(k-1)})$ and by Theorem 2.1.2.3,

$\tilde{B}^{(k)}_{\tau_k(h)} = B^{(k-1)}_h$, for $1 \le h \le m$. It can be easily seen that the row permutation τ_k is the permutation:

$$\tau_k(h) = \begin{cases} h, & \text{if } 1 \le h < k \text{ or } t < h \le m \\ k, & \text{if } h = t \\ h+1, & \text{if } k \le h < t \end{cases}$$

If $\sigma_k = \tau_k^{-1}$, then σ_k is defined by:

$$\sigma_k(h) = \begin{cases} h, & \text{if } 1 \le h < k \text{ or } t < h \le m \\ t, & \text{if } h = k \\ h-1, & \text{if } k < h \le t. \end{cases}$$

Note that $\tilde{B}^{(k)}_h = \tilde{B}^{(k)}_{\tau_k(\sigma_k(h))} = B^{(k-1)}_{\sigma_k(h)}$, $1 \le h \le m$.

In step (3) $I_k = (i_{k,1},\ldots,i_{k,m})$ is computed from I_{k-1} such that $i_{k,h} = i_{k-1,\sigma_k(h)}$, $1 \le h \le m$. Let $\sigma_o = (1,2,\ldots,m) = I_o$ and assume inductively that $I_{k-1} = \sigma_o \cdots \sigma_{k-1}$, for $k > 0$. Then $i_{k-1,u} = \sigma_o \cdots \sigma_{k-1}(u)$, $1 \le u \le m$, and so $i_{k-1,\sigma_k(h)} = \sigma_o \cdots \sigma_{k-1}(\sigma_k(h)) = \sigma_o \cdots \sigma_k(h)$, $1 \le h \le m$. Thus, $i_{k,h} = \sigma_o \cdots \sigma_k(h)$, $1 \le h \le m$, and so $I_k = \sigma_o \cdots \sigma_k = I_{k-1}\,\sigma_k \in \mathcal{P}_m$.

In step (4) the k^{th} pivot operation is completed by transforming $\tilde{B}^{(k)}$ to an m by n matrix $B^{(k)} \in \mathbb{M}(\mathcal{J})$ by elementary row operations other than row interchanges. More specifically, $B^{(k)}_h = \tilde{B}^{(k)}_h$, for $1 \le h \le k$, and, if $k < m$,

then for $k < h \leq m$, $B_h^{(k)}$ is computed by multiplying $\tilde{B}_h^{(k)}$ by $\tilde{B}^{(k)}(k,j_k')$, then adding $\tilde{B}_k^{(k)}$ multiplied by $-\tilde{B}^{(k)}(h,j_k')$, and finally dividing by $\tilde{B}^{(k)}(k-1,j_{k-1}') \neq 0$: $B_h^{(k)} =$

$$\left[\tilde{B}^{(k)}(k,j_k')\cdot\tilde{B}_h^{(k)} - \tilde{B}^{(k)}(h,j_k')\cdot\tilde{B}_k^{(k)}\right]/\tilde{B}^{(k)}(k-1,j_{k-1}').$$

To assure that $B^{(k)} \in \mathbb{M}(\mathcal{J})$, for $\mathcal{J}$ an arbitrary integral domain, it must be shown that the division is possible in $\mathcal{J}$. This will be done in Theorem 2.2.1.4. However, assuming that $\mathcal{J}$ is a field, it has already been shown that $B^{(k)} \in \mathbb{M}(\mathcal{J})$ is produced from $B^{(k-1)}$ by elementary row operations in $\mathbb{M}(\mathcal{J})$ and so is row equivalent in $\mathbb{M}(\mathcal{J})$ to $B^{(k-1)}$. This completes the discussion of the k^{th} pivot operation. We note that, since $B^{(k)}$ is row equivalent to $B^{(k-1)}$, for $k = 1,2,\ldots,q$, then $B = B^{(q)}$ is obtained from $B^{(o)} = A$ by elementary row operations in $\mathbb{M}(\mathcal{J})$ and, hence, B is row equivalent to A.

We now present two lemmas which will be required for Theorem 2.2.1.4. The first concerns the permutations I_k and the second concerns the matrices $B^{(k)}$.

Lemma 2.2.1.2. Let $I_o, I_1,\ldots,I_q$, where $I_k = (i_{k,1},\ldots,i_{k,m}) \in \mathcal{P}_m$, $0 \leq k \leq q$, be the sequence of permutations generated by Algorithm 2.2.1.1 for an m by n matrix $A \in \mathbb{M}(\mathcal{J})$. Then $1 \leq i_{k,k+1} < i_{k,k+2} < \cdots < i_{k,m} \leq m$, for each k: $0 \leq k \leq q$.

Proof: We have initially that $1 \le i_{o,1} < i_{o,2} <\cdots< i_{o,m} \le m$, since $i_{o,h} = h$, $1 \le h \le m$. If $q > 0$, suppose inductively, for k: $0 < k \le q$, that $1 \le i_{k-1,k} <\cdots< i_{k-1,m} \le m$. Let $t = \sigma_k(k)$ and recall that $i_{k,h} = i_{k-1,\sigma_k(h)}$, $1 \le h \le m$. For $k < h \le t$, $\sigma_k(h) = h-1$ and so $i_{k,h} = i_{k-1,h-1}$. Applying the inductive hypothesis we have $i_{k,k+1} < i_{k,k+2} <\cdots< i_{k,t}$. For $t < h \le m$, $\sigma_k(h) = h$ and so $i_{k,h} = i_{k-1,h}$. Again applying the inductive hypothesis we have $i_{k,t+1} < i_{k,t+2} <\cdots< i_{k,m}$. Also, $i_{k,t} < i_{k,t+1}$. since we have by the previous observations that $i_{k,t} = i_{k-1,t-1} < i_{k-1,t+1} = i_{k,t+1}$. Thus, $i_{k,k+1} <\cdots< i_{k,t} < i_{k,t+1} <\cdots< i_{k,m}$. Of course, $1 \le i_{k,k+1}$ and $i_{k,m} \le m$, completing the inductive step.//

The next lemma, stated without proof, follows as a direct consequence of Sylvester's determinant identity, as stated in [GAF59].

Lemma 2.2.1.3. Let M be an $s + 2$ by $s + 2$ matrix in $\mathfrak{m}(\mathfrak{I})$, $s \ge 1$. Then

$$M\begin{pmatrix}1,\ldots,s+1\\1,\ldots,s+1\end{pmatrix}\cdot M\begin{pmatrix}1,\ldots,s,s+2\\1,\ldots,s,s+2\end{pmatrix}-M\begin{pmatrix}1,\ldots,,s+1\\1,\ldots,s,s+2\end{pmatrix}\cdot M\begin{pmatrix}1,\ldots,s,s+2\\1,\ldots,,s+1\end{pmatrix}$$

$$= M\begin{pmatrix}1,\ldots,s\\1,\ldots,s\end{pmatrix}\cdot M\begin{pmatrix}1,\ldots,s+2\\1,\ldots,s+2\end{pmatrix} .//$$

We now state and prove the key theorem of this subsection, establishing our earlier assertions about the mapping

Γ_1 defined by Algorithm 2.2.1.1.

Theorem 2.2.1.4. Let $A \in \mathbb{m}(\mathcal{J})$, $\mathcal{J}$ an arbitrary integral domain, and suppose $\Gamma_1(A) = (B,J,I)$. Then $J = J_A$, $I = I_A$, and $B = \bar{A}$, where $\bar{A}$ is an RE form in $\mathbb{m}(\mathcal{J})$ for A.

Proof: Let A be m by n and let $(B^{(o)}, B^{(1)},\ldots,B^{(q)})$, $(J_o, J_1,\ldots,J_q)$, and $(I_o, I_1,\ldots,I_q)$ be the sequences generated by Γ_1. We have seen that, if A is a zero matrix, then $J = \emptyset = J_A$, $I = (1,\ldots,m) = I_A$, and $B = A = \bar{A}$, trivially an RE form for A. For A nonzero, let $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$. Then $q > 0$ and we let $J = (j'_1,\ldots,j'_q)$ and $I = (i'_1,\ldots,i'_m)$. Let the inductive hypothesis for v: $0 \le v < q$ be: $J_v = (j'_1,\ldots,j'_v) = (j_1,\ldots,j_v)$, $I_v = (i_{v,1},\ldots,i_{v,m}) = (i_1,\ldots,i_v, i_{v,v+1},\ldots,i_{v,m})$, and $B^{(v)}$ is row equivalent in $\mathbb{m}(\mathcal{J})$ to A and is defined by:

$$B^{(v)}(h,j) = \begin{cases} A\begin{pmatrix} i_1,\ldots,i_{h-1},i_h \\ j_1,\ldots,j_{h-1},j \end{pmatrix}, & 1 \le h \le v \text{ and } 1 \le j \le n \\ A\begin{pmatrix} i_1,\ldots,i_v,i_{v,h} \\ j_1,\ldots,j_v,j \end{pmatrix}, & v < h \le m \text{ and } 1 \le j \le n. \end{cases}$$

For v = 0, the inductive hypothesis is satisfied vacuously by I_o and J_o. Also, $B^{(o)}$ is trivially row equivalent in $\mathbb{m}(\mathcal{J})$ to A and is defined by : $B^{(o)}(h,j) = A(h,j) = A\begin{pmatrix} h \\ j \end{pmatrix} = A\begin{pmatrix} i_o,h \\ j \end{pmatrix}$, for $0 = v < h \le m$ and $1 \le j \le n$. We

suppose that the inductive hypothesis is true for $v = k-1$, $0 \leq k-1 < q$, and show that it holds for $v = k$. Let $j_o = j'_o = 0$. We know that j'_k is the least integer j: $j_{k-1} = j'_{k-1} < j \leq n$ such that $A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1,h} \\ j_1,\ldots,j_{k-1},j \end{pmatrix} = B^{(k-1)}(h,j) \neq 0$, for some h: $k \leq h \leq m$. Hence, $r \geq k$, since $r = \text{rank}(A) \geq \text{rank}\left(A\begin{bmatrix} i_1,\ldots,i_{k-1},i_{k-1,h} \\ j_1,\ldots,j_{k-1},j \end{bmatrix}\right) = k$, and so j_k and i_k exist.

We first show that $j'_k = j_k$. We have $\text{rank}\left(A\begin{bmatrix} 1,\ldots,m \\ 1,\ldots,j'_k \end{bmatrix}\right) \geq \text{rank}\left(A\begin{bmatrix} i_1,\ldots,i_{k-1},i_{k-1,h} \\ j_1,\ldots,j_{k-1},j'_k \end{bmatrix}\right) = k$, for some $h \geq k$. However, j_k is the least j such that $\text{rank}\left(A\begin{bmatrix} 1,\ldots,m \\ 1,\ldots,j \end{bmatrix}\right) \geq k$ and so $j_k \leq j'_k$. If $j'_k > j_k$, then j_k is an integer j: $j'_{k-1} < j < j'_k$ and so $A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1,h} \\ j_1,\ldots,j_{k-1},j_k \end{pmatrix} = B^{(k-1)}(h,j_k) = 0$, for all h: $k \leq h \leq m$. However, $i_k = i_{k-1,h'}$ for some h': $k \leq h' \leq m$ and so $A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1,h'} \\ j_1,\ldots,j_k \end{pmatrix} = A\begin{pmatrix} i_1,\ldots,i_{k-1},i_k \\ j_1,\ldots,j_k \end{pmatrix} \neq 0$, a contradiction. Therefore, $j'_k = j_k$.

We next show that $i'_k = i_k$. We know that $t = \sigma_k(k)$ is the least integer h: $k \leq h \leq m$ such that $B^{(k-1)}(h,j_k) = A\begin{pmatrix} i_1,\ldots,i_{k-1},\ i_{k-1,h} \\ j_1,\ldots,j_k \end{pmatrix} \neq 0$. By Lemma 2.2.1.2 we also know that $1 \leq i_{k-1,k} < i_{k-1,k+1} <\cdots< i_{k-1,m} \leq m$. Therefore, $i_{k-1,t}$ is the least integer i: $1 \leq i \leq m$ and $i \neq i_u$,

$1 \le u < k$, such that $A\begin{pmatrix} i_1,\ldots,i_{k-1},i \\ j_1,\ldots,\ldots,j_k \end{pmatrix} \neq 0$, and so $i_{k-1,t} = i_k$. Since $i_{k,u} = i_{k-1,\sigma_k(u)} = i_{k-1,u}$, for $1 \le u < k$, and since $i_{k,k} = i_{k-1,\sigma_k(k)} = i_{k-1,t} = i_k$, we have that $I_k = (i_1,\ldots,i_k,i_{k,k+1},\ldots,i_{k,m})$.

We now show that $B^{(k)}$ satisfies the inductive hypothesis. For $1 \le h < k$, $B_h^{(k)} = \tilde{B}_h^{(k)} = B_{\sigma_k(h)}^{(k-1)} = B_h^{(k-1)}$, giving $B^{(k)}(h,j) = A\begin{pmatrix} i_1,\ldots,i_h \\ j_1,\ldots,j_{h-1},j \end{pmatrix}$, $1 \le j \le n$. For $h = k$ we have $B_k^{(k)} = \tilde{B}_k^{(k)} = B_{\sigma_k(k)}^{(k-1)} = B_t^{(k-1)}$ and so $B^{(k)}(k,j) = B^{(k-1)}(t,j) = A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1,t} \\ j_1,\ldots,j_{k-1},j \end{pmatrix} = A\begin{pmatrix} i_1,\ldots,i_{k-1},i_k \\ j_1,\ldots,j_{k-1},j \end{pmatrix}$, $1 \le j \le n$. Consider rows $B_{k+1}^{(k)},\ldots,B_m^{(k)}$, assuming $k < m$. Recall that the elementary row operations in $\mathfrak{M}(\mathcal{I})$ transforming row h: $k < h \le m$ of $\tilde{B}^{(k)}$ compute each of the elements of $B_h^{(k)}$ by: $B^{(k)}(h,j) = \left[\tilde{B}^{(k)}(k,j_k)\cdot\tilde{B}^{(k)}(h,j) - \tilde{B}^{(k)}(h,j_k)\cdot\tilde{B}^{(k)}(k,j)\right]/\tilde{B}^{(k)}(k-1,j_{k-1})$, $1 \le j \le n$. Letting $t = \sigma_k(k)$ and $s = \sigma_k(h)$, this becomes: $B^{(k)}(h,j) = \left[B^{(k-1)}(t,j_k)\cdot B^{(k-1)}(s,j) - B^{(k-1)}(s,j_k)\cdot B^{(k-1)}(t,j)\right]/B^{(k-1)}(k-1,j_{k-1}) = \left[A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1,t} \\ j_1,\ldots,\ldots,j_k \end{pmatrix}\cdot A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1,s} \\ j_1,\ldots,j_{k-1},j \end{pmatrix} - A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1,s} \\ j_1,\ldots,\ldots,j_k \end{pmatrix}\cdot A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1,t} \\ j_1,\ldots,j_{k-1},j \end{pmatrix}\right]/A\begin{pmatrix} i_1,\ldots,i_{k-1} \\ j_1,\ldots,j_{k-1} \end{pmatrix}$.

Noting that $i_{k-1,t} = i_k$, if we apply Lemma 2.2.1.3 to the matrix $M = A\begin{bmatrix} i_1,\ldots,i_k,i_{k-1},s \\ j_1,\ldots,j_k,j \end{bmatrix}$, we obtain the identity:

$$A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix}\cdot A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1},s \\ j_1,\ldots,j_{k-1},j \end{pmatrix} - A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1},s \\ j_1,\ldots,j_k \end{pmatrix}\cdot$$

$$A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_{k-1},j \end{pmatrix} = A\begin{pmatrix} i_1,\ldots,i_{k-1} \\ j_1,\ldots,j_{k-1} \end{pmatrix}\cdot A\begin{pmatrix} i_1,\ldots,i_k,i_{k-1},s \\ j_1,\ldots,j_k,j \end{pmatrix}.$$

Note that this applies only for $k > 1$; for $k = 1$, we know the divisions by $B^{(o)}(o,j_o) = 1$, taken as a convention here, are exact. Thus, all divisions are exact in $\mathcal{J}$ and so $B^{(k)}(h,j) \in \mathcal{J}$, for $\mathcal{J}$ an arbitrary integral domain, and in fact, $B^{(k)}(h,j) = A\begin{pmatrix} i_1,\ldots,i_k,i_{k-1},\sigma_k(h) \\ j_1,\ldots,j_k,j \end{pmatrix} = A\begin{pmatrix} i_1,\ldots,i_k,i_k,h \\ j_1,\ldots,j_k,j \end{pmatrix}$, $k < h \le m$ and $1 \le j \le n$. All elementary row operations are defined in $\mathfrak{M}(\mathcal{J})$ and $B^{(k)}$ is row equivalent in $\mathfrak{M}(\mathcal{J})$ to $B^{(k-1)}$, and hence to A. This completes the inductive step.

We have now established that

$$B^{(q)}(h,j) = \begin{cases} A\begin{pmatrix} i_1,\ldots,i_{h-1},i_h \\ j_1,\ldots,j_{h-1},j \end{pmatrix}, & 1 \le h \le q \text{ and } 1 \le j \le n \\ A\begin{pmatrix} i_1,\ldots,i_q,i_q,h \\ j_1,\ldots,j_q,j \end{pmatrix}, & q < h \le m \text{ and } 1 \le j \le n. \end{cases}$$

Also, we know that $B^{(q)}(h,j) = 0$, for $h > q$ and $1 \le j \le n$. It follows readily that $B^{(q)}$ is an RE matrix with RE

sequence $(j_1,\ldots,j_q)$ and, hence, $\text{rank}(B^{(q)}) = q$. However, $B^{(q)}$ is row equivalent to A and so $r = \text{rank}(A) = \text{rank}(B^{(q)}) = q$. Thus, from the definition of $B^{(q)}$ given above, we clearly have $B^{(q)} = B^{(r)} = \bar{A}$, an RE form for A. Also, $J = (j_1,\ldots,j_r) = J_A$.

Finally, we know from Lemma 2.2.1.2 that $i'_{r+1} < i'_{r+2} <\ldots< i'_m$, since $i'_u = i_{q,u}$, for $r < u \leq m$. This is the ordering of the last m - r components of I which would make I the minimal element of $\wp_A$. Hence, $I = I_A$.//

2.2.2. The Computation of a Reduced Row Echelon Form by Exact Division

The effective mapping Γ such that $\Gamma(A) = \bar{\bar{A}}$, for every $A \in \mathbb{m}(\mathcal{J})$, will be defined by the following algorithm. It will be indicated in the subsequent discussion that Γ is, in fact, an algorithm or effective mapping. It will then be shown in a sequence of theorems that $\Gamma(A)$ is a RRE form in $\mathbb{m}(\mathcal{J})$, $\mathcal{J}$ an arbitrary integral domain, and that $\Gamma(A) = \bar{\bar{A}}$.

Algorithm 2.2.2.1. Transforming a Matrix over an Integral Domain $\mathcal{J}$ to Reduced Row Echelon Form by Exact Division

Input: An m by n matrix $A \in \mathbb{m}(\mathcal{J})$.

Output: The m by n matrix $C \in \mathbb{m}(\mathcal{J})$, where $C = \bar{\bar{A}}$.

(1) [Triangularize by exact division.] Compute $\Gamma_1(A) = (B, J, I)$ by Algorithm 2.2.1.1.

(2) [Initialize the diagonalization.] $C \leftarrow B$; $r \leftarrow \text{rank}(A)$; if $r = 0$ or $r = 1$, return C; otherwise, have $J = (j_1,\ldots,j_r)$; $i \leftarrow r$; $D \leftarrow B(r,j_r)$.

(3) [Diagonalize by exact division.] $i \leftarrow i-1$; $E \leftarrow C(i,j_i)$; for $t = i+1,\ldots,r$, set $F[t] \leftarrow C(i,j_t)$; $C_i \leftarrow D \cdot C_i$; for $t = i+1,\ldots,r$, compute $C_i \leftarrow C_i - F[t]\cdot C_t$; $C_i \leftarrow C_i/E$; if $i > 1$, go to (3); return C.

This algorithm is, in fact, an algorithm or effective mapping Γ of $\mathfrak{m}(\mathfrak{I})$ into $\mathfrak{m}(\mathfrak{I})$, which can be seen in the following discussion. $B = \bar{A}$, $J = J_A$, and $I = I_A$ are computed in step (1) for any $A \in \mathfrak{m}(\mathfrak{I})$ by Algorithm 2.2.1.1 and so step (1) is effective. Step (2) is effective, terminating with $C = B = \bar{A}$, if A is a zero matrix or if $\text{rank}(A) = 1$; otherwise, it initializes step (3). Let A be m by n and $J_A = (j_1,\ldots,j_r)$. In step (3) a sequence of matrices is generated: $C^{(o)}, C^{(1)},\ldots,C^{(r-1)}$, where $C^{(o)} = B$ and $C = C^{(r-1)}$. Each $C^{(i)}$ is computed from $C^{(i-1)}$, $0 < i \leq r-1$, by a finite sequence of elementary row operations in $\mathfrak{m}(\mathfrak{I})$, which computation may be summarized as follows. Note that $B(r,j_r) = \bar{A}(r,j_r) = d = A\binom{i_1,\ldots,i_r}{j_1,\ldots,j_r}$, where $I_A = (i_1,\ldots,i_m)$. For $1 \leq h \leq m$, $h \neq r-i$,

$C_h^{(i)} = C_h^{(i-1)}$, and for h = r-i: $C_{r-i}^{(i)} =$

$$\left[d \cdot C_{r-i}^{(i-1)} - \Sigma_{t=r-i+1}^{r} C^{(i-1)}(r-i,j_t) \cdot C_t^{(i-1)}\right] / C^{(i-1)}(r-i,j_{r-i}).$$

Thus, to obtain $C^{(i)}$ from $C^{(i-1)}$ we multiply row r - i by d, add row t multiplied by $- C^{(i-1)}(r-i,j_t)$, for t = r-i+1,...,r, and then divide by $C^{(i-1)}(r-i,j_{r-i})$. All elementary row operations are defined in $\mathfrak{m}(\mathfrak{J})$, except possibly the divisions. It will be shown below that, for $1 \leq i \leq r-1$, $C^{(i-1)}(r-i,j_{r-i}) \neq 0$ and in Theorem 2.2.2.4 it will be seen that the division is possible in $\mathfrak{J}$, for $\mathfrak{J}$ an arbitrary integral domain. For the present, however, we allow that $\mathfrak{J}$ may be required to be a field.

It will now be shown that Γ is an effective mapping of $\mathfrak{m}(\mathfrak{J})$ into $\mathfrak{m}(\mathfrak{J})$ and that $C = \Gamma(A)$ is a RRE matrix in $\mathfrak{m}(\mathfrak{J})$. If r = rank(A) = 0, $C = \bar{A} = \bar{\bar{A}}$, a zero matrix and an RRE matrix trivially. Suppose $r \geq 1$. Note that row C_r satisfies: $C(r,j_r) = C^{(o)}(r,j_r) = \bar{A}(r,j_r) = d \neq 0$ and $C(r,j) = C^{(o)}(r,j) = \bar{A}(r,j) = 0$, for $1 \leq j < j_r$. Also, rows $C_i = C_i^{(o)} = \bar{A}_i$ are zero, $r < i \leq m$, if $r < m$. If r = 1, then $C = \Gamma(A)$ is defined and is an RRE matrix in $\mathfrak{m}(\mathfrak{J})$. Suppose $r > 1$ and let the inductive hypothesis for v: $0 \leq v < r$ be: for $1 \leq h < r-v$, $C_h^{(v)} = \bar{A}_h$; for $r-v \leq h \leq r$, $C^{(v)}(h,j_h) = d \neq 0$ and $C^{(v)}(h,j) = 0$, $1 \leq j < j_h$ and $j = j_t$,

$h < t \leq r$; and for $r < h \leq m$, $C_h^{(v)}$ is a zero row. This hypothesis is clearly true for $v = 0$. Assuming it is true for $v = k-1$, $0 < k \leq r-1$, it will be shown to hold for $v = k$.

Since $C_h^{(k)} = C_h^{(k-1)}$, for $1 \leq h \leq m$, $h \neq r-k$, we have only to show that row C_{r-k} satisfies the hypthesis. Letting $z = r-k$, we know that, for $1 \leq j \leq n$:

$$C^{(k)}(z,j) = \left[d \cdot C^{(k-1)}(z,j) - \Sigma_{t=z+1}^{r} C^{(k-1)}(z,j_t) \cdot C^{(k-1)}(t,j)\right] / C^{(k-1)}(z,j_z).$$

For $1 \leq j < j_z$, $C^{(k-1)}(z,j) = \bar{A}(z,j) = 0$ and $C^{(k-1)}(t,j) = 0$, $z < t \leq r$, by the inductive hypothesis, since $j_z < j_t$, $z < t \leq r$. Therefore, $C^{(k)}(z,j) = \left[d \cdot 0 - \Sigma_{t=z+1}^{r} C^{(k-1)}(z,j_t) \cdot 0\right] / C^{(k-1)}(z,j_z) = 0$, for $1 \leq j < j_z$. For $j = j_z$ we have as above that $C^{(k-1)}(t,j_z) = 0$, for $z < t \leq r$, and so $C^{(k)}(z,j_z) = \left[d \cdot C^{(k-1)}(z,j_z) - \Sigma_{t=z+1}^{r} C^{(k-1)}(z,j_t) \cdot 0\right] / C^{(k-1)}(z,j_z) = d$. Finally, for $j = j_u$, $z < u \leq r$, we have by the inductive hypothesis that $C^{(k-1)}(t,j_u) = 0$, $z < t \leq r$ and $t \neq u$, and $C^{(k-1)}(t,j_t) = d$, for $t = u$. Hence, $C^{(k)}(z,j_u) = \left[d \cdot C^{(k-1)}(z,j_u) - C^{(k-1)}(z,j_u) \cdot d\right] / C^{(k-1)}(z,j_u) = 0$, $z < u \leq r$. This completes the inductive step. These results are summarized in the following theorem.

$N = N^{(r-1)}$, and let $\sigma_1', \ldots, \sigma_r'$ be the sequence of permutations employed to define I_M. Of course, rank(M) = r, since $A\begin{pmatrix} i_1, \ldots, i_r \\ j_1, \ldots, j_r \end{pmatrix} \neq 0$. Let $J_M = (j_1', \ldots, j_r')$ and for convenience let $j_{r+1} = j$ and $\rho_k = \sigma_{k+1} \cdots \sigma_r$, $0 \leq k < r$.

We first show that $M^{(v)} = B^{(v)}\begin{bmatrix} 1, \ldots, v, \rho_v(v+1), \ldots, \rho_v(r) \\ j_1, \ldots, j_r, j_{r+1} \end{bmatrix}$ for $0 \leq v \leq r$. For v = 0, note that $\rho_o = \sigma_1 \cdots \sigma_r = I_A$ and hence $\rho_o(z) = i_z$, $1 \leq z \leq r$. So, $M^{(o)} = M = A\begin{bmatrix} i_1, \ldots, i_r \\ j_1, \ldots, j_{r+1} \end{bmatrix} = B^{(o)}\begin{bmatrix} \rho_o(1), \ldots, \rho_o(r) \\ j_1, \ldots, j_{r+1} \end{bmatrix}$. Assume inductively, for some k: $0 < k \leq r$, that $M^{(k-1)} = B^{(k-1)}\begin{bmatrix} 1, \ldots, k-1, \rho_{k-1}(k), \ldots, \rho_{k-1}(r) \\ j_1, \ldots, j_{r+1} \end{bmatrix}$. Note that $M^{(k-1)}(z,z) = B^{(k-1)}(z, j_z) \neq 0$, for $1 \leq z < k$ and so $j_z' = z$, $1 \leq z < k$. Note that $\rho_{k-1}(k) = \sigma_k(\sigma_{k+1} \cdots \sigma_r(k)) = \sigma_k(k)$, for $1 \leq k \leq r$. So, $M^{(k-1)}(k,k) = B^{(k-1)}(\rho_{k-1}(k), j_k) = B^{(k-1)}(\sigma_k(k), j_k) = B^{(k)}(k, j_k) \neq 0$ is the k^{th} pivot for computing $M^{(k)}$ and no row interchanges are performed, making $\sigma_k' = (1, 2, \ldots, r)$, the identity in ρ_r. For $1 \leq z < k$, $M^{(k)}(z,h) = M^{(k-1)}(z,h) = B^{(k-1)}(z, j_h) = B^{(k)}(z, j_h)$, $1 \leq h \leq r+1$. For z = k, $M^{(k)}(k,h) = M^{(k-1)}(k,h) = B^{(k-1)}(\rho_{k-1}(k), j_h) = B^{(k-1)}(\sigma_k(k), j_h) = B^{(k)}(k, j_h)$, $1 \leq h \leq r+1$. Consider z:

Theorem 2.2.2.2. For every $A \in \mathfrak{M}(\mathcal{J})$, $\mathcal{J}$ an integral domain, $\Gamma(A) = C$ is defined and is an RRE matrix in $\mathfrak{M}(\mathcal{J})$. If A is nonzero with $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$, then C has diagonal elements $C(i,j_i) = A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_r \end{pmatrix}$, $1 \leq i \leq r$.

It will be for us to show in Theorem 2.2.2.4 that $\Gamma(A)$ is obtained by elementary row operations in $\mathfrak{M}(\mathcal{J})$, for $\mathcal{J}$ an arbitrary integral domain, and that, in fact, $\Gamma(A) = \bar{\bar{A}}$. We first prove the following lemma.

Lemma 2.2.2.3. Let A be an m by n nonzero matrix in $\mathfrak{M}(\mathcal{J})$ with $r = \text{rank}(A) < n$. Let $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$ and define a matrix $M = A\begin{bmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_r,j \end{bmatrix}$, for some j: $1 \leq j \leq n$. If $N = \Gamma(M)$ and $C = \Gamma(A)$, then $N = C\begin{bmatrix} 1,\ldots,r \\ j_1,\ldots,j_r,j \end{bmatrix}$.

Proof: Let $B^{(o)}, B^{(1)},\ldots,B^{(r)}$ and $C^{(o)}, C^{(1)},\ldots,C^{(r-1)}$ be the sequences of matrices generated by Algorithm 2.2.2.1, where $B^{(o)} = A$, $C^{(o)} = B^{(r)} = \bar{A}$, and $C = C^{(r-1)}$, and let $\sigma_1,\ldots,\sigma_r$ be the sequence of permutations employed to define I_A. Analogously, let $M^{(o)}, M^{(1)},\ldots,M^{(r)}$ and $N^{(o)}, N^{(1)},\ldots,N^{(r-1)}$ be the sequences of matrices generated by Algorithm 2.2.2.1, where $M^{(o)} = M$, $N^{(o)} = M^{(r)} = \bar{M}$, and

$k < z \leq r$ and let $z' = \sigma_{k+1}\cdots\sigma_r(z)$. For $1 \leq h \leq r+1$ we obtain:

$$M^{(k)}(z,h) = \left[M^{(k-1)}(k,k)\cdot M^{(k-1)}(z,h) - M^{(k-1)}(z,k)\cdot M^{(k-1)}(k,h)\right]$$
$$/M^{(k-1)}(k-1,k-1) = \left[B^{(k-1)}(\rho_{k-1}(k),j_k)\cdot B^{(k-1)}(\rho_{k-1}(z),j_h) - \right.$$
$$\left. B^{(k-1)}(\rho_{k-1}(z),j_k)\cdot B^{(k-1)}(\rho_{k-1}(k),j_h)\right]/B^{(k-1)}(k-1,j_{k-1})$$
$$= \left[B^{(k-1)}(\sigma_k(k),j_k)\cdot B^{(k-1)}(\sigma_k(z'),j_h) - B^{(k-1)}(\sigma_k(z'),j_k)\cdot \right.$$
$$\left. B^{(k-1)}(\sigma_k(k),j_h)\right]/B^{(k-1)}(k-1,j_{k-1}) = B^{(k)}(z',j_h) =$$
$$B^{(k)}(\rho_k(z),j_h).$$

Thus, $M^{(k)} = B^{(k)}\begin{bmatrix}1,\ldots,k,\rho_k(k+1),\ldots,\rho_k(r)\\ j_1,\ldots,j_{r+1}\end{bmatrix}$, completing the inductive step. Hence, $M^{(r)} = B^{(r)}\begin{bmatrix}1,\ldots,r\\ j_1,\ldots,j_{r+1}\end{bmatrix}$.

We next show that $N^{(v)} = C^{(v)}\begin{bmatrix}1,\ldots,r\\ j_1,\ldots,j_{r+1}\end{bmatrix}$, for $0 \leq v \leq r-1$. By definition, $N^{(o)} = M^{(r)} = B^{(r)}\begin{bmatrix}1,\ldots,r\\ j_1,\ldots,j_{r+1}\end{bmatrix}$ $= C^{(o)}\begin{bmatrix}1,\ldots,r\\ j_1,\ldots,j_{r+1}\end{bmatrix}$. Assume inductively that, for $0 < k \leq r-1$, $N^{(k-1)} = C^{(k-1)}\begin{bmatrix}1,\ldots,r\\ j_1,\ldots,j_{r+1}\end{bmatrix}$. For $1 \leq z \leq r$ and $z \neq s = r-k$, we have $N^{(k)}(z,h) = N^{(k-1)}(z,h) = C^{(k-1)}(z,j_h) = C^{(k)}(z,j_h)$, $1 \leq h \leq r+1$. We need only show that $N^{(k)}(s,h) = C^{(k)}(s,j_h)$, for $1 \leq h \leq r+1$. Noting that $e = N^{(o)}(r,r) = B^{(r)}(r,j_r) = A\begin{pmatrix}i_1,\ldots,i_r\\ j_1,\ldots,j_r\end{pmatrix} = d$, we have

$$N^{(k)}(s,h) = \left[e\cdot N^{(k-1)}(s,h) - \Sigma_{t=s+1}^{r} N^{(k-1)}(s,t)\cdot N^{(k-1)}(t,h)\right]$$
$$/N^{(k-1)}(s,s) = \left[d\cdot C^{(k-1)}(s,j_h) - \Sigma_{t=s+1}^{r} C^{(k-1)}(s,j_t)\cdot \right.$$
$$\left. C^{(k-1)}(t,j_h)\right]/C^{(k-1)}(s,j_s) = C^{(k)}(s,j_h).$$

Thus, $N^{(k)} = C^{(k)}\begin{bmatrix}1,\ldots,r\\ j_1,\ldots,j_{r+1}\end{bmatrix}$, completing the inductive step. Thus, $N = N^{(r-1)} = C^{(r-1)}\begin{bmatrix}1,\ldots,r\\ j_1,\ldots,j_{r+1}\end{bmatrix} = C\begin{bmatrix}1,\ldots,r\\ j_1,\ldots,j_r,j\end{bmatrix}$.

The main theorem of this subsection can now be proved.

Theorem 2.2.2.4. For every $A \in \mathbb{M}(\mathcal{J})$, $\mathcal{J}$ an arbitrary integral domain, $\Gamma(A)$ is a RRE form in $\mathbb{M}(\mathcal{J})$ for A and, in fact, $\Gamma(A) = \bar{\bar{A}}$.

Proof: Let $C = \Gamma(A)$. If A is a zero matrix, then $C = A = \bar{\bar{A}}$ is trivially a RRE form for A. Suppose A is m by n and nonzero, with $r = \text{rank}(A)$, $J_A = (j_1,\ldots,j_r)$, and $I_A = (i_1,\ldots,i_m)$. Recall from Theorem 2.1.2.1 that $\bar{\bar{A}}$ is a RRE matrix. If $r = n$, then $j_h = h$, for $1 \le h \le r$, and the only nonzero elements of C are the diagonal elements $C(h,h) = A\begin{pmatrix}i_1,\ldots,i_r\\ j_1,\ldots,j_r\end{pmatrix}$. In this case, $C = \bar{\bar{A}}$, all divisions in step (3) of Algorithm 2.2.2.1 are exact, and C is derived from A by elementary row operations in $\mathbb{M}(\mathcal{J})$.

Suppose $r < n$. Let $M = A\begin{bmatrix}i_1,\ldots,i_r\\ j_1,\ldots,j_r,j\end{bmatrix}$, for some j: $1 \le j \le n$. By Lemma 2.2.2.3 we know that $N = \Gamma(M) = C\begin{bmatrix}1,\ldots,r\\ j_1,\ldots,j_r,j\end{bmatrix}$. In particular, $N(h,h) = C(h,j_h) = A\begin{pmatrix}i_1,\ldots,i_r\\ j_1,\ldots,j_r\end{pmatrix} \neq 0$, $1 \le h \le r$. Let $\mathcal{F}$ be a field containing

$\mathfrak{J}$. By dividing each row of N by $A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_r \end{pmatrix}$ in $\mathfrak{m}(\mathfrak{F})$ we obtain $N' \in \mathfrak{m}(\mathfrak{F})$ by elementary row operations in $\mathfrak{m}(\mathfrak{F})$ from M, where N' is defined by:

$$N'(h,u) = \begin{cases} 1, \text{ if } h=u=1,2,\ldots,r \\ \dfrac{C(h,j)}{A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_r \end{pmatrix}}, \text{ for } 1 \le h \le r \text{ and } u = r+1 \\ 0, \text{ otherwise.} \end{cases}$$

Hence, N' is a RRE form in $\mathfrak{m}(\mathfrak{F})$ for M. Moreover, we know that, by Cramer's Rule, M can be transformed by elementary row operations in $\mathfrak{m}(\mathfrak{F})$ to a matrix $N'' \in \mathfrak{m}(\mathfrak{F})$ defined by:

$$N''(h,u) = \begin{cases} 1, \text{ if } h=u=1,2,\ldots,r \\ \dfrac{M\begin{pmatrix} 1,\ldots,r \\ 1,\ldots,h-1,r+1,h+1,\ldots,r \end{pmatrix}}{M\begin{pmatrix} 1,\ldots,r \\ 1,\ldots,r \end{pmatrix}}, \text{ for } 1 \le h \le r \text{ and } u = r+1 \\ 0, \text{ otherwise.} \end{cases}$$

N' is also a RRE form in $\mathfrak{m}(\mathfrak{F})$ for M.

It is shown in Chapter 7 of [BIG65] that every matrix in $\mathfrak{m}(\mathfrak{F})$, $\mathfrak{F}$ a field, has exactly one RRE form with diagonal elements equal to unity. Therefore, N' = N" and so, for $1 \le h \le r$,

$$\frac{C(h,j)}{A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_r \end{pmatrix}} = \frac{M\begin{pmatrix} 1,\ldots,r \\ 1,\ldots,h-1,r+1,h+1,\ldots,r \end{pmatrix}}{M\begin{pmatrix} 1,\ldots,r \\ 1,\ldots,r \end{pmatrix}}$$

Since $M\begin{pmatrix}1,\ldots,r\\1,\ldots,r\end{pmatrix} = A\begin{pmatrix}i_1,\ldots,i_r\\j_1,\ldots,j_r\end{pmatrix}$ and $M\begin{pmatrix}1,\ldots,r\\1,\ldots,h-1,r+1,h+1,\ldots,r\end{pmatrix}$ $= A\begin{pmatrix}i_1,\ldots,i_r\\j_1,\ldots,j_{h-1},j,j_{h+1},\ldots,j_r\end{pmatrix}$, we have that $C(h,j) = A\begin{pmatrix}i_1,\ldots,i_r\\j_1,\ldots,j_{h-1},j,j_{h+1},\ldots,j_r\end{pmatrix}$, for $1 \leq h \leq r$ and $1 \leq j \leq n$. All other elements of C are known to be zero. Hence, the single division performed in step (3) of Algorithm 2.2.2.1 on each element must be exact. Thus, for every $A \in \mathbb{m}(\mathcal{J})$, $\Gamma(A) = C = \bar{\bar{A}}$ by definition of $\bar{\bar{A}}$ and $\Gamma(A)$ is a RRE form in $\mathbb{m}(\mathcal{J})$ for A.//

We will call $\bar{\bar{A}}$ the <u>determinantal RRE form</u> for A, for every $A \in \mathbb{m}(\mathcal{J})$.

Section 2.3. The Commutativity of an Induced Mapping with the Effective Mapping Γ

Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be integral domains and suppose θ is a homomorphism of $\mathcal{I}_1$ to $\mathcal{I}_2$. θ can be extended to a mapping θ' of $\mathbb{m}(\mathcal{I}_1)$ to $\mathbb{m}(\mathcal{I}_2)$ as follows. If $A \in \mathbb{m}(\mathcal{I}_1)$, define $A^* = \theta'(A)$ such that each element $A^*(i,j)$ of A^* satisfies: $A^*(i,j) = \theta(A(i,j))$. For simplicity let θ', called the mapping induced by θ, also be denoted by θ. In this section the equality of $\theta\Gamma(A)$ and $\Gamma\theta(A)$ is discussed, for any $A \in \mathbb{m}(\mathcal{I}_1)$, and sufficient conditions for this equality derived. In the case that $\theta\Gamma(A) = \Gamma\theta(A)$, we will say that θ is a commuting induced mapping for A; otherwise, θ will be called a noncommuting induced mapping for A.

We begin with the following lemma.

Lemma 2.3.1 Let θ be an induced mapping of $\mathbb{m}(\mathcal{I}_1)$ to $\mathbb{m}(\mathcal{I}_2)$, where $\mathcal{I}_1$ and $\mathcal{I}_2$ are integral domains, and let $A^* = \theta(A)$, for $A \in \mathbb{m}(\mathcal{I}_1)$. Then $\text{rank}(A^*) \leq \text{rank}(A)$.

Proof: Let $s = \text{rank}(A^*)$. Then there exists an s by s submatrix B^* such that $\det(B^*) \neq 0$, where $B^* = \theta(B)$ for the corresponding submatrix B of A. Hence, $\det(B) \neq 0$, for otherwise we would have $\det(B^*) = \det(\theta(B)) = \theta(\det(B)) = \theta(0) = 0$, a contradiction. Therefore, $r = \text{rank}(A) \geq \text{rank}(B) = \text{rank}(B^*) = \text{rank}(A^*) = s$.//

This lemma will now be used to prove the following theorem, which will be employed in the algorithms of Chapter 4.

Theorem 2.3.2 Let θ be an induced mapping of $\mathbb{M}(\mathcal{S}_1)$ to $\mathbb{M}(\mathcal{S}_2)$. Let $A \in \mathbb{M}(\mathcal{S}_1)$, $A^* = \theta(A)$, $r = \text{rank}(A)$, and $r^* = \text{rank}(A^*)$. Then either $r > r^*$ or $(J_A, I_A) \leq (J_{A^*}, I_{A^*})$.

Proof: If A is a zero matrix, then $J_A = \emptyset = J_{A^*}$ and $I_A = (1,2,\ldots,m) = I_{A^*}$ and so $(J_A, I_A) = (J_{A^*}, I_{A^*})$. Suppose A is m by n and nonzero. By Lemma 2.3.1, $r \geq r^*$. Supposing that $r = r^*$, it need only be shown that $(J_A, I_A) \leq (J_{A^*}, I_{A^*})$. Let $J_A = (j_1,\ldots,j_r)$ and $J_{A^*} = (j_1^*,\ldots,j_r^*)$. By Lemma 2.3.1 we know that $\text{rank}\left(A\begin{bmatrix}1,\ldots,m\\ 1,\ldots,j_i^*\end{bmatrix}\right) \geq \text{rank}\left(A^*\begin{bmatrix}1,\ldots,m\\ 1,\ldots,j_i^*\end{bmatrix}\right) = i$. Since j_i is the least integer j: $1 \leq j \leq n$ such that $\text{rank}\left(A\begin{bmatrix}1,\ldots,m\\ 1,\ldots,j\end{bmatrix}\right) = i$, clearly $j_i \leq j_i^*$. Hence, if $j_i < j_i^*$, for some i: $1 \leq i \leq r$, then $J_A < J_{A^*}$ and so $(J_A, I_A) < (J_{A^*}, I_{A^*})$.

Suppose $J_A = J_{A^*}$. If $I_A = I_{A^*}$, we are finished, since then $(J_A, I_A) = (J_{A^*}, I_{A^*})$. Hence, assume $I_A \neq I_{A^*}$. Let $I_A = (i_1,\ldots,i_m)$ and $I_{A^*} = (i_1^*,\ldots,i_m^*)$ and suppose u is the least integer, $1 \leq u \leq m$, such that $i_u \neq i_u^*$. But then $u \leq r$; so, i_u^* is the least integer i: $1 \leq i \leq m$ and $i \neq i_h$, $1 \leq h < u$, such that $A^*\begin{pmatrix}i_1^*,\ldots,i_{u-1}^*,i\\ j_1^*,\ldots\ldots,j_u^*\end{pmatrix} \neq 0$. Hence, $A\begin{pmatrix}i_1,\ldots,i_{u-1},i_u^*\\ j_1,\ldots\ldots,j_u\end{pmatrix} \neq 0$. Since i_u is the least integer i:

$1 \le i \le m$ and $i \neq i_h$, $1 \le h < u$, such that $A\begin{pmatrix} i_1,\ldots,i_{u-1},i \\ j_1,\ldots,\ldots,j_u \end{pmatrix} \neq 0$, we see that $i_u < i_u^*$. Thus, $I_A < I_{A^*}$ and so $(J_A,I_A) < (J_{A^*},I_{A^*})$.//

The next theorem gives a sufficient condition for the equality of $\Gamma\theta(A)$ and $\theta\Gamma(A)$, hence, a sufficient condition for a given induced mapping to be a commuting mapping for A.

Theorem 2.3.3 For every $A \in \mathfrak{m}(\mathfrak{I}_1)$ and any induced mapping θ of $\mathfrak{m}(\mathfrak{I}_1)$ to $\mathfrak{m}(\mathfrak{I}_2)$, $(\overline{\overline{A^*}}) = (\overline{\overline{A}})^*$ whenever $(J_A,I_A) = (J_{A^*},I_{A^*})$.

Proof: Suppose $(J_A,I_A) = (J_{A^*},I_{A^*})$. The theorem is trivially true for A a zero matrix; so, assume A is nonzero and m by n. Let $J_A = J_{A^*} = (j_1,\ldots,j_r)$ and $I_A = I_{A^*} = (i_1,\ldots,i_m)$. Then, for $1 \le h \le r$ and $1 \le j \le n$, $(\overline{\overline{A}})^*(h,j) = (\overline{\overline{A}}(h,j))^* = \theta\left(A\begin{pmatrix} i_1,\ldots,\ldots,i_r \\ j_1,\ldots,j_{h-1},j,j_{h+1},\ldots,j_r \end{pmatrix}\right) = A^*\begin{pmatrix} i_1,\ldots,\ldots,i_r \\ j_1,\ldots,j_{h-1},j,j_{h+1},\ldots,j_r \end{pmatrix}$ $= (\overline{\overline{A^*}})(h,j)$. Also, for $r < h \le m$ and $1 \le j \le n$, $(\overline{\overline{A}})^*(h,j) = 0 = (\overline{\overline{A^*}})(h,j)$. Thus, we have shown that $(\overline{\overline{A}})^* = (\overline{\overline{A^*}})$.//

The following theorem provides another sufficient condition for an induced mapping to be commuting.

Theorem 2.3.4 Let A be an m by n nonzero matrix in $\mathfrak{m}(\mathfrak{I}_1)$ with $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$ and let $A^* = \theta(A)$, where θ is an induced mapping of $\mathfrak{m}(\mathfrak{I}_1)$ to $\mathfrak{m}(\mathfrak{I}_2)$. Then $(J_A,I_A) = (J_{A^*},I_{A^*})$ if and only if $A^*\begin{pmatrix} i_1,\ldots i_k \\ j_1,\ldots j_k \end{pmatrix} \neq 0$, for $1 \le k \le r$.

Proof: Let $J_{A^*} = (j_1^*,\ldots,j_s^*)$ and $I_{A^*} = (i_1^*,\ldots,i_m^*)$. To show necessity, we simply observe that, if $(J_A,I_A) = (J_{A^*},I_{A^*})$, then $r = s$ and $A^*\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix} = A^*\begin{pmatrix} i_1^*,\ldots,i_k^* \\ j_1^*,\ldots,j_k^* \end{pmatrix} \neq 0$, for $1 \leq k \leq r$. To show sufficiency, suppose that $(J_A,I_A) \neq (J_{A^*},I_{A^*})$. Then applying Theorem 2.3.2 either 1) $r > r^*$, or 2) $r = r^*$ and $J_A < J_{A^*}$, or 3) $r = r^*$, $J_A = J_{A^*}$, and $I_A < I_{A^*}$. If 1) holds, clearly $A^*\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_r \end{pmatrix} = 0$. If 2) holds, let t be the least integer k such that $j_k \neq j_k^*$. Then $j_t < j_t^*$ and so $\operatorname{rank}\left(A^*\begin{bmatrix} i_1,\ldots,i_t \\ j_1,\ldots,j_t \end{bmatrix}\right) \leq \operatorname{rank}\left(A^*\begin{bmatrix} 1,\ldots,m \\ 1,\ldots,j_t \end{bmatrix}\right) < t$; hence, $A^*\begin{pmatrix} i_1,\ldots,i_t \\ j_1,\ldots,j_t \end{pmatrix} = 0$. Finally, if 3) holds, let t be the least integer k such that $i_k \neq i_k^*$, where in fact, $i_t < i_t^*$. Then $A^*\begin{pmatrix} i_1,\ldots,i_t \\ j_1,\ldots,j_t \end{pmatrix} = A^*\begin{pmatrix} i_1^*,\ldots,i_{t-1}^*,i_t \\ j_1^*,\ldots\ldots,j_t^* \end{pmatrix} = 0$, since i_t^* is the least i: $1 \leq i \leq m$ and $i \neq i_u$, $1 \leq u < t$, such that $A^*\begin{pmatrix} i_1^*,\ldots,i_{t-1}^*,i \\ j_1^*,\ldots\ldots,j_t^* \end{pmatrix} \neq 0$. Thus, it has been shown that, if $(J_A,I_A) \neq (J_{A^*},I_{A^*})$, then there is a k: $1 \leq k \leq r$ such that $A^*\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix} = 0$, proving sufficiency.//

Let an induced mapping θ be rejected if $(J_A,I_A) \neq (J_{A^*},I_{A^*})$ and accepted if $(J_A,I_A) = (J_{A^*},I_{A^*})$. Thus, there may be commuting mappings θ which are also rejected mappings.

For example, let $A = \begin{bmatrix} t & 1 & 0 \\ 0 & t & 1 \\ 1 & 0 & t \end{bmatrix}$ and let t be a nonzero element such that $\theta(t) = 0$ and $t^3 + 1 \neq 0$. Then $J_A = J_{A^*} = (1,2,3)$; but $I_A = (1,2,3)$ and $I_{A^*} = (3,1,2)$. Hence, θ is a rejected mapping for A. Yet, $\bar{A}$ and $\bar{\bar{A}}^*$ are diagonal matrices with diagonal elements $t^3 + 1$ and 1, respectively, and so $(\bar{A})^* = \bar{\bar{A}}^*$, since $\theta(t^3+1) = 1$. Thus, θ is also a commuting mapping for A. A specific example of such a mapping is found in the next section; that is, take θ to be a mod-p mapping φ_p, p prime, and take t = p.

We note that every non-commuting mapping is a rejected mapping. Theorem 2.3.3, therefore, provides a test for detecting rejected mappings for a matrix A and, hence, a sufficient criterion for eliminating all non-commuting mappings for A. This criterion will be employed in the algorithms of Chapter 4. By Theorem 2.3.4, θ is an accepted mapping for A if and only if $A^*\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix} \neq 0$, for $1 \leq k \leq r$. This fact will be employed in subsequent sections to bound the number of rejected mappings for a matrix A.

Section 2.4 Reduction Modulo a Prime and the Chinese Remainder Theorem For Matrices

Let I be the ring of integers and $I[x_1,\ldots,x_s]$ be the ring of polynominals in s variables over I. For any prime $p \in I$, let φ_p be the unique homomorphism of I onto GF(p), the finite field of p elements, such that $\varphi_p(a) = a$, for $0 \le a < p$. Let φ_p' be the homomorphism of $I[x]$ onto $GF(p)[x]$ induced by φ_p; that is, for $A = \sum_{i=0}^{n} a_i x^i \in I[x]$, $\varphi_p'(A) = \sum_{i=0}^{n} \varphi_p(a_i)x^i$. Denote φ_p' simply by φ_p. φ_p can be extended to $I[x_1,\ldots,x_s]$, for any $s > 1$, in the following way. Suppose the homomorphism φ_p has been defined on $I[x_1,\ldots,x_{s-1}]$. Note that, for any $A \in I[x_1,\ldots,x_s]$, $A(x_1,\ldots,x_s) = \sum_{i=0}^{n} A_i x_s^i$, where each $A_i \in I[x_1,\ldots,x_{s-1}]$, since $I[x_1,\ldots,x_s] \cong I[x_1,\ldots,x_{s-1}][x_s]$. Thus, let φ_p' be the homomorphism of $I[x_1,\ldots,x_s]$ onto $GF(p)[x_1,\ldots,x_s]$ <u>induced</u> by φ_p; that is, for $A \in I[x_1,\ldots,x_s]$, $\varphi_p'(A) = \sum_{i=0}^{n} \varphi_p(A_i)x_s^i$. Again denote φ_p' by φ_p. Each such homomorphism φ_p is called a <u>mod-p homomorphism</u>.

We define the mapping <u>norm</u> on I and $I[x_1,\ldots,x_s]$, $s \ge 1$, as follows. For $a \in I$, let $\text{norm}(a) = |a|$ and, for $A = \sum_{i=0}^{n} a_i x^i \in I[x]$, let $\text{norm}(A) = \sum_{i=0}^{n} \text{norm}(a_i)$. Next we define recursively the mapping norm on $I[x_1,\ldots,x_s]$, $s > 1$. For $A = \sum_{i=0}^{n} A_i x_s^i \in I[x_1,\ldots,x_s]$, where each $A_i \in I[x_1,\ldots,x_{s-1}]$,

let norm(A) = $\sum_{i=0}^{n}$ norm(A_i). The mapping norm has the important properties that, for A, B in $I[x_1,\ldots,x_s]$ or in I, norm(A+B) $\leq$ norm(A) + norm(B) and norm(A·B) $\leq$ norm(A) · norm(B). Note that, for A ϵ $I[x_1,\ldots,x_s]$, $s \geq 1$, each integer coefficient of A is bounded in magnitude by norm(A). The mappings φ_p and norm are discussed in [COG69b].

Suppose $p_1,\ldots,p_t$ are distinct odd primes and let $a_i \in GF(p_i)$, $1 \leq i \leq t$. By the Chinese Remainder Theorem, there is a unique $a \in I$ such that $a_i = \varphi_{p_i}(a)$, for $1 \leq i \leq t$, and $-p_1\cdots p_t/2 < a < p_i\cdots p_t/2$. (See [COG69b] or [KND69], Section 4.3.2) A sequence $(a_1,\ldots,a_t)$, where $a_i \in GF(p_i)$, $1 \leq i \leq t$, is called a <u>modular representation of an integer</u> $a \in I$ if $\varphi_{p_k}(a) = a_k$, for $1 \leq k \leq t$, and $-p_1\cdots p_t/2 < a < p_1\cdots p_t/2$. Hence, by the Chinese Remainder Theorem, $(a_1,\ldots,a_t)$ uniquely determines a. A method for computing $a \in I$ from a modular representation $(a_1,\ldots,a_t)$ with respect to the distinct odd primes $p_1,\ldots,p_t$ is as follows. Let $b_1 = a_1 \in I$, if $a_1 < p_1/2$, and $b_1 = a_1 - p_1$, otherwise. Then, if $t > 1$, for $i = 2, 3,\ldots,t$, perform the following computations: $q_i = [\varphi_{p_i}(p_1\cdots p_{i-1})]^{-1} \cdot (a_i - \varphi_{p_i}(b_{i-1}))$ in $GF(p_i)$; next $q_i' = q_i - p_i$, if $q_i \geq p_i/2$, and $q_i' = q_i$, otherwise; then $b_i = b_{i-1} + q_i' \cdot p_1\cdots p_{i-1}$. At the conclusion of these computations we have $a = b_t$. This method is referred to as <u>Garner's Method</u>. (See [TAI61] or [KND69], Section 4.3.2)

We can extend Garner's Method and the concept of a modular representation to $I[x_1,\ldots,x_s]$ as follows. A sequence $(B_1,\ldots,B_t)$, where $B_i \in GF(p_i)[x_1,\ldots,x_s]$, $1 \leq i \leq t$, and the p_i are distinct odd primes, is called a <u>modular representation of a polynomial</u> $A \in I[x_1,\ldots,x_s]$ if $\omega_{p_k}(A) = B_k$, for $1 \leq k \leq t$, and there is a bound c on the magnitudes of the integer coefficients of A such that $c < p_1\cdots p_t/2$. Garner's Method can be extended to $I[x_1,\ldots,x_s]$, for all $s \geq 1$, in the following way. For $s = 1$, let $(B_1,\ldots,B_t)$ be a modular representation of $A = \sum_{i=0}^{n} a_i x^i \in I[x]$ with respect to odd primes $p_1,\ldots,p_t$. For $0 \leq i \leq n$, the i^{th} coefficients of $B_1,\ldots,B_t$ form a modular representation of a_i with respect to primes $p_1,\ldots,p_t$ and, hence, a_i may be computed from these coefficients by Garner's Method for the integers. This constitutes Garner's Method for $I[x]$.

Suppose Garner's Method has been defined for $I[x_1,\ldots,x_{s-1}]$, $s > 1$, such that $B \in I[x_1,\ldots,x_{s-1}]$ can be computed from a modular representation for B. Let $A = \sum_{i=0}^{n} A_i x_s^i \in I[x_1,\ldots,x_s]$, where each $A_i \in I[x_1,\ldots,x_{s-1}]$, have the modular representation $(B_1,\ldots,B_t)$ with respect to odd primes $p_1,\ldots,p_t$. Garner's Method for $I[x_1,\ldots,x_s]$ consists simply in applying Garner's Method for $I[x_1,\ldots,x_{s-1}]$ recursively to the coefficients of $B_1,..,B_t$. This is possible,

since, for $0 \leq i \leq n$, the i^{th} coefficients of $B_1,\ldots,B_t$ form a modular representation of A_i with respect to $p_1,\ldots,p_t$, and this modular representation uniquely determines A_i.

These concepts can be extended to matrices in a straightforward way. Let M be an m by n matrix in $\mathbb{m}(I[x_1,\ldots,x_s])$ and let φ_p be a mod-p homomorphism on $I[x_1,\ldots,x_s]$. The mapping φ_p' of $\mathbb{m}(I[x_1,\ldots,x_s])$ onto $\mathbb{m}(GF(p)[x_1,\ldots,x_s])$ induced by φ_p is defined such that $N = \varphi_p'(M)$ if and only if $N(i,j) = \varphi_p(M(i,j))$, for $1 \leq i \leq m$ and $1 \leq j \leq n$. The mod-p mapping φ_p' of $\mathbb{m}(I)$ onto $\mathbb{m}(GF(p))$ is defined analogously. Again we denote φ_p' by φ_p.

We say that a sequence of matrices $(N^{(1)},\ldots,N^{(t)})$, where $N^{(k)} \in \mathbb{m}(GF(p_k)[x_1,\ldots,x_s])$, $1 \leq k \leq t$, and the p_k are distinct odd primes, is a <u>modular representation of a matrix</u> $M \in \mathbb{m}(I[x_1,\ldots,x_s])$ if $\varphi_{p_k}(M) = N^{(k)}$, for $1 \leq k \leq t$, and there is a bound c on the magnitudes of the integer coefficients of the elements of M such that $c < p_1\cdots p_t/2$. Given such a modular representation for an m by n matrix $M \in \mathbb{m}(I[x_1,\ldots,x_s])$, we note that, for $1 \leq i \leq m$ and $1 \leq j \leq n$, $(N^{(1)}(i,j),\ldots,N^{(t)}(i,j))$ is a modular representation for $M(i,j)$ with respect to the primes $p_1,\ldots,p_t$. Hence, Garner's Method is extended to $\mathbb{m}(I[x_1,\ldots,x_s])$ by applying Garner's

Method for $I[x_1,\ldots,x_s]$ elementwise to $N^{(1)},\ldots,N^{(t)}$. The definition of a modular representation and Garner's Method are extended to $\mathbb{m}(I)$ similarly.

Let $c \geq \max_{\substack{1\leq i\leq m\\ 1\leq j\leq n}} \{\text{norm}(M(i,j))\}$, for an m by n matrix $M \in \mathbb{m}(I[x_1,\ldots,x_s])$. Note that c bounds the magnitude of the integer coefficients of each M(i,j). Hence, if $p_1,\ldots,p_t$ are distinct odd primes such that $c < p_1\cdots p_t/2$, then the sequence $(\omega_{p_1}(M),\ldots,\omega_{p_t}(M))$ of matrices, where $\omega_{p_k}(M) \in \mathbb{m}(GF(p_k)[x_1,\ldots,x_s])$, is a modular representation of M, from which M may be constructed by Garner's Method. Similarly for $M \in \mathbb{m}(I)$.

Given an m by n nonzero matrix A in $\mathbb{m}(I)$ or $\mathbb{m}(I[x_1,\ldots,x_s])$ with $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$, we wish to find an upper bound on the number of primes required to obtain a modular representation of $\bar{\bar{A}}$, from which to construct $\bar{\bar{A}}$ by Garner's Method. From Theorem 2.1.2.1 we know that the only possibly nonzero elements of $\bar{\bar{A}}$ are $\bar{\bar{A}}(i,j) = A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_{i-1},j,j_{i+1},\ldots,j_r \end{pmatrix}$, $1 \leq i \leq r$ and $j_i \leq j \leq n$ and $j \neq j_u$, $i < u \leq r$. Thus, bounds on the norms of these elements will have to be found, which will require bounds on the norms of determinants. The following lemma gives one such bound.

Lemma 2.4.1 For every m by m matrix B in $\mathbb{m}(I)$ or in

$\mathbb{m}(I[x_1,\ldots,x_s])$,

$$\text{norm}(\det(B)) \le m! \cdot \prod_{i=1}^{m} \max_{1\le j\le m} \text{norm}(B(i,j)).$$

Proof: Let an arbitrary element $(t_1,\ldots,t_m) \in \wp$ be designated by τ. Since $\det(B) = \sum_{\tau\in\wp_m} \text{sgn}(\tau) \cdot B(1,t_1)\cdots B(m,t_m)$,

$$\text{norm}(\det(B)) \le \sum_{\tau\in\wp_m} \text{norm}(B(1,t_1)\cdots B(m,t_m)) \le$$

$$\sum_{\tau\in\wp_m} \prod_{i=1}^{m} \text{norm}(B(i,t_i)) \le m!\cdot\max_{\tau\in\wp_m} \prod_{i=1}^{m} \text{norm}(B(i,t_i)) \le$$

$$m!\cdot\prod_{i=1}^{m} \max_{\tau\in\wp_m} \text{norm}(B(i,t_i)) = m!\cdot\prod_{i=1}^{m} \max_{1\le j\le m} \text{norm}(B(i,j)).//$$

Using this lemma, a different bound on the norm of a determinant is derived and will be applied in Theorem 2.4.3 to certain elements of $\bar{\bar{A}}$.

Lemma 2.4.2 Let B be an m by m matrix in $\mathbb{m}(I)$ or in $\mathbb{m}(I[x_1,\ldots,x_s])$. Then, for any integer i: $1 \le i \le m$,

$$\text{norm}(\det(B)) < m!\cdot\left(\max_{1\le k\le m} \text{norm}(B(k,i))\right)\cdot\left(\prod_{h=1}^{m} e_h\right)/\min_{1\le h\le m} e_h,$$

where $e_h = 1 + \max_{\substack{1\le t\le m \\ t\ne i}} \text{norm}(B(h,t))$, $1 \le h \le m$.

Proof: From the Laplace expansion theorem:

$$\det(B) = \sum_{k=1}^{m} (-1)^{i+k} B(k,i)\cdot B\begin{pmatrix}1,\ldots,k-1,k+1,\ldots,m\\ 1,\ldots,i-1,i+1,\ldots,m\end{pmatrix}.$$

Thus, $\text{norm}(\det(B))$

$$\le \sum_{k=1}^{m} \text{norm}(B(k,i))\cdot\text{norm}\left(B\begin{pmatrix}1,\ldots,k-1,k+1,\ldots,m\\ 1,\ldots,i-1,i+1,\ldots,m\end{pmatrix}\right)$$

$$\le m\cdot\left(\max_{1\le k\le m} \text{norm}(B(k,i))\right)\cdot\left(\max_{1\le k\le m} \text{norm}\left(B\begin{pmatrix}1,\ldots,k-1,k+1,\ldots,m\\ 1,\ldots,i-1,i+1,\ldots,m\end{pmatrix}\right)\right)$$

$$\leq m\cdot f\cdot\left(\max_{1\leq k\leq m}\left((m-1)!\prod_{\substack{h=1\\h\neq k}}^{m}\max_{\substack{1\leq t\leq m\\t\neq i}}\text{norm}(B(h,t))\right)\right),$$

where $f = \max_{1\leq k\leq m} \text{norm}(B(k,i))$, applying Lemma 2.4.1. Let

$e_h = 1+\max_{\substack{1\leq t\leq m\\t\neq i}} \text{norm}(B(h,t))$, $1 \leq h \leq m$. Then

$$\text{norm}(\det(B)) < m!\cdot f\cdot\max_{1\leq k\leq m}\prod_{\substack{h=1\\h\neq k}}^{m} e_h \leq m!\cdot f\cdot (\prod_{h=1}^{m} e_h)/\min_{1\leq h\leq m} e_h .//$$

The following theorem obtains bounds on the norms of the possibly nonzero elements of the determinantal RRE form $\bar{\bar{A}}$.

Theorem 2.4.3 For A an m by n nonzero matrix in $\mathbb{M}(I)$ or in $\mathbb{M}(I[x_1,\ldots,x_s])$ with $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$, for $1 \leq i \leq r$ and $1 \leq j \leq n$,

$$\text{norm}(\bar{\bar{A}}(i,j)) < r!\left(\max_{1\leq k\leq r}\text{norm}(A(i_k,j))\right)\cdot (\prod_{h=1}^{r} e_h)/\min_{1\leq h\leq r} e_h,$$

where $e_h = 1 + \max_{\substack{1\leq t\leq r\\t\neq i}} \text{norm}(A(i_h,j_t))$, $1 \leq h \leq r$.

Proof: This theorem follows readily if, for $1 \leq i \leq r$ and $1 \leq j \leq n$, we apply Lemma 2.4.2 to the matrix

$$B = A\begin{bmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_{i-1},j,j_{i+1},\ldots,j_r \end{bmatrix}, \text{ noting that } \bar{\bar{A}}(i,j) = \det(B).//$$

Let δ be the maximum of the bounds of Theorem 2.4.3. Then δ could be used to obtain a modular representation of $\bar{\bar{A}}$. We now proceed to find a lower bound w on the number of

primes sufficient to obtain a modular representation of $\bar{\bar{A}}$.

Theorem 2.4.4 Let M be an m by n nonzero matrix in either $\mathbb{m}(I)$ or $\mathbb{m}(I[x_1,\ldots,x_s])$ such that $\text{norm}(M(i,j)) \leq c$, $1 \leq i \leq m$ and $1 \leq j \leq n$, for some positive integer c. Given any integer $\mu \geq 2$ and any base $\beta \geq 2$, let $w = \lceil \log_\beta 2c/\log_\beta \mu \rceil$. If $p_1,\ldots,p_w$ are distinct odd primes such that $p_k \geq \mu$, $1 \leq k \leq w$, then $(\varphi_{p_1}(M),\ldots,\varphi_{p_w}(M))$ is a modular representation of M.

Proof: We have $w \geq \log_\beta 2c/\log_\beta \mu$ and so $\log_\beta \mu^w = w\cdot\log_\beta \mu \geq \log_\beta 2c$; hence, $\mu^w \geq 2c$. Moreover, $p_1\cdots p_w \geq \mu^w \geq 2c$ so $p_1\cdots p_w/2 > c$ and we have $p_1\cdots p_w/2 > \text{norm}(M(i,j))$, for $1 \leq i \leq m$ and $1 \leq j \leq n$. Thus, $(N^{(1)},N^{(2)},\ldots,N^{(w)})$, where $N^{(k)} = \varphi_{p_k}(M)$, $1 \leq k \leq w$, is a modular representation of M with respect to primes $p_1,\ldots,p_w$.//

This theorem can now be applied to $M = \bar{\bar{A}}$ with $c = \delta$, giving the next theorem directly.

Theorem 2.4.5 Let A be an m by n nonzero matrix in $\mathbb{m}(I)$ or in $\mathbb{m}(I[x_1,\ldots,x_s])$ and let $c = \delta$ be the maximum of the bounds on the norms of the elements of $\bar{\bar{A}}$ from Theorem 2.4.3. If μ, β, and w and the primes $p_1,\ldots,p_w$ are defined as in Theorem 2.4.4, then $(\varphi_{p_1}(\bar{\bar{A}}),\ldots,\varphi_{p_w}(\bar{\bar{A}}))$ is a modular representation for $\bar{\bar{A}}$.

If the ω_{p_k} of this theorem are commuting mappings for A, then $\omega_{p_k}(\bar{\bar{A}}) = \overline{\overline{\omega_{p_k}(A)}}$, for $1 \le k \le w$, and we have the following corollary.

Corollary 2.4.6 If the mod-p mappings $\omega_{p_1},\ldots,\omega_{p_k}$ of Theorem 2.4.5 are commuting mappings for A, then $(\overline{\overline{\omega_{p_1}(A)}},\ldots,\overline{\overline{\omega_{p_w}(A)}})$ is a modular representation for $\bar{\bar{A}}$.

We will now find an upper bound ρ on the number of non-commuting mod-p mappings for a nonzero matrix A. Let $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$. By the remarks following Theorem 2.3.4, we need only compute ρ as a bound on the number of rejected mod-p mappings for A. That is, we will compute a bound on the number of primes p for which

$$\omega_p\left(A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix}\right) = 0, \text{ for some } k\colon\ 1 \le k \le r.$$

Theorem 2.4.7 Let A be an m by n nonzero matrix in either $\mathbb{M}(I)$ or $\mathbb{M}(I[x_1,\ldots,x_s])$ with $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$. For $1 \le k \le r$, if $\rho'_k = k! \cdot \prod_{u=1}^{k} \max_{1\le v\le k} \operatorname{norm}(A(i_u,j_v))$, then $\rho_k = \lceil \log_2 \rho'_k \rceil$ is an upper bound on the number of prime divisors of $\operatorname{norm}\left(A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix}\right)$.

Proof: Let $\rho''_k = \operatorname{norm}\left(A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix}\right)$, for $1 \le k \le r$. For any k: $1 \le k \le r$, by applying Lemma 2.4.1 to the matrix

$B = A\begin{bmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{bmatrix}$ we see that $\rho_k'' = \text{norm}(\det(B)) \leq k! \cdot \prod_{u=1}^{k} \max_{1\leq v\leq k} \text{norm}(B(u,v)) = \rho_k'$. Letting $\rho_k = \lceil \log_2 \rho_k' \rceil$, we have $\rho_k \geq \log_2 \rho_k'$ and so $2^{\rho_k} \geq \rho_k' \geq \rho_k''$. Let $\rho_k'' = \prod_{i=1}^{q} z_i$, where the z_i are primes, not necessarily distinct. Then $\prod_{i=1}^{q} z_i \leq 2^{\rho_k}$. Suppose $q > \rho_k$. Since each $z_i \geq 2$, it follows that $\rho_k'' = \prod_{i=1}^{q} z_i \geq 2^q > 2^{\rho_k}$, contradicting the fact that $2^{\rho_k} \geq \rho_k''$. Thus, $q \leq \rho_k$ and, since q is an upper bound on the number of prime divisors of ρ_k'', so is ρ_k.//

Using this theorem, we are now able to obtain an upper bound on the number of non-commuting mod-p mappings for a nonzero matrix A.

Theorem 2.4.8 Let A be an m by n nonzero matrix in either $\mathbb{m}(I)$ or $\mathbb{m}(I[x_1,\ldots,x_s])$ and let ρ_k be as defined in Theorem 2.4.7, for $1 \leq k \leq r$. Then $\rho = \sum_{k=1}^{r} \rho_k$ is an upper bound on the number of non-commuting mod-p mappings for A.

Proof: A bound on the number of primes p for which $\varphi_p\left(A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix}\right) = 0$, for some k: $1 \leq k \leq r$, will give us a bound on the number of non-commuting mod-p mappings for A. For $1 \leq k \leq r$, $\varphi_p\left(A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix}\right) = 0$ if and only if p divides each integer coefficient of $A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix}$; hence, we seek a bound d_k on the number of distinct primes, each of which

divide the coefficients of $A\binom{i_1,\ldots,i_k}{j_1,\ldots,j_k}$, for $1 \leq k \leq r$.

For $1 \leq k \leq r$, consider the following argument. Let c_k be the largest absolute value of the integer coefficients of $A\binom{i_1,\ldots,i_k}{j_1,\ldots,j_k}$. Then $\lceil \log_2 c_k \rceil \geq d_k$ and also ρ_k' of Theorem 2.4.7 satisfies $\rho_k' \geq \text{norm}\left(A\binom{i_1,\ldots,i_k}{j_1,\ldots,j_k}\right) \geq c_k$. Thus, $\rho_k = \lceil \log_2 \rho_k' \rceil \geq \lceil \log_2 c_k \rceil \geq d_k$ and so ρ_k is an upper bound on the number of distinct primes, each of which divides the integer coefficients of $A\binom{i_1,\ldots,i_k}{j_1,\ldots,j_k}$. Finally, by summing over k, we have that $\rho = \sum_{k=1}^{r} \rho_k$ is an upper bound on the number of non-commuting mod-p mappings for A.//

There are an infinite number of primes p. Hence, Theorem 2.4.8 implies that, given the integer $\mu \geq 2$, there are an infinite number of commuting mod-p mappings φ_p with $p \geq \mu$, for any nonzero matrix A in $\mathbb{M}(I)$ or $\mathbb{M}(I[x_1,\ldots,x_s])$.

Section 2.5 Evaluation and Interpolation of Matrices

Let $\mathcal{I}$ be an integral domain and $\mathcal{I}[x]$ the ring of polynomials over $\mathcal{I}$. For any $a \in \mathcal{I}$, let Ψ_a be the mapping of $\mathcal{I}[x]$ onto $\mathcal{I}$ such that $\Psi_a(A)=A(a)$, for every $A \in \mathcal{I}[x]$. Ψ_a is a homomorphism, called an evaluation homomorphism for $\mathcal{I}[x]$. Let $Q(\mathcal{I})$ be the quotient field of $\mathcal{I}$ and let $a_0, a_1, \ldots, a_q$ and $b_0, b_1, \ldots, b_q$ be sequences of elements of $Q(\mathcal{I})$, where the a_i are distinct. It is well-known that there exists a unique polynomial $A \in Q(\mathcal{I})[x]$ of degree at most q, which interpolates to b_i at a_i, that is, such that $b_i = \Psi_{a_i}(A)$, for $0 \leq i \leq q$. A sequence $(b_0, b_1, \ldots, b_q)$, where $b_i \in \mathcal{I}$, $0 \leq i \leq q$, is called a value representation of a polynomial $A \in \mathcal{I}[x]$ if $\deg(A) \leq q$ and there exist distinct elements $a_0, a_1, \ldots, a_q$ of $\mathcal{I}$ such that $b_i = \Psi_{a_i}(A)$, $0 \leq i \leq q$.

Thus, every sequence $(a_0, a_1, \ldots, a_q)$ of distinct elements of $\mathcal{I}$ determines a value representation of some polynomial $A \in Q(\mathcal{I}[x])$, for which A is the unique interpolating polynomial. This polynomial may be constructed in $Q(\mathcal{I})[x]$ by a method called incremental interpolation, which is described as follows. For $0 \leq k \leq q$, let $P_k(x) = \prod_{i=0}^{k}(x-a_i)$ and suppose $A_k(x)$ is the unique element of $Q(\mathcal{I})[x]$ with degree at most k such that $A_k(a_i) = b_i$, for $0 \leq i \leq k$. Then, letting $A_{-1}(x) = 0$ and $P_{-1}(x) = 1$,

$$A_k(x) = \{[b_k - A_{k-1}(a_k)]/P_{k-1}(a_k)\}\cdot P_{k-1}(x) + A_{k-1}(x),$$

for $k = 0, 1, \ldots, q$, whence $A = A_q$. These concepts are discussed in [COG69b] and have been applied more recently in [COG71a] and [BRO71].

Let GF(p) be the finite field with p elements and let $GF(p)[x_1, \ldots, x_s]$, $s \geq 1$, be the domain of polynomials in s variables over GF(p). If $s = 1$, then the above definitions hold for $\mathfrak{J} = GF(p)$. If $s > 1$, then we take $\mathfrak{J} = GF(p)[x_1, \ldots, x_{s-1}]$ and apply the definitions to $\mathfrak{J}[x_s]$, for $GF(p)[x_1, \ldots, x_s] \cong GF(p)[x_1, \ldots, x_{s-1}][x_s]$. In this latter case, $P_k(x_s) = \Pi_{i=0}^{k}(x_s - a_i)$, $0 \leq k \leq q$, and if we choose $a_i \in GF(p)$, for $0 \leq i \leq q$, then $[b_k - A_{k-1}(a_k)]/P_{k-1}(a_k) \in \mathfrak{J}$, for $0 \leq i \leq q$. Thus, for $\mathfrak{J} = GF(p)$ or $\mathfrak{J} = GF(p)[x_1, \ldots, x_{s-1}]$, we will have $A_k \in \mathfrak{J}[x]$, for $0 \leq k \leq q$.

It is a simple matter to extend these concepts to $\mathfrak{m}(\mathfrak{J}[x])$, for $\mathfrak{J}$ an arbitrary integral domain. If Ψ_a is an evaluation homomorphism on $\mathfrak{J}[x]$, we define the mapping Ψ_a' of $\mathfrak{m}(\mathfrak{J}[x])$ onto $\mathfrak{m}(\mathfrak{J})$ such that, for an m by n matrix M in $\mathfrak{m}(\mathfrak{J}[x])$, $N = \Psi_a'(M)$ if and only if $N(i,j) = \Psi_a(M(i,j))$, $1 \leq i \leq m$ and $1 \leq j \leq n$. Ψ_a' is called the <u>evaluation mapping</u> on $\mathfrak{m}(\mathfrak{J}[x])$ induced by Ψ_a and will be denoted simply by Ψ_a. We say that a sequence $(N^{(0)}, N^{(1)}, \ldots, N^{(q)})$ of matrices in $\mathfrak{m}(\mathfrak{J})$ is a <u>value representation of a matrix</u>

$M \in \mathbb{m}(\mathcal{I}[x])$, m by n, if $\deg(M(i,j)) \leq q$, for $1 \leq i \leq m$ and $1 \leq j \leq n$, and there exist distinct elements $a_0, a_1, \ldots, a_q$ of $\mathcal{I}$ such that $N^{(k)} = \psi_{a_k}(M)$, for $0 \leq k \leq q$.

Given a sequence $a_0, a_1, \ldots, a_q$ of distinct elements of $Q(\mathcal{I})$ and a sequence $N^{(0)}, N^{(1)}, \ldots, N^{(q)}$ of m by n matrices in $\mathbb{m}(Q(\mathcal{I}))$, there is clearly a unique m by n matrix $M \in \mathbb{m}(Q(\mathcal{I})[x])$, each of whose elements has degree at most q, such that $\psi_{a_k}(M) = N^{(k)}$, for $0 \leq k \leq q$. (i.e., simply apply the results for $Q(\mathcal{I})[x]$ elementwise to the $N^{(k)}$.) Given such a value representation $(N^{(0)}, N^{(1)}, \ldots, N^{(q)})$ of M, one can construct M by incremental interpolation. That is, letting $M^{(-1)}$ be the m by n zero matrix and letting $P_{-1}(x) = 1$ and $P_k(x) = \prod_{i=0}^{k}(x-a_i)$, the sequence $M^{(0)}, M^{(1)}, \ldots, M^{(q)}$ of m by n matrices in $\mathbb{m}(Q(\mathcal{I})[x])$, where $M = M^{(q)}$, can be constructed by applying incremental interpolation elementwise. Hence, for $k = 0,1,\ldots,q$, the polynomial $M^{(k)}(i,j)(x)$ of degree at most k in $Q(\mathcal{I})[x]$ such that $M^{(k)}(i,j)(a_h) = N^{(h)}(i,j)$, for $0 \leq h \leq k$, is obtained by:

$$M^{(k)}(i,j)(x) = \{[N^{(k)}(i,j) - M^{(k-1)}(i,j)(a_k)]$$
$$/P_{k-1}(a_k)\} \cdot P_{k-1}(x) + M^{(k-1)}(i,j)(x),$$

for $1 \leq i \leq m$ and $1 \leq j \leq n$. In the cases where $\mathcal{I} = GF(p)$ or $\mathcal{I} = GF(p)[x_1, \ldots, x_{s-1}]$, $s > 1$, if we choose $a_h \in GF(p)$ and $N^{(h)} \in \mathbb{m}(\mathcal{I})$, $0 \leq h \leq q$, then we will have $M^{(k)} \in \mathbb{m}(\mathcal{I}[x])$,

for $0 \leq k \leq q$.

As in Section 2.4, recall from Theorem 2.1.2.1 that, for A an m by n nonzero matrix in $\mathfrak{m}(\mathfrak{I}[x])$ with $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$, the only possibly nonzero elements of $\bar{\bar{A}}$ are $\bar{\bar{A}}(i,j) = A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_{i-1},j,j_{i+1},\ldots,j_r \end{pmatrix}$, for $1 \leq i \leq r$ and $j_i \leq j \leq n$ and $j \neq j_u$, $i < u \leq r$. Hence, if a bound g on the degrees of these elements is known, then a value representation of g+1 elements for $\bar{\bar{A}}$ is sufficient to construct $\bar{\bar{A}}$ by incremental interpolation. The computation of such a bound will require bounds on the degrees of determinants. We begin with the following lemma, giving one such bound.

Lemma 2.5.1 For every m by m matrix $B \in \mathfrak{m}(\mathfrak{I}[x])$,

$$\deg(\det(B)) \leq \sum_{i=1}^{m} \max_{1\leq j\leq m} \deg(B(i,j)).$$

Proof: Let τ designate an arbitrary element $(t_1,\ldots,t_m)$ in $\wp_m$. Since $\det(B) = \sum_{\tau\in\wp_m} \operatorname{sgn}(\tau)\cdot B(1,t_1)\cdots B(m,t_m)$,

$$\deg(\det(B)) \leq \max_{\tau\in\wp_m} \deg(B(1,t_1)\cdots B(m,t_m)) =$$

$$\max_{\tau\in\wp_m} \sum_{i=1}^{m} \deg(B(i,t_i)) \leq \sum_{i=1}^{m} \max_{\tau\in\wp_m} \deg(B(i,t_i))$$

$$= \sum_{i=1}^{m} \max_{1\leq j\leq m} \deg(B(i,j)).//$$

The next lemma, paralleling Lemma 2.4.2, applies this

result to obtain another bound on the degree of a determinant.

Lemma 2.5.2 Let B be an m by m matrix in $\mathbb{M}(\mathcal{J}[x])$. Then, for any integer i: $1 \le i \le m$,

$$\deg(\det(B)) \le \max_{1\le k\le m} \deg(B(k,i)) + \left(\sum_{h=1}^{m} e_h\right) - \min_{1\le h\le m} e_h,$$

where $e_h = \max_{\substack{1\le t\le m \\ t\ne i}} \deg(B(h,t))$, $1 \le h \le m$.

Proof: By the Laplace expansion theorem:

$$\det(B) = \sum_{k=1}^{m} (-1)^{i+k} B(k,i)\cdot B\begin{pmatrix} 1,\ldots,k-1,k+1,\ldots,m \\ 1,\ldots,i-1,i+1,\ldots,m \end{pmatrix} -$$

Thus, $\deg(\det(B))$

$$\le \max_{1\le k\le m} \left\{\deg(B(k,i)) + \deg\left(B\begin{pmatrix} 1,\ldots,k-1,k+1,\ldots,m \\ 1,\ldots,i-1,i+1,\ldots,m \end{pmatrix}\right)\right\}$$

$$\le \max_{1\le k\le m} \deg(B(k,i)) + \max_{1\le k\le m} \deg\left(B\begin{pmatrix} 1,\ldots,k-1,k+1,\ldots,m \\ 1,\ldots,i-1,i+1,\ldots,m \end{pmatrix}\right)$$

$$\le \max_{1\le k\le m} \deg(B(k,i)) + \max_{1\le k\le m} \sum_{\substack{h=1 \\ h\ne k}}^{m} \max_{\substack{1\le t\le m \\ t\ne i}} \deg(B(h,t)),$$

applying Lemma 2.5.1. Thus, letting $e_h = \max_{\substack{1\le t\le m \\ t\ne i}} \deg(B(h,t))$,

$1 \le h \le m$, $\deg(\det(B)) \le \max_{1\le k\le m} \deg(B(k,i)) + \max_{1\le k\le m} \sum_{\substack{h=1 \\ h\ne k}}^{m} e_h$

$$= \max_{1\le k\le m} \deg(B(k,i)) + \left(\sum_{h=1}^{m} e_h\right) - \min_{1\le h\le m} e_h .//$$

This lemma is applied in the next theorem to obtain bounds on the degrees of individual elements of the determinantal RRE form $\bar{\bar{A}}$.

Theorem 2.5.3 For A an m by n nonzero matrix in $\mathfrak{m}(\mathcal{J}[x])$ with $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$, for $1 \le i \le r$ and $1 \le j \le n$,

$$\deg(\bar{\bar{A}}(i,j)) \le \max_{1\le k\le r} \deg(A(i_k,j)) + \sum_{h=1}^{r} e_{ih} - \min_{1\le h\le r} e_{ih},$$

where $e_{ih} = \max_{\substack{1\le t\le r \\ t\ne i}} \deg(A(i_h,j_t))$, $1 \le h \le r$.

Proof: The theorem follows directly from Lemma 2.5.2, if, for $1 \le i \le r$ and $1 \le j \le n$, we take $B = A\begin{bmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_{i-1},j,j_{i+1},\ldots,j_r \end{bmatrix}$ and note that $\bar{\bar{A}}(i,j) = \det(B)$.//

Let $e_h = \max_{1\le t\le r} \deg(A(i_h,j_t))$, for $1 \le h \le r$, for A nonzero. By Lemma 2.5.1, the diagonal elements of $\bar{\bar{A}}$ satisfy: $\deg(\bar{\bar{A}}(i,j_i)) = \deg(\delta(A)) \le \sum_{h=1}^{r} e_h$, $1 \le i \le r$. Consider the remaining possibly nonzero elements of $\bar{\bar{A}}$: $\bar{\bar{A}}(i,j)$, for $1 \le i \le r$ and $j_i < j \le n$ and $j \ne j_u$, $i < u \le r$. It is not difficult to see, for each of these elements, that

$$\deg(\bar{\bar{A}}(i,j)) \le \max_{1\le k\le r} \max_{\substack{j_k<h\le n \\ h\ne j_u, \\ k<u\le r}} \deg(A(i_k,h)) + \sum_{h=1}^{r} e_{ih} - \min_{1\le h\le r} e_{ih}.$$

Since $\sum_{h=1}^{r} e_{ih} - \min_{1\le h\le r} e_{ih} \le \sum_{h=1}^{r} e_h - \min_{1\le h\le r} e_h$, we thus have the following theorem.

Theorem 2.5.4 For A an m by n nonzero matrix in $\mathfrak{m}(\mathfrak{J}[x])$ with $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$, $\deg(\bar{\bar{A}}(i,j_i)) \le \sum_{h=1}^{r} e_h$ for $1 \le i \le r$, and $\deg(\bar{\bar{A}}(i,j)) \le f + \sum_{h=1}^{r} e_h - \min_{1\le h\le r} e_h$, for $j \ne j_i$, where

$$f = \max_{1\le k\le r} \max_{\substack{j_k<h\le n \\ h\ne j_u, \\ k<u\le r}} \deg(A(i_k,h)) \quad \text{and} \quad e_h = \max_{1\le t\le r} \deg(A(i_h,j_t)),$$

$1 \le h \le r$.

As a consequence of this theorem and the fact that $\Psi_a(\bar{\bar{A}}) = \overline{\overline{\Psi_a(A)}}$, for a commuting mapping Ψ_a for A, we have the following sufficient criterion for a sequence of matrices in $\mathfrak{m}(\mathfrak{J})$ to be a value representation for $\bar{\bar{A}}$.

Theorem 2.5.5 Let η be the larger of the bounds of Theorem 2.5.4 and suppose $a_0,a_1,\ldots,a_\eta$ are distinct elements of $\mathfrak{J}$ such that each Ψ_{a_i} is a commuting mapping for $A \in \mathfrak{m}(\mathfrak{J})$. Then $(\overline{\overline{\Psi_{a_0}(A)}}, \overline{\overline{\Psi_{a_i}(A)}},\ldots,\overline{\overline{\Psi_{a_\eta}(A)}})$ is a value representation for $\bar{\bar{A}}$.

This criterion will be applied in Chapter 4 for $A \in \mathfrak{m}(GF(p)[x_1,\ldots,x_s])$.

As for the mod-p mappings of Section 2.4, a bound on the number of non-commuting evaluation mappings for a matrix A will now be obtained. This will be done by computing an upper bound ν on the number of rejected evaluation mappings for A.

Theorem 2.5.6 For any nonzero matrix $A \in \mathbb{M}(\mathcal{J}[x])$ with $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$, an upper bound ν on the number of rejected evaluation mappings for A is given by

$$\nu = \sum_{k=1}^{r} \sum_{u=1}^{k} \max_{1\le v\le k} \deg(A(i_u,j_v)).$$

Proof: From Theorem 2.3.4 we know that Ψ_a is a rejected evaluation mapping for A if and only if $\Psi_a\left(A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix}\right) = 0$, for some k: $1 \le k \le r$, that is, if and only if a is a root of $A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix}$. Let $\nu_k = \deg\left(A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix}\right)$, $1 \le k \le r$. Since $A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix}$ has at most ν_k distinct roots a in $\mathcal{J}$, there are at most ν_k distinct mappings Ψ_a such that $\Psi_a\left(A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix}\right) = 0$, for $1 \le k \le r$. Thus, $\nu = \sum_{k=1}^{r} \nu_k$ bounds the number of rejected evaluation mappings for A.//

If $\mathcal{J} = GF(p)$ or $\mathcal{J} = GF(p)[x_1,\ldots,x_s]$ and if, for each mapping Ψ_a, we require $a \in GF(p)$, then there are at least $p - \nu$ accepted evaluation mappings for A. Moreover, if the bound η of Theorem 2.5.5 is used to obtain a value representation for $\bar{\bar{A}}$ and if $\eta \le p - \nu$, then the value representation exists and can be found.

Section 2.6 The Computation of a Reduced Row Echelon Form of a Matrix Over a Field

In this section an algorithm or effective mapping Δ of $\mathbb{m}(\mathfrak{F})$ into $\mathbb{m}(\mathfrak{F})$, for $\mathfrak{F}$ an arbitrary field, is defined such that $\Delta(A) = \bar{\bar{A}}$, the determinantal RRE form for A. This algorithm is a Gaussian elimination method and applies only elementary row operations in $\mathbb{m}(\mathfrak{F})$ to obtain $\bar{\bar{A}}$. The matrix A is transformed first to a matrix $\hat{A}$ and then to a matrix $\hat{\hat{A}}$, which are similar in form to $\bar{A}$ and $\bar{\bar{A}}$, respectively. These are defined as follows. Let A be m by n and nonzero and let $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$. Then the m by n matrix $\hat{A}$ in $\mathbb{m}(\mathfrak{F})$ is defined by:

$$\hat{A}(k,j) = \begin{cases} A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_{k-1},j \end{pmatrix} \Big/ A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix}, & 1 \le k \le r \\ & \text{and } 1 \le j \le n \\ 0, & r < k \le m \text{ and } 1 \le j \le n. \end{cases}$$

Similarly, we define the m by n matrix $\hat{\hat{A}}$ in $\mathbb{m}(\mathfrak{F})$ by:

$$\hat{\hat{A}}(k,j) = \begin{cases} A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_{k-1},j,j_{k+1},\ldots,j_r \end{pmatrix} \Big/ A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_r \end{pmatrix}, & \\ & 1 \le k \le r \text{ and } 1 \le j \le n \\ 0, & r < k \le m \text{ and } 1 \le j \le n. \end{cases}$$

If A is a zero matrix, then we define $\hat{\hat{A}} = \hat{A} = A$.

The effective mapping Δ is constructed by first defining an effective mapping Δ_1 of $\mathbb{m}(\mathfrak{F})$ into $\mathfrak{F} \times \mathbb{m}(\mathfrak{F}) \times \mathcal{J} \times \mathcal{P}$ such that $\Delta_1(A) = (\delta(A), \hat{A}, J_A, I_A)$. Then, using Δ_1, an ef-

fective mapping Δ_2 of $\mathbb{m}(\mathfrak{F})$ into $\mathfrak{F} \times \mathbb{m}(\mathfrak{F})$ is defined such $\Delta_2(A) = (\delta(A), \hat{\hat{A}})$. Finally, using Δ_2, the mapping Δ is obtained. In effect, Δ_1 performs a triangularization of a matrix $A \in \mathbb{m}(\mathfrak{F})$, Δ_2 performs a complete diagonalization of A, and Δ diagonalizes A and then modifies that result.

Note that this section parallels Section 2.2, in which the effective mapping Γ is constructed. Hence, frequent reference will be made to that section. In particular, the algorithms and theorems of this section closely parallel those of Section 2.2. The algorithms are likewise stated in terms of elementary row operations, this time for matrices over a field $\mathfrak{F}$. Also, as in that section, the effectiveness of the algorithms will not be proved but will be assumed to be clear from the discussion.

We begin with the algorithm defining Δ_1.

Algorithm 2.6.1 Transforming a Matrix over a Field $\mathfrak{F}$ to Row Echelon Form by Gaussian Elimination

Input: An m by n matrix $A \in \mathbb{m}(\mathfrak{F})$.

Output: The quadruple $(D,B,J,I) \in \mathfrak{F} \times \mathbb{m}(\mathfrak{F}) \times \mathcal{J} \times \mathcal{P}$, where $D = \delta(A)$, $B = \hat{A}$, $J = J_A$, and $I = I_A$.

(1) [Initialize.] $B \leftarrow A$; $J \leftarrow \emptyset$; for $1 \leq h \leq m$, $I[h] \leftarrow h$; $k \leftarrow 1$; $s \leftarrow 0$; $D \leftarrow 1$, the identity of $\mathfrak{F}$.

(2) [Locate a nonzero pivot, if any.] $s \leftarrow s+1$; if

$s > n$, go to (5); for $t = k, k+1, \ldots, m$, search for $B(t,s) \neq 0$; if not found, go to (2); $G \leftarrow B(t,s)$; suffix s to the sequence J.

(3) [Change the row order, if necessary.] If $t = k$, go to (4); $E \leftarrow B_t$; $f \leftarrow I[t]$; for $u = t, t-1, \ldots, k+1$, set $B_u \leftarrow B_{u-1}$ and $I[u] \leftarrow I[u-1]$; then set $B_k \leftarrow E$ and $I[k] \leftarrow f$.

(4) [Recompute rows $B_k, \ldots, B_m$.] $D \leftarrow D \cdot G$; $B_k \leftarrow B_k / G$; if $k = m$, go to (5); for $h = k+1, \ldots, m$, set $G \leftarrow B(h,s)$ and compute $B_h \leftarrow B_h - G \cdot B_k$; $k \leftarrow k+1$; go to (2).

(5) [Return.] If $J = \emptyset$, set $D \leftarrow 0 \in \mathfrak{F}$; return D, B, J, and I.

This algorithm is very similar to Algorithm 2.2.1.1. Let A be an m by n matrix in $\mathfrak{M}(\mathfrak{F})$ and suppose Algorithm 2.2.1.1 generates the sequences $(B^{(0)}, B^{(1)}, \ldots, B^{(r)})$, $(J_0, J_1, \ldots, J_r)$, and $(I_0, I_1, \ldots, I_r)$ in computing $\Gamma_1(A) = (\bar{A}, J_A, I_A)$. Let $\tau_1, \ldots, \tau_r$ be the sequence of row permutations employed by Γ_1 and $\sigma_1, \ldots, \sigma_r$ be the sequence of permutations used to define $I_1, \ldots, I_r$. As in Algorithm 2.2.1.1, Algorithm 2.6.1 performs a sequence of pivot operations, producing the sequences $(B'^{(0)}, B'^{(1)}, \ldots, B'^{(q)})$, $(J'_0, J'_1, \ldots, J'_q)$, $(I'_0, I'_1, \ldots, I'_q)$, and $(D_0, D_1, \ldots, D_q)$, $q \geq 0$, where $B'^{(0)} = A$, $J'_0 = \emptyset$, $I'_0 = (1,2,\ldots,m)$, and $D_0 = 1 \in \mathfrak{F}$.

Also, a sequence of row permutations $\tau_1', \ldots, \tau_q'$ are employed, with $\sigma_1', \ldots, \sigma_q'$ being their inverses used to define $I_1', \ldots, I_q'$.

The k^{th} pivot operation is accomplished in steps (2)-(4) and produces $B'^{(k)} \in \mathbb{m}(\mathfrak{F})$, $J_k' \in \mathcal{J}$, $I_k' \in \mathcal{P}$, and $D_k \in \mathfrak{F}$. Step (2) is identical to step (2) of Algorithm 2.2.1.1 in performing the pivot search and computing J_k'. Also, step (3) is identical in performing the row interchange to define $\tilde{B}'^{(k)}$ and in computing I_k'. In step (4) $B'^{(k)}$ is computed from $\tilde{B}'^{(k)}$ by transforming rows $\tilde{B}_h'^{(k)}$, $k \le h \le m$, by elementary row operations; also, D_k is computed.

If the pivot search of step (2) fails, then $B = B'^{(q)}$, $J = J_q'$, $I = I_q'$, and $D = D_q$ (if $q = 0$, then $D = 0$), where $q = k-1$. If the pivot $B'^{(k-1)}(t,s)$ is found, then $J_k' = (j_1', \ldots, j_{k-1}', j_k')$, where $J_{k-1}' = (j_1', \ldots, j_{k-1}')$ and $j_k' = s$; also, τ_k' and σ_k' are determined and $\tilde{B}_h'^{(k)} = B'^{(k-1)}_{\sigma_k'(h)}$, $1 \le h \le m$, and $I_k' = I_{k-1}' \sigma_k'$ in step (3). Then, in step (4), $D_k = D_{k-1} \cdot B'^{(k-1)}(t,s)$ and $B'^{(k)}$ is computed as follows: $B_h'^{(k)} = \tilde{B}_h'^{(k)}$, for $1 \le h < k$, and $B_k'^{(k)} = \tilde{B}_k'^{(k)} / \tilde{B}'^{(k)}(k, j_k')$, and finally $B_h'^{(k)} = \tilde{B}_h'^{(k)} - \tilde{B}'^{(k)}(h, j_k') \cdot B_k'^{(k)}$, for $k < h \le m$. Clearly, $B'^{(k)}$ is obtained from $B'^{(k-1)}$ by elementary row operations in $\mathbb{m}(\mathfrak{F})$.

We now present a theorem, justifying the claims of Algorithm 2.6.1, which parallels Theorem 2.2.1.4.

Theorem 2.6.2 Let $A \in \mathfrak{m}(\mathfrak{F})$ and let $\Delta_1(A) = (D,B,J,I)$. Then $D = \delta(A)$, $B = \hat{A}$, $J = J_A$, and $I = I_A$, and $\hat{A}$ is an RE form in $\mathfrak{m}(\mathfrak{F})$ for A.

Proof: If A is a zero matrix, then $D = 0 = \delta(A)$, $J = J_0' = \emptyset = J_A$, $I = (1,2,\ldots,m) = I_A$, and $B = B'^{(0)} = A = \hat{A}$, an RE form for A. Suppose A is m by n and nonzero and let $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$. Recall that Algorithm 2.2.1.1 generates $I_v = (i_{v,1},\ldots,i_{v,m})$ and $\tilde{B}^{(v)}$ at the v^{th} pivot operation, $v > 0$, in computing $\Gamma_1(A)$. Consider the following inductive hypothesis: for v:

$0 \le v \le q$, $J_v' = (j_1,\ldots,j_v)$, $I_v' = (i_1,\ldots,i_v,i_{v,v+1},\ldots,i_{v,m})$, $D_v = A\begin{pmatrix} i_1,\ldots,i_v \\ j_1,\ldots,j_v \end{pmatrix}$, and $B'^{(v)}$ is row equivalent in $\mathfrak{m}(\mathfrak{F})$ to A and is defined by:

$$B'^{(v)}(h,j) = \begin{cases} A\begin{pmatrix} i_1,\ldots,i_h \\ j_1,\ldots,j_{h-1},j \end{pmatrix} / A\begin{pmatrix} i_1,\ldots,i_h \\ j_1,\ldots,j_h \end{pmatrix}, & 1 \le h \le v \text{ and } 1 \le j \le n \\ A\begin{pmatrix} i_1,\ldots,i_v,i_{v,h} \\ j_1,\ldots,j_v,j \end{pmatrix} / A\begin{pmatrix} i_1,\ldots,i_v \\ j_1,\ldots,j_v \end{pmatrix}, & v < h \le m \text{ and } 1 \le j \le n. \end{cases}$$

Note that for $v = 0$ we take $A\begin{pmatrix} i_1,\ldots,i_v \\ j_1,\ldots,j_v \end{pmatrix} = 1$ by convention. For $v = 0$, the hypothesis is satisfied, since $J_0' = \emptyset$, $I_0' = (1,\ldots,m) = (i_{0,1},\ldots,i_{0,m}) = I_0$, $D_0 = 1$, and $B^{(0)}$ is defined by: $B'^{(0)}(h,j) = A(h,j) = A\binom{h}{j} = A\binom{i_0,h}{j}/1$, $0 = v < h \le m$ and $1 \le j \le n$. Suppose that the inductive hypothesis holds for $v = k-1$, $0 < k \le q$; we show that it

holds for $v = k$.

For $k \le h \le m$ and $1 \le j \le n$, we know from Theorem 2.2.1.4 that $B^{(k-1)}(h,j) = A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1},h \\ j_1,\ldots,j_{k-1},j \end{pmatrix}$ and from the inductive hypothesis that $B'^{(k-1)}(h,j) = A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1},h \\ j_1,\ldots,j_{k-1},j \end{pmatrix} / A\begin{pmatrix} i_1,\ldots,i_{k-1} \\ j_1,\ldots,j_{k-1} \end{pmatrix}$. Hence, for $k \le h \le m$ and $1 \le j \le n$, $B'^{(k-1)}(h,j) = 0$ if and only if $B^{(k-1)}(h,j) = 0$ and so $B'^{(k-1)}(t,s)$ is the k^{th} pivot in computing $\Delta_1(A)$ if and only if $B^{(k-1)}(t,s)$ is the k^{th} pivot in computing $\Gamma_1(A)$. Therefore, τ_k is the k^{th} row permutation in computing $\Delta_1(A)$ and so $\tilde{B}'^{(k)}_h = B'^{(k-1)}_{\sigma_k(h)}$, $1 \le h \le m$. Also, $J'_k = (j_1,\ldots,j_{k-1},j_k) = J_k$ and $I'_k = I'_{k-1}\ \sigma_k = I_{k-1}\ \sigma_k = I_k = (i_1,\ldots,i_k,i_{k,k+1},\ldots,i_{k,m})$. Moreover, $D_k = D_{k-1}\cdot B'^{(k-1)}(t,s) = D_{k-1}\cdot B'^{(k-1)}(\sigma_k(k),j_k) = A\begin{pmatrix} i_1,\ldots,i_{k-1} \\ j_1,\ldots,j_{k-1} \end{pmatrix}\cdot A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1,\sigma_k(k)} \\ j_1,\ldots,j_{k-1},j_k \end{pmatrix} / A\begin{pmatrix} i_1,\ldots,i_{k-1} \\ j_1,\ldots,j_{k-1} \end{pmatrix} = A\begin{pmatrix} i_1,\ldots,i_{k-1},i_k \\ j_1,\ldots,j_{k-1},j_k \end{pmatrix}$, since $i_{k-1,\sigma_k(k)} = i_{k,k} = i_k$.

Finally, we show that $B'^{(k)}$ satisfies the inductive hypothesis. For $1 \le h < k$, $B'^{(k)}_h = \tilde{B}'^{(k)}_h = B'^{(k-1)}_{\sigma_k(h)} = B'^{(k-1)}_h$, satisfying the hypothesis. For $h = k$, $B'^{(k)}_k = \tilde{B}'^{(k)}_k / \tilde{B}'^{(k)}(k,j_k) = B'^{(k-1)}_{\sigma_k(k)} / B'^{(k-1)}(\sigma_k(k),j_k)$ and so, for $1 \le j \le n$,

$$B'^{(k)}(k,j) = B'^{(k-1)}(\sigma_k(k),j)/B'^{(k-1)}(\sigma_k(k),j_k)$$

$$= \frac{A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1},\sigma_k(k) \\ j_1,\ldots,j_{k-1},j \end{pmatrix}/A\begin{pmatrix} i_1,\ldots,i_{k-1} \\ j_1,\ldots,j_{k-1} \end{pmatrix}}{A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1},\sigma_k(k) \\ j_1,\ldots,j_{k-1},j_k \end{pmatrix}/A\begin{pmatrix} i_1,\ldots,i_{k-1} \\ j_1,\ldots,j_{k-1} \end{pmatrix}}$$

$$= A\begin{pmatrix} i_1,\ldots,i_{k-1},i_k \\ j_1,\ldots,j_{k-1},j \end{pmatrix} /A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix}$$

For $k < h \leq m$, the elements of row $B'^{(k)}_h$ are computed by:

$$B'^{(k)}(h,j) = \tilde{B}'^{(k)}(h,j) - \tilde{B}'^{(k)}(h,j_k)\cdot B'^{(k)}(k,j),\ 1 \leq j \leq n.$$

So, letting $t = \sigma_k(k)$ and $u = \sigma_k(h)$ and recalling that $i_{k-1,\sigma_k(k)} = i_{k,k} = i_k$ and $i_{k-1,\sigma_k(h)} = i_{k,h}$, we obtain:

$$B'^{(k)}(h,j) = B'^{(k-1)}(u,j) - B'^{(k-1)}(u,j_k)\cdot B'^{(k-1)}(t,j)/$$

$$B'^{(k-1)}(t,j_k)$$

$$= \frac{A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1},u \\ j_1,\ldots,j_{k-1},j \end{pmatrix}}{A\begin{pmatrix} i_1,\ldots,i_{k-1} \\ j_1,\ldots,j_{k-1} \end{pmatrix}} - \frac{A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1},u \\ j_1,\ldots,j_k \end{pmatrix}}{A\begin{pmatrix} i_1,\ldots,i_{k-1} \\ j_1,\ldots,j_{k-1} \end{pmatrix}} \cdot$$

$$\frac{A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1},t \\ j_1,\ldots,j_{k-1},j \end{pmatrix}}{A\begin{pmatrix} i_1,\ldots,i_{k-1} \\ j_1,\ldots,j_{k-1} \end{pmatrix}} \Bigg/ \frac{A\begin{pmatrix} i_1,\ldots,i_{k-1},i_{k-1},t \\ j_1,\ldots,j_{k-1},j \end{pmatrix}}{A\begin{pmatrix} i_1,\ldots,i_{k-1} \\ j_1,\ldots,j_{k-1} \end{pmatrix}}$$

$$= \frac{A\begin{pmatrix} i_1,\ldots,i_{k-1},i_k,h \\ j_1,\ldots,j_{k-1},j \end{pmatrix}}{A\begin{pmatrix} i_1,\ldots,i_{k-1} \\ j_1,\ldots,j_{k-1} \end{pmatrix}} - \frac{A\begin{pmatrix} i_1,\ldots,i_{k-1},i_k,h \\ j_1,\ldots,j_k \end{pmatrix}\cdot A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_{k-1},j \end{pmatrix}}{A\begin{pmatrix} i_1,\ldots,i_{k-1} \\ j_1,\ldots,j_{k-1} \end{pmatrix}\cdot A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix}}$$

$$= \left[A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix} \cdot A\begin{pmatrix} i_1,\ldots,i_{k-1},i_k,h \\ j_1,\ldots,j_{k-1},j \end{pmatrix} - A\begin{pmatrix} i_1,\ldots,i_{k-1},i_k,h \\ j_1,\ldots,j_k \end{pmatrix} \cdot A\begin{pmatrix} i_1,\ldots,\ldots,i_k \\ j_1,\ldots,j_{k-1},j \end{pmatrix} \right]$$

$$\Big/ \left[A\begin{pmatrix} i_1,\ldots,i_{k-1} \\ j_1,\ldots,j_{k-1} \end{pmatrix} \cdot A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix} \right].$$

As in Theorem 2.2.1.4, if we apply Lemma 2.2.1.3 to the matrix $M = A\begin{bmatrix} i_1,\ldots,i_k,i_k,h \\ j_1,\ldots,j_k,j \end{bmatrix}$, we obtain the identity:

$$A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix} \cdot A\begin{pmatrix} i_1,\ldots,i_{k-1},i_k,h \\ j_1,\ldots,j_{k-1},j \end{pmatrix} - A\begin{pmatrix} i_1,\ldots,i_{k-1},i_k,h \\ j_1,\ldots,\ldots,j_k \end{pmatrix} \cdot A\begin{pmatrix} i_1,\ldots,\ldots,i_k \\ j_1,\ldots,j_{k-1},j \end{pmatrix} = A\begin{pmatrix} i_1,\ldots,i_{k-1} \\ j_1,\ldots,j_{k-1} \end{pmatrix} \cdot A\begin{pmatrix} i_1,\ldots,i_k,i_k,h \\ j_1,\ldots,\ldots,j_k,j \end{pmatrix}$$

Hence, $B'^{(k)}(h,j) = A\begin{pmatrix} i_1,\ldots,i_k,i_k,h \\ j_1,\ldots,j_k,j \end{pmatrix} \Big/ A\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix}$, for $k < h \leq m$ and $1 \leq j \leq n$. This completes the inductive step.

It has been shown above that the same number of pivot operations are performed in computing $\Gamma_1(A)$ and $\Delta_1(A)$. Hence, $q = r$ and we have $I' = I'_r = I_r = I_A$, $J' = J'_r = J_r = J_A$, and $D = D_r = A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_r \end{pmatrix} = \delta(A)$. Since $B'^{(r)}$ is defined by:

$$B'^{(r)}(h,j) = \begin{cases} A\begin{pmatrix} i_1,\ldots,i_h \\ j_1,\ldots,j_{h-1},j \end{pmatrix} \Big/ A\begin{pmatrix} i_1,\ldots,i_h \\ j_1,\ldots,j_h \end{pmatrix}, & 1 \leq h \leq r \text{ and } 1 \leq j \leq n \\ A\begin{pmatrix} i_1,\ldots,i_r,i_h \\ j_1,\ldots,j_r,j \end{pmatrix} \Big/ A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_r \end{pmatrix}, & r < h \leq m \text{ and } 1 \leq j \leq n, \end{cases}$$

we have that, for $1 \le j \le r$, $B'^{(r)}(h,j) = B^{(r)}(h,j)/A\begin{pmatrix} i_1,\ldots,i_h \\ j_1,\ldots,j_h \end{pmatrix}$, $1 \le h \le r$, and $B'^{(r)}(h,j) = B^{(r)}(h,j)/A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_r \end{pmatrix}$, $r < h \le m$. Thus, for $1 \le h \le m$ and $1 \le j \le n$, $B'^{(r)}(h,j) = 0$ if and only if $B^{(r)}(h,j) = \bar{A}(h,j) = 0$. This implies that, for $1 \le h \le r$, $B'^{(r)}(h,j) = 0$, $1 \le j < j_h$, and also $B'^{(r)}(h,j) = 0$, for $1 \le j \le n$ and $r < h \le m$. Therefore, $B = B'^{(r)} = \hat{A}$, which is also row equivalent to A, making $\hat{A}$ an RE form for A.//

Note that the diagonal elements $\hat{A}(h,j_h) = 1$, for $1 \le h \le r$, where A is nonzero.

We next present the algorithm defining the effective mapping Δ_2 of $\mathbb{m}(\mathfrak{F})$ into $\mathfrak{F} \times \mathbb{m}(\mathfrak{F})$. This algorithm closely parallels Algorithm 2.2.2.1, which defines the effective mapping Γ on $\mathbb{m}(\mathfrak{I})$.

Algorithm 2.6.3 Transforming a Matrix over a Field $\mathfrak{F}$ to Reduced Row Echelon Form by Gaussian Elimination

Input: An m by n matrix $A \in \mathbb{m}(\mathfrak{F})$.

Output: The pair (D,C) in $\mathfrak{F} \times \mathbb{m}(\mathfrak{F})$, where $D = \delta(A)$ and $C = \hat{\hat{A}}$.

(1) [Triangularize by Gaussian Elimination.] Compute $\Delta_1(A) = (D,B,J,I)$ by Algorithm 2.6.1.

(2) [Initialize the diagonalization.] $C \leftarrow B$; $r \leftarrow$ rank(A); if $r = 0$ or $r = 1$, return D and C; otherwise, have $J = (j_1,\ldots,j_r)$; $i \leftarrow r$.

(3) [Diagonalize by Gaussian elimination.] $i \leftarrow i-1$; for $t = i+1,\ldots,r$, set $F \leftarrow C(i,j_t)$ and compute $C_i \leftarrow C_i - F\cdot C_t$; if $i > 1$, go to (3); return D and C.

Similar to Algorithm 2.2.2.1, a sequence of m by n matrices in $\mathbb{m}(\mathfrak{F})$ is generated, where $C^{(0)} = B = \hat{A}$ and $C = C^{(r-1)}$. Each $C^{(i)}$, $1 \le i \le r-1$, is computed from $C^{(i-1)}$ by a finite sequence of elementary row operations. This computation is summarized as follows: for $1 \le h \le m$, $h \ne r-i$, $C_h^{(i)} = C_h^{(i-1)}$, and for $h = r-i$:

$$C_{r-i}^{(i)} = C_{r-i}^{(i-1)} - \sum_{t=r-i+1}^{r} C^{(i-1)}(r-i,j_t)\cdot C_t^{(i-1)}.$$

Thus, $C^{(i)}$ is row equivalent in $\mathbb{m}(\mathfrak{F})$ to $C^{(i-1)}$, for $1 \le i \le r-1$, and hence $C = C^{(r-1)}$ is row equivalent to $C^{(0)} = B = \hat{A}$. Since $\hat{A}$ is row equivalent to A, C is row equivalent to A.

We now show that C is an RRE matrix with diagonal elements $C(i,j_i)$, $1 \le i \le r$, equal to unity. Let the inductive hypothesis for v: $0 \le v < r$ be: for $1 \le h < r-v$ or $r < h \le m$, $C_h^{(v)} = \hat{A}_h$, and for $r-v \le h \le r$, $C^{(v)}(h,j_h) = 1$ and $C^{(v)}(h,j) = 0$, $1 \le j < j_h$ or $j = j_t$, $h < t \le r$. Letting $z = r-v$, $0 < v \le r-1$, the elements of row $C_z^{(v)}$ are computed

by

$$C^{(v)}(z,j) = C^{(v-1)}(z,j) - \sum_{t=z+1}^{r} C^{(v-1)}(z,j_t)\cdot C^{(v-1)}(t,j).$$

Using this equation, the proof of the inductive hypothesis follows exactly as in Theorem 2.2.2.2 and we shall not repeat it here for Algorithm 2.6.3. We note, however, that $C^{(v)}(z,j_z) = C^{(v-1)}(z,j_z) - \sum_{t=z+1}^{r} C^{(v-1)}(z,j_t)\cdot 0 = C^{(v-1)}(z,j_z)$ and so $C(z,j_z) = C^{(v)}(z,j_z) = C^{(0)}(z,j_z) = \hat{A}(z,j_z) = 1$.

Thus, $C = C^{(r-1)}$ is an RRE matrix row equivalent to A and so is an RRE form in $\mathbb{m}(\mathfrak{F})$ for A. These results are summarized in the following theorem, which is similar to Theorem 2.2.2.2.

Theorem 2.6.4 For every $A \in \mathbb{m}(\mathfrak{F})$, $\mathfrak{F}$ a field, $\Delta_2(A) = (D,C)$ is defined, where $D = \delta(A)$ and C is an RRE form in $\mathbb{m}(\mathfrak{F})$ for A. If A is nonzero with $J_A = (j_1,\ldots,j_r)$, then the diagonal elements $C(i,j_i)$, $1 \le i \le r$, equal unity.

The following lemma will be required to prove Theorem 2.6.6. It parallels Lemma 2.2.2.3.

Lemma 2.6.5 Let A be an m by n nonzero matrix in $\mathbb{m}(\mathfrak{F})$ with $J_A = (j_1,\ldots,j_r)$ and $I_A = (i_1,\ldots,i_m)$. Define the matrix $M = A\begin{bmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_r,j \end{bmatrix}$, for some j: $1 \le j \le n$. If $\Delta_2(M) = (E,N)$ and $\Delta_2(A) = (D,C)$, then $N = C\begin{bmatrix} 1,\ldots,r \\ j_1,\ldots,j_r,j \end{bmatrix}$.

This lemma is proved in a manner identical to that of Lemma 2.2.2.3 and its proof is not included. Simply replace $\mathcal{J}$ by $\mathfrak{F}$ and the mapping Γ by Δ_2. The next theorem shows that the RRE form C computed by Δ_2 is, in fact, $\hat{\hat{A}}$.

Theorem 2.6.6 Let $A \in \mathbb{M}(\mathfrak{F})$ and suppose $\Delta_2(A) = (D,C)$. Then $C = \hat{\hat{A}}$.

Proof: If A is a zero matrix, then $C = A = \hat{\hat{A}}$. Suppose A is m by n and nonzero, with $J_A = (j_1,\ldots,j_r)$, and $I_A = (i_1,\ldots,i_m)$. If $r = n$, then $j_h = h$, for $1 \le h \le r$, and the only nonzero elements of C are the diagonal elements $C(h,h) = 1$; so, $C = \hat{\hat{A}}$.

Suppose $r < n$. Let $M = A\begin{bmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_r,j \end{bmatrix}$, for some j: $1 \le j \le n$. By Lemma 2.6.5, if $\Delta_2(M) = (E,N)$, then $N = C\begin{bmatrix} 1,\ldots,r \\ j_1,\ldots,j_r,j \end{bmatrix}$. Moreover, by Cramer's Rule M can be transformed by elementary row operations in $\mathbb{M}(\mathfrak{F})$ to a matrix N' defined by:

$$N'(h,k) = \begin{cases} 1, \text{ if } h = k = 1,2,\ldots,r \\ M\begin{pmatrix} 1,\ldots,r \\ 1,\ldots,h-1,r+1,h+1,\ldots,r \end{pmatrix} \Big/ M\begin{pmatrix} 1,\ldots,r \\ 1,\ldots,r \end{pmatrix}, \\ \qquad\qquad \text{for } 1 \le h \le r \text{ and } k = r+1 \\ 0, \text{ elsewhere.} \end{cases}$$

Hence, N' is an RRE form in $\mathbb{M}(\mathfrak{F})$ for A with diagonal elements equal to unity. By a theorem in Chapter 7 of [BIG65],

we know that there is exactly one such RRE form for a matrix over a field and so N' = N. In particular, for $1 \le h \le r$ and for any j: $1 \le j \le n$,

$$C(h,j) = N(h,r+1)$$

$$= M\begin{pmatrix}1,\ldots,r\\ 1,\ldots,h-1,r+1,h+1,\ldots,r\end{pmatrix}/M\begin{pmatrix}1,\ldots,r\\ 1,\ldots,r\end{pmatrix}$$

$$= A\begin{pmatrix}i_1,\ldots,i_r\\ j_1,\ldots,j_{h-1},j,j_{h+1},\ldots,j_r\end{pmatrix}/A\begin{pmatrix}i_1,\ldots,i_r\\ j_1,\ldots,j_r\end{pmatrix}$$

$$= \hat{\hat{A}}(h,j).$$

We know that $C(h,j) = 0 = \hat{\hat{A}}(h,j)$, elsewhere, and so we have that $C = \hat{\hat{A}}$.//

We finally present the algorithm defining the effective mapping Δ of $\mathbb{m}(\mathfrak{F})$ into $\mathbb{m}(\mathfrak{F})$.

Algorithm 2.6.7 Transforming a Matrix over a Field $\mathfrak{F}$ to Determinantal RRE Form by Gaussian Elimination

Input: An m by n matrix $A \in \mathbb{m}(\mathfrak{F})$.

Output: The m by n matrix $W = \bar{\bar{A}}$.

(1) [Diagonalize by Gaussian elimination.] Compute $\Delta_2(A) = (D,C)$ by Algorithm 2.6.3.

(2) [Modify the first r rows of C, if necessary.] $W \leftarrow C$; $r \leftarrow \text{rank}(A)$; if $r = 0$, go to (3); for $i = 1,2,\ldots,r$, compute $W_i \leftarrow D \cdot W_i$.

(3) [Return.] Return W.

This algorithm simply employs Algorithm 2.6.3 to obtain $D = \delta(A)$ and $C = \hat{A}$ and possibly modifies C, obtaining $W \in \mathfrak{m}(\mathfrak{F})$. If A is a zero matrix, then $W = C = \hat{A} = A = \bar{\bar{A}}$. If A is nonzero, then W is produced by multiplying rows $C_1,\ldots,C_r$ by $\delta(A) = A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_r \end{pmatrix}$. Hence, for $1 \le i \le r$, $W_i = \delta(A)\cdot\hat{\hat{A}}_i$ and so $W(i,j) =$

$$A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_r \end{pmatrix}\cdot A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_{i-1},j,j_{i+1},\ldots,j_r \end{pmatrix}/A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_r \end{pmatrix} =$$

$$A\begin{pmatrix} i_1,\ldots,i_r \\ j_1,\ldots,j_{i-1},j,j_{i+1},\ldots,j_r \end{pmatrix} = \bar{\bar{A}}(i,j),\ j_i \le j \le n \text{ and}$$

$j \neq j_u$, $i < u \le r$. All other elements of row W_i are zero. Since rows $W_{r+1},\ldots,W_m$ are zero, we have that $W = \bar{\bar{A}}$. We have just proved the following theorem.

Theorem 2.6.8 For every $A \in \mathfrak{m}(\mathfrak{F})$, $\Delta(A)$ is defined and $\Delta(A) = \bar{\bar{A}}$.

Possible fields $\mathfrak{F}$ to which Algorithm 2.4.7 may be applied to compute $\Delta(A) = \bar{\bar{A}}$, for $A \in \mathfrak{m}(\mathfrak{F})$, are the finite field GF(p) with p elements, the rational number field Q(I), and the field of rational functions $Q(I[x_1,\ldots,x_s])$.

CHAPTER III.

ALGORITHMS FOR MATRIX ALGEBRA

In this chapter we present algorithms for computing the sum, difference, and product of two matrices. Auxiliary algorithms related to these algorithms will also be included. Computing times will be obtained for all algorithms in this and succeeding chapters; hence, the basic concepts and notation for the computing time analysis are presented in Section 3.1. The classical algorithms for computing the sum, difference, and product of two matrices are given in Section 3.2. In Section 3.3 algorithms for applying mod-p and evaluation mappings to matrices and for applying Garner's method and incremental interpolation to matrices are presented. These auxiliary algorithms are employed in an evaluation mapping algorithm for multiplying matrices over $GF(p)[x_1,\ldots,x_s]$, $s \geq 1$, and in a mod-p mapping (or modular) algorithm for multiplying matrices over $I[x_1,\ldots,x_s]$, $s \geq 0$. The new algorithms for matrix multiplication are given in Section 3.4. These algorithms will be required in the algorithms for solving systems of linear equations, which will be presented in the next chapter.

3.1 Computing Time Analysis

Let f and g be real-valued functions on a set $\mathcal{S}$. If there is a real number c such that $|f(x)| \leq c \cdot |g(x)|$, for all $x \in \mathcal{S}$, then we say that f is dominated by g and we write $f \preceq g$. If $f \preceq g$ and $g \preceq f$, we say that f and g are codominant and we write $f \sim g$. Of course, $f \succeq g$ means $g \preceq f$ and is read: f dominates g. This is the notation to be used to analyse the computing times of the algorithms and has been introduced by G. E. Collins, [COG71a], and W. S. Brown, [BRO71].

The general framework for the analyses is as follows. We are given an algorithm $\mathcal{A}$ and a set $\mathcal{S}$ of "inputs" to $\mathcal{A}$. Note that each $x \in \mathcal{S}$ can be considered an n-tuple. We also have a set $\mathcal{R}$ of m-tuples $y = (y_1, \ldots, y_m)$ describing the inputs; more precisely, there is a mapping which assigns to each $y \in \mathcal{R}$ a unique finite nonempty subset $\mathcal{S}_y$ of $\mathcal{S}$. Let $t_{\mathcal{A}}(x)$ denote the time (in some convenient unit) to apply algorithm $\mathcal{A}$ to the input x. We then define the function $T_{\mathcal{A}}$ on $\mathcal{R}$ by: $T_{\mathcal{A}}(y) = \max\{t_{\mathcal{A}}(x) : x \in \mathcal{S}_y\}$. We call $T_{\mathcal{A}}$ the maximum computing time function for the algorithm $\mathcal{A}$ with respect to the set $\mathcal{R}$. Analogously, we can define the minimum computing time function $U_{\mathcal{A}}$ for algorithm $\mathcal{A}$ by: $U_{\mathcal{A}}(y) = \min\{t_{\mathcal{A}}(x) : x \in \mathcal{S}_y\}$. For some algorithms there is a great discrepancy between the maximum computing time (i.e., the worst case)

and the time for a typical performance. A computing time function V_G reflecting the expected behavior of an algorithm G is thus defined as follows: $V_G(y) = (\sum_{x \in S_y} t_G(x))/\|S_y\|$, where $\|Q\|$ is the number of elements in a finite set Q. The function V_G is called the <u>average computing time function</u> for algorithm G with respect to $\mathcal{R}$.

The relation between T_G, U_G, and V_G is given, in terms of dominance relations, by: $U_G \lesssim V_G \lesssim T_G$. For we clearly have, for all $y \in \mathcal{R}$, that $V_G(y) = (\sum_{x \in S_y} t_G(x))/\|S_y\| \leq (\sum_{x \in S_y} T_G(y))/\|S_y\| = \|S_y\| \cdot T_G(y)/\|S_y\| = T_G(y)$. Similarly, for each $y \in \mathcal{R}$, $U_G(y) \leq V_G(y)$. For each algorithm, a dominance relation will be found. That is, a function f on $\mathcal{R}$ will be found such that $T_G \lesssim f$. Hence, without it explicitly being stated, we will have $U_G \lesssim f$ and $V_G \lesssim f$. For some algorithms, however, it will be possible to show that $U_G \sim T_G \sim f$. In such cases it is clearly also known that $V_G \sim f$. Such a function f as above will be referred to, when the need arises, appropriately as a <u>dominating</u> (or <u>codominating</u>) <u>function</u> for the maximum computing time function and will be denoted by T'_G.

Roughly, the approach in analysing an algorithm G will be to obtain the computing time for each step from the number of times it is executed and the times for the individual executions. The times for these steps are then combined to

form the total computing time for the algorithm $\mathcal{A}$. To be more precise, we let N_i denote the maximum number of times step i is executed, $T_{i,j}$ denote the maximum computing time for the j^{th} execution of step i, and T_i denote the maximum total computing time for all executions of step i. Each of these quantities will have a dominating (or codominating) function, denoted by N_i', $T_{i,j}'$, and T_i', respectively. For the more complicated algorithms, the analysis may be organized by means of a computing time table. Such a table will usually have four columns, labeled: i, N_i, $T_{i,j}$, and T_i. The entries in the latter three columns are explicitly given functions N_i', $T_{i,j}'$, and T_i', which are assumed to be dominating functions, unless otherwise stated. If $N_i' = 1$, then $T_{i,j}'$ will be omitted, since in this case $T_i' = T_{i,j}'$.

The elements of the input set $\mathcal{S}$ will usually be defined in terms of the integers I, the elements of a finite field GF(p), or polynomials over either I or GF(p). We note that in what follows, when $s = 0$, $I[x_1,\ldots,x_s]$ denotes I and $GF(p)[x_1,\ldots,x_s]$ denotes GF(p). It will be necessary to define the sets $\mathcal{S}_y$ in order to do the computing time analyses and frequently this will require the definition of certain subsets of $I[x_1,\ldots,x_s]$, $s \geq 0$, and $GF(p)[x_1,\ldots,x_s]$, $s \geq 1$. The required subsets are defined as follows. Let P(d) denote the set of all integers b such that $|b| \leq d$ and

let $P(d,m_1,\ldots,m_s)$ denote the set of all polynomials A in $I[x_1,\ldots,x_s]$ such that norm(A) $\leq$ d and the degree of A in x_i is at most m_i. We introduce the convention that $P(d,m_1,\ldots,m_s)$ denotes P(d), for the case s = 0. Also, let $P^*(m_1,\ldots,m_s)$ be the set of all polynomials A in $GF(p)[x_1,\ldots,x_s]$, $s \geq 0$, such that the degree of A in x_i is at most m_i. Similarly, $P^*(m_1,\ldots,m_s)$ denotes GF(p), when s = 0.

Typically, some components of the inputs $x \in S$ will be matrices over $I[x_1,\ldots,x_s]$, $P(d,m_1,\ldots,m_s)$, $GF(p)[x_1,\ldots,x_s]$, or $P^*(m_1,\ldots,m_s)$, for $s \geq 0$. Hence, we define M(m,n,S) as the set of all m by n matrices with elements in the set S.

Since the sequence $m_1,\ldots,m_s$ of degree bounds occurs often, we let $\bar{m}_s$ denote the vector $(m_1,\ldots,m_s)$, $s \geq 1$, and $\bar{m}_o$ denote the null vector $\emptyset$. Similarly, $\bar{n}_s$ denotes the vector $(n_1,\ldots,n_s)$ of degree bounds and $\bar{n}_o = \emptyset$. Thus, for example, $P(d,\bar{m}_s)$ denotes $P(d,m_1,\ldots,m_s)$ and $P^*(\bar{m}_s)$ denotes $P^*(m_1,\ldots,m_s)$, for $s \geq 0$. In fact, the sequence $m_1,\ldots,m_s$ and the vector $\bar{m}_s$ can be used interchangeably. Similarly for $n_1,\ldots,n_s$ and $\bar{n}_s$.

The algorithms of this and subsequent chapters are designed to expect mathematical objects which are represented as list structures in the SAC-1 Mathematical Symbol Manipulation System. The list representations for elements of I, $I[x_1,\ldots,x_s]$, $s \geq 1$, and $GF(p)[x_1,\ldots,x_s]$, $s \geq 0$, are as

described in [COG68a], [COG68b], and [COG69a], respectively. The construction of other kinds of lists is done by means of the capabilities supplied in [COG67]. The algorithms described in these references will be referred to by name when employed in the algorithms. Their computing times can usually be found in [COG69a] or [COG69b]. If they are not given there, then they may be easily derived.

The data representation for several mathematical objects consists in single-precision integer representation (i.e., atoms), that is, integers with magnitudes less than γ, the maximum single-precision positive integer (Fortran integer), plus one (see [COG68a]). Some integer quantities assumed to be less than γ are: the radix β for the infinite-precision integer (L-integer) representation, prime numbers p when used to specify a finite field GF(p), the elements of GF(p), the degrees and degree bounds for polynomials, the integer elements of sequences in Ω, and the dimensions of matrices.

Two list structures of special importance which should be described are those for sequences (vectors) in Ω and for matrices. Thus, a non-null sequence $H = (h_1,\ldots,h_k)$ in Ω is represented as the first-order list $(h_1',\ldots,h_k')$, where h_i' is the atom representing h_i. The null sequence $\emptyset$ is represented by the null list (). A matrix A in M(m,n,S), for

some set S, is represented <u>rowwise</u> by the list $(A_1,\ldots,A_m)$, where A_i is the list $(A_{i,1},\ldots,A_{i,m})$ representing row i. Thus, if S = GF(p), then $A_{i,j}$ is the atom representing A(i,j) and if S is I, $I[x_1,\ldots,x_s]$, or $GF(p)[x_1,\ldots,x_s]$, $s \geq 1$, or a subset of one of these sets, then $A_{i,j}$ is the list representing the integer or polynomial A(i,j). The <u>columnwise</u> representation is defined similarly as the list $(A_1,\ldots,A_n)$, where A_j is the list $(A_{1,j},\ldots,A_{m,j})$ representing column j. The rowwise representation will be taken to be standard and exceptions will be properly noted.

A polynomial in $P(d,m_1,\ldots,m_s)$ or in $P^*(m_1,\ldots,m_s)$, $s \geq 1$, may conceivably have $\pi_{i=1}^{s}(m_i+1)$ terms. This expression will occur often enough to warrant a more compact notation and so we define $\mu_i = m_i+1$, for $1 \leq i \leq s$, and $\mu = \max_{1\leq i\leq s} \mu_i$. In some algorithms, polynomials in $P(e,n_1,\ldots,n_s)$ or $P^*(n_1,\ldots,n_s)$, for a different set of degree bounds, are also involved and similarly we define $\nu_i = n_i+1$, for $1 \leq i \leq s$, and $\nu = \max_{1\leq i\leq s} \nu_i$. Again, for convenience we define $\kappa_i = \mu_i + \nu_i$, $1 \leq i \leq s$, and $\kappa = \max_{1\leq i\leq s} \kappa_i$.

By consulting the indicated references, one finds that the number of cells needed to represent an integer in P(d) is $\lesssim \log d$, that the number of cells to represent a polynomial in $P(d,m_1,\ldots,m_s)$ is $\lesssim (\log d)(\pi_{i=1}^{s}\mu_i)$, and the number of

cells to represent a polynomial in $P^*(m_1,\ldots,m_s)$ is $\lesssim \pi_{i=1}^{s}\mu_i$. To represent an m by n matrix requires m + mn cells for the matrix structure plus the cells to represent the elements of the matrix. Hence, the number of cells to represent a matrix in M(m,n,GF(p)) is ~ m + mn ~ mn and to represent a matrix in M(m,n,P(d)) is $\lesssim$ mn(log d). Also, the number of cells to represent a matrix in $M(m,n,P(d,m_1,\ldots,m_s))$ is $\lesssim$ mn(log d)$(\pi_{i=1}^{s}\mu_i)$ and to represent a matrix in $M(m,n,P^*(m_1,\ldots,m_s))$ is $\lesssim$ mn $(\pi_{i=1}^{s}\mu_i)$.

We adopt the convention that the expression $\pi_{i=1}^{s}t_i$ has the value 1 for s = 0 and that $\sum_{i=1}^{s}t_i$ has the value 0 for s = 0. Hence, for example, we have that the number of cells to represent an element of $P(d,m_1,\ldots,m_s) = P(d,\bar{m}_s)$ is $\lesssim$ (log d)$(\pi_{i=1}^{s}\mu_i)$, for $s \geq 0$. Similarly for an element of $P^*(\bar{m}_s)$, $s \geq 0$.

The SAC-1 System, being a reference count system, requires the erasure of all lists which are to be discarded. The computing time for any erasure is codominant with n+1, where n is the number of cells returned to the available space list by the erasure. Consequently, dominance relations for the times to erase integers, polynomials, and matrices can be deduced from the above dominance relations for the number of cells. The times to erase other kinds of lists (i.e., sequences) can be similarly dominated. It will be

seen in the analyses below that the times for erasures do not dominate the times for the algorithms (except, of course, for erasure algorithms).

Before concluding this section, some remarks concerning the primes p will be made. Several of the algorithms will require a list of distinct odd single-precision primes $(p_1,\ldots,p_t)$. SAC-1 provides such a list, called PRIME, which may be generated by GENPR with a reference such as: PRIME = GENPR(A,k,m), where A is a one-dimensional array of length k and m is an odd integer ≥ 3 (see [COG69a]). Usually, m is chosen such that $m \geq \gamma/2^u$, for u a small positive integer and so each prime p_i of the list PRIME is nearly as large as the maximum single-precision integer. The list PRIME will be communicated to the SAC-1 routines implementing the algorithms by declaring the identifier PRIME as the first element of COMMON block TR4. Recalling that each prime p satisfies $p < \gamma$, it follows that each arithmetic operation in GF(p), including the computation of multiplicative inverses, is bounded. Hence, the computing time for each of these operations is ~ 1.

We note that each algorithm is identified both by a name and by a number. Hence, reference to an algorithm may be made by either of these.

Section 3.2 The Classical Matrix Arithmetic Algorithms

In this section we describe and analyze the classical algorithms for computing the sum, difference, and product of two matrices over $I[x_1,\ldots,x_s]$, for $s \geq 0$. We begin with the algorithm for matrix addition.

Algorithm 3.2.1 (MSUM) Matrix Addition

Input: m by n matrices A and B over $I[x_1,\ldots,x_s]$, $s \geq 0$.

Output: The m by n matrix C = MSUM(A,B) over $I[x_1,\ldots,x_s]$ such that each element satisfies: C(i,j) = A(i,j) + B(i,j).

(1) [Initialize.] X ← A; Y ← B; C ← ().

(2) [Begin next row D.] ADV(E,X); ADV(F,Y); D ← ().

(3) [Compute and prefix next element of row D.] D ← PFL(PSUM(FIRST(E),FIRST(F)),D); E ← TAIL(E); F ← TAIL(F); if E ≠ (), go to (3).

(4) [Prefix next row of sum matrix.] C ← PFL(INV(D),C); if X ≠ (), go to (2).

(5) [Return.] C ← INV(C); return C.

This algorithm obtains the elements of C = A + B row-by-row from top-to-bottom and left-to-right by applying algorithm PSUM to corresponding elements of A and B. The next algorithm (MDIF) for matrix subtraction is almost identical to algorithm MSUM with the exception that PSUM is replaced by PDIF in step (3).

Algorithm 3.2.2 (MDIF) Matrix Subtraction

Input: m by n matrices A and B over $I[x_1,\ldots,x_s]$, $s \geq 0$.

Output: The m by n matrix $C = MDIF(A,B)$ over $I[x_1,\ldots,x_s]$ such that each element satisfies: $C(i,j) = A(i,j) - B(i,j)$

(1) [Initialize.] $X \leftarrow A$; $Y \leftarrow B$; $C \leftarrow ()$.

(2) [Begin next row D.] ADV(E,X); ADV(F,Y); $D \leftarrow ()$.

(3) [Compute and prefix next element of row D.] $D \leftarrow$ PFL(PDIF(FIRST(E),FIRST(F)),D); $E \leftarrow$ TAIL(E); $F \leftarrow$ TAIL(F); if $E \neq ()$, go to (3).

(4) [Prefix next row of difference matrix.] $C \leftarrow$ PFL(INV(D),C); if $X \neq ()$, go to (2).

(5) [Return.] $C \leftarrow$ INV(C); return C.

Theorem 3.2.3. Let $T_{MSUM}(m,n,d,\bar{m}_s,e,\bar{n}_s)$ be the maximum computing time of MSUM for $A \in M(m,n,P(d,\bar{m}_s))$ and $B \in M(m,n,P(e,\bar{n}_s))$, $s \geq 0$. Then $T_{MSUM} \lesssim mn(\log d + \log e)(\pi_{i=1}^{s} \mu_i + \pi_{i=1}^{s} \nu_i)$. Similarly for MDIF.

Proof: Step (1), being executed once, clearly has $T_1 \sim 1$. Each of the m executions of step (2) is ~ 1 and so $T_2 \sim m$. Step (3) is executed mn times, the time for each being dominated by T_{PSUM} and so $T_3 \lesssim mn \cdot T_{PSUM}(d,\bar{m}_s,e,\bar{n}_s) \lesssim mn(\log d + \log e)(\pi_{i=1}^{s} \mu_i + \pi_{i=1}^{s} \nu_i)$. Step (4) is executed m times, the time for each being dominated by $T_{INV}(n) \sim n$ and so $T_4 \lesssim mn$. Finally, $T_5 = T_{INV}(m) \sim m$.

Clearly, T_3 dominates the times for the other steps and so $T_{MSUM} \sim T_3$, giving the theorem.//

Since the algorithm for matrix multiplication will require the computation of the transpose of a matrix, there now follows an algorithm to accomplish this.

Algorithm 3.2.4 (MTRAN) Matrix Transpose

Input: An m by n matrix A over $GF(p)[x_1,\ldots,x_s]$ or $I[x_1,\ldots,x_s]$, $s \geq 0$.

Output: The n by m transpose matrix B such that B(i,j) = A(j,i).

(1) [Initialize.] G ← CINV(A); F ← G; C ← FIRST(F); t ← TYPE(C); n ← LENGTH(C); construct the list B = ((),(),...,()) of length n.

(2) [Begin next column of transpose.] ADV(C,F); S ← B.

(3) [Prefix next element of current column.] ADV(D,C); if t = 0, ALTER(PFA(D,FIRST(S)),S); if t ≠ 0, ALTER(PFL(BORROW(D),FIRST(S)),S); S ← TAIL(S); if S ≠ (), go to (3); if F ≠ (), go to (2).

(4) [Return.] Erase G; return B.

This algorithm in step (1) constructs the matrix G, consisting in the rows of A in reverse order, and the list B of n null vectors. Then in steps (2) and (3) the columns of the transpose of A are added to the matrix B in the

order m, m-1,...,2, 1. Finally, in step (4) B is returned as the transpose.

Theorem 3.2.5. Let $T_{MTRAN}(m,n)$ be the maximum computing time of MTRAN for an m by n matrix A. Then $T_{MTRAN} \sim mn$.

Proof: Step (1) is executed once, the time being $T_1 \sim T_{CINV}(m) + T_{LENGTH}(n) \sim m + n$. Step (2) is executed m times and the time for each is ~ 1; hence, $T_2 \sim m$. Step (3) is executed mn times and the time for each is ~ 1; hence, $T_3 \sim mn$. The time for step (4) is ~ m, the time to return m cells to the available space list. Clearly, T_3 dominates these times and so $T_{MTRAN}(m,n) \sim T_3 \sim mn$.//

We note that this time is, in fact, codominant with the minimum computing time of MTRAN, since the time for every m by n matrix is ~ mn.

The final algorithm of this section is the classical algorithm for computing matrix products.

Algorithm 3.2.6 (MPROD) Classical Matrix Multiplication

Input: An m by n matrix A and an n by q matrix B, both over $I[x_1,...,x_s]$, $s \geq 0$.

Output: The m by q matrix C = MPROD(A,B) over $I[x_1,...,x_s]$, where C = AB.

(1) [Initialize.] X ← CINV(A); Y ←INV(MTRAN(B)); C ← ();
V ← X.

(2) [Begin next row of product C.] ADV(R,V); W ← Y; D ← ().

(3) [Begin next element of current row.] S ← R; ADV(T,W); E ← ().

(4) [Add next term of inner product.] F ← PPROD(FIRST(S), FIRST(T)); G ← PSUM(E,F); erase E and F; E ← G; S ← TAIL(S); T ← TAIL(T); if S ≠ (), go to (4).

(5) [Prefix next element of current row.] D ← PFL(E,D); if W ≠ (), go to (3).

(6) [Prefix next row of product C.] C ← PFL(D,C); if V ≠ (), go to (2).

(7) [Return.] Erase X and Y; return C.

C is formed by taking inner products of the rows of A and the columns of B: $C(i,j) = \sum_{k=1}^{n} A(i,k)B(k,j)$. In step (1) the matrix X, consisting of the rows of A reversed in order, and the matrix Y, consisting of the columns of B in reverse order, are constructed. Using X and Y, in steps (2)-(4) the elements of C = AB are computed from row m to row 1 and within each row from column q to column 1. Steps (5) and (6) are clear and in step (7) the product C is returned, after erasing the auxiliary matrices X and Y.

Theorem 3.2.7. Let $T_{MPROD}(m,n,q,d,\bar{m}_s,e,\bar{n}_s)$ be the maximum computing time of MPROD for $A \in M(m,n,P(d,\bar{m}_s))$ and $B \in M(n,q,P(e,\bar{n}_s))$, $s \geq 0$. Then $T_{MPROD} \lesssim mnq(\log d)(\log e)(\pi_{i=1}^{s}\mu_i)(\pi_{i=1}^{s}\nu_i) + mnq(\log n)(\pi_{i=1}^{s}\kappa_i)$.

Proof: The following is a maximum computing time table for MPROD.

i	N_i	$T_{i,j}$	T_i
1	1	-	$m + nq$
2	m	1	m
3	mq	1	mq
4	mnq	(see below)	$mnq\ (\log d)(\log e)(\pi_{i=1}^{s}\mu_i)\cdot(\pi_{i=1}^{s}\nu_i) + mnq\ (\log n)\cdot(\pi_{i=1}^{s}\kappa_i)$
5	mq	1	mq
6	m	1	m
7	1	-	$m + nq$

Selected explanations of these computing times follow. For step (1), $T_1 \sim T_{CINV}(m) + T_{INV}(q) + T_{MTRAN}(m,n) \sim m + q + nq \sim m + nq$. Steps (2) and (3) are clear. In step (4) each of the elements $C(i,j)$ is computed as a sum $\sum_{k=1}^{n} F_k$, where $F_k = A(i,k)\cdot B(k,j)$. This requires computing the n products F_k and the $n - 1$ sums $G_k = F_k + G_{k-1}$, for $k = 2,\ldots,n$, where $G_t = \sum_{u=1}^{t} F_u$. Since $A(i,k) \in P(d,\bar{m}_s)$ and $B(k,j) \in P(e,\bar{n}_s)$, the maximum time to compute each F_k is $T_{PPROD}(d,\bar{m}_s,e,\bar{n}_s) \lesssim (\log d)(\log e)\cdot(\pi_{i=1}^{s}\mu_i)(\pi_{i=1}^{s}\nu_i)$. Also, since $F_k \in P(de,\bar{k}_s)$ and $G_{k-1} \in P(kde,\bar{k}_s)$, where $\bar{k}_s = (m_1+n_1,\ldots,m_s+n_s)$, the maximum time to compute each G_k is $\lesssim T_{PSUM}(de,\bar{k}_s,kde,\bar{k}_s) \lesssim (\log kde)\cdot(\pi_{i=1}^{s}\kappa_i) \lesssim (\log nde)(\pi_{i=1}^{s}\kappa_i)$. Note that $\log de = \log d + \log e \lesssim (\log d)(\log e)$ and that $\pi_{i=1}^{s}\kappa_i \lesssim (\pi_{i=1}^{s}\mu_i)(\pi_{i=1}^{s}\nu_i)$.

Summing over k, we find that the maximum time to compute $C(i,j)$ is $\lesssim \sum_{k=1}^{n}[(\log d)(\log e)(\pi_{i=1}^{s}\mu_i)(\pi_{i=1}^{s}\nu_i) + (\log n + \log de)\cdot(\pi_{i=1}^{s}\kappa_i)] \lesssim n(\log d)(\log e)(\pi_{i=1}^{s}\mu_i)(\pi_{i=1}^{s}\nu_i) + n(\log n)(\pi_{i=1}^{s}\kappa_i) + n(\log d)(\log e)(\pi_{i=1}^{s}\mu_i)(\pi_{i=1}^{s}\nu_i) \sim n(\log d)(\log e)(\pi_{i=1}^{s}\mu_i)(\pi_{i=1}^{s}\nu_i) + n(\log n)(\pi_{i=1}^{s}\kappa_i)$. Since there are mq such elements $C(i,j)$ computed, we have the dominance relation for T_4. Steps (5) and (6) are clear. Finally, in step (7) m cells are returned by erasing X and nq cells are returned by erasing Y and so $T_7 \sim m + nq$. Clearly, T_4' dominates and so $T_{MPROD} \lesssim T_4'$, giving the theorem.//

3.3 Algorithms for Applying Mod-p and Evaluation Mappings

The algorithms of this section fall into two categories: those for applying mod-p mappings and Garner's method to matrices and those for applying evaluation mappings and incremental interpolation. Each category will require some auxiliary algorithms. We begin with two auxiliary algorithms related to the former. The first of these obtains a list of variables for a matrix, which will be required by the algorithm for applying Garner's method.

Algorithm 3.3.1 (MVLIST) Matrix Variable List

Input: An m by n nonzero matrix A over $I[x_1,\ldots,x_s]$, $s \geq 0$.

Output: The list L = MVLIST(A), where L is the null list (), if A is over I, or the list of variables $(x_1,\ldots x_s)$, if A is over $I[x_1,\ldots,x_s]$, $s > 0$.

(1) [Initialize.] X ← A.

(2) [Get next row.] ADV(B,X).

(3) [Get next row element.] ADV(C,B); if C ≠ (), go to (4); if B ≠ (), go to (3); go to (2).

(4) [Compute and return variable list.] L ← PVLIST(C); return L.

This algorithm simply searches the matrix A row-by-row from top-to-bottom and left-to-right for a nonzero element and, when found, obtains the variable list L from this element.

Theorem 3.3.2. Let $T_{MVLIST}(m,n)$ be the maximum computing time of MVLIST for $A \in M(m,n, I[x_1,\ldots,x_s])$, $s \geq 0$. Then $T_{MVLIST} \sim mn$.

Proof: The time to apply PVLIST to a nonzero element of A is bounded, since s is not considered variable, and so the time for step (4), which is executed once, is ~ 1. The maximum computing time for step (3) is dominated by the zero tests, of which there may be mn. Hence, $T_3 \sim mn$, which dominates the times for steps (1) and (2). Thus, $T_{MVLIST} \sim T_3 \sim mn$.//

We next present an algorithm for the erasure of matrices of integers or of polynomials over the integers.

Algorithm 3.3.3 (MERASE) Matrix Erasure

Input: An m by n matrix A over $I[x_1,\ldots,x_s]$, $s \geq 0$.

Output: A is erased by executing MERASE(A). The variable A is redefined to consist in its last k rows, where A_k is the first row which overlaps another list. In general, however, all rows are erased and A = () is returned.

(1) [Begin erasure of next row.] If A = (), return; k ← COUNT(A)-1; if k > 0, (SCOUNT(k,A); return); DECAP(B,A).

(2) [Erase next row element.] k ← COUNT(B)-1; if k > 0, (SCOUNT(k,B); go to (1)); DECAP(C,B); PERASE(C); if B ≠ (), go to (2); go to (1).

Theorem 3.3.4. Let $T_{MERASE}(m,n,d,\bar{m}_s)$ denote the maximum computing time of MERASE for $A \in M(m,n,P(d,\bar{m}_s))$. Then $T_{MERASE} \lesssim mn(\log d)\cdot(\pi_{i=1}^{s}\mu_i)$.

Proof: The number of cells for the list structure of A is $\lesssim mn(\log d)(\pi_{i=1}^{s}\mu_i)$, as mentioned in 3.1. The applications of DECAP in steps (1) and (2) return the cells for the matrix structure proper and the applications of PERASE in step (2) return the cells for the matrix elements. Since the time to return a cell to available space is bounded, the maximum time to erase A is $\lesssim mn(\log d)(\pi_{i=1}^{s}\mu_i)$.//

Next, an algorithm for applying a mod-p mapping to a matrix is presented.

Algorithm 3.3.5 (MMOD) Applying a Mod-p Mapping

Input: An m by n matrix A over $I[x_1,\ldots,x_s]$, $s \geq 0$, a prime number p, and the non-negative integer s.

Output: The m by n matrix B = MMOD(p,A,s) over $GF(p)[x_1,\ldots,x_s]$ such that the corresponding elements of A and B satisfy: $B(i,j) = \varphi_p(A(i,j))$.

(1) [Initialize.] X ← A; B ← ().

(2) [Begin next row.] ADV(T,X); C ← ().

(3) [Apply φ_p to next row element.] ADV(R,T); R ← CPMOD(p,R); if s = 0, C ← PFA(R,C); if s ≠ 0, C ← PFL(R,C); if T ≠ (), go to (3).

(4) [Prefix next row.] B ← PFL(INV(C),B); if X ≠ (), go to (2).

(5) [Return.] B ← INV(B); return B.

A is traversed row-by-row from top-to-bottom and left-to-right. The elements of B are generated in this order.

Theorem 3.3.6. Let $T_{MMOD}(m,n,d,\bar{m}_s)$ be the maximum computing time of MMOD for $A \in M(m,n,P(d,\bar{m}_s))$, $s \geq 0$. Then $T_{MMOD} \prec mn(\log d)(\pi_{i=1}^{s}\mu_i)$.

Proof: The maximum computing time for this algorithm is dominated by the time for the applications of CPMOD in step (3). Since step (3) is executed mn times, we have $T_{MMOD} \sim mn \cdot T_{CPMOD}(d,\bar{m}_s) \sim mn \cdot (\log d)(\pi_{i=1}^{s}\mu_i)$.//

As a companion for algorithm MMOD an algorithm for applying Garner's method to matrices now follows.

Algorithm 3.3.7 (MGARN) Garner's Method for Matrices

Input: An odd prime number p and an odd positive infinite-precision integer Q relatively prime to p. An m by n matrix B over $I[x_1,\ldots,x_s]$ and an m by n matrix A over $GF(p)[x_1,\ldots,x_s]$, for $s \geq 0$. Also, the list L, which is the null list (), if s = 0, or is the list $(x_1,\ldots,x_s)$ of variables, if $s \geq 1$. The integer coefficients of B are less than Q/2 in magnitude.

Output: The m by n matrix C = MGARN(Q,B,p,A,L) over $I[x_1,\ldots,x_s]$. Each element C(i,j) of C is that unique polynomial or integer such that C(i,j) ≡

$B(i,j) \pmod Q$, $C(i,j) \equiv A(i,j) \pmod p$ (i.e., $A(i,j) = \varphi_p(C(i,j))$), and the integer coefficients of $C(i,j)$ are each less than $Qp/2$ in magnitude.

(1) [Initialize.] $X \leftarrow B$; $Y \leftarrow A$; $C \leftarrow ()$.

(2) [Begin next row of C.] ADV(S,X); ADV(T,Y); $D \leftarrow ()$.

(3) [Compute next row element by Garner's Method.] ADV(U,S); ADV(V,T); $E \leftarrow$ CPGARN(Q,U,p,V,L); $D \leftarrow$ PFL(E,D); if $S \neq ()$, go to (3).

(4) [Prefix next row.] $C \leftarrow$ PFL(INV(D),C); if $X \neq ()$, go to (2).

(5) [Return.] $C \leftarrow$ INV(C); return C.

Elements of C are computed row-by-row from top-to-bottom and left-to-right as the matrices B and A are traversed.

Theorem 3.3.8. Let $T_{MGARN}(m,n,Q,\bar{m}_s)$ be the maximum computing time of MGARN for $A \in M(m,n,P^*(\bar{m}_s))$, $s \geq 0$, and B an m by n matrix over $I[x_1,\ldots,x_s]$, each of whose elements have degree in x_i at most m_i and whose integer coefficients are less than $Q/2$ in magnitude. Then $T_{MGARN} \sim mn(\log Q)(\pi_{i=1}^{s}\mu_i)$.

Proof: The maximum computing time for MGARN is dominated by the time for the mn executions of step (3). The maximum time for the mn executions of CPGARN is $\sim mn \cdot T_{CPGARN} \sim mn(\log Q)(\pi_{i=1}^{s}\mu_i)$. This time dominates the time for the applications of INV in step (4) and for the remaining steps, giving the theorem.//

We now present the algorithms of the second category: those for applying evaluation mappings and incremental interpolation. Analogous to the preceding set, two related auxiliary algorithms are presented. The first computes the number of variables for the elements of a matrix.

Algorithm 3.3.9 (MCPNV) Congruence Matrix Number of Variables

Input: An m by n nonzero matrix A over $GF(p)[x_1,\ldots,x_s]$, $s \geq 0$.

Output: The integer s = MCPNV(A).

(1) [Initialize.] B ← A; if TYPE(FIRST(B))=0, (s←0;return s).

(2) [Begin next row search.] ADV(C,B).

(3) [Obtain and test next row element.] ADV(D,C); if $D \neq 0$, go to (4); if $C \neq ()$, go to (3); go to (2).

(4) [Compute and return variable count.] s ← CPNV(D); return s.

The analysis of this algorithm is similar to that for algorithm MVLIST and may be easily translated. We note that algorithm CPNV is described in [COG71b] and has a maximum computing time which is ~ 1.

Theorem 3.3.10. Let $T_{MCPNV}(m,n)$ be the maximum computing time of MCPNV for $A \in M(m,n,GF(p)[x_1,\ldots,x_s])$, $s \geq 0$. Then $T_{MCPNV} \sim mn$, if $s > 0$, and $T_{MCPNV} \sim 1$, if $s = 0$.

The second auxiliary algorithm erases a matrix over $GF(p)[x_1,\ldots,x_s]$, $s \geq 0$.

Algorithm 3.3.11 (MCPERS) Congruence Matrix Erasure

Input: An m by n matrix A over $GF(p)[x_1,\ldots x_s]$, $s \geq 0$.

Output: A is erased by executing MCPERS(A). (See output description of MERASE concerning the new value of A.)

(1) [Initialize.] t ← TYPE(FIRST(A)).

(2) [Begin erasure of next row.] If A = (), return; k ← COUNT(A)-1; if k > 0, (SCOUNT(k,A); return); DECAP(B,A).

(3) [Erase next row element.] k ← COUNT(B)-1; if k > 0, (SCOUNT(k,B); go to (2)); DECAP(C,B); if t = 1, CPERAS(C); if B ≠ (), go to (3); go to (2).

The analysis of this algorithm is very similar to that for algorithm MERASE and is omitted. The result is stated as follows.

Theorem 3.3.12. Let $T_{MCPERS}(m,n,\bar{m}_s)$ be the maximum computing time of MCPERS for $A \in M(m,n,P^*(\bar{m}_s))$, $s \geq 0$. Then $T_{MCPERS} \leq mn\pi_{i=1}^{s}\mu_i$.

The next algorithm effects the application of an evaluation mapping to a matrix.

Algorithm 3.3.13 (MCPEVL) Applying an Evaluation Mapping

Input: A prime number p, an element b of GF(p), an m by n matrix A over $GF(p)[x_1,\ldots,x_s]$, $s \geq 1$, and the integer s.

Output: The m by n matrix C = MCPEVL(p,A,b,s) over $GF(p)[x_1,\ldots,x_{s-1}]$, such that each element of C

satisfies: $C(i,j) = A(i,j)(x_1,\ldots,x_{s-1},b) = \Psi_b(A(i,j))$.

(1) [Initialize.] X ← A; C ← ().

(2) [Begin next row.] ADV(W,X); D ← ().

(3) [Compute and prefix next row element.] ADV(U,W); if U ≠ (), U ← CPEVAL(p,U,b); if s = 1, D ← PFA(U,D); if s ≠ 1, D ← PFL(U,D); if W ≠ (), go to (3).

(4) [Prefix next row.] C ← PFL(INV(D),C); if X ≠ (), go to (2).

(5) [Return.] C ← INV(C); return C.

Theorem 3.3.14. Let $T_{MCPEVL}(m,n,\bar{m}_s)$ be the maximum computing time of MCPEVL for $A \in M(m,n,\mathcal{P}^*(\bar{m}_s))$, $s \geq 1$. Then $T_{MCPEVL} \lesssim mn\pi_{i=1}^{s}\mu_i$.

Proof: The computing time of MCPEVL is dominated by the mn applications of CPEVAL in step (3), which is $\lesssim mn \cdot T_{CPEVAL} \lesssim mn\pi_{i=1}^{s}\mu_i$.//

We close this section with an algorithm for applying incremental interpolation to the elements of a matrix.

Algorithm 3.3.15 (MCPINT) Interpolating the Elements of a Matrix

Input: A prime number p, an element b of GF(p), an m by n matrix A over $GF(p)[x_1,\ldots,x_s]$, $s \geq 1$, the integer s, an m by n matrix C over $GF(p)[x_1,\ldots x_{s-1}]$, and a univariate polynomial $D(x) = \pi_{i=0}^{k}(x-b_i)$ over

GF(p), where $b \neq b_t$, $0 \leq t \leq k$, and the b_t are distinct. The degree of each element of A in x_s is at most k.

Output: The unique m by n matrix H = MCPINT(p,A,b,C,D,s) over $GF(p)[x_1,\ldots,x_s]$, where each element H(i,j) is the unique interpolating polynomial of degree k+1 or less in x_s such that $H(i,j)(x_1,\ldots,x_{s-1},b_t) = A(i,j)(x_1,\ldots,x_{s-1},b_t)$, $0 \leq t \leq k$, and also $H(i,j)(x_1,\ldots,x_{s-1},b) = C(i,j)(x_1,\ldots,x_{s-1})$.

Note that the special case k = -1 is permitted in MCPINT, in which case D(x) = 1 and A is the m by n zero matrix.

(1) [Initialize.] X ← A; Y ← C; H ← ().

(2) [Begin next row.] ADV(W,X); ADV(T,Y); E ← ().

(3) [Interpolate to get next row element.] ADV(U,W); ADV(V,T); F ← CPINT(p,U,b,V,D,s); E ← PFL(F,E); if W ≠ (), go to (3).

(4) [Prefix next row.] H ← PFL(INV(E),H); if X ≠ (), go to (2).

(5) [Return.] H ← INV(H); return H.

Theorem 3.3.16. Let $T_{MCPINT}(m,n,\bar{m}_s)$ be the maximum computing time of MCPINT for $A \in M(m,n,P^*(\bar{m}_s))$, where $\bar{m}_s = (m_1,\ldots,m_{s-1},k)$, and $C \in M(m,n,P^*(\bar{m}_{s-1}))$. Then $T_{MCPINT} \lesssim mn((k+2)(\pi_{i=1}^{s-1}\mu_i))$.

Proof: The time for each execution of step (3) is dominated by the time for CPINT, which is $\lesssim (k+2)\cdot\pi_{i=1}^{s-1}\mu_i$. Since step (3) is executed mn times, we have $T_3 \lesssim mn(k+2)\pi_{i=1}^{s}\mu_i$, which dominates the times for the remaining steps.//

3.4 A Modular Algorithm for Matrix Multiplication

In this section a new algorithm for multiplying two matrices with integer or polynomial entries is presented. This algorithm actually consists of three basic algorithms: the outer, middle, and inner algorithms. The outer or main algorithm applies mod-p mappings and Garner's method to compute the product of two matrices over $I[x_1,\ldots,x_s]$, $s \geq 0$, requiring a special algorithm to compute a bound on the integer coefficients of the elements of the product. The outer algorithm employs the middle algorithm to multiply two matrices over $GF(p)[x_1,\ldots,x_s]$, $s \geq 0$. Evaluation mappings and incremental interpolation are applied by the middle algorithm and a special algorithm is used to compute a bound on the degrees in the main variable of the elements of the product. This middle algorithm is recursive, applying itself to the images (under evaluation mappings) of the input matrices. In case these images are matrices over $GF(p)$, the inner algorithm is applied to compute the product by the classical algorithm. After each algorithm a computing time analysis is done and so we present the inner algorithm first, and then the middle and outer algorithms in that order.

We begin with an auxiliary algorithm which tests for a zero matrix.

Algorithm 3.4.1 (MZERO) Zero Matrix Test

Input: An m by n matrix A over $I[x_1,\ldots,x_s]$ or $GF(p)[x_1,\ldots,x_s]$, $s \geq 0$.

Output: The integer k = MZERO(A), where k = 0, if A is nonzero, and k = 1, if A is zero.

(1) [Initialize.] X ← A; k ← 0.

(2) [Obtain next row.] ADV(Y,X).

(3) [Test next row element.] If FIRST(Y) ≠ 0, return k; Y ← TAIL(Y); if Y ≠ (), go to (3); if X ≠ (), go to (2); k ← 1; return k.

This algorithm scans the matrix A row-by-row from top-to-bottom and left-to-right, testing for a nonzero element. If none is found, k is set to 1 and returned.

Theorem 3.4.2. Let $T_{MZERO}(m,n)$ be the maximum computing time of MZERO for A an m by n matrix. Then $T_{MZERO} \sim mn$.

Proof: The highest number of elements tested nonzero in step (3) is mn, which occurs, for example, for A a zero matrix. Thus, $T_3 \sim mn$ and this time dominates the time for the other steps.//

We now give the inner algorithm for matrix multiplication over GF(p).

Algorithm 3.4.3 (MCMPY) Classical Matrix Multiplication Over GF(p)

Input: A prime number p and matrices A and B over GF(p),

where A is m by n and represented rowwise and B is n by q and represented columnwise.

Output: The m by q matrix C = MCMPY(p,A,B) over GF(p) such that C = AB.

(1) [Initialize.] X ← CINV(A); Y ← CINV(B); C ← (); V ← X.

(2) [Begin next row of product.] ADV(R,V); D ← (); W ← Y.

(3) [Begin to compute next element as an inner product.] S ← R; ADV(T,W); e ← 0.

(4) [Add next term of inner product.] e ← CSUM(p,e,CPROD(p, FIRST(S),FIRST(T))); S ← TAIL(S); T ← TAIL(T); if S ≠ (), go to (4); D ← PFA(e,D); if W ≠ (), go to (3); C ← PFL(D,C); if V ≠ (), go to (2).

(5) [Return.] Erase X,Y; return C.

A is represented as the list of rows $(A_1,\ldots,A_m)$ and B as the list of columns $(B_1,\ldots,B_q)$. The element C(i,j) of the product is computed as the inner product of A_i and B_j in steps (3) and (4): $C(i,j) = \sum_{k=1}^{n} A(i,k)\ B(k,j)$.

Theorem 3.4.4. Let $T_{MCMPY}(m,n,q)$ be the maximum computing time for MCMPY for $A \in M(m,n,GF(p))$ and $B \in M(n,q,GF(p))$. Then $T_{MCMPY} \sim mnq$, which is codominant with the minimum computing time also.

Proof: It can be easily shown that $T_1 \sim m+q$, $T_2 \sim m$, and $T_5 \sim m+q$. Step (3) is executed mq times, where each execution is ~ 1, and so $T_3 \sim mq$. Step (4) is executed mnq

times, where each execution time is ~ 1, since each algorithm evoked has a bounded computing time. Thus, T_4 ~ mnq. Since T_4 dominates the times for the other steps, T_{MCMPY} ~ T_4 ~ mnq. We note that, since the time for applying MCMPY to any such matrices A and B is ~ mnq, this is also the minimum computing time for MCMPY.//

There next follows an algorithm for computing a degree bound for the product of two matrices whose entries are polynomials over GF(p).

Algorithm 3.4.5 (MCPMDB) Congruence Matrix Multiplication Degree Bound

Input: Matrices A and B over GF(p)[$x_1,\ldots,x_s$], $s \geq 1$, where A is m by n and represented rowwise and B is n by q and represented columnwise.

Output: A non-negative integer z = MCPMDB(A,B), where z is a bound for the degrees in the main variable x_s of the elements of the product AB.

(1) [Initialize.] X ← A; Y ← B; z ← 0.

(2) [Begin degree bound for next row.] ADV(R,X); W ← Y.

(3) [Get degree of next row element.] S ← R; ADV(T,W).

(4) [Compare degree of next term of inner product.] ADV(U,S); ADV(V,T); if U = 0 or V = 0, go to (5); e ← deg(U) + deg(V); if e > z, z ← e.

(5) [Termination tests and return.] If $S \neq ()$, go to (4); if $W \neq ()$, go to (3); if $X \neq ()$, go to (2); return z.

This algorithm is very similar to MCMPY in the way the matrices A and B are scanned. However; in place of adding the product of A(i,k) and B(k,j) to the partial inner product sum in MCMPY, we have in MCPMDB that the sum of the degrees (in x_s) of A(i,k) and B(k,j) is compared with the previous maximum degree sum. This occurs in step (4). The degree bound computed is $z = \max_{1\le i\le m} \max_{1\le j\le q} \max_{1\le k\le n} \{\deg(A(i,k)) + \deg(B(k,j))\}$.

Theorem 3.4.6. Let $T_{MCPMDB}(m,n,q)$ be the maximum computing time of MCPMDB for $A \in M(m,n,GF(p)[x_1,\ldots,x_s])$ and $B \in M(n,q,GF(p)[x_1,\ldots x_s])$, $s \geq 1$. Then $T_{MCPMDB} \sim mnq$, which is codominant with the minimum computing time also.

Proof: As for MCMPY, the maximum computing time for this algorithm is dominated by the mnq executions of step (4), each of which has a bounded computing time. Hence, $T_{MCPMDB} \sim T_4 \sim mnq$. This is true of every pair A,B of input matrices and so the minimum computing time for this algorithm is $\sim mnq$.//

Algorithm MCPMDB is applied in the following algorithm for computing the product of matrices of polynomials over GF(p), using evaluation mappings and incremental interpolation.

Algorithm 3.4.7 (MCPMPY) Congruence Matrix Multiplication Using Evaluation Mappings

Input: A prime number p and matrices A and B over $GF(p)[x_1, \ldots, x_s]$, $s \geq 0$, where A is m by n and represented rowwise and B is n by q and represented columnwise.

Output: The m by q matrix C = MCPMPY(p,A,B) over $GF(p)[x_1, \ldots, x_s]$ such that C = AB.

(1) [Initialize.] m ← LENGTH(A); q ← LENGTH(B).

(2) [Compute product over GF(p).] If TYPE(FIRST(A)) ≠ 0, go to (3); C ← MCMPY(p,A,B); return C.

(3) [Construct the m by q zero matrix C .] C ← (); for i = 1,m do: (D ← (); for j = 1, q do: (D ← PFL(0,D)); C ← PFL(D,C)); if MZERO(A) = 1 or MZERO(B) = 1, return C .

(4) [Finish initializing the interpolation.] s ← MCPNV(A); k ← MCPMDB(A,B); i ← 0; D ← PFA(0,PFA(1,0)).

(5) [Apply another evaluation mapping, if one exists.] If i = p, print error message and stop; A* ← MCPEVL(p,A, i,s); B* ← MCPEVL(p,B,i,s).

(6) [Apply algorithm recursively.] C* ← MCPMPY(p,A*,B*); erase A* and B*.

(7) [Apply incremental interpolation.]C' ← MCPINT(p,C ,i, C*,D,s); erase C , C*; C ← C'; if i < k, go to (8); erase D; return C .

(8) [Recompute polynomial D for next interpolation.] E ← PFA(1,PFA(1,PFA(CDIF(p,0,i),0))); W ← CPPROD(p,D,E);

erase D and E; D ← W; i ← i+1; go to (5).

If A and B are over GF(p), the product C is obtained in step (2) by applying MCMPY. Otherwise, the initialization is completed in steps (3) and (4), including computing the variable count s and degree bound k in x_s. Steps (5)-(8) are executed in order k+1 times. The algorithm applies itself recursively to the images A* and B* of A and B under evaluation mappings Ψ_i to obtain the products $C^* = A^*B^* = \Psi_i(AB) = \Psi_i(C)$, $0 \le i \le k$, which form a value representation of C. The k+1 applications of incremental interpolation in step (7) thus assures that the product AB is constructed. Note that it may be safely assumed that each prime p satisfies $p >> k$ and so in general the algorithm will not fail.

Theorem 3.4.8. Let $T_{MCPMPY}(m,n,q,\bar{m}_s,\bar{n}_s)$ be the maximum computing time of MCPMPY for $A \in M(m,n,P^*(\bar{m}_s))$ and $B \in M(n,q,P^*(\bar{n}_s))$, $s \ge 0$. Then, for $s = 0$, $T_{MCPMPY} \sim mnq$ and, for $s \ge 1$, $T_{MCPMPY} \lesssim mnq(\pi_{i=1}^{s}\kappa_i) + mn\{\sum_{i=1}^{s}(\pi_{j=1}^{i}\mu_j)(\pi_{j=i}^{s}\kappa_j)\} + nq\{\sum_{i=1}^{s}(\pi_{j=1}^{i}\nu_j)(\pi_{j=i}^{s}\kappa_j)\} + mq\{(\pi_{i=1}^{s}\kappa_i)(\sum_{i=1}^{s}\kappa_i)\}$.

Proof: For A and B over GF(p), only steps (1) and (2) are executed and so $T_{MCPMPY}(m,n,q,\bar{m}_o,\bar{n}_o) \sim T_{MCMPY}(m,n,q) \sim mnq$. Supposing $s \ge 1$, the following is a maximum computing time table for MCPMPY.

i	N_i	$T_{i,j}$	T_i
1	1	-	$m+q$
2	1	-	1
3	1	-	$mq + mn + nq$
4	1	-	mnq
5	κ_s	$mn(\pi_{i=1}^{s}\mu_i)+nq(\pi_{i=1}^{s}\nu_i)$	$mn\kappa_s(\pi_{i=1}^{s}\mu_i)+nq\kappa_s(\pi_{i=1}^{s}\nu_i)$
6	κ_s	$T_{MCPMPY}(m,n,q,\bar{m}_{s-1},\bar{n}_{s-1})$ $+ mn(\pi_{i=1}^{s-1}\mu_i)+nq(\pi_{i=1}^{s-1}\nu_i)$	$\kappa_s \cdot T_{MCPMPY}(m,n,q,\bar{m}_{s-1},\bar{n}_{s-1})$ $+ mn\kappa_s(\pi_{i=1}^{s-1}\mu_i)+nq\kappa_s($ $\pi_{i=1}^{s-1}\nu_i)$
7	κ_s	$mqj(\pi_{i=1}^{s-1}\kappa_i)$	$mq\kappa_s(\pi_{i=1}^{s}\kappa_i)$
8	κ_s	j	κ_s^2

Selected explanations of these computing times follow. Steps (1) and (2) are clear. In step (3) the time to construct C is $\sim mq$ and the maximum computing times to test A and B zero are $T_{MZERO}(m,n) \sim mn$ and $T_{MZERO}(n,q) \sim nq$, respectively. For step (4), $T_4 \sim T_{MCPMDB}(m,n,q) \sim mnq$. Clearly, the degree bound $k \leq m_s+n_s < \mu_s+\nu_s = \kappa_s$ and so the maximum number of times steps (5)-(8) are executed is $\lesssim \kappa_s$. In step (5) the maximum time to compute A* is $T_{MCPEVL}(m,n,\bar{m}_s) \lesssim mn(\pi_{i=1}^{s}\mu_i)$ and that to compute B* is $T_{MCPEVL}(n,q,\bar{n}_s) \lesssim nq(\pi_{i=1}^{s}\nu_i)$. We note that $A^* \in M(n,n,P^*(\bar{m}_{s-1}))$ and $B^* \in M(n,q,P^*(\bar{n}_{s-1}))$. Hence, in step (6) the time to erase A* is $\lesssim mn(\pi_{i=1}^{s-1}\mu_i)$ and the time to erase B* is $\lesssim nq(\pi_{i=1}^{s-1}\nu_i)$. This also explains the arguments of T_{MCPMPY}

in step (6). Before the j^{th} execution of step (7), $C' \in M(m,q,P^*(m_1+n_1,\ldots,m_{s-1}+n_{s-1},j-1))$ and $C^* \in M(m,q,P^*(m_1+n_1,\ldots,m_{s-1}+n_{s-1}))$. So, for $j = 1,2,\ldots,k+1$, $T_{7,j} \lesssim T_{MCPINT} \lesssim mqj(\pi_{i=1}^{s-1}\kappa_i)$. Since $T_{7,j} \lesssim mq\kappa_s(\pi_{i=1}^{s-1}\kappa_i)$, $T_7 \lesssim \sum_{j=1}^{k+1} T_{7,j} \lesssim (k+1)mq(\pi_{i=1}^{s}\kappa_i) \lesssim \kappa_s mq(\pi_{i=1}^{s}\kappa_i)$. Finally, $T_{8,j} \lesssim j$, since the j^{th} application of CPPROD is to $D \in P^*(j-1)$ and $E \in P^*(1)$, and so $T_8 \lesssim \sum_{j=1}^{k+1} j \lesssim (k+1)^2 \lesssim \kappa_s^2$.

We now obtain T_{MCPMPY}. Clearly, $T_1+T_2+T_3 \lesssim T_4' = mnq$ and $T_8 \lesssim T_7' = mq\kappa_s(\pi_{i=1}^{s}\kappa_i)$. So, $T_{MCPMPY}(m,n,q,\bar{m}_s,\bar{n}_s) \lesssim T_4' + T_5' + T_6' + T_7' \sim mnq + mn\kappa_s(\pi_{i=1}^{s}\mu_i) + nq\kappa_s(\pi_{i=1}^{s}\nu_i) + \kappa_s \cdot T_{MCPMPY}(m,n,q,\bar{m}_{s-1},\bar{n}_{s-1}) + mq\kappa_s(\pi_{i=1}^{s}\kappa_i)$. For $s-1 = 1$, $T_{MCPMPY}(m,n,q,\bar{m}_1,\bar{n}_1) \lesssim mnq + mn\kappa_1\mu_1 + nq\kappa_1\nu_1 + \kappa_1 \cdot T_{MCPMPY}(m,n,q,\bar{m}_o,\bar{n}_o) + mq\kappa_1 \cdot \kappa_1 \sim mnq + mn\kappa_1\mu_1 + nq\kappa_1\nu_1 + \kappa_1 \cdot mnq + mq\kappa_1^2 \sim \kappa_1 mnq + mn\kappa_1\mu_1 + nq\kappa_1\nu_1 + mq\kappa_1^2$.

Suppose inductively, for $s \geq 2$, that $T_{MCPMPY}(m,n,q,\bar{m}_{s-1},\bar{n}_{s-1}) \lesssim mnq(\pi_{i=1}^{s-1}\kappa_i) + mn\{\sum_{i=1}^{s-1}(\pi_{j=1}^{i}\mu_j)(\pi_{j=i}^{s-1}\kappa_j)\} + nq\{\sum_{i=1}^{s-1}(\pi_{j=1}^{i}\nu_j)(\pi_{j=i}^{s-1}\kappa_j)\} + mq\{(\pi_{i=1}^{s-1}\kappa_i)(\sum_{i=1}^{s-1}\kappa_i)\}$. One can easily see that this hypothesis is satisfied for $s-1 = 1$. We show that it holds for s. For $T_{MCPMPY}(m,n,q,\bar{m}_s,\bar{n}_s) \lesssim mnq + mn\kappa_s(\pi_{j=1}^{s}\mu_j) + nq\kappa_s(\pi_{j=1}^{s}\nu_j) + \kappa_s \cdot [\, mnq(\pi_{i=1}^{s-1}\kappa_i) + mn\{\sum_{i=1}^{s-1}(\pi_{j=1}^{i}\mu_j)(\pi_{j=i}^{s-1}\kappa_j)\} + nq\{\sum_{i=1}^{s-1}(\pi_{j=1}^{i}\nu_j)(\pi_{j=i}^{s-1}\kappa_i)\} + mq\{(\pi_{i=1}^{s-1}\kappa_i)(\sum_{i=1}^{s-1}\kappa_i)\}\,] + mq\kappa_s(\pi_{i=1}^{s}\kappa_i) \sim mnq(\pi_{i=1}^{s}\kappa_i) + mn\{\sum_{i=1}^{s-1}(\pi_{j=1}^{i}\mu_j)(\pi_{j=i}^{s}\kappa_j) + (\pi_{j=1}^{s}\mu_j)\cdot\kappa_s\} + nq\{\sum_{i=1}^{s-1}(\pi_{j=1}^{i}\nu_j)(\pi_{j=i}^{s}\kappa_j) + (\pi_{j=1}^{s}\nu_j)\kappa_s\} + mq\{(\pi_{i=1}^{s}\kappa_i)(\sum_{i=1}^{s-1}\kappa_i) + (\pi_{i=1}^{s}\kappa_i)\kappa_s\} =$

$mnq(\pi_{i=1}^{s}\kappa_i) + mn\{\sum_{i=1}^{s}(\pi_{j=1}^{i}\mu_j)(\pi_{j=i}^{s}\kappa_j)\} + nq\{\sum_{i=1}^{s}(\pi_{j=1}^{i}\nu_j)(\pi_{j=i}^{s}\kappa_j)\} + mq\{(\pi_{i=1}^{s}\kappa_i)(\sum_{i=1}^{s}\kappa_i)\}$. This completes the inductive step.//

We observe that $\sum_{i=1}^{s}(\pi_{j=1}^{i}\mu_j)(\pi_{j=1}^{s}\kappa_j) \lesssim \sum_{i=1}^{s}\mu_i(\pi_{j=1}^{s}\kappa_j)$ and that $\sum_{i=1}^{s}(\pi_{j=1}^{i}\nu_j)(\pi_{j=i}^{s}\kappa_j) \lesssim \sum_{i=1}^{s}\nu_i(\pi_{j=1}^{s}\kappa_i)$. The following corollary, giving a weaker result, follows directly.

Corollary 3.4.9. $T_{MCPMPY}(m,n,q,\bar{m}_s,\bar{n}_s) \lesssim (\pi_{i=1}^{s}\kappa_i)(mnq + mn(\sum_{i=1}^{s}\mu_i) + nq(\sum_{i=1}^{s}\nu_i) + mq(\sum_{i=1}^{s}\kappa_i))$.

Applying the definitions of μ, ν, and κ from 3.1, we obtain as an immediate consequence the following compact but crude dominance relation.

Corollary 3.4.10. $T_{MCPMPY}(m,n,q,\bar{m}_s,\bar{n}_s) \lesssim \kappa_5(mnq + mn\mu + nq\nu + mq\kappa)$.

The main algorithm will require the following algorithm, which computes a bound on the norms of the elements of the product of two matrices with integer or polynomial coefficients.

Algorithm 3.4.11 (MMCB) Matrix Multiplication Coefficient Bound

Input: Matrices A and B over $I[x_1,\ldots,x_s]$, $s \geq 0$, where A is m by n and represented rowwise and B is n by q and represented columnwise.

Output: An infinite-precision integer K = MMCB(A,B) such that $K/2 \geq \text{norm}(C(i,j))$, for all elements of C = AB.

(1) [Initialize.] X ← A; Y ← B; K ← 0; n ← LENGTH(FIRST (X)); I ← ((),(),...,()), a list of length n; J ← ((),(),...,()), a list of length n.

(2) [Begin column bound list for A.] ADV(R,X); S ← I.

(3) [Compare next row element.] ADV(U,R); U ← PNORMF(U); V ← FIRST(S); h ← ICOMP(U,V); if h = 1, (erase V; ALTER(U,S); if h ≠ 1, erase U; S ← TAIL(S); if S ≠ (), go to (3); if X ≠ (), go to (2).

(4) [Begin row bound list for B.] ADV(R,Y); S ← J.

(5) [Compare next column element.] ADV(U,R); U ← PNORMF(U); V ← FIRST(S); h ← ICOMP(U,V); if h = 1, (erase V; ALTER(U,S)); if h ≠ 1, erase U; S ← TAIL(S); if S ≠ (), go to (5); if Y ≠ (), go to (4).

(6) [Compute and return norm bound.] DECAP(U,I); DECAP (V, J); R ← IPROD(U,V); erase U and V; S ← ISUM(K,R); erase K and R; K ← S; if I ≠ (), go to (6); U ← PFA(2,0); V ← IPROD(U,K); erase U, K; K ← V; return K.

In steps (2) and (3) the vector $I = (I_1,\ldots,I_n)$ is computed such that $I_h = \max_i \text{norm}(A(i,h))$. In steps (4) and (5) the vector $J = (J_1,\ldots,J_m)$ is computed such that $J_h = \max_j \text{norm}(B(h,j))$. Finally, in step (6) $K = 2\cdot\sum_{h=1}^{n} I_h\cdot J_h$ is computed. Since $\text{norm}(C(i,j)) \le \sum_{h=1}^{n} \text{norm}(A(i,h))\cdot \text{norm}(B(h,j)) \le \sum_{h=1}^{n} I_h\cdot J_h = K/2$, for each element of C, K is a bound on the norms of the elements of C and, hence, a

bound on the integer coefficients of these elements. The algorithm PNORMF is described in [COG70].

Theorem 3.4.12. Let $T_{MMCB}(m,n,q,d,\bar{m}_s,e,\bar{n}_s)$ be the maximum computing time of MMCB for $A \in M(m,n,P(d,\bar{m}_s))$ and $B \in M(m,n,P(e,\bar{n}_s))$, $s \geq 0$. Then $T_{MMCB} \lesssim mn(\log d)(\pi_{i=1}^{s}\mu_i) + nq(\log e)(\pi_{i=1}^{s}\nu_i) + n(\log d)(\log e) + n(\log n)$.

Proof: For step (1), $T_1 \sim T_{LENGTH}(n) \sim n$. The time for steps (2) and (3) is dominated by the maximum time for the mn applications of PNORMF. The maximum time for each is $T_{PNORMF}(d,\bar{m}_s) \lesssim (\log d)(\pi_{i=1}^{s}\mu_i)$, since the time for each of the possibly $\pi_{i=1}^{s}\mu_i$ additions is $\lesssim \log d$, and so $T_2 + T_3 \lesssim mn(\log d)(\pi_{i=1}^{s}\mu_i)$. Similarly, $T_4 + T_5 \lesssim nq.T_{PNORMF}(e,\bar{n}_s) \lesssim nq(\log e)(\pi_{i=1}^{s}\nu_i)$. For step (6) the maximum time is dominated by the n applications of IPROD and ISUM. Since $I_h \in P(d)$ and $J_h \in P(e)$, the time for the multiplications $I_h \cdot J_h$ is $\lesssim n(\log d)(\log e)$. Letting $L_k = \sum_{h=1}^{k} I_h J_h$, clearly each $I_h \cdot J_h \in P(de)$ and $L_k \in P(kde)$ and so the time for the sums is $\lesssim n(\log nde)$. Thus, $T_6 \lesssim n(\log d)(\log e) + n(\log n + \log d + \log e) \sim n(\log d)(\log e) + n(\log n)$. Combining the maximum computing times for all the steps, we find that $T_{MMCB} \lesssim mn(\log d)(\pi_{i=1}^{s}\mu_i) + nq(\log e)(\pi_{i=1}^{s}\nu_i) + n(\log d)(\log e) + n(\log n)$.//

The main or outer algorithm for computing the products of matrices of integer or polynomial entries, using mod-p

mappings and Garner's Method, is now presented.

Algorithm 3.4.13 (MMPY) Matrix Multiplication Using Mod-p Mappings

Input: Matrices A and B over $I[x_1,\ldots,x_s]$, $s \geq 0$, where A is m by n and B is n by q.

Output: The m by q matrix C = MMPY(A,B) over $I[x_1,\ldots,x_s]$ such that C = AB.

Note that this algorithm requires the list PRIME of odd single-precision primes which was discussed in 3.1.

(1) [Initialize.] B' ← MTRAN(B); Z ←PRIME; m ← LENGTH(A); q ← LENGTH(B').

(2) [Construct the m by q zero matrix C .] C ← (); for i = 1, m do: (D ← (); for j = 1, q do: (D ← PFL(0,D)); C ← PFL(D,C)); if MZERO(A) = 1 or MZERO(B) = 1, go to (8).

(3) [Finish initializing Garner's method.] V ← MVLIST(A); s ← LENGTH(V); I ← PFA(1,0); J ← MMCB(A,B').

(4) [Apply a mod-p mapping, if available.] If Z = (), (print error message; stop); ADV(p,Z); A* ← MMOD(p,A,s); B* ← MMOD(p,B',s).

(5) [Apply evaluation mapping algorithm.] C* ← MCPMPY(p, A*,B*); erase A*,B*.

(6) [Apply Garner's method.] C' ← MGARN(I,C ,p,C*,V); erase C,C*; C ← C'.

(7) [Recompute I and test.] T ← PFA(p,0); U ← IPROD(I,T); erase I,T; I ← U; if ICOMP(I,J) ≠ 1, go to (4); erase V,I,J.

(8) [Return.] Erase B'; return C .

In steps (1)-(3) initialization and initial zero matrix tests are performed. An integer coefficient bound J for the elements of the product C = AB is obtained which satisfies $J \geq 2 \cdot \text{norm}(C(i,j))$, for each element of C. Steps (4)-(7) are iterated in order, once for each new prime p_j employed. In step (5), MCPMPY is applied to the images $A^* = \varphi_{p_j}(A)$ and $B^* = \varphi_{p_j}(B)$, computed in step (4), and the product $C^* = A^*B^* = \varphi_{p_j}(AB) = \varphi_{p_j}(C)$ is obtained. Garner's method is then applied to C' and C* in step (6) yielding a new matrix C' with coefficients $< p_1 \cdots p_j/2$. The j^{th} execution of step (7) computes $I = p_1 \cdots p_j$ and tests $I > J$. When this test succeeds, say for $j = k$, we have $I = p_1 \cdots p_k > J \geq 2.\text{norm}(C(i,j))$ or $p_1 \cdots p_k/2 > \text{norm}(C(i,j))$, for all i,j, and so (referring to the discussion in 2.4) C' = C.

We show that $k \leq \log \text{nde}$ as follows. From the proof of Theorem 3.4.12 it can be seen that $C \in M(m,q,P(\text{nde},m_1+n_1,\ldots,m_s+n_s))$. Also, it is easily seen that $J/2 \leq \text{nde}$ from the discussion following algorithm MMCB. We know that $p_1 \cdots p_{k-1} \leq J$ and so $p_1 \cdots p_{k-1} \leq 2\text{nde}$. Hence, $p_1 \cdots p_k \leq 2\text{ndep}_k$; thus, $\log p_1 \cdots p_k \leq \log 2\text{ndep}_k = \log 2 + \log \text{nde} +$

$\log p_k < \log nde + \log 2 + \log \gamma$, whence $\log p_1 \cdots p_k \lesssim \log nde$. We conclude by showing that $\log p_1 \cdots p_k \sim k$. First, note that $p_1 \cdots p_k < \gamma^k$ and so $\log p_1 \cdots p_k < \log \gamma^k = k(\log \gamma)$. Hence, $\log p_1 \cdots p_k \lesssim k$. On the other hand, from the discussion of 3.1, each prime $p_j \geq \gamma/c$, for some small positive number c, and so $\log p_1 \cdots p_k \geq \log(\gamma/c)^k = k(\log \gamma/c)$, whence $\log p_1 \cdots p_k \gtrsim k$.

Thus, the number k of primes employed to compute C = AB (and the number of iterations of steps (4)-(7)) is $\lesssim \log nde$. We now obtain the maximum computing time of MMPY.

Theorem 3.4.14. Let $T_{MMPY}(m,n,q,d,\bar{m}_s,e,\bar{n}_s)$ be the maximum computing time of MMPY for $A \in M(m,n,P(d,\bar{m}_s))$ and $B \in M(n,q,P(e,\bar{n}_s))$, $s \geq 0$. Then $T_{MMPY} \lesssim (\log nde)\ [\ mnq(\pi_{i=1}^{s} \kappa_i) + mn\{(\log d)(\pi_{i=1}^{s} \mu_i) + \sum_{i=1}^{s} (\pi_{j=1}^{i} \mu_j)(\pi_{j=i}^{s} \kappa_j)\} + nq\{(\log e)(\pi_{i=1}^{s} \nu_i) + \sum_{i=1}^{s} (\pi_{j=1}^{i} \nu_j)(\pi_{j=i}^{s} \kappa_j)\} + mq\{(\pi_{i=1}^{s} \kappa_i)(\log de + \sum_{i=1}^{s} \kappa_i)\}\ ]$.

Proof: We first give a maximum computing time table for MMPY.

i	N_i	$T_{i,j}$	T_i
1	1	-	$m + nq$
2	1	-	$mq + mn + nq$
3	1	-	$mn(\log d)(\pi_{i=1}^{s} \mu_i) + nq(\log e)(\pi_{i=1}^{s} \nu_i) + n(\log d)(\log e) + n(\log n)$

i	N_i	$T_{i,j}$	T_i
4	log nde	$mn(\log d)(\pi_{i=1}^{s}\mu_i) + nq(\log e)(\pi_{i=1}^{s}\nu_i)$	$(\log nde)(mn(\log d)(\pi_{i=1}^{s}\mu_i) + nq(\log e)(\pi_{i=1}^{s}\nu_i))$
5	log nde	$T_{MCPMPY}(m,n,q,\bar{m}_s,\bar{n}_s) + mn(\pi_{i=1}^{s}\mu_i) + nq(\pi_{i=1}^{s}\nu_i)$	$(\log nde)\cdot[mnq(\pi_{i=1}^{s}\kappa_i) + mn\{\sum_{i=1}^{s}(\pi_{j=1}^{i}\mu_j)(\pi_{j=i}^{s}\kappa_j)\} + nq\{\sum_{i=1}^{s}(\pi_{j=1}^{i}\nu_i)(\pi_{j=i}^{s}\kappa_j)\} + mq\{(\pi_{i=1}^{s}\kappa_i)(\sum_{i=1}^{s}\kappa_i)\}]$
6	log nde	$mqj(\pi_{i=1}^{s}\kappa_i)$	$mq(\log nde)^2(\pi_{i=1}^{s}\kappa_i)$
7	log nde	j	$(\log nde)^2$
8	1	-	nq -

Selected explanations of these computing times follow. For step (1), $T_1 \sim T_{MTRAN}(n,q) + T_{LENGTH}(m) \sim nq + m$. For step (2), the time to construct C is $\sim mq$, the time to test A zero is $\sim mn$, and the time to test B zero is $\sim nq$. So, $T_2 \sim mq + mn + nq$. Assuming that neither A nor B is zero, steps (3)-(7) are each executed at least once. $T_3 \sim T_{MMCB}(m,n,q,d,\bar{m}_s,e,\bar{n}_s)$, which is given in Theorem 3.4.13.

As shown in the discussion preceding the theorem, $N_i \lesssim \log nde$, for $i = 4, 5, 6, 7$. In step (4) the maximum time to compute A* is $T_{MMOD}(m,n,d,\bar{m}_s) \lesssim mn(\log d)(\pi_{i=1}^{s}\mu_i)$ and the maximum time to compute B* is $T_{MMOD}(n,q,e,\bar{n}_s) \lesssim nq(\log e)(\pi_{i=1}^{s}\nu_i)$. In step (5), the time to erase A* is $\lesssim mn(\pi_{i=1}^{s}\mu_i)$ and the time to erase B* is $\lesssim nq(\pi_{i=1}^{s}\nu_i)$, each of which is

dominated by $mnq(\pi_{i=1}^{s}K_i)$. Hence, $T_{5,j} \lesssim T'_{MCPMPY}(m,n,q,\bar{m}_s, \bar{n}_s)$. We note that for the j^{th} execution of step (6) MGARN is applied to $C' \in M(m,q,P(p_1\cdots p_j, m_1+n_1,\ldots,m_s+n_s))$ and $C^* \in M(m,q,P^*(m_1+n_1,\ldots,m_s+n_s))$. Hence, for this execution $T_{MGARN}(m,q,p_1\cdots p_j,m_1+n_1,\ldots,m_s+n_s) \lesssim mq(\log p_1\cdots p_j)(\pi_{i=1}^{s}K_i) \sim mqj(\pi_{i=1}^{s}K_i)$. Since the time to erase C' is $\lesssim mq(\log p_1\cdots p_j)(\pi_{i=1}^{s}K_i)$, which dominates the time to erase C*, we have $T_{6,j} \lesssim mqj(\pi_{i=1}^{s}K_i) \lesssim mq(\log nde)(\pi_{i=1}^{s}K_i)$. So, $T_6 \lesssim \sum_{j=1}^{k} T_{6,j} \lesssim kmq(\log nde)(\pi_{i=1}^{s}K_i) \lesssim mq(\log nde)^2(\pi_{i=1}^{s}K_i)$. For the j^{th} execution of step (7), $I = p_1\cdots p_{j-1}$ is multiplied by p_j and the maximum time for this is $\lesssim \log p_1\cdots p_j \sim j$, which dominates the time to compare I and J. Hence $T_7 \lesssim \sum_{j=1}^{k} T_{7,j} \lesssim k^2 \lesssim (\log nde)^2$. In step (8) the erasure of B' always returns nq cells to available space and so $T_8 \sim nq$.

We now obtain the maximum computing time for the algorithm. Clearly, T'_1, T'_2, and T'_8 are dominated by $mnq(\log nde)(\pi_{i=1}^{s}K_i) \lesssim T'_5$ and $T'_7 \lesssim T'_6$. Also, $mn(\log d)(\pi_{i=1}^{s}\mu_i) + n(\log d)(\log e) + n(\log n) \lesssim mn(\log nde)(\log e)(\pi_{i=1}^{s}\mu_i)$ and $nq(\log e)(\pi_{i=1}^{s}\nu_i) \lesssim nq(\log nde)(\log e)(\pi_{i=1}^{s}\nu_i)$, which gives $T'_3 \lesssim T'_4$. We note that $T'_6 = mq(\log n)(\log nde)(\pi_{i=1}^{s}K_i) + mq(\log de)(\log nde)(\pi_{i=1}^{s}K_i)$ and that $mq(\log n)(\log nde)(\pi_{i=1}^{s}K_i) \lesssim mnq(\log nde)(\pi_{i=1}^{s}K_i) \lesssim T'_5$.

Thus, $T_{MMPY} \lesssim T'_4 + T'_5 + T'_6 \sim (\log nde)[mn(\log d)(\pi_{i=1}^{s}\mu_i) + nq(\log e)(\pi_{i=1}^{s}\nu_i) + mnq(\pi_{i=1}^{s}K_i) + mn\{\sum_{i=1}^{s}(\pi_{j=1}^{i}\mu_j)(\pi_{j=i}^{s}K_i)\}$

$+ nq\{\sum_{i=1}^{s}(\pi_{j=1}^{i}\nu_j)(\pi_{j=i}^{s}\kappa_j)\} + mq\{(\pi_{i=1}^{s}\kappa_i)(\sum_{i=1}^{s}\kappa_i)\} + mq(\log de)(\pi_{i=1}^{s}\kappa_i)\ ] = (\log nde)\cdot[\ mnq(\pi_{i=1}^{s}\kappa_i) + mn\{(\log d)(\pi_{i=1}^{s}\mu_i) + \sum_{i=1}^{s}(\pi_{j=1}^{i}\mu_j)(\pi_{j=i}^{s}\kappa_j)\} + nq\{(\log e)(\pi_{i=1}^{s}\nu_i) + \sum_{i=1}^{s}(\pi_{j=1}^{i}\nu_j)(\pi_{j=i}^{s}\kappa_j)\} + mq\{(\pi_{i=1}^{s}\kappa_i)(\log de + \sum_{i=1}^{s}\kappa_i)\}\ ]$.//

The following somewhat weaker result, paralleling Corollary 3.4.9, follows easily from the same observations which gave that corollary.

Corollary 3.4.15. $T_{MMPY}(m,n,q,d,\bar{m}_s,e,\bar{n}_s) \leq (\log nde)(\pi_{i=1}^{s}\kappa_i)$ $[\ mnq + mn(\log d + \sum_{i=1}^{s}\mu_i) + nq(\log e + \sum_{i=1}^{s}\nu_i) + mq(\log de + \sum_{i=1}^{s}\kappa_i)\ ]$.

Paralleling Corollary 3.4.10, we have the following still weaker but yet more compact dominance relation for the maximum computing time.

Corollary 3.4.16. $T_{MMPY}(m,n,q,d,\bar{m}_s,e,\bar{n}_s) \leq (\log nde)\kappa^s($ $mnq + mn(\mu + \log d) + nq(\nu + \log e) + mq(\kappa + \log de)\ )$.

CHAPTER IV

ALGORITHMS FOR THE SOLUTION OF LINEAR EQUATIONS

In this chapter are presented the algorithms for computing general solutions to systems of linear equations with integer or polynomial coefficients. Some basic theoretical results and definitions are obtained in Section 4.1. Although the algorithms of Chapter 3 will be employed in the algorithms of this chapter, additional more specialized algorithms will be required in the solution of linear equations. These are given in Section 4.2. The next three sections present the modular algorithm for solving linear equations. The innermost part of this algorithm, the nucleus, is presented in Section 4.3. This is a Gaussian elimination algorithm for computing the determinantal RRE form of a matrix over GF(p) and is essentially Algorithm 2.6.7 but rewritten for SAC-1 data structures. As for all the algorithms presented in this thesis, a computing time analysis is performed immediately following a discussion of the algorithm. The intermediate algorithm, employing evaluation and interpolation to compute an RRE form for a matrix of polynomials over GF(p), is given in Section 4.4. Then the complete modular algorithm follows in Section 4.5.

A modular algorithm for computing determinants is pre-

sented in Section 4.6. The great similarity between this algorithm and the algorithm for solving linear equations is noted.

Finally, in Section 4.7 algorithms are described for computing matrix inverses and for null space basis generation. These two algorithms turn out to be simply skeletons for applying the linear equation solving algorithm. We begin now with some basic theory.

4.1. Basic Definitions and Results

We first consider the construction of null space bases. Suppose E is an m by n nonzero RRE matrix with RE sequence $J_E = (j_1,\ldots,j_r)$ and with common diagonal value d. Let $1 \le h_1 < h_2 <\cdots< h_{n-r} \le n$ be the sequence of integers which complements $j_1, j_2,\ldots,j_r$ with respect to $1, 2,\ldots,n$. We define the n by n-r matrix Z such that, for $1 \le j \le n-r$,

$$Z(u,j) = \begin{cases} E(i,h_j), \text{ if } u = j_i \text{ and } j_i < h_j \\ -d, \text{ if } u = h_j \\ 0, \text{ otherwise} \end{cases} \qquad (1)$$

We now prove the following theorem concerning the matrix Z.

Theorem 4.1.1. Let E be an m by n nonzero RRE matrix with $J_E = (j_1,\ldots,j_r)$, $r < n$, and common diagonal value d. If Z is the n by n-r matrix defined by (1), then Z is a null space basis for E.

Proof: Let $M = EZ$ and consider any element $M(i,j)$. If $j_i < h_j$, let k be the largest integer, $k \le r$, such that $j_k < h_j$; otherwise, let $k = 0$. By the definition of Z, $M(i,j) = \sum_{t=1}^{k} E(i,j_t)Z(j_t,j) + E(i,h_j)Z(h_j,j) = \sum_{t=1}^{k} E(i,j_t)E(t,h_j) - dE(i,h_j)$. Since E is an RRE matrix, $E(i,j_t) = 0$ for $i \neq t$. Hence, if $j_i < h_j$, then $M(i,j) = E(i,j_i)E(i,h_j) - dE(i,h_j) = 0$, since $E(i,j_i) = d$. If $j_i > h_j$, then $E(i,h_j) = 0$ and so $M(i,j) = -dE(i,h_j) = 0$. Thus, $M = 0$ and so every column of Z is in the null space of E.

By the definition of Z, $Z(h_j,j) = -d \neq 0$, while $Z(h_j,u) = 0$ for $u \neq j$. It follows that the columns of Z are linearly independent. Since there are n-r columns, they therefore constitute a basis for the null space of E.//

Suppose $E = \overline{\overline{C}}$, for a nonzero matrix C. Then Z is a null space basis for $\overline{\overline{C}}$ and, since C and $\overline{\overline{C}}$ are row equivalent, $C = A\overline{\overline{C}}$ for some m by m matrix A. Thus, $CZ = (A\overline{\overline{C}})Z = A(\overline{\overline{C}}Z) = 0$ and so Z is also a null space basis for C, giving the following corollary.

Corollary 4.1.2. Let C be an m by n nonzero matrix with rank $r < n$. If $E = \overline{\overline{C}}$ and Z is defined by (1), then Z is a null space basis for C.

If C is the augmented matrix of a consistent linear system of equations, then this corollary, in fact, provides a general solution to the linear system.

Theorem 4.1.3. Let C = (A,B) be the augmented matrix of a consistent linear system AX = B, where A is m by n and nonzero and B is m by q. Let E be an m by n' RRE matrix, n' = n+q, with common diagonal value d and rank r, and suppose $J_E = J_C$. Also, let Z' be the n' by n'-r matrix constructed from E according to (1) and assume CZ' = 0. Then, if Z consists of the first n rows and first n-r columns of Z' and if Y consists of the first n rows and last q columns of Z', then (d,Y,Z) is a general solution of AX = B.

Proof: Let Z' be partitioned as $\begin{bmatrix} Z^{(1)} & Z^{(2)} \\ Z^{(3)} & Z^{(4)} \end{bmatrix}$, where $Z^{(1)}$ is n by n-r. We have $CZ' = (A,B)Z' = (AZ^{(1)} + BZ^{(3)}, AZ^{(2)} + BZ^{(4)})$. By referring to the definition of Z', one sees that $Z^{(3)}$ is the q by n-r zero matrix and $Z^{(4)}$ is the q by q diagonal matrix with diagonal value -d. Since $CZ' = 0$, it follows that $0 = AZ^{(1)} + BZ^{(3)} = AZ^{(1)}$ and $0 = AZ^{(2)} + BZ^{(4)} = AZ^{(2)} - dB$. Hence, letting $Z = Z^{(1)}$ and $Y = Z^{(2)}$, we have $AZ = 0$ and $AY = dB$. Since the columns of Z' are linearly independent by Theorem 4.1.1, it follows immediately that the columns of Z are also linearly independent.//

If $E = \overline{\overline{C}}$, then $CZ' = 0$ by Corollary 4.1.2. Hence, since $d = \delta(C) = \delta(A)$, the triple $(\delta(A),Y,Z)$ constitutes a general solution, called the determinantal general solution.

The next result provides a sufficient condition for deciding if the RE sequence of some RRE matrix is the RE sequence of another matrix. This test will be applied in Sections 4.4 and 4.5.

Theorem 4.1.4. Let C be an m by n nonzero matrix of rank $r < n$ and E an m by n nonzero RRE matrix of rank $r' \leq r$, where $J_C \leq J_E$ if $r' = r$. Suppose Z is the matrix constructed from E as defined by (1). Then, if $CZ = 0$, it follows that $J_C = J_E$.

Proof: If $r' < r$, then CZ is nonzero, for Z then has more than n-r linearly independent columns. So, $r' = r$. Sup-

pose $J_E = (j'_1,\ldots,j'_r) > (j_1,\ldots,j_r) = J_C$. Then there is an integer k: $1 \le k \le r$ such that $j'_i = j_i$, $1 \le i < k$, and $j'_k > j_k$. Let $1 \le h_1 < h_2 <\cdots< h_{n-r} \le n$ be the complement of $j'_1,\ldots,j'_r$. For some j, $h_j = j_k$. Let $Z^{(t)}$ and $C^{(t)}$ denote the t^{th} columns of Z and C, respectively. Then

$$Z^{(j)}[i] = \begin{cases} -E(k,j'_k) \neq 0, \text{ if } i = h_j = j_k \\ E(u,k), \text{ if } i = j'_u = j_u,\ 1 \le u < k \\ 0, \text{ elsewhere} \end{cases}$$

Hence, if $CZ = 0$, then $CZ^{(j)} = \sum_{u=1}^{k} Z^{(j)}[j_u]C^{(j_u)}$ is a zero vector. Since $Z^{(j)}[j_k] \neq 0$, either $C^{(j_k)}$ is a zero vector or $C^{(j_k)}$ is linearly dependent on columns $C^{(j_1)},\ldots,C^{(j_{k-1})}$, each of which is contrary to the definition of J_C. Therefore, $j'_k = j_k$ and we have $J_C = J_E$.//

We know that a large number of elements of a nonzero RRE matrix E are known to be zero. The algorithms of Sections 4.3-4.5 can be made more efficient by discarding these elements from the representation of E. To do this we let k be the least nonnegative integer such that $j_{i+1} = j_i + 1$, for $k < i \le r$, where $J_E = (j_1,\ldots,j_r)$ and we take $j_{r+1} = n+1$ and $j_o = 0$. Thus, k is in the range: $0 \le k \le r$. If $k = 0$, then the only nonzero elements of E are the diagonal elements. If $k > 0$, then rows $1,\ldots,k$ of E have non-diagonal elements which are possibly nonzero; for row i, these are: $E(i,j)$, $j_i < j \le n$ and $j \neq j_u$, $i < u \le r$.

We define the list V as follows: If k = 0, V is the null list (). If k > 0, V is the list $(V_1,\ldots,V_k)$, where V_i is the list of the possibly nonzero non-diagonal elements of row i of V_i. That is, V_i is the list ($E(i,j_i+1)$, $\ldots,E(i,j_{i+1}-1),E(i,j_{i+1}+1),\ldots,E(i,j_r-1),E(i,j_r+1),\ldots,$ $E(i,n)$). Clearly, E can be constructed from J_E, d, V. Hence, we call the triple (J_E,d,V) the <u>compact representation</u> of the RRE matrix E. The list V will be referred to as the <u>non-diagonal part</u> of the compact representation of E.

We note that, when the non-diagonal part V is non-null, it has a structure similar to that for a matrix. However, its "rows" $V_1,\ldots,V_k$ are of unequal lengths. Nevertheless, certain algorithms which receive matrices as inputs apply as well to non-diagonal parts which are non-null. Algorithms already presented to which a non-null non-diagonal part may be input with the desired result are MVLIST, MERASE, MMOD, MGARN,MCPNV, MCPEVL, MCPINT, MZERO. More will be presented in the next section: MZCON, MEQ, MCPEQ.

It is easy to verify that (1) is equivalent to:

$$Z(j_i,j) = E(i,h_j)$$
$$Z(h_j,u) = \begin{cases} -d, & u = j \\ 0, & u \neq j \end{cases} \qquad (2)$$

Thus, when V is non-null, its rows $V_1,\ldots,V_k$ are identical to the rightmost segments of rows $j_1,\ldots,j_k$ of the null

space basis matrix Z constructed from E by definition (1). The remaining elements of these rows of Z are zeros. The only nonzero elements of the remaining rows of Z have the value -d and the positions of these elements are computable from J_E. Hence, Z can be constructed directly from the compact representation (J_E,d,V) of E. We note that the integer n is equal to $J_E[r]$, if V = (), or $J_E[r]$ plus the length of the first row of V, if $V \neq ()$. It may be more convenient to simply input n along with (J_E,d,V). This is done in an auxiliary algorithm, given in the next section, which accomplishes this construction.

4.2. Auxiliary Algorithms for Solving Linear Equations

Several supporting algorithms will be required by the modular algorithm for solving systems of linear equations. These algorithms are given in this section, the first of which performs the lexicographical comparison of any two vectors in Ω.

Algorithm 4.2.1. (VCOMP) Vector Comparison

Input: Vectors H and K in Ω.

Output: The integer z = VCOMP(H,K), where z = -1, 0, or +1, depending on whether H < K, H = K, or H > K, respectively.

(1) [Initialize.] S ← H; T ← K.

(2) [Test if H exhausted.] If S ≠ (), go to (3); if T ≠ (), go to (5); z ← 0; return z.

(3) [Test if K exhausted.] If T ≠ (), go to (4); go to (6).

(4) [Compare next elements.] ADV(u,S); ADV(v,T); w ← u-v; if w = 0, go to (2); if w > 0, go to (6).

(5) [H < K return.] z ← -1; return z.

(6) [H > K return.] z ← 1; return z.

Theorem 4.2.2. Let $T_{VCOMP}(m,n)$ be the maximum computing time of VCOMP for H of length m and K of length n. Then $T_{VCOMP} \sim \min(m,n) + 1$.

Proof: For each step $T_{i,j} \sim 1$. Steps (2), (3), and (4) are each executed at most min(m,n) + 1 times and so

$T_2 + T_3 + T_4 \lesssim \min(m,n) + 1$. Since T_1, T_5, and T_6 are ~ 1, $T_{VCOMP} \lesssim \min(m,n) + 1$. Also, if H is an initial segment of K, then $N_2 = m+1 = \min(m,n) + 1$ and so $T_{VCOMP} \sim \min(m,n) + 1$.//

The next algorithm constructs a zero matrix of the same size as the input matrix.

Algorithm 4.2.3. (MZCON) Zero Matrix Construction

Input: An m by n matrix A over $I[x_1,\ldots,x_s]$, $s \geq 0$, or $GF(p)[x_1,\ldots,x_s]$, $s \geq 1$.

Output: An m by n zero matrix B obtained by B = MZCON(A).

(1) [Initialize.] B ← (); S ← A.

(2) [Begin next row.] ADV(U,S); X ← ().

(3) [Prefix next row element.] X ← PFL(0,X); U ← TAIL(U); if U ≠ (), go to (3); B ← PFL(X,B); if S ≠ (), go to (2).

(4) [Return.] B ← INV(B); return B.

This algorithm traverses the matrix A, prefixing a zero element to a row of B for each element of A visited in step (3). Thus, $N_3 = mn$ and $T_{3,j} \sim 1$ and so $T_3 \sim mn$, which dominates the time for the other steps. The computing time is given in the following theorem.

Theorem 4.2.4. Let $T_{MZCON}(m,n)$ be the computing time of MZCON for A an m by n matrix. Then $T_{MZCON} \sim mn$.

There now follow two simple algorithms for testing matrix equality.

Algorithm 4.2.5. (MEQ) Matrix Equality Test

Input: m by n matrices A and B over $I[x_1,\ldots,x_s]$, $s \geq 0$.

Output: The integer z = MEQ(A,B), where z = 1, if A = B, and z = 0, if $A \neq B$.

(1) [Initialize.] $X \leftarrow A$; $Y \leftarrow B$.

(2) [Obtain next rows.] ADV(S,X); ADV(T,Y).

(3) [Test next row elements.] $C \leftarrow$ PDIF(FIRST(S), FIRST(T)); if $C \neq 0$, go to (4); $S \leftarrow$ TAIL(S); $T \leftarrow$ TAIL(T); if $S \neq ()$, go to (3); if $X \neq ()$, go to (2); $z \leftarrow 1$; return z.

(4) [Unequal return.] Erase C; $z \leftarrow 0$; return z.

Theorem 4.2.6. Let $T_{MEQ}(m,n,d,\bar{m}_s,e,\bar{n}_s)$ be the maximum computing time of MEQ for $A \in M(m,n,P(d,\bar{m}_s))$ and $B \in M(m,n,P(e,\bar{n}_s))$, $s \geq 0$. Then $T_{MEQ} \lesssim mn(\log d + \log e)(\pi_{i=1}^{s}\mu_i + \pi_{i=1}^{s}\nu_i)$.

Proof: The maximum time for this algorithm is dominated by the maximum time for the applications of PDIF in step (3). There are at most mn such applications and the time for each is $T_{PDIF}(d,\bar{m}_s,e,\bar{n}_s) \lesssim (\log d + \log e)(\pi_{i=1}^{s}\mu_i + \pi_{i=1}^{s}\nu_i)$.//

The next algorithm performs such a test for matrices of polynomials over GF(p).

Algorithm 4.2.7. (MCPEQ) Congruence Matrix Equality Test

Input: A prime number p and m by n matrices A and B over $GF(p)[x_1,\ldots,x_s]$, $s \geq 1$.

Output: The integer z = MCPEQ(p,A,B), where z = 1, if A = B, and z = 0, if A ≠ B.

(1) [Initialize.] X ← A; Y ← B.

(2) [Obtain next rows.] ADV(S,X); ADV(T,Y).

(3) [Test next row elements.] C ← CPDIF(p,FIRST(S), FIRST(T)); if C ≠ 0, go to (4); S ← TAIL(S); T ← TAIL(T); if S ≠ (), go to (3); if X ≠ (), go to (2); z ← 1; return z.

(4) [Unequal return.] Erase C; z ← 0; return z.

This algorithm is identical to the preceding algorithm, with the exception that CPDIF is applied in Step (3). Since $T_{CPDIF}(\bar{m}_s,\bar{n}_s) \leq \pi_{i=1}^{s}\mu_i + \pi_{i=1}^{s}\nu_i$, the computing time is as follows.

Theorem 4.2.8. Let $T_{MCPEQ}(m,n,\bar{m}_s,\bar{n}_s)$ be the maximum computing time of MCPEQ for $A \in M(m,n,P^*(\bar{m}_s))$ and $B \in M(m,n, P^*(\bar{n}_s))$, $s \geq 1$. Then $T_{MCPEQ} \leq mn(\pi_{i=1}^{s}\mu_i + \pi_{i=1}^{s}\nu_i)$.

The next algorithm constructs a null space basis for an RRE matrix from its compact represention. This was discussed briefly at the end of the preceding section. It will be employed in the substitution tests of the modular algorithm, for solving linear equations.

Algorithm 4.2.9 (NULCON) Null Space Basis Construction

Input: A positive integer n, the RE sequence J of an m by n RRE matrix E of rank r: $1 \leq r < n$, over $I[x_1,\ldots,x_s]$,

$s \geq 0$, or $GF(p)[x_1,\ldots,x_s]$, $s \geq 1$, the integer or polynomial P, where -P is the common diagonal value of E, and the list W such that (J,-P,W) is the compact representation of E.

Output: The n by n-r null space basis matrix Z = NULCON(n, J,P,W) for E, as defined by formula (2) of Section 4.1.

(1) [Initialize.] L ← J; W' ← CINV(W); r ← LENGTH(J); t ← r - LENGTH(W); if t > 0, apply W' ← PFL(0,W') t times; W' ← INV(W'); e ← n-r; Z ← (); ADV(h,L); i ← 0; k ← 0.

(2) [Begin next row of Z.] i ← i+1; if i > n, go to (5); if i = h, go to (4).

(3) [Construct next row using P.] k ← k+1; U ← (); t ← e-k; if t > 0, do U ← PFL(0,U) t times; U ← PFL(BORROW(P),U); t ← k-1; if t > 0, do U ← PFL(0,U) t times; Z ← PFL(U,Z); go to (2).

(4) [Construct next row from row of W'.] DECAP(U,W'); U ← INV(CINV(U)); t ← e - LENGTH(U); if t > 0, do U ← PFL(0,U) t times; Z ← PFL(U,Z); if L ≠ (), ADV(h,L); go to (2).

(5) [Return.] Z ← INV(Z); return Z.

Let $h_1,\ldots,h_{n-r}$ be the complement of $j_1,\ldots,j_r$. The k^{th} execution of step (3) constructs row h_k of Z, which consists of all zeros except for the value P in position k.

The k^{th} execution of step (4) constructs row j_k of Z, which consists of the elements of row k of W, if such a row exists, preceded by sufficiently many zeros to make its length n-r. If W has less than k rows, then the new row of Z is a row of zeros. The rows are constructed in the order 1, 2,...,n. In step (4) the statement: U ← INV(CINV(U)) constructs a new row U indentical to the row of W by borrowing the elements of W. The list W' constructed in step (1) is identical to W, except that sufficently many null vectors have been suffixed so that W' has length r.

Theorem 4.2.10. Let $T_{NULCON}(n,r)$ be the maximum computing time of NULCON for E a matrix with n columns and rank r. Then $T_{NULCON} \sim n(n-r)$.

Proof: Step (1) is executed once, the time being dominated by $T_{LENGTH}(r) + T_{INV}(r) \sim r$. Step (2) is executed n+1 times, where $T_{2,j} \sim 1$, and so $T_2 \sim n$. Step (3) is executed n-r times and $T_{3,j} \sim n-r$; so, $T_3 \sim (n-r)^2$. Step (4) is executed r times, where $T_{4,j} \sim n-r$, and so $T_4 \sim r(n-r)$. $T_5 \sim T_{INV}(n) \sim n$. Since $T_3' + T_4' = (n-r)^2 + r(n-r) = n(n-r)$, the maximum time for steps (3) and (4) dominates the times for the other steps, giving the theorem.//

The last algorithm of this section computes a degree bound for the determinantal RRE form of a matrix of polynomials over GF(p). In fact, this bound will also hold for

certain other RRE forms for C, which are described as follows. Let C be an m by n nonzero matrix with RE sequence $J_C = (j_1,\ldots,j_r)$ and pick any permutation $I = (k_1,\ldots,k_m) \in \mathcal{P}_C$. Define the m by n matrix F as follows:

$$F(h,j) = \left\{ \begin{array}{l} c\begin{pmatrix} k_1,\ldots,k_r \\ j_1,\ldots,j_{h-1},j,j_{h+1},\ldots,j_r \end{pmatrix}, \ 1 \le h \le r \\ \quad \text{and } 1 \le j \le n \\ 0, \text{ otherwise} \end{array} \right\} \quad (1)$$

That F is an RRE matrix with RE sequence $J = J_C$ follows by Theorem 2.1.2.1 with I replacing I_C. By applying row interchanges to C the matrix C' can be obtained, where $C'_i = C_{k_i}$, $1 \le i \le m$. Then the exact division algorithm, Algorithm 2.2.2.1, applied to C' produces $\overline{\overline{C'}}$ with $J_{C'} = J_C$ and $I_{C'} = (1, 2,\ldots,m)$. This can be shown in a straightforward manner, using the methods of Section 2.2. We note that $\overline{\overline{C'}}$ is an RRE form for C. For $1 \le h \le r$ and $1 \le j \le n$, $\overline{\overline{C'}}(h,j) = C'\begin{pmatrix} 1,2,\ldots,r \\ j_1,\ldots,j_{h-1},j,j_{h+1},\ldots,j_r \end{pmatrix} =$ $c\begin{pmatrix} k_1,k_2,\ldots,k_r \\ j_1,\ldots,j_{h-1},j,j_{h+1},\ldots,j_r \end{pmatrix} = F(h,j)$. Since all other elements of $\overline{\overline{C'}}$ and F are zero, $F = \overline{\overline{C'}}$ and so F is an RRE form for C. Formula (1) thus defines a certain class of RRE forms for C, which includes $\overline{\overline{C}}$.

Algorithm 4.2.11. (DBRRE) Degree Bound for a Class of RRE Forms of a Matrix

Input: An m by n nonzero matrix C over $GF(p)[x_1,\ldots,x_s]$,

$s \geq 1$, and its RE sequence $J = (j_1,\ldots,j_r)$.

Output: An integer b = DBRRE(J,C), which bounds the degrees in the main variable x_s of the elements of each RRE form F for C as defined by (1).

(1) [Initialize.] X ← MTRAN(C); Y ← X; K ← J; e ← 0; f ← 0; L ← (); i ← 0.

(2) [Obtain next column of C.] If Y = (), go to (5); ADV(U,Y); i ← i+1; if K = (), go to (4); if i ≠ FIRST(K), go to (4); g ← 0.

(3) [Get maximum degree of pivot column.] ADV(T,U); if T ≠ 0, (d ← deg(T); if d > g, g ← d); if U ≠ (), go to (3); L ← PFA(g,L); K ← TAIL(K); go to (2).

(4) [Get maximum degree of non-pivot column.] ADV(T,U); if T ≠ 0, (d ← deg(T); if d > f, f ← d); if U ≠ (), go to (4); go to (2).

(5) [Compute sum and minimum of elements of list L.] DECAP(d,L); e ← e+d; if d < g, g ← d; if L ≠ (), go to (5).

(6) [Complete computation of degree bound.] b ← e; h ← f-g; if h > 0, b ← b+h; erase X; return b.

In step (1) the transpose X of C is computed, yielding X as a columnwise representation of C. This facilitates the computation of the maximum column degrees required to compute the bound b. In step (3) the maximum column degrees e_k, $1 \leq k \leq r$, are computed, where $e_k = \max_{1\leq i\leq m} \deg(C(i,j_k))$,

and the list $L = (e_r, e_{r-1}, \ldots, e_1)$ is constructed. In step (5) the sum $e = \sum_{k=1}^{r} e_k$ and also $g = \min_k e_k$ are computed.

We note that Lemma 2.5.1 might just as easily have established:

$$\deg(\det(B)) \leq \sum_{j=1}^{m} \max_{1 \leq i \leq m} \deg(B(i,j))$$

Thus, Theorem 2.5.4 could have been obtained with the e_h's, which will be called e_h' here, defined by: $e_h' = \max_{1 \leq u \leq r} \deg(C(i_u, j_h))$, $1 \leq h \leq r$. Clearly, $e_h' \leq e_h$, $1 \leq h \leq r$, and so $\sum_{h=1}^{r} e_h' - \min_{1 \leq h \leq r} e_h' \leq \sum_{h=1}^{r} e_h - \min_{1 \leq h \leq r} e_h = e - g$. In step (4) the maximum degree f of the elements in columns $j \neq j_u$ is computed. Clearly, this f bounds the f of Theorem 2.5.4 and so $\deg(\bar{\bar{C}}(h,j)) \leq f + e - g$, for $j \neq j_u$, $1 \leq u \leq r$. It is also true in a completely analogous fashion, for F as defined by (1), that $\deg(F(h,j)) \leq f + e - g$, for $j \neq j_u$, $1 \leq u \leq r$. Of course, we also have $\deg(\delta(C)) \leq \sum_{h=1}^{r} e_h = e$ and similarly $\deg(F(h, j_h)) \leq e$. Finally, since b is computed in step (6) so that $e \leq b$ and $f + e - g \leq b$, we see that b is a single degree bound on the elements of $\bar{\bar{C}}$ and, in fact, on the elements of every RRE form F for C defined by (1).

Theorem 4.2.12. Let $T_{DBRRE}(m,n)$ be the computing time of DBRRE for $C \in M(m,n,GF(p)[x_1, \ldots, x_s])$, $s \geq 1$. Then $T_{DBRRE} \sim mn$.

Proof: Step (1) is executed once and so $T_1 \sim T_{MTRAN}(m,n) \sim mn$. For steps (2)-(5), $T_{i,j} \sim 1$, $i = 2, 3, 4, 5$, and so $T_i \sim N_i$, $i = 2, 3, 4, 5$. Step (2) is executed n+1 times; hence, $T_2 \sim n$. Step (3) is executed m times for each of r pivot columns and so $T_3 \sim mr$. Step (4) is executed m times for each of the n-r non-pivot columns; if $n = r$, step (4) is not executed. Hence, $T_4 \sim m(n-r)$. Since step (5) is executed r times, $T_5 \sim r$. Finally, step (6), being executed once, has $T_6 \sim mn$, the time to return mn cells to available space by the erasure of X. Therefore, since $T_2 + T_5 \leq T_3 + T_4 \sim mr + m(n-r) \sim mn \sim T_1 \sim T_6$, we have $T_{DBRRE} \sim mn$.//

4.3. Computing the Determinantal RRE Form over GF(p)

In this section the inner algorithm CRRE, employed in the solution of linear equations is presented and analyzed. The middle and outer algorithms will be presented in the next two sections. CRRE employs a Gaussian elimination algorithm to produce, in effect, a diagonalized matrix with 1's on the diagonal. The elements of this matrix are then multiplied by an element of GF(p) to give $\overline{\overline{A}}$. Not all elements of $\overline{\overline{A}}$ are retained; instead the compact form of $\overline{\overline{A}}$ is computed. The algorithm now follows.

Algorithm 4.3.1. (CRRE) Computing the Determinantal RRE Form by Gaussian Elimination over GF(p)

Input: A prime number p and an m by n nonzero matrix A over GF(p), whose list does not overlap another list.

Output: The list W = (J,I,d,V) obtained by W = CRRE(p,A), where $J = J_A$, $I = I_A$, $d = \delta(A)$, and V is the non-diagonal part of the compact representation of $\overline{\overline{A}}$. The input list A is erased.

(1) [Initialize.] B' ← A; m ← LENGTH(B'); n ← LENGTH(FIRST(B')); I ← (); J ← (); d ← 1; V ← (); B ← (); I' ← (); for k = 1, m do: (I' ← PFA(m+1-k,I')); z ← 0.

(2) [Begin next step of triangularization.] z ← z+1; B" ← B'; B' ← (); I" ← I'; I' ← ().

(3) [Search column z for next pivot element.] DECAP(F,B");

DECAP(e,F); DECAP(k,I"); if e ≠ 0, go to (4); I' ← PFA(k,I'); if F ≠ (), B' ← PFL(F,B'); if B" ≠ (), go to (3); go to (8).

(4) [Update d, J, I, and B, and begin elimination.] d ← CPROD(p,e,d); J ← PFA(z,J); I ← PFA(k,I); B ← PFL(F,B); if F = (), go to (8); e ← CRECIP(p,e); G ← F.

(5) [Transform pivot row.] ALTER(CPROD(p,e,FIRST(G)), G); G ← TAIL(G); if G ≠ (), go to (5).

(6) [Begin to transform next row, if any.] If B" = (), go to (8); DECAP(G,B"); DECAP(u,G); B' ← PFL(G,B'); if u = 0, go to (6); H ← F.

(7) [Recompute next element of row G.] s ← CPROD(p,u,FIRST(H)); t ← CDIF(p,FIRST(G),s); ALTER(t,G); G ← TAIL(G); H ← TAIL(H); if G ≠ (), go to (7); go to (6).

(8) [Test for end of triangularization.] B' ← INV(B'); I' ← CONC(INV(I'),I"); if B' ≠ (), go to (2); I ← CONC(INV(I), I'); if B" ≠ (), erase B".

(9) [Initialize splitting of row.] If B = (), (U ← V; V ← (); go to (15)); DECAP(H,B); H ← INV(H); $\bar{H}$ ← (); M ← (); z ← n; L ← J.

(10) [Split row.] If H = (), go to (11); DECAP(u,H); k ← FIRST(L); if z = k, (M ← PFA(u,M); L ← TAIL(L)); if z ≠ k, $\bar{H}$ ← PFA(u,$\bar{H}$); z ← z-1; go to (10).

(11) [Initialize diagonalization of row.] T ← V; $\bar{H}$ ← INV($\bar{H}$).

(12) [Prepare transformation by next row of V.] If M = (), go to (14); DECAP(u,M); if T = (), go to (12); ADV(S,T); $H \leftarrow \bar{H}$.

(13) [Transform by next row of V.] If S = (), go to (12); ADV(x,S); w ← FIRST(H); w ← CDIF(p,w,CPROD(p,u,x)); ALTER(w,H); H ← TAIL(H); go to (13).

(14) [Prefix row to V.] If $\bar{H} \neq ()$, V ← PFL($\bar{H}$,V); go to (9).

(15) [Compute next row of non-diagonal part.] If U = (), go to (17); DECAP(T,U); S ← ().

(16) [Multiply next row element by δ(A).] DECAP(e,T); g ← CPROD(p,d,e); S ← PFA(g,S); if $T \neq ()$, go to (16); V ← PFL(S,V); go to (15).

(17) [Construct output list and return.] V ← INV(V); J ← INV(J); W ← PFL(J,PFL(I,PFA(d,PFL(V,0)))); return W.

This algorithm accomplishes the same computations as does the algorithm of Section 2.6 in transforming A to determinantal RRE form. That is, the triangularization performed in steps (1)-(8) of CRRE is essentially the same as would be performed by Algorithm 2.6.1. The back substi-

tution phase of the complete diagonalization, as done in steps (2) and (3) of Algorithm 2.6.3, is accomplished in steps (9)-(14). The multiplication of the elements by $d = \delta(A)$, as done in step (2) of Algorithm 2.6.7, is accomplished in steps (15)-(16). In place of computing $\bar{\bar{A}}$, CRRE computes the compact representation (J,d,V) of $\bar{\bar{A}}$ and returns it along with $I = I_A$. An explanation of the individual steps now follows.

In step (1), $J = J_o = \emptyset$, $I' = I_o = (1,2,\ldots,m)$, and $B' = B'^{(o)} = A$ are initially defined. Steps (2)-(8) accomplish each pivot operation. For the k^{th} pivot operation the following lists have the indicated roles. B is the list of the first k-1 triangularized rows, with latest, k-1, being first: $B = (B_{k-1}, B_{k-2}, \ldots, B_1)$. For row B_i the leading j_i-1 zeros and the 1 in column j_i have been discarded, the first of the remaining elements being in column j_i+1. B" is the list of rows B_k, $B_{k+1}, \ldots, B_m$, which have yet to be searched for a pivot element in column z or, if one has been found, have yet to be transformed by the pivot row. The first z-1 elements of a row of B", which are zero, have been discarded. The non-pivot rows, which either have been searched in column z or have been transformed by the pivot row, appear in reverse order as the elements of B'. The first z elements of these rows are zero and have been dis-

carded. The lists I, I', I" have an analogous function with respect to the permutation $I_{k-1} = (i_{k-1,1},\ldots,i_{k-1,m})$ as have B, B', B" with respect to the matrix $B'^{(k-1)}$, referring to Section 2.6. Thus, just after the element in the i^{th} row has been tested against zero or the i^{th} row has been transformed, the three lists appear as: $I = (i_{k-1,k-1},\ldots,i_{k-1,1})$, $I' = (i_{k-1,i},\ldots,i_{k-1,k})$, and $I" = (i_{k-1,i+1},\ldots,i_{k-1,m})$. If the element in the last row has been tested nonzero unsuccessfully, then $I' = (i_{k-1,m},\ldots,i_{k-1,k})$ and I" = (). If the k^{th} pivot has been found in row t and all the rows transformed, then $I = (i_{k-1,t},i_{k-1,k-1},\ldots,i_{k-1,1}) = (i_{k,k},\ldots,i_{k,1})$, $I' = (i_{k-1,m},\ldots,i_{k-1,t+1},i_{k-1,t-1},\ldots,i_{k-1,k}) = (i_{k,m},\ldots,i_{k,k+1})$, and I" = ().

In step (2), B" and I" receive the elements of B' and I', respectively, which are then redefined as null lists to begin the next pivot operation. During the pivot search in step (3), elements are removed from B" and I" and prefixed to B' and I', respectively. When the pivot element is found, d is recomputed and the required elements are prefixed to the lists J, I, and B in step (4). We note that $z = j_k$ is prefixed to J to form the list $(j_k,j_{k-1},\ldots,j_1)$. Multiplying the pivot row, which now appears as the first element of B, by the inverse e of the pivot element in step (5) prepares it for the elimination on the remaining rows,

which constitute B". The leading elements of these rows, which are to be zeroed, are not recomputed but are simply discarded, after being used in the transformation in steps (6) and (7). The transformed rows are removed from B" and prefixed to B'. In step (8) the inversion of B' and I' puts their elements in the correct order. If B' is non-null, the next pivot operation is begun in step (2). Otherwise, I is inverted to form the list $(i_{r,1},\ldots,i_{r,r})$ and concatenated with I' = $(i_{r,r+1},\ldots,i_{r,m})$ to form I = $(i_{r,1},\ldots,i_{r,m}) = (i_1,\ldots,i_m)$. However, if the r^{th} pivot element was found in column $j_r = n$, then transfer is made directly to step (8) from step (4) and no further row transformations (eliminations) need be performed. In this case, in step (8) we have B" either as a single column matrix (of elements to be zeroed) or the null list and also I' = $(i_{r,r+1},\ldots,i_{r,t})$ and I" = $(i_{r,t+1},\ldots,i_{r,m})$, for $r \leq t \leq m$. So, if B" $\neq$ (), B" is erased and, after the two concatenations, I = $(i_1,\ldots,i_m)$. B is left with the triangularized rows in the reverse order.

The diagonalization (back substitution) now begins in step (9), where the initializations for diagonalizing the next row of B are performed. Suppose the list B is $(B_r, B_{r-1}, \ldots, B_1)$ upon finishing the triangularization (i.e., rows in reverse order). The list V, which is initially (), is to

receive the diagonalized rows as they are generated. Let V_i be the list resulting from transforming row B_i. The following lists have the indicated rolls in transforming B_i. H is initially set to B_i in step (9), but with elements reversed: $H = (B(i,n),\ldots,B(i,j_i+1))$. Row H is then "split" into two lists: $\bar{H}$ and M. $\bar{H}$ is to consist in the elements in non-diagonal columns: $\bar{H} = (B(i,n),\ldots,B(i,j_r+1),B(i,j_{r-1}),\ldots,B(i,j_i+1))$. M is to consist in the elements in the diagonal columns: $M = (B(i,j_{i+1}),\ldots,B(i,j_r))$, done in steps (9)-(11).

Suppose that rows $B_r,\ldots,B_{i+1}$ have been diagonalized, for $i \geq 1$, with $V = (V_{i+1},\ldots,V_r)$ and $B = (B_i,\ldots,B_1)$ resulting. $\bar{H}$ and M are first computed as defined above. There then follows a sequence of iterations of steps (12) and (13), which accomplish the transformation of $\bar{H}$ by $V_{i+1},\ldots,V_r$. That is $\bar{H}$ is replaced by $\bar{H} - B(i,j_t)\cdot V_t$ for $t = i+1,\ldots,r$. Control finally goes from step (9) to step (15) with $U = (V_1,\ldots,V_r)$ and $V = ()$.

In steps (15) and (16) V is redefined as the nondiagonal part of the compact representation of $\bar{\bar{A}}$ but with rows reversed in order. This involves discarding those V_t at the end of V which are empty, reversing the order of the elements of each remaining V_t while multiplying each element by d. In step (17) V is inverted to properly order its rows, J is inverted

to form $J = (j_1,\ldots,j_r)$ and the output list $W = (J,I,d,V)$ is constructed and returned.

For the maximum computing time of CRRE a codominance relation will be obtained. This will require the following result.

Lemma 4.3.2. If r, m, and n are integers such that $1 \le r \le \min(m,n)$, then $\sum_{i=1}^{r}(m-i+1)(n-i+1) \ge mnr/8$.

Proof: Let $k = \min(m,n)$ and assume $1 \le r \le k$. If $1 \le i \le \lceil k/2 \rceil$, then $m-i+1 > m-k/2$ and $n-i+1 > n-k/2$. Thus, if $r \le \lceil k/2 \rceil$, then $\sum_{i=1}^{r}(m-i+1)(n-i+1) > \sum_{i=1}^{r}(m-k/2)(n-k/2) \ge \sum_{i=1}^{r}(m/2)(n/2) = mnr/4$. On the other hand, if $r > \lceil k/2 \rceil$, then $\sum_{i=1}^{r}(m-i+1)(n-i+1) > \sum_{i=1}^{\lceil k/2 \rceil}(m-i+1)(n-i+1) > \sum_{i=1}^{\lceil k/2 \rceil}(m/2)(n/2) = \lceil k/2 \rceil mn/4 \ge \lceil r/2 \rceil mn/4 \ge mnr/8$.//

Theorem 4.3.3. Let $T_{CRRE}(m,n,r)$ be the maximum computing time of CRRE for $A \in M(m,n,GF(p))$ and A of rank $r \ge 1$. Then $T_{CRRE} \sim mnr$.

Proof: To establish codominance requires showing that dominance holds in both directions. We first show that $T_{CRRE} \lesssim mnr$. To begin, the following maximum computing time table is derived.

i	N_i	$T_{i,j}$	T_i
1	1	-	m+n
2	n	1	n
3	mn	1	mn

i	N_i	$T_{i,j}$	T_i
4	r	1	r
5	nr	1	nr
6	mr	1	mr
7	mnr	1	mnr
8	n	m	mn
9	r	n	nr
10	nr	1	nr
11	r	n-r+1	r(n-r+1)
12	r^2	1	r^2
13	$r^2(n-r+1)$	1	$r^2(n-r+1)$
14	r	1	r
15	r	1	r
16	r(n-r+1)	1	r(n-r+1)
17	1	-	r

Selected explanation of these computing times now follow. In step (1) the times to compute m, n, and I' are $\sim$ m, n, m, respectively, and so $T_1 \sim m + n + m \sim m + n$. Step (2) is executed once for each pivot operation, plus possibly once more, and so $T_2 \sim N_2 \sim r$. In step (3) the maximum number of elements of A searched for the pivot elements is $\leq mn$ and so $T_3 \sim N_3 \leq mn$. Step (4) is executed r times; so $T_4 \sim N_4 = r$. Step (5) is executed at most n-1 times for each pivot operation and so $T_5 \sim N_5 \lesssim nr$. Step (6) is executed at most m-i+1 times for the i^{th} pivot operation and so $T_6 \sim N_6 \lesssim mr$. Step (7) is executed at most (m-i+1)(n-i+1) times for the i^{th} pivot operation and so

$T_7 \sim N_7 \lesssim \sum_{i=1}^{r}(m-i+1)(n-i+1) \leq \sum_{i=1}^{r} mn = mnr$. For step (8), which is executed at most n times and for which $T_{8,j} \lesssim m$ for $1 \leq j \leq r$, we have $T_8 \lesssim rm$. Thus $T_7' =$ mnr clearly dominates the times for the forward elimination steps.

Step (9) is executed r+1 times, where $T_{9,j} \lesssim T_{INV}($ $n-r+j) \lesssim n$, and so $T_9 \lesssim nr$. In splitting row H, step (10) is executed once for each of possibly n-1 elements of each of possibly r rows of B. So, $T_{10} \sim N_{10} \lesssim nr$. Step (11) is executed once for each of possibly r rows of B, where the time for each execution is $\lesssim n$, and so $T_{11} \lesssim rn$. Step (12) is executed r-k times to transform B_k, $1 \leq k < r$, and so $T_{12} \sim N_{12} \lesssim \sum_{k=1}^{r}(r-k) \leq r^2$. The number of executions of step (13) to transform B_k by $S = V_t$ is equal to the number of elements of V_t, plus one, which is $n-j_t-(r-t) + 1 \leq n-t$ $-r+t+1 = n-r+1$. Since $V_{k+1},\ldots,V_r$ are used for each B_k, $1 \leq k < r$, the total number of executions of step (13) is $\leq$ $\sum_{k=1}^{r-1}\sum_{t=k+1}^{r}(n-r+1) = \sum_{k=1}^{r-1}(r-k)(n-r+1) < r^2(n-r+1)$. Hence, $T_{13} \sim N_{13} \lesssim r^2(n-r+1)$. For step (14), $T_{14} \sim N_{14} \sim r$ and very clearly, the total time for the individual steps of the back substitution is $\lesssim T_9' + T_{13}' = nr + r^2(n-r+1) \lesssim r^2 n$.

In step (15), since U has r elements initially, $N_{15} \sim$ r+1 and $T_{15} \sim N_{15} \sim r$. Step (16) multiplies each of the elements of U by d, of which there at most r(n-r+1), and

so we have $T_{16} \sim N_{16} \lesssim r(n-r+1)$. Finally, $T_{17} \sim T_{INV}(r) \sim r$. Therefore, $T_{CRRE} \lesssim mnr + nr^2 \sim mnr$.

To show that $T_{CRRE} \gtrsim mnr$, we shall produce a matrix A in $M(m,n,GF(p))$ of rank $r \geq 1$ for which the time for step (7) is $\gtrsim mnr$. This will be the case if the k^{th} pivot is found in row k and column k (i.e., $J_A = (1,2,\ldots,r)$ and $I_A = (1,2,\ldots,m)$) and each element below the pivot element in rows $k+1,\ldots,m$ is nonzero. Then at least $(m-k+1)(n-k+1)$ arithmetic operations are required by the k^{th} pivot operation and at least $N = \sum_{k=1}^{r}(m-k+1)(n-k+1)$ arithmetic operations are required in the forward elimination. From Lemma 4.3.2 we have that $8N \geq mnr$ and so we would have $T_{CRRE} \sim N \sim mnr$.

Such an m by n matrix A can be defined as follows:

$$A(i,j) = \begin{cases} i^{j-1}, & i \leq r \\ r^{j-1}, & i > r \end{cases} \qquad (1)$$

It is easy to see, for $k = 1,2,\ldots,r$, that each upper left-hand minor of order k is a Vandermonde determinant and, hence, is nonzero; that is,

$$A\begin{pmatrix} 1,\ldots,k \\ 1,\ldots,k \end{pmatrix} = \det\left(\begin{bmatrix} 1 & 1 & 1 & . & 1 \\ 1 & 2 & 3 & . & k \\ . & . & . & . & . \\ 1 & 2^{k-1} & 3^{k-1} & \ldots & k^{k-1} \end{bmatrix}\right)$$

$$= \prod_{1 \leq s < t \leq k} (t-s) \neq 0.$$

This of course follows, provided that p does not divide $\prod_{1\le s<t\le k}(t-s)$. This is assured by the assumption in Section 3.1 that dimensions of matrices are less than any prime p on the list PRIME. For then we have that, if $p \mid \prod_{1\le s<t\le k}(t-s)$, then $p \mid (t-s)$ for some pair t,s, which is impossible, since $(t-s) < k < p$. Note that, since the initial segments of the same length of rows r,r+1,...,m are identical, all minors of the form (2) vanish for k > r.

Thus, r = rank(A), $J_A = (1,2,\ldots,r)$, and $I_A = (1,2,\ldots,m)$. Also, $\delta_k(A) = A\begin{pmatrix}1,\ldots,k\\1,\ldots,k\end{pmatrix}$, $1 \le k \le r$. Finally we show that for every pivot operation, the elements below the k^{th} pivot element in column k are nonzero. From the proof of Theorem 2.6.2 we know that these elements $B'^{(k-1)}(h,k)$, $k < h \le m$, have the form:

$$B'^{(k-1)}(h,k) = A\begin{pmatrix}i_1,\ldots,i_{k-1},i_{k-1},h\\j_1,\ldots,j_{k-1},k\end{pmatrix} \Big/ A\begin{pmatrix}i_1,\ldots,i_{k-1}\\j_1,\ldots,j_{k-1}\end{pmatrix}$$

$$= A\begin{pmatrix}1,\ldots,k-1,h\\1,\ldots,\ldots,k\end{pmatrix} \Big/ A\begin{pmatrix}1,\ldots,k-1\\1,\ldots,k-1\end{pmatrix}$$

These elements will be shown to be nonzero if the numerators are shown to be nonzero. It is not difficult to show that each minor $A\begin{pmatrix}1,\ldots,k-1,h\\1,\ldots,\ldots,k\end{pmatrix}$ is a Vandermonde determinant and, hence, is nonzero.

Thus, CRRE will require at least N arithmetic operations when applied to the matrix A defined by (1). Therefore, it has been shown that $T_{CRRE} \gtrsim mnr$, establishing dominance in

the reverse direction and so $T_{CRRE} \sim mnr.//$

4.4. An Evaluation Mapping Algorithm for Computing an RRE Form of a Matrix

We now present the middle algorithm in the complete modular algorithm for solving linear equations. This algorithm computes some RRE form for a matrix of polynomials over GF(p) or, as a special case, a matrix over GF(p). In this latter case, $\bar{\bar{C}}$ is obtained. For a matrix of polynomials over GF(p), images C* under evaluation mappings are obtained, to which the algorithm applies itself recursively to obtain RRE forms F* for the C*. The F* are, in fact, images of some RRE matrix F, assuming certain tests have been passed, which is being constructed by incremental interpolation. When a substitution test is satisfied, it is known that F is an RRE form for C. When a degree bound is attained, it is known that F has been obtained. It is shown that it is reasonable to expect that with high probability $F = \bar{\bar{C}}$.

Algorithm 4.4.1. (CPRRE) Computing an RRE Form by Evaluation and Interpolation

Input: A prime number p and an m by n nonzero matrix C over $GF(p)[x_1,\ldots,x_s]$, $s \geq 0$.

Output: The list W = (J,I,D,V) obtained by W = CPRRE(p,C), where $J = J_C, I \in \mathcal{P}_C$, and (J,D,V) is the compact representation of the RRE form F defined from C, J, and I by formula (1) in Section 4.2. C is erased.

(1) [Apply Gaussian elimination algorithm.] s ← MCPNV(C); if s > 0, go to (2); W ← CRRE(p,C); return W.

(2) [Initialize evaluation mapping algorithm.] r ← 0; a ← p; h ← 0.

(3) [Apply evaluation mapping Ψ_a.] a ← a-1; if a < 0, print error message and stop; otherwise, C* ← MCPEVL(p,C,a,s); if MZERO(C*) = 1, (erase C*; go to (3)).

(4) [Apply algorithm recursively.] W* ← CPRRE(p,C*); DECAP(J*,W*); DECAP(I*,W*); DECAP(D*,W*); DECAP(V*,W*).

(5) [Rejection tests.] r* ← LENGTH(J*); if r* < r, go to (6); if r* > r, go to (7); u ← VCOMP(J*,J); if u = 1, go to (6); if u = -1, go to (7); u ← VCOMP(I*,I); if u = 1, go to (6); if u = -1, go to (7); erase J*, I*; go to (8).

(6) [Discard Ψ_a.] Erase J*, I*, V*; if s > 1, erase D*; go to (3).

(7) [Discard previously retained evaluation mappings and initialize interpolation.] If r > 0, (erase J, I, D, V, E); J ← J*; I ← I*; D ← 0; V ← (); if V* ≠ (), V ← MZCON(V*); E ← PFA(0,PFA(1,0)); r ← r*.

(8) [Retain Ψ_a and apply interpolation.] D' ← CPINT(p,D,a,D*,E,s); if s > 1, erase D*; if V ≠ (), (V' ← MCPINT(p,V,a,V*,E,s); erase V*); T ← PFA(1,PFA(1,PFA(CDIF(p,0,a),0))); E' ← CPPROD(p,T,E); erase T,E; E ← E'; if

h = 0, go to (9); erase D; D ← D'; if V ≠ (), (erase V; V ← V'); go to (12).

(9) [Equality test.] T ← CPDIF(p,D,D'); erase D; D ← D'; u ← 1; if V ≠ (), (u ← MCPEQ(p,V,V'); erase V; V ← V'); if u = 1 and T = 0, go to (10); erase T; go to (3).

(10) [Substitution test.] n ← LENGTH(FIRST(C)); if r = n, (h ← 1; go to (11)); D' ← CPNEG(p,D); Z ← NULCON(n, J,D',V); T ← MCPMPY(p,C,Z); u ← MZERO(T); erase D',Z,T; if u = 0, go to (3); h ← 1.

(11) [Compute degree bound.] b ← DBRRE(J,C).

(12) [Degree test.] If CPDEG(E) ≤ b, go to (3).

(13) [Construct output list and return.] Erase E,C; W ← PFL(J,PFL(I,PFL(D,PFL(V,0)))); return W.

A discussion of this algorithm now follows, which will attempt to show that, for every input matrix C, CPRRE computes an output list W = (J,I,D,V), where $J = J_C, I \in P_C$, and (J,D,V) is the compact representation of the RRE form F for C defined from C,J, and I by formula (1) of Section 4.2. If C is a matrix over GF(p), the list W = $(J_C, I_C, \delta(C), V)$, where $(J_C, \delta(C), V)$ is the compact representation of $\bar{\bar{C}}$, is computed by CRRE and returned in step (1). C is altered and erased by CRRE. This establishes the case s = 0.

If C is over $GF(p)[x_1,\ldots,x_s]$, $s \geq 1$, we assume that CPRRE computes a list $W^* = (J^*,I^*,D^*,V^*)$ satisfying the output definition for every matrix C^* over $GF(p)[x_1,\ldots,x_{s-1}]$. We show in what follows that CPRRE computes a list $W = (J,I,D,V)$ satisfying the output definition. In step (1) s is computed and the evaluation-interpolation algorithm is then initialized in step (2). In particular, the variable r, representing the common rank obtained by the most recently retained evaluation mappings, is set to 0 and the flag h, which is set to 1 when a degree bound b has been computed, is set to 0.

There then follows a sequence of iterations involving various subsets of steps (3)-(12). In step (3) the image $C^* = \Psi_a(C)$ is computed, for a distinct $a \in GF(p)$, and, of course, Ψ_a is discarded as rejected evaluation mapping for C, if C^* is a zero matrix. CPRRE then applies itself recursively in step (4) to C^*. We have by the inductive assumption that a list $W^* = (J^*,I^*,D^*,V^*)$ is obtained, where $J^* = J_{C^*}$, $I^* \in \wp_{C^*}$, and (J^*,D^*,V^*) is the compact representation of an RRE form F^* for C^*. Letting $J^* = (j'_1,\ldots,j'_{r^*})$ and $I^* = (i'_1,\ldots,i'_m)$, F^* is defined by:

$$F^*(h,j) = \begin{cases} C^*\begin{pmatrix} i'_1,\ldots,i'_m \\ j'_1,\ldots,j'_{h-1},j,j'_{h+1},\ldots,j'_{r^*} \end{pmatrix}, & 1 \leq h \leq r \text{ and } 1 \leq j \leq n \\ 0, \text{ otherwise} \end{cases} \quad (1)$$

This is simply definition (1) of Section 4.2 for C*. W* is obtained, of course, barring termination of the algorithm in step (2) during the recursive application.

In step (5) Theorem 2.3.2 is applied in the rejection tests, for the induced mapping θ being either Ψ_a or one of the previously retained (and not yet discarded) evaluation mappings Ψ_{a_i}, $0 \leq i < k$. We note that each of the matrices $H = \Psi_{a_i}(C)$ have rank r, RE sequence $J = J_H$, and permutation $I \in \wp_H$ in common. Let $\hat{r} = \text{rank}(C)$. It can be inferred from Theorem 2.3.2 that exactly one of the following disjoint cases occurs:

$$\begin{aligned}
&\text{(a)} \quad r^* < \hat{r};\\
&\text{(b)} \quad r^* = \hat{r} \text{ and } J^* > J_C;\\
&\text{(c)} \quad r^* = \hat{r},\ J^* = J_C, \text{ and } I^* > I_C;\\
&\text{(d)} \quad r^* = \hat{r},\ J^* = J_C, \text{ and } I^* = I_C.
\end{aligned} \tag{2}$$

Of course, these cases hold for r,J, and I replacing r*,J*, and I*. In each of the cases (2a)-(2c), Ψ_a is a rejected evaluation mapping and should be discarded. The following cases, paralleling formulas (2a)-(2d), might be detected in step (5):

$$\begin{aligned}
&\text{(a)} \quad r^* < r;\\
&\text{(b)} \quad r^* = r \text{ and } J^* > J;\\
&\text{(c)} \quad r^* = r,\ J^* = J, \text{ and } I^* > I;\\
&\text{(d)} \quad r^* = r,\ J^* = J, \text{ and } I^* = I.
\end{aligned} \tag{3}$$

If (3a) is detected, then $r^* < \hat{r}$, since $r \leq \hat{r}$. If (3b) is detected, two possible subcases occur: $r = \hat{r}$ or $r < \hat{r}$. If $r = \hat{r}$, then $J \geq J_C$ and so $J^* > J_C$. If $r < \hat{r}$, then $r^* < \hat{r}$. If (3c) is detected, two subcases occur: i) $r = \hat{r}$ and $J = J_C$, or ii) $r < \hat{r}$ or $J > J_C$. If i) occurs, then $I \geq I_C$ and so $I^* > I_C$. If ii) occurs, then either $r^* < \hat{r}$ or $J^* > J_C$. In each of cases (3a)-(3c), Ψ_a is found to be rejected and is discarded, with the consequent erasures of J*,I*,D*, and V* in step (6). If case (3d) is detected, Ψ_a is retained, although it may be shown in a subsequent iteration to be rejected along with the other retained evaluation mappings. In case (3d), transfer is to step (8) for the interpolation step.

Three other cases can occur and may be detected in step (5). These are given by cases (3a)-(3c), if we merely reverse the roles of r* and r, J* and J, and I* and I. In these cases we find, using arguments similar to those above, that the previously retained evaluation mappings Ψ_{a_i} are rejected mappings for C. The Ψ_{a_i} are then discarded, with the result that in step (7) J,I,D,E, and V are erased and replaced by J*, I*, and the zero of $GF(p)[x_1,\ldots,x_{s-1}]$, the univariate polynomial 1, and a non-diagonal part with the same structure as V* but consisting of all zeros, respectively. Also, r is replaced by r*. This constitutes a re-initiali-

zation of the interpolation. We note that when a nonzero C* is first obtained in step (3), r* > r = 0 is found in step (5), whereupon step (7) performs the first initialization of the interpolation.

Thus, whenever the new evaluation mapping Ψ_a is retained, whether or not the previously retained Ψ_{a_i} are discarded, the interpolation in step (8) is performed. This involves applying CPINT to D and D* and, if V and V* are non-null, MCPINT to V and V* to obtain new interpolants D' and V', respectively. Also, the univariate polynomial E(x) is replaced by $(x-a)\cdot E(x)$.

Let G be the RRE matrix for which (J,D,V) is the compact representation. Suppose the matrix F of formula (1) of Section 4.2 were defined using $J = (j_1', \ldots, j_r')$. Then the matrices F* as defined by formula (1) above, where r* = r, satisfy: $F^* = \Psi_{a_i}(F)$, letting $C^* = \Psi_{a_i}(C)$. Hence, the F*'s are the values used in the sequence of interpolations of step (8) for constructing F, and G is (in its compact form) the most recent interpolant. In step (8) the compact representation (J,D',V') of the next interpolant G' is obtained. Thus, G and G' are the k^{th} and $(k+1)^{th}$ interpolants, respectively.

If in step (8) flag h = 1, the degree bound b has been previously obtained and, after redefining D and V as D' and

V', transfer is made directly to step (12) to test if this bound has been attained. If h = 0, the equality of D and D' and of V and V' are tested in step (9). If one or the other pair are not equal (i.e., $G \neq G'$), transfer is made to step (3) to begin another iteration. In any case, D' and V' replace D and V, respectively, and D and V are erased.

By Theorem 2.5.6, we know that there are a finite number of rejected evaluation mappings for C. Hence, it is assured that eventually sufficiently many evaluation mappings will be retained to produce successive interpolants which are equal. For example, if in the worst case all rejected evaluation mappings had been applied and later discarded, then retaining accepted mappings would eventually result in $G = G' = \bar{\bar{C}}$ being obtained. It is possible, however, that equality could occur with all retained mappings being rejected mappings. Thus, the equality test is guaranteed to succeed at some time, whereupon the substitution test will be applied in step (10).

If in step (10) it is found that r = n, then r = r' = n, since $r \leq r' \leq n$. In this case it is known that $J = J_C$, since $J = (1,2,\ldots,n) \leq J_C$ and $J \geq J_C$, and so h is set to 1 and transfer is made to compute the degree bound. If $r < n$, then the matrix Z satisfying formula (1) of Section 4.1 can be computed from n,J,-D, and V by NULCON. By Theorem

4.1.4, if T = CZ is a zero matrix, then we know that J = $J_G = J_C$ and the substitution test is successful. In this case, h is set to 1 and the degree bound b is computed in step (11). If the substitution test fails, that is, if CZ is non-zero, then transfer is made to step (3) to begin another iteration. We note that the matrix product CZ is computed by the evaluation mapping algorithm MCPMPY.

As for the equality test, it is assured that eventually the substitution test will be satisfied. For, if in the worst case, the equality test resulted in G = G' = $\bar{\bar{C}}$, then by Corollary 4.1.2 the matrix Z constructed by NULCON from n, J = J_C, -D = -δ(C), and V would result in CZ being a zero matrix.

In step (11) DBRRE is applied to J = J_C = $(j_1,\ldots,j_r)$ and to C to compute a bound b on the degrees in x_s of the elements of every matrix F defined by formula (1) of Section 4.2. This includes the particular matrix F being constructed by interpolation in step (8). We note that h is set to 1 and the bound b is computed only once for C. Thereafter, during subsequent iterations transfer is directly from step (8) following the interpolation to step (12) to test the degree bound. It may happen that all retained mappings will be discarded following the computation of b. In this case, the interpolation is begun again and the first sequence of

b+1 retained evaluation mappings will produce an RRE form F for C. Note that E(x) > b is required for return in step (12). Suppose k+1 iterations have been performed. Then the maximum degree attainable by the elements of G is k. However, deg(E(x)) = k+1. So, to assure that the maximum degree attainable is k = b, we must require that deg(E(x)) = b+1 > b. Until this criterion is satisfied, transfer is back to step (3). When it is satisfied, the list W is constructed and returned in step (13) and E and C are erased.

This concludes the inductive step in an informal verification of CPRRE. Although a formal proof of the algorithm has not been intended, the discussion should be sufficient to show that the output description is correct.

We now consider what would be the most likely behavior of CPRRE for the input matrix C consisting of randomly chosen entries from $GF(p)[x_1,\ldots,x_s]$, more specifically, from $P^*(m_1,\ldots,m_s)$, $s \geq 1$. A rigorous treatment of those parts of the discussion relying on the properties and implications of randomness will not be attempted. In such cases the asserted properties seem plausible; but attempts to establish them rigorously would probably be quite difficult and would distract from our main concern here. We first sketch an argument showing that the probability of the occurrence of a rejected evaluation mapping is small.

Let $\delta_k(C) = C\begin{pmatrix} i_1,\ldots,i_k \\ j_1,\ldots,j_k \end{pmatrix}$, $1 \le k \le r$. From Theorem 2.5.6 we know that an upper bound on the number of rejected evaluation mappings for C is given by $\alpha = \sum_{k=1}^{r} \alpha_k$, where $\alpha_k = \deg(\delta_k(C))$. By applying Lemma 2.5.1 to each $\delta_k(C)$, we find that $\alpha_k \le \sum_{u=1}^{k} \max_{1\le v\le k} \deg(C(i_u,j_v)) \le \sum_{u=1}^{k} m_s = km_s < k\mu_s$, for $1 \le k \le r$. Thus, $\alpha < \sum_{k=1}^{r} k\mu_s = r(r+1)\mu_s/2 \le r^2 \mu_s$, giving a crude upper bound on the number of rejected evaluation mappings which could be chosen. The crudeness derives from the fact that each $\delta_k(C)$ has been considered to have roots only in GF(p), from which the elements a for the mappings Ψ_a are selected.

By interchanging each of $x_1,\ldots,x_{s-1}$ with the main variable x_s and applying Lemma 2.5.1 as above, we find that $\deg_{x_i}(\delta_k(C)) \le km_i$, $1 \le i \le s$. Hence, $\delta_k(C) \in P^*(km_1,\ldots, km_s)$, for $1 \le k \le r$. We assert as plausible that, since each element of C is randomly chosen from $P^*(m_1,\ldots,m_s)$, each determinant $\delta_k(C)$ possesses similar properties of being randomly generated in $P^*(km_1,\ldots,km_s)$. It then is reasonable to expect that any element of GF(p) has equal likelihood of being a root of $\delta_k(C)$, for each k. Hence, each of the p mappings Ψ_a has equal likelihood of being a rejected evaluation mapping for C. Thus, a crude upper bound on the probability that a rejected evaluation mapping occurs is $\zeta = r^2\mu_s/p > P(\delta_k(a) = 0 \mid a \in GF(p) \text{ and } 1 \le k \le r)$.

Recall from Section 3.1 that dimensions of matrices and degree bounds of polynomials have been assumed to be less than any prime p on the list PRIME. It is, in fact, true that these quantities are much smaller than p, where typically $p \geq 10^{10}$. Thus, $r << p$ and $r^2 << p$ and also $\mu_s << p$, whereupon it is still reasonable to expect that $\zeta << 1$. Thus, the probability that a rejected evaluation mapping occurs is small. This implies that with high probability $J = J_C$ and $I = I_C$ are computed for all mappings Ψ_a employed. Hence, with high probability $F = \bar{\bar{C}}$ is being constructed by incremental interpolation.

Another aspect of the expected behavior of CPRRE concerns the equality test in step (9). Recall that the equality test succeeds when $D = D'$ and the corresponding elements of V and V' are equal (i.e., successive polynomial interpolants are equal). Suppose that k consecutive evaluation mappings $\Psi_{a_0}, \Psi_{a_1}, \ldots, \Psi_{a_{k-1}}$ have been retained and let A_{k-1} be some specific k^{th} interpolant obtained (i.e., $A_{k-1} = D$ or A_{k-1} an element of V). Given that B_{k-1} is the next value obtained, the next interpolant A_k is obtained by the formula:

$$A_k(x_1,\ldots,x_s) = \{[B_k(x_1,\ldots,x_{s-1}) - A_{k-1}(x_1,\ldots,x_{s-1},a_k)] / E(a_k)\}.E(x_s) + A_{k-1}(x_1,\ldots,x_s) \quad (4)$$

Let $f(x_s) = B_k(x_1,\ldots,x_{s-1}) - A_{k-1}(x_1,\ldots,x_{s-1},x_s)$, that is, a polynomial in x_s with coefficients in $GF(p)[x_1,\ldots,x_{s-1}]$.

Then $A_k = A_{k-1}$ if and only if $f(a_k) = 0$. Since f has degree at most k-1, f has at most k-1 roots and, hence, at most k-1 roots in GF(p). Assuming that each element of GF(p) has equal likelihood of being a root of f, the probability that a_k is a root of f, and hence that $A_k = A_{k-1}$, is at most k/p.

We now can obtain a single bound on the probability that the equality test in step (9) succeeds prematurely. We saw above that with high probability the compact representation of $F = \bar{\bar{C}}$ is being constructed by the interpolation. Since the polynomial elements of this representation are minors of order r, it can be shown by the same argument as that used above that these polynomials are in $P^*(rm_1,\ldots,rm_s)$. The bound b computed in step (11) by DBRRE satisfies $b \leq rm_s$, since it is computed in such a way as to be attainable. Thus, $r\mu_s$ is an upper bound on the number of iterations of the interpolation. If t is the degree of a particular polynomial element R of $F = \bar{\bar{C}}$ being constructed by interpolation, then the probability of two successive interpolants being equal before R has been obtained is $P(A_k = A_{k-1} \mid 1 \leq k \leq t) \leq \sum_{k=1}^{t}(k/p) = t(t+1)/2p \leq t^2/p < (r\mu_s)^2/p = \zeta'$. Since the compact representation has at least one polynomial element, this is an upper bound on the probability that the equality test succeeds prematurely. Since $r << p$ and $\mu_s << p$, it is still reasonable to expect that $(r\mu_s)^2 << p$, which implies

$\zeta' << 1$.

To illustrate further the unlikelihood of a premature equality test success occurring, suppose t is the maximum degree in x_s among all elements of $F = \bar{\bar{C}}$ and suppose there are z polynomial elements in the compact representation of F. Let t' be the minimum of their degrees. The probability that the equality of successive interpolants occurs at the k^{th} iteration, $k \leq t'$ for all z polynomials is at most $(k/p)^z$. Thus, the probability of a premature equality test success before the $(t'+1)^{th}$ iteration is $\leq \sum_{k=1}^{t'} (k/p)^z \leq \sum_{k=1}^{t} (k/p)^z \leq \left[\sum_{k=1}^{t} (k/p)\right]^z = (\zeta')^z$, which is indeed much smaller than ζ'. This implies that the probability of a premature equality occurring at all is some value much less than $\zeta' = (r\mu_s)^2/p$, when more than one polynomial interpolation is involved.

Consider now the matter of obtaining computing time dominance relations for CPRRE. Suppose dominance relations were desired for the strict maximum or average computing time functions, as defined in Section 3.1. Then those input matrices C would have to be accounted for, for which a large number of rejected evaluation mappings exist, possibly close to $r^2\mu_s$. Moreover, matrices C, for which all possible premature equality test successes and the resulting substitution tests might occur, would also have to be included.

Such premature substitution tests would most likely fail, thus requiring further tests. Unusual cases such as these would be relatively few; however, their computing times $t_{CPRRE}(p,C)$ could be extremely large.

The effect of the possible existence of such cases on a maximum computing time analysis of CPRRE is to cause the dominating functions to be exceedingly large. Again, if an average computing time analysis were to be performed, the computing times for all extreme cases would have to be included in computing the sum: $\Sigma_{(p,C)\in S_y} t_{CPRRE}(p,C)$, or a bound for this sum. Here, for each $y = (m,n,r,m_1,\ldots,m_s)$, S_y is the set of all input pairs (p,C), where $C \in M(m,n,P^*(m_1,\ldots,m_s))$ and $r = rank(C) \geq 1$.

It seems likely however, that, in view of the very small probability with which these extreme cases occur, the expected performance of the algorithm will not be seriously affected by these cases. Hence, what will be obtained for CPRRE are dominance relations on the maximum computing times, under the assumption that no rejected evaluation mappings are used and no premature equality test successes occur. Then we can expect that $F = \bar{\bar{C}}$ is being constructed by the interpolation and that the first equality test success occurs when $J = J_C$, $I = I_C$, $D = \delta(C)$, and (J,D,V) is the compact representation of $\bar{\bar{C}}$. Subsequent iterations will then

produce the same interpolants until the bound b is attained.

We note that a similar analysis and related assumptions were made in [BRO71] for an evaluation-interpolation algorithm for multivariate GCD calculation.

The following elementary result will be convenient to have in the computing time analysis of CPRRE.

Lemma 4.4.2. Let r and n be integers such that $1 \leq r < n$. Then $n \leq 2r(n-r)$.

Proof: If $r \leq n/2$, then $r(n-r) \geq r(n-n/2) = rn/2 \geq n/2$. If $r > n/2$, then $r(n-r) \geq (n/2)(n-r) \geq n/2$.//

Theorem 4.4.3. Let $T_{CPRRE}(m,n,r,\bar{m}_s)$ be the maximum computing time of CPRRE under the assumption that no rejected evaluation mappings are used and every substitution test is successful, for C a nonzero matrix of rank r in $M(m,n,P^*(\bar{m}_s))$, $s \geq 0$. Then $T_{CPRRE} \sim mnr$, for $s = 0$, and $T_{CPRRE} \sim r^s(\Pi_{i=1}^{s}\mu_i)[mn^2 + (m+n)(n-r+1)\, r(\Sigma_{i=1}^{s}\mu_i)]$, for $s > 0$.

Proof: If C is over GF(p), only step (1) is executed. Thus, $T_{CPRRE}(m,n,r,\bar{m}_o) \sim T_{MCPNV}(m,n) + T_{CRRE}(m,n,r) \sim mn + mnr \sim mnr$. Suppose C is over $GF(p)[x_1,\ldots,x_s]$, $s \geq 1$. The following maximum computing time table (under the given assumptions) is obtained for CPRRE.

i	N_i	$T_{i,j}$	T_i
1	1	-	mn
2	1	-	1
3	$r\mu_s$	$mn(\Pi_{i=1}^{s}\mu_i)$	$mnr\mu_s(\Pi_{i=1}^{s}\mu_i)$
4	$r\mu_s$	$T_{CPRRE}(m,n,r,\bar{m}_{s-1})$	$r\mu_s \cdot T_{CPRRE}(m,n,r,\bar{m}_{s-1})$
5	$r\mu_s$	m	$mr\mu_s$
6	0	-	0
7	1	-	$r(n-r) + 1$
8	$r\mu_s$	$(r(n-r)+1)r^{s-1}j(\Pi_{i=1}^{s-1}\mu_i)$	$(r(n-r)+1)r^{s+1}\mu_s(\Pi_{i=1}^{s}\mu_i)$
9	$r\mu_s$	$(r(n-r)+1)r^{s-1}j(\Pi_{i=1}^{s-1}\mu_i)$	$(r(n-r)+1)r^{s+1}\mu_s(\Pi_{i=1}^{s}\mu_i)$
10	1	-	$(n-r+1)r^{s}(\Pi_{i=1}^{s}\mu_i)\cdot[mn + (m+n)r(\sum_{i=1}^{s}\mu_i)]$
11	1	-	mn
12	$r\mu_s$	1	$r\mu_s$
13	1	-	$mn(\Pi_{i=1}^{s}\mu_i)$

Selected explanations of the times obtained for the individual steps is now given. For step (1), $T_1 \sim T_{MCPNV}(m,n) \sim mn$. Step (2) is trivial. It was noted above that the bound b computed in step (11) satisfies: $b < r\mu_s$. Thus, $r\mu_s$ bounds the number of executions of steps (3)-(5), (8), (9), and (12), since b+1 executions of steps (3)-(5), and (8) occur, and at most b executions of each of steps (9) and (12). We note that C* is never zero in step (3), since Ψ_a

has been assumed to be accepted. In step (5) we always have $r^* = r$, $J^* = J$, and $I^* = I$ and so step (8) is always executed next, except for the first execution for which $r^* > r = 0$ and step (7) is next. Step (9) is not executed after the degree bound has been computed, which may not occur until just after the $(b+1)^{th}$ execution of step (9). The degree test in step (12) is executed after every interpolation once the degree bound has been computed. Thus, $N_i \lesssim r\mu_s$, for $i = 3,4,5,8,9,12$. Step (6) is not executed, given the assumption that no rejected mappings occur, and similarly the only execution of step (7) is to initialize the interpolation for the first iteration. Since it has been also assumed that no substitution test failures occur, $N_{10} = 1$. Also, only one bound b is computed and so $N_{11} = 1$. Of course, $N_{13} = 1$. Thus, all the N_i' have been established and we now turn to the $T_{i,j}$ and the T_i.

Clearly, $T_{3,j} \sim T_{MCPEVL}(m,n,\bar{m}_s) \lesssim mn(\Pi_{i=1}^{s}\mu_i)$, giving T_3' easily. Since a dominating function is not yet known for T_{CPRRE}, the function name must be used for step (4). For step (5), $T_{5,j} \lesssim T_{LENGTH}(r^*) + T_{VCOMP}(r^*,r) + T_{VCOMP}(m,m)$ $\sim r^* + r^* + m \sim m$. For step (7), the largest $T_{7,j}$ would occur when MZCON is applied to a non-diagonal part V* with the maximum number of elements. In Theorem 4.3.2 this was found to be $r(n-r)$, when $r < n$. Thus, $T_7 \sim T_{7,j} \lesssim r(n-r) + 1$. In

step (8) at most $r(n-r)+1$ polynomial interpolations occur: at most $r(n-r)$ when applying MCPINT to V and V* and one for D and D*. For the j^{th} iteration, D and the elements of V have degree at most j-1 in x_s and degree at most $r\mu_i$ in the variable x_i, $i < s$. This latter fact follows, since each compact form (J*,D*,V*) computed in step (4) represents a matrix in $M(m,n,P^*(rm_1,\ldots,rm_{s-1}))$. Thus, $T_{8,j} \lesssim T_{CPINT}(rm_1,\ldots,rm_{s-1},j) + T_{MCPINT}(r,n-r,rm_1,\ldots,rm_{s-1},j) \lesssim (r(n-r)+1)j(\Pi_{i=1}^{s-1} r\mu_i) = (r(n-r)+1)r^{s-1}j(\Pi_{i=1}^{s-1}\mu_i)$, which clearly dominates the time for the erasures and the application of CPPROD. This assumes that $r < n$; but the relation clearly holds for $r = n$. Thus, $T_8 \lesssim \sum_{j=1}^{r\mu_s} T'_{8,j} \leq \sum_{j=1}^{r\mu_s}(r(n-r)+1)r^{s-1}(\Pi_{i=1}^{s-1}\mu_i) \leq (r(n-r)+1)r^{s+1}\mu_s(\Pi_{i=1}^{s}\mu_i)$. We find that for step (9) $T_{9,j} \lesssim T_{CPDIF}(rm_1,\ldots,rm_{s-1},j-1,rm_1,\ldots,rm_{s-1}) + T_{MCPEQ}(r,n-r,rm_1,\ldots,rm_{s-1},j-1,rm_1,\ldots,rm_{s-1},j) \lesssim T'_{8,j}$, assuming $r < n$; but $T_{9,j} \lesssim T'_{8,j}$ for $r = n$ also. Thus, similar to step (8), $T_9 \lesssim T'_8$. Skipping step (10) for the moment, we see that $T_{11} \sim T_{DBRRE}(m,n) \sim mn$. T_{12} is trivial and $T_{13} \lesssim mn(\Pi_{i=1}^{s}\mu_i)$, the function required to dominate the time to erase C.

We now return to step (10). The time to construct D' is $\lesssim T_{CPNEG}(rm_1,\ldots,rm_s) \lesssim r^s(\Pi_{i=1}^{s}\mu_i)$ and the time to construct Z is $T_{NULCON}(n,r) \sim n(n-r)$. We have $Z \in M(n,n-r,P^*(rm_1,\ldots,rm_s))$. Thus, the time to compute CZ is $\lesssim \hat{T} = T_{MCPMPY}(m,n,n-r,m_1,\ldots,m_s,rm_1,\ldots,rm_s) \lesssim mn(n-r)(\Pi_{i=1}^{s}(\mu_i+r\mu_i)) +$

$mn\{\sum_{i=1}^{s}(\prod_{j=1}^{i}\mu_j)(\prod_{j=i}^{s}(\mu_j+r\mu_j))\} + n(n-r)\{\sum_{i=1}^{s}(\prod_{j=1}^{i}r\mu_j)(\prod_{j=i}^{s}(\mu_j+r\mu_j))\} + m(n-r)\{(\prod_{i=1}^{s}(\mu_i+r\mu_i))(\sum_{i=1}^{s}(\mu_i+r\mu_i))\}$. Since $\mu_j + r\mu_j \sim r\mu_j$, $\hat{T} \lesssim mn(n-r)r^s(\prod_{i=1}^{s}\mu_i) + mnr(\prod_{j=1}^{s}\mu_j)(\sum_{i=1}^{s}\mu_i r^{s-i}) + n(n-r)r^{s+1}(\prod_{i=1}^{s}\mu_i)(\sum_{i=1}^{s}\mu_i) + m(n-r)r^{s+1}(\prod_{i=1}^{s}\mu_i)(\sum_{i=1}^{s}\mu_i) = (\prod_{i=1}^{s}\mu_i)\cdot[mn(n-r)r^s + mnr(\sum_{i=1}^{s}\mu_i r^{s-i}) + (m+n)(n-r)r^{s+1}(\sum_{i=1}^{s}\mu_i)]$. Applying Lemma 4.4.2, we see that $mnr(\sum_{i=1}^{s}\mu_i r^{s-i}) \lesssim m(n-r)r^2(\sum_{i=1}^{s}\mu_i r^{s-i}) \lesssim (m+n)(n-r)r^{s+1}(\sum_{i=1}^{s}\mu_i)$. Thus, $\hat{T} \lesssim (\prod_{i=1}^{s}\mu_i)\cdot[mn(n-r)r^s + (m+n)(n-r)r^{s+1}(\sum_{i=1}^{s}\mu_i)] = (n-r)r^s(\prod_{i=1}^{s}\mu_i)\cdot[mn + (m+n)r(\sum_{i=1}^{s}\mu_i)] = \hat{T}'$. We note that $\hat{T}' \lesssim \hat{T}$ and that, if we replace $(n-r)$ by $(n-r+1)$ in $\hat{T}'$, this relation is not changed. Clearly, $\hat{T}'$ dominates the time for CPNEG, NULCON, MZERO, the erasures, and all other operations in step (10). Hence, $T_{10} \lesssim \hat{T}' = T'_{10}$.

We now obtain a single dominance relation for the total time for all steps. It is easily seen that $T'_i \lesssim T'_3$, for $i = 1,2,5,6,7,11,12,13$. Moreover, $T'_3 = mnr\mu_s(\prod_{i=1}^{s}\mu_i) \lesssim (m+n)r(n-r+1)r^s(\sum_{i=1}^{s}\mu_i)(\prod_{i=1}^{s}\mu_i) \lesssim T'_{10}$, since $n \lesssim r(n-r+1)$. Also, $T'_8 + T'_9 \lesssim (n-r+1)r^{s+2}\mu_s(\prod_{i=1}^{s}\mu_i) \lesssim (n-r+1)(m+n)r^{s+1}(\sum_{i=1}^{s}\mu_i)(\prod_{i=1}^{s}\mu_i) \lesssim T'_{10}$. Thus, we have $T_{CPRRE} \lesssim T'_4 + T'_{10}$.

For $s = 1$, $T_{CPRRE}(m,n,r,\bar{m}_1) \lesssim r\mu_1\cdot T_{CPRRE}(m,n,r,\bar{m}_o) + T'_{10} \sim r\mu_1\cdot mnr + (n-r+1)r\mu_1[mn + (m+n)r\mu_1] = mnr\mu_1(r+(n-r+1)) + (m+n)(n-r+1)r^2\mu_1^2 \sim mn^2r\mu_1 + (m+n)(n-r+1)r^2\mu_1^2$. Let the inductive hypothesis be: $T_{CPRRE}(m,n,r,\bar{m}_t) \lesssim mn^2r^t(\prod_{i=1}^{t}\mu_i) + (m+n)(n-r+1)r^{t+1}(\prod_{i=1}^{t}\mu_i)(\sum_{i=1}^{t}\mu_i)$, for $t \geq 1$. Clearly, the

hypothesis holds for t = 1. Assuming that it holds for t = s-1, where s > 1, we show that it holds for s. For

$$T_{CPRRE}(m,n,r,\bar{m}_s) \lesssim r\mu_s \cdot T_{CPRRE}(m,n,r,\bar{m}_{s-1}) + T'_{10} \lesssim r\mu_s \cdot \{ mn^2 r^{s-1} (\Pi_{i=1}^{s-1} \mu_i) + (m+n)(n-r+1) r^s (\Pi_{i=1}^{s-1} \mu_i)(\Sigma_{i=1}^{s-1} \mu_i) \} + (n-r+1) r^s (\Pi_{i=1}^{s} \mu_i) \cdot \{ mn + (m+n) r (\Sigma_{i=1}^{s} \mu_i) \} = r^s (\Pi_{i=1}^{s} \mu_i) \cdot \{ mn^2 + (m+n)(n-r+1) r (\Sigma_{i=1}^{s-1} \mu_i) + (n-r+1) mn + (n-r+1)(m+n) r (\Sigma_{i=1}^{s} \mu_i) \} \sim r^s (\Pi_{i=1}^{s} \mu_i) [mn^2 + (m+n)(n-r+1) r (\Sigma_{i=1}^{s} \mu_i)] .//$$

Thus, the time for the substitution test nearly dominates the time for the remaining steps, were it not for the mn^2-term in brackets, contributed by the recursive application of CPRRE. The probability that an accepted evaluation mapping occurs at the first execution of step (3) is at least $1 - \alpha/\rho = (\rho-\alpha)/\rho$, which is close to 1. So, with high probability any rejected mappings occur after the first iteration, are discarded immediately, and no interpolations are wasted on such mappings. Moreover, the probability that a succession of t rejected evaluation mappings are retained, with the required interpolations being performed, is some value less than $[\alpha/\rho]^t$, an extremely small number. Thus, it appears that, even if rejected mappings do occur, the time for the computation required for applying such mappings will not be significant.

4.5. A Modular Algorithm for Solving Linear Equations

In this section the outer or main algorithm for computing a general solution to a system of linear equations AX = B with integer or polynomial coefficients is presented. If the system is inconsistent, this fact is detected and reported. There is a clear similarity between this algorithm and CPRRE, which will be detailed below. The basic method is to compute images C* under mod-p mappings of the augmented system matrix C = (A,B), to which algorithm CPRRE is applied to obtain RRE forms F* for the C*. The F* are images of some RRE matrix F, which is being constructed from these images by Garner's Method, assuming certain tests have been satisfied. When at some point a substitution test is satisfied, F is known to be an RRE form for C with RE sequence $J = J_C$. Using J, it can be decided whether or not system C is consistent. If it is, then a general solution (D,Y,Z) to the system AX = B is constructed. As was done for CPRRE, arguments are sketched which make it reasonable to expect that with high probability $F = \bar{\bar{C}}$ is obtained and, hence, that the determinantal general solution (δ(A),Y,Z) is computed. The complete modular algorithm now follows.

Algorithm 4.3.5. (PLES) Computing a General Solution to a System of Linear Equations

Input: A positive integer n and an m by n' matrix C over

$I[x_1,\ldots,x_s]$, $s \geq 0$, where $n < n'$. C is the augmented matrix for a linear system $AX = B$, where A is m by n and nonzero and B is m by q.

Output: A list G obtained by G = PLES(n,C). If the system C is consistent, G = (D,Y,Z), representing a general solution to the system AX = B. If the system is inconsistent, G = (), the null list.

Note that this algorithm requires the list PRIME of distinct odd primes.

(1) [Initialize.] X ← MVLIST(C); L ← PRIME; r ← 0.

(2) [Apply mod-p mapping φ_p.] If L = (), print error message and stop; otherwise, ADV(p,L); C* ← MMOD(p,C,s); if MZERO(C*) = 1, (erase C*; go to (2)).

(3) [Compute an RRE form F* for C* by the evaluation mapping algorithm.] W* ← CPRRE(p,C*); DECAP(J*,W*); DECAP(I*,W*); DECAP(D*,W*); DECAP(V*,W*).

(4) [Rejection tests.] r* ← LENGTH(J*); if r* < r, go to (5); if r* > r, go to (6); u ← VCOMP(J*,J); if u = 1, go to (5); if u = -1, go to (6); u ← VCOMP(I*,I); if u = 1, go to (5); if u = -1, go to (6); erase J*,I*; go to (7).

(5) [Discard φ_p.] Erase J*,I*; if X ≠ (), erase D*; if V* ≠ (), erase V*; go to (2).

(6) [Discard previously retained mod-p mappings and initialize interpolation.] If r > 0, (erase J,I,D,V,

E); J ← J*; I ← I*; D ← 0; V ← (); if V* ≠ (), V ← MZCON(V*); E ← PFA(1,0); r ← r*.

(7) [Retain φ_p and apply Garner's Method.] D' ← CPGARN(E, D,p,D*,X); if X ≠ (), erase D*; if V ≠ (), (V' ← MGARN(E,V,p,V*,X); erase V*); T ← PFA(p,0); E' ← IPROD(T,E); erase T,E; E ← E'.

(8) [Equality test.] T ← PDIF(D,D'); erase D; D ← D'; u ← 1; if V ≠ (), (u ← MEQ(V,V'); erase V; V ← V'); if u = 1 and T = 0, go to (9); erase T; go to (2).

(9) [Substitution test.] If r > n, (G ← (); erase D; go to (12)); n' ← LENGTH(FIRST(C)); D' ← PNEG(D); Z' ← NULCON(n',J, D',V); T ← MMPY(C,Z'); u ← MZERO(T); erase D',T; if u = 0, (erase Z'; go to (2)).

(10) [Inconsistent system test.] J ← INV(J); if FIRST(J) > n, (erase Z'; G ← (); erase D; go to (12)); Y ← (); Z ← (); h ← n-r-1; u ← n.

(11) [Construct general solution.] u ← u-1; DECAP(T,Z'); if h ≥ 0, (Z ← PFL(T,Z); if h > 0, do T ← TAIL(T) h times; W ← T; T ← TAIL(T); SSUCC(0,W)); Y ← PFL(T,Y); if u > 0, go to (11); erase Z'; G ← PFL(D,PFL(INV(Y),PFL(INV(Z),0))).

(12) [Final erasures and return.] Erase J,I,V,E,X; return G.

An explanation of the individual steps is now given, which is intended to show that the list G, as defined in the

output description, is obtained for every valid input pair (n,C). In step (1) the initialization obtains the list X = $(x_1,\ldots,x_s)$ of variables, if C is over $I[x_1,\ldots,x_s]$, $s \geq 1$, or X = (), if C is over I. Also, L is set to the list of primes and the variable r, giving the common rank obtained by the most recently retained mod-p mappings, is set to 0. A sequence of iterations then follow involving various subsets of steps (2)-(9). These steps are a direct parallel with steps (3)-(10) of CPRRE, the main differences due to replacing evaluation mappings by mod-p mappings, incremental interpolation by Garner's Method, and, in general, operations in $GF(p)[x_1,\ldots,x_s]$ by operations in $I[x_1,\ldots,x_s]$. Hence, much of the discussion of CPRRE pertains directly to PLES.

In step (2) the next prime p is obtained and the mod-p mapping φ_p applied to obtain an image matrix $C^* = \varphi_p(C)$. Of course, if C* is zero, then φ_p is discarded as a rejected mapping, C* is erased, and the next mod-p mapping selected. If the list PRIME is insufficiently long and is exhausted, the algorithm terminates in failure. However, since another odd prime can in theory always be found, this is an implementation problem which can be corrected and does not invalidate PLES as an algorithm. In step (3) C* is input to CPRRE,

which returns the list W* = (J*,I*,D*,V*), where (J*,D*V*) is the compact representation of an RRE form F* for C* and I* ϵ $\wp_{C*}$. The matrix F* is as defined by formula (1) of Section 4.4 from C*,J* = $(j'_1,\ldots,j'_{r*})$, and I* = $(i'_1,\ldots,i'_m)$. This was shown by the discussion of CPRRE. The elements J*, I*,D*, and V* are removed from W*.

In step (4) tests for rejecting φ_p, or the previously retained (and not yet discarded) mod-p mappings φ_{p_i}, $1 \le i \le k$, are applied. Each of the matrices H = φ_{p_i}(C) have rank r, RE sequence J = J_H, and permutation I ϵ $\wp_H$ in common. These rejection tests are identical to those of CPRRE and, hence, the discussion of steps (5)-(7) of CPRRE may be applied without any significant change to steps (4)-(6) of PLES. In so doing, φ_p replaces Ψ_a, φ_{p_i} replaces Ψ_{a_i}, $I[x_1,\ldots,x_s]$ replaces $GF(p)[x_1,\ldots,x_s]$, observing that the case s = 0 is now included, and Garner's Method replaces incremental interpolation. We note that, in initializing (or re-initializing) Garner's method in step (6) of PLES, the L-integer E = 1 is defined in place of the univariate polynomial $E = 1x^0$ over GF(p) required in CPRRE.

The net result is that Garner's Method is applied in step (7), whenever the new mod-p mapping φ_p is retained, whether or not the previously retained φ_{p_i} are discarded. Thus, by applying CPGARN to D and D* and, if V and V* are

non-null, MGARN to V and V*, new iterates D' and V' are obtained. In addition, the L-integer $E = p_1 \cdots p_k$ is replaced by $p \cdot E$. If the ω_{p_i} have been retained, $p = p_{k+1}$ and $E = p_1 \cdots p_{k+1}$ and, if the ω_{p_i} have been discarded, $p = p_1$ and $E = p_1$.

Let G be the RRE matrix for which (J,D,V) is the compact representation. If F is the m by n matrix as defined by formula (1) of Section 4.2, using $J = (j_1', \ldots, j_r')$, then each RRE form F* for a $C^* = \omega_{p_i}(C)$ satisfies: $F^* = \omega_{p_i}(F)$. Thus, the F*'s are part of a modular representation of F which are being used in a sequence of iterative applications of Garner's Method to construct F and G is the most recent iterate (i.e., its compact representation). In step (7) the compact representation (J,D',V') of the next iterate G' is computed. G and G' are then the k^{th} and $(k+1)^{th}$ iterates, respectively.

The equality of these iterates is tested next in step (8) by comparing their compact representations: D with D' and V with V'. If one or the other pair are unequal, step (2) then follows to begin another iteration. Regardless of the outcome, D and V are erased and are replaced by D' and V', respectively. We are assured by Theorem 2.4.8 that there are a finite number of rejected mod-p mappings for C. Hence, if equality does not occur for all retained mappings being

rejected mappings for C, then it will occur for all retained mappings being accepted mappings. For each accepted mapping φ_p produces $F^* = \overline{\overline{\varphi_p(C)}} = \varphi_p(\bar{\bar{C}})$, from which $F = \bar{\bar{C}}$ will eventually be constructed, resulting in G = G', if the iterations are not terminated sooner. Thus, at some time the substitution test will be applied in step (9).

In step (9) the test r > n is first applied. If it succeeds, the system C is inconsistent, since rank(A) ≤ n < r = rank(G) ≤ rank(C). In this case no substitution test is required and so G is defined as the null list and returned in step (12). If r ≤ n, then r < n' and the matrix Z' can be constructed from n',J,-D, and V by NULCON, satisfying formula (1) of Section 4.1. If T = CZ' is a nonzero matrix, the substitution test fails and so Z' is erased and the next iteration is begun in step (2). If CZ is a zero matrix (i.e., the test succeeds), then by Theorem 4.1.4, $J \neq J_C$. If J[r] > n, then the submatrix of C consisting of its first J[r] columns has rank greater than A and so rank(C) = r > rank(A). In fact, this condition is also necessary for the system C to be inconsistent. Hence, this test is applied in step (10) to eliminate all inconsistent systems. In this case, Z' is erased, G is defined as the null list, and return is made in step (12).

If J[r] ≤ n, the system is consistent. Thus, since

$CZ' = 0$, we have from Theorem 4.1.3 that a general solution (D,Y,Z) to the system $AX = B$ can be obtained from Z'. In particular, the theorem specifies that the submatrix of Z' consisting of the first n rows and first n-r columns constitutes a null space basis Z for A. Moreover, the submatrix consisting of the first n rows and last q columns constitutes the matrix part Y of a particular solution, when paired with D. The matrices Z and Y are constructed in step (11). Initially h is set to the number of columns of Z minus one and u is set to the number n of rows required. If $r = n$, then $h = -1$, indicating that there is no null space basis to construct. In this case Z remains as the null list. If $r < n$, then $h \geq 0$ and the first h+1 elements of each of the first n rows of Z' are removed to form the n rows $Z_1,\ldots,Z_n$ of Z. They do, however, appear in the reverse order: $Z = (Z_n,\ldots,Z_1)$. As the rows of Z' are being prefixed to Z, the final segments of q elements of each row of Z' are being prefixed to Y. This occurs even when $h = -1$ and results in the entire row of Z' becoming the corresponding row of Y. The rows $Y_1,\ldots,Y_n$ of Y also appear in the reverse order: $Y = (Y_n,\ldots,Y_1)$. Since $n < n'$, Z' now consists in its last n'-n rows, whereupon Z' is erased.

Finally, the general solution list $G = (D,Y,Z)$ is constructed, reversing the order of the rows of Y and Z. The

last step, step (12), performs the erasures of all remaining lists, which have been constructed by the algorithm, and returns the list G.

As was done for CPRRE, we now consider the expected behavior of PLES. First consider the probability that a mod-p mapping φ_p is rejected for C. From Theorem 2.3.4 we know that φ_p is a rejected mapping if and only if $\varphi_p(\delta_k(C)) = 0$, for some k: $1 \le k \le r$, where $\delta_k(C) = C\binom{i_1,\ldots,i_k}{j_1,\ldots,j_k}$, $J_C = (j_1,\ldots,j_r)$, and $I_C = (i_1,\ldots,i_m)$. Furthermore, $\varphi_p(\delta_k(C)) = 0$ if and only if p divides all integer coefficients of $\delta_k(C)$.

Consider the case $s = 0$, i.e., $d_k = \delta_k(C)$ an integer. Recall that each prime p selected from the list PRIME satisfies: $\gamma/c \le p < \gamma$. Typically, on a binary computer $\gamma = 2^t$, for some positive integer t, and $c = 2$ (i.e., on the UNIVAC 1108 one might have $t = 33$). Thus, if $|d_k| < \gamma/2$, then p does not divide d_k, since $p > |d_k|$. If $\gamma/2 \le |d_k| < \gamma$, then p divides d_k if and only if $d_k = \pm p$, since $2p \ge \gamma$, for all such primes p. Assuming that γ is approximately 10^{10} (i.e., for $\gamma = 2^{33}$), by using the Prime Number Theorem (see [KND69], p. 340) the frequency of primes in the interval $[\gamma/2, \gamma]$ is approximately 1/20. Since the number of primes in $[\gamma/2, \gamma]$ is then approximately $(\gamma/2)(1/20) = \gamma/40$, an estimate of the probability that $p \mid d_k$ in this interval is $40/\gamma$. Finally, if $|d_k| > \gamma$,

the probability decreases rapidly with the number of potential prime divisors of d_k. The probability is small that $|d_k|$ falls in an interval where the probability that $p \mid \delta_k$ is anything but minute, since the interval $[\gamma/2, \gamma]$ is small.

Thus, it is quite reasonable to expect that the probability that φ_p is a rejected mapping for the integer case $s = 0$ is quite small. It may then be inferred that the probability that φ_p is rejected, for $s \geq 1$, is much smaller. For p must divide not just one integer but all the integer coefficients of a polynomial $\delta_k(C)$. In all the cases $s \geq 0$, the summations of the probabilities for $k = 1,2,\ldots,r$ should not produce a probability of φ_p being rejected which is anything but minute. It can thus be expected that all mod-p mappings employed are accepted. Hence, with high probability $J = J_C$ and $I = I_C$ are computed for all mappings φ_p and, hence, $F = \overline{\overline{C}}$ is being constructed by Garner's Method.

We turn now to the equality and substitution tests, as they affect and are affected by the expected behavior of the algorithm. Equality occurs when the successive iterates D and D' and also V and V', when they are non-null, are equal (i.e., all pairs of successive polynomial or integer iterates). Thus, suppose k-1 consecutive mod-p mappings $\varphi_{p_1},\ldots,\varphi_{p_{k-1}}$ have been retained and let b_{k-1} be any $(k-1)^{th}$ integer iterate,

where we are considering only the case s = 0. Referring to Section 4.2, we have in applying Garner's Method that $b_k = b_{k-1}$ if and only if $a_k - \varphi_{p_k}(b_{k-1}) \equiv 0 (\text{mod } p_k)$, where a_k is the next image in $GF(p_{k_i})$ of the integer being constructed and b_k is the k^{th} iterate. Letting $b = a_k - b_{k-1}$ and considering b to be an arbitrary integer such that $|b| < p_1 \cdots p_k/2$, then the probability that $\varphi_{p_k}(b) = \varphi_{p_k}(a_k - b_{k-1}) = a_k - \varphi_{p_k}(b_{k-1}) \equiv 0 (\text{mod } p_k)$ is very nearly $1/p_k$.

Thus, if C consists in integer entries (i.e., s = 0), the probability of a premature equality success in constructing $F = \bar{\bar{C}}$ by Garner's Method can be inferred to be small. For an estimate of the probability that a premature equality of integer iterates occurs is at most $\sum_{i=1}^{k}(1/p_i) \leq \sum_{i=1}^{k}(2/\gamma) = 2k/\gamma = \eta$, where k is the number of iterations required to construct $F = \bar{\bar{C}}$. Since γ and the primes p are large integers (i.e., on the order of 10^{10}), it is reasonable to assume (for any tractable problem) that the number k of primes required so that $p_1 \cdots p_k/2$ is greater than all integer coefficients of $F = \bar{\bar{C}}$ is small relative to γ: $k << \gamma$. Hence, the above probability η is small. Moreover, if C is a matrix of polynomials ($s \geq 1$) or if more than one integer or polynomial constitutes the compact representation of $F = \bar{\bar{C}}$, then the probability of premature equality is much, much smaller. The expected be-

havior of PLES will, therefore, be such that no rejected mod-p mappings occur and also no premature equality test success occurs, which implies that only one substitution test is applied.

Suppose that the elements of C are in $P(d,\bar{m}_s)$, $s \geq 0$, and recall that the elements of the compact representation of $\bar{\bar{C}}$ are subdeterminants of C of order r. If M is any such subdeterminant, then by Lemma 2.4.1 we see that $\text{norm}(M) \leq r!\sum_{i=1}^{r} d = r!d^r \leq r^r d^r = (rd)^r$. It can also be shown, using Lemma 2.5.1, that the degree in x_i of each such subdeterminant is $\leq rm_i$, $1 \leq i \leq r$, as was done in Section 4.4. Thus, each element of the compact representation of $F = \bar{\bar{C}}$ is in $P((rd)^r, rm_1, \ldots, rm_s)$, $s \geq 0$.

We show that the number k of primes used satisfies: $k \lesssim r(\log rd)$. Let N be the maximum magnitude of the integer coefficients of the elements of $F = \bar{\bar{C}}$, which is being constructed by Garner's Method. Then $N \leq (rd)^r$. Moreover, for the last three iterations (j=k-2,k-1,k), we have: $N \geq p_1 \cdots p_{k-2}/2$, $N < p_1 \cdots p_{k-1}/2$, and $N < p_1 \cdots p_k/2$. So, $\log N \geq \log(p_1 \cdots p_{k-2}/2) = \log p_1 \cdots p_{k-2} - \log 2 \sim k-2 - \log 2$. Hence, $k \sim \log N + 2 + \log 2 \sim \log N \leq \log(rd)^r = r(\log rd)$.

For the same reasons as given in Section 4.4 for CPRRE, dominance relations on the maximum computing time

functions for PLES will be obtained, but under the assumption that no rejected mod-p mappings (or evaluation mappings) occur and no premature equality test successes occur.

Theorem 4.5.2. Let $T_{PLES}(m,n',r,d,\bar{m}_s)$ be the maximum computing time of PLES under the assumption that no rejected mod-p or evaluation mappings are used and every substitution test is successful, for the augmented system matrix C of rank $r \geq 1$ in $M(m,n',d,\bar{m}_s)$, $s \geq 0$. Then, for $s = 0$, $T_{PLES} \lesssim (\log n' + r(\log rd))[m(n')^2 + (m+n')r(n'-r+1)(\log rd)]$ and, for $s \geq 1$, $T_{PLES} \sim (\log n' + r(\log rd))\, r^s (\Pi_{i=1}^{s}\mu_i)[m(n')^2 + (m+n')r(n'-r+1)(\log rd + \sum_{i=1}^{s}\mu_i)]$.

Proof: We begin with a maximum computing time table (under the given assumptions) for PLES, which will be followed by an explanation of the dominance relations obtained for the individual steps. Note that in steps (3), (9)-(12) the entries i are followed by inequalities. These are used to specify the particular subset of the input matrices C to which the corresponding dominating functions N_i', $T_{i,j}'$, and T_i' apply.

i	N_i	$T_{i,j}$	T_i
1	1	-	mn'
2	r(log rd)	mn'(log d)$(\Pi_{i=1}^{s}\mu_i)$	mn'r(log d)(log rd)$(\Pi_{i=1}^{s}\mu_i)$

i	N_i	$T_{i,j}$	T_i
3:s=0	r(log rd)	$mn'r$	$mn'r^2(\log rd)$
3:s>0	r(log rd)	$r^s(\Pi_{i=1}^s \mu_i)[m(n')^2 + (m+n')(n'-r+1)r(\sum_{i=1}^s \mu_i)]$	$r^{s+1}(\Pi_{i=1}^s \mu_i)(\log rd)[m(n')^2+(m+n')(n'-r+1)r(\sum_{i=1}^s \mu_i)]$
4	r(log rd)	m	$rm(\log rd)$
5	0	-	0
6	1	-	$r(n'-r)+1$
7	r(log rd)	$(r(n'-r)+1)r^s j(\Pi_{i=1}^s \mu_i)$	$(r(n'-r)+1)r^{s+2}(\log rd)^2(\Pi_{i=1}^s \mu_i)$
8	r(log rd)	$(r(n'-r)+1)r^s j(\Pi_{i=1}^s \mu_i)$	$(r(n'-r)+1)r^{s+2}(\log rd)^2(\Pi_{i=1}^s \mu_i)$
9:r>n	1	-	1
9:r≤n	1	-	$(\log n'+r(\log rd))(n'-r+1)r^s(\Pi_{i=1}^s \mu_i)\cdot[mn'+(m+n')r(\log rd + \sum_{i=1}^s \mu_i)]$
10:J[r] >n	1	-	$n'(n'-r)$
10:J[r] ≤n	1	-	r
11:J[r] ≤n	n	j<n: n-r j=n: $(n'-r)(n'-r+1)$	$n'(n'-r)$
12:J[r] ≤n	1	-	$m+r(n'-r+1)+r(\log rd)$
12:J[r] >n	1	-	$m+(r(n'-r)+1)r^{s+1}(\log rd)(\Pi_{i=1}^s \mu_i)$

As the discussion preceding the theorem showed, if it is assumed that no rejected mod-p mappings occur, then the number k of mod-p mappings which must be applied before the equality and substitution tests are satisfied is $\sim \log(rd)^r = r(\log rd)$. For each of these mappings steps (2),(3), and (4) are executed. In step (4) each pair r and r*, J and J*, and I and I* will be compared and found equal and so step (7) will always be executed next, except for the first execution, when step (6) will be inserted to initialize Garner's Method. This is the only execution of step (6). Step (8) always follows step (7) and is executed only when preceded by step (7). Thus, we have $N_i \lesssim r(\log rd)$, for $i = 2,3,4,7,8$, and $N_6 = 1$. Also, since no mod-p mappings are discarded, $N_5 = 0$. Since it is assumed that the substitution test of step (9) succeeds when first applied, we have $N_9 = 1$. Step (10) is always executed just once. Step (11) is executed only for a consistent system and in this case $N_{11} = n$. Of course, $N_1 = N_{12} = 1$.

In step (1), $T_1 \sim T_{MVLIST}(m,n') \sim mn'$. For step (2), $T_{2,j} \sim T_{MMOD}(m,n',d,\bar{m}_s) \lesssim mn'(\log d)(\Pi_{i=1}^{s}\mu_i)$. Letting k be the number of iterations required to obtain the output list G, then $T_2 = \sum_{j=1}^{k} T_{2,j} = kmn'(\log d)(\Pi_{i=1}^{s}\mu_i) \lesssim r(\log rd)mn'(\log d)(\Pi_{i=1}^{s}\mu_i) = T_2'$. For step (3), if $s = 0$, $T_{3,j} \sim T_{CPRRE}(m,n',r,\bar{m}_o) \sim mn'r = T_{3,j}'$ and, if $s \geq 1$, we have $T_{3,j} \sim T_{CPRRE}(m,$

$n', r, \bar{m}_s) \lesssim r^s (\Pi_{i=1}^s \mu_i)[m(n')^2 + (m+n')(n'-r+1) r (\Sigma_{i=1}^s \mu_i)] = T'_{3,j}$. We have used rank $r = r^*$ for C^*, since each mod-p mapping is assumed to be accepted. Thus, $T_3 = \Sigma_{j=1}^k T_{3,j} \lesssim r(\log rd)$. $T'_{3,j} = T'_3$ follows directly. In step (4), $T_{4,j} \sim T_{LENGTH}(r^*) + T_{VCOMP}(r^*, r) + T_{VCOMP}(m,m) \sim r^* + r^* + m \sim m$ and so $T_4 \lesssim \Sigma_{j=1}^k m = km \lesssim r(\log rd) \cdot m = T'_4$. Since $N_5 = 0$, $T_5 = 0$. The maximum time for the execution of step (6) occurs when MZCON is applied to a non-diagonal part V^* with the maximum number of elements. Since $r^* = r$ is assumed, this is $r(n'-r)+1$, as shown in Theorem 4.3.2. Hence, $T_6 \sim T_{6,1} \lesssim r(n'-r)+1$.

In step (7) Garner's Method is applied to at most $r(n-r)+1$ integers or polynomials -- at most $r(n-r)$ in applying MGARN to V and V^* and one in applying CPGARN to D and D^*. Thus, there occur at the j^{th} iteration at most $r(n-r)+1$ computations: $H' = CPGARN(E, H, p_j, H^*, L)$, where $H \in P(E, rm_1, \ldots, rm_s)$, $H^* \in P^*(rm_1, \ldots, rm_s)$, and $E = p_1 \cdots p_{j-1}$. It was shown preceding Theorem 3.4.14 that $\log p_1 \cdots p_t \sim t$. Thus, the computing time for each of these computations is $\lesssim T_{CPGARN}(p_1 \cdots p_{j-1}, rm_1, \ldots, rm_s) \lesssim (\log p_1 \cdots p_{j-1})(\Pi_{i=1}^s r\mu_i) \sim jr^s(\Pi_{i=1}^s \mu_i)$. Hence, $T_{7,j} \lesssim jr^s(r(n-r)+1)(\Pi_{i=1}^s \mu_i)$ and so $T_7 = \Sigma_{j=1}^k T_{7,j} \lesssim \Sigma_{j=1}^k jr^s(r(n-r)+1)(\Pi_{i=1}^s \mu_i) = k(k+1) r^s(r(n-r)+1)(\Pi_{i=1}^s \mu_i)/2 \sim k^2 r^s(r(n-r)+1)(\Pi_{i=1}^s \mu_i) \lesssim r^2(\log rd)^2 r^s(r(n-r)+1)(\Pi_{i=1}^s \mu_i) = T'_7$. Step (8) is very similar to step (7) and requires possibly $r(n-r)+1$ subtractions of ele-

ments in $P(p_1 \cdots p_j, rm_1, \ldots, rm_s)$ at the j^{th} iteration. Since $T_{PDIF}(p_1 \cdots p_j, rm_1, \ldots, rm_s) \lesssim jr^s(\Pi_{i=1}^s \mu_i)$, we obtain in the same way $T_8' = T_7'$.

We now consider the substitution test of step (9). Those inconsistent systems C for which $r > n$ require no substitution test and so $T_9 \sim 1$ in such cases. If $r \leq n$, then the substitution test must be performed. NULCON is applied to construct a matrix Z' satisfying formula (1) of Section 4.1 from n', J, D, and V, where (J,D,V) is the compact form of the most recent iterate G in constructing $F = \bar{\bar{C}}$. G is m by n' and of rank r and so the time to construct Z' is $T_{NULCON}(n',r) \sim n'(n'-r)$. Clearly, $Z' \in M(n', n'-r, P((rd)^r, rm_1, \ldots, rm_s))$. Thus, the time to compute $T = CZ'$ is $\lesssim \hat{T} = T_{MMPY}(m,n',n'-r, d, m_1, \ldots, m_s, (rd)^r, rm_1, \ldots, rm_s)$ $\lesssim (\log n'de)[\, mn'q(\Pi_{i=1}^s \kappa_i) + mr\{(\log d)(\Pi_{i=1}^s \mu_i) + \sum_{i=1}^s (\Pi_{j=1}^i \mu_j)(\Pi_{j=i}^s \kappa_j)\} + rq\{(\log e)(\Pi_{i=1}^s \nu_i) + \sum_{i=1}^s (\Pi_{j=1}^i \nu_j)(\Pi_{j=i}^s \kappa_j)\} + mq\{(\Pi_{i=1}^s \kappa_i)(\log de + \sum_{i=1}^s \kappa_i)\}\,]$, where $e = (rd)^r$, $\nu_i = rm_i + 1$, $\kappa_i = \mu_i + \nu_i$, and $q = n'-r$.

Let $\lambda = \log n'de = \log n'd(rd)^r = \log n' + \log d + \log(rd)^r \sim \log n' + r(\log rd)$. We note that $\nu_i \lesssim rm_i + r = r\mu_i$ and $\kappa_i \lesssim \mu_i + r\mu_i \sim r\mu_i$. Thus $\Pi_{i=1}^s \kappa_i \lesssim \Pi_{i=1}^s r\mu_i = r^s(\Pi_{i=1}^s \mu_i)$, $\sum_{i=1}^s (\Pi_{j=1}^i \mu_j)(\Pi_{j=i}^s \kappa_j) \lesssim \sum_{i=1}^s (\Pi_{j=1}^i \mu_j)(\Pi_{j=i}^s r\mu_j) = r(\Pi_{i=1}^s \mu_i)$ $(\sum_{i=1}^s \mu_i r^{s-i})$, $\Pi_{i=1}^s \nu_i \lesssim \Pi_{i=1}^s r\mu_i = r^s(\Pi_{i=1}^s \mu_i)$, $\sum_{i=1}^s (\Pi_{j=1}^i \nu_j)$ $(\Pi_{j=i}^s \kappa_j) \lesssim \sum_{i=1}^s r^i (\Pi_{j=1}^i \mu_j) r^{s-i+1} (\Pi_{j=i}^s \mu_j) = r^{s+1}(\Pi_{i=1}^s \mu_i)$

$(\sum_{i=1}^{s}\mu_i)$, and finally $\sum_{i=1}^{s}\kappa_i \lesssim \sum_{i=1}^{s} r\mu_i = r(\sum_{i=1}^{s}\mu_i)$. Applying these dominance relations, $\hat{T} \lesssim \lambda\cdot[mn'(n'-r)r^s(\prod_{i=1}^{s}\mu_i) + mn'\{(\log d)(\prod_{i=1}^{s}\mu_i) + r(\prod_{i=1}^{s}\mu_i)(\sum_{i=1}^{s}\mu_i r^{s-i})\} + n'(n'-r)\{r(\log rd)r^s(\prod_{i=1}^{s}\mu_i) + r^{s+1}(\prod_{i=1}^{s}\mu_i)(\sum_{i=1}^{s}\mu_i)\} + m(n'-r)\{r^s(\prod_{i=1}^{s}\mu_i)(r(\log rd) + r(\sum_{i=1}^{s}\mu_i))\}] = \lambda(\prod_{i=1}^{s}\mu_i)\cdot[mn'\{(n'-r)r^s + \log d + r(\sum_{i=1}^{s}\mu_i r^{s-i})\} + (m+n')(n'-r)r^{s+1}(\log rd + \sum_{i=1}^{s}\mu_i)]$. Since $n' \lesssim r(n'-r)$, by Lemma 4.4.2, $mn'r(\sum_{i=1}^{s}\mu_i r^{s-i}) \lesssim mn'r^s(\sum_{i=1}^{s}\mu_i) \lesssim (m+n')[r(n'-r)]r^s(\sum_{i=1}^{s}\mu_i)$. Similarly, $mn'(\log d) \lesssim (m+n')[r(n'-r)]r^s(\log rd)$. Thus, $\hat{T} \lesssim \lambda(\prod_{i=1}^{s}\mu_i)\cdot[mn'(n'-r)r^s + (m+n')(n'-r)r^{s+1}(\log rd + \sum_{i=1}^{s}\mu_i)] = \hat{T}'$. $\hat{T}'$ can easily be shown to dominate the times for PNEG, NULCON, MZERO, and the erasures in step (9) and so $T_9 \lesssim \hat{T}' = T_9'$, when $r < n$.

In step (10), when $J[r] > n$, the time to erase Z' is $\sim n'(n'-r)$, since this many cells are returned to available space, and this time dominates the times for the other operations. Thus, $T_{10} \sim n'(n'-r)$ for an inconsistent system. If the system is consistent (i.e., $J[r] \leq n$), $T_{10} \sim T_{INV}(r) \sim r$. If $J[r] \leq n$, then step (11) is executed, n times. For the first n-1 executions, at most n-r elements of a row of Z' are traversed and so $T_{11,j} \lesssim n-r$, when $j < n$. During the n^{th} execution, Z' is erased, returning (n'-r)(n'-r+1) cells to available space, and Y and Z are inverted, the time for each being $\sim n$. So, $T_{11} \lesssim n'(n'-r)$. The time

for step (12) is dominated by the times for the erasures. The numbers of cells returned for erasing J,I,E,X are $\lesssim$ r, m, r(log rd), 1, respectively. If the system is consistent, D and the elements of V belong to the general solution (D, Y,Z); hence, their erasure returns at most r(n-r) + 1 cells. If the system is inconsistent, the number of cells returned by the erasure of D and V is $\lesssim (r(n'-r)+1)r^{s+1}(\log rd)(\Pi_{i=1}^{s}\mu_i)$. Thus, for a consistent system, $T_{12} \lesssim m + r(n'-r+1) + r(\log rd)$ and for an inconsistent system, $T_{12} \lesssim m + (r(n'-r)+1)r^{s+1}(\log rd)(\Pi_{i=1}^{s}\mu_i)$.

We may now combine the dominating functions for all the steps. Clearly, for $s \geq 0$, $T_1' + T_4' + T_5' + T_6' \lesssim T_2'$, $T_6' \lesssim T_7' = T_8'$, and $T_{12}' \lesssim T_3'$. Also, $T_{10}' + T_{11}' \lesssim n'(n'-r) \lesssim mn'r(n'-r+1) \lesssim T_9'$, since $n' \lesssim r(n'-r+1)$. So, $T_{PLES} \lesssim T_2' + T_3' + T_7' + T_9'$. Let $\lambda = \lambda_1 + \lambda_2$, where $\lambda_1 = \log n'$ and $\lambda_2 = r(\log rd)$. Consider the case s = 0. We have $T_{PLES} \lesssim (T_2'+T_3'+T_7') + T_9' = \lambda_2 \cdot [mn'(\log d) + mn'r + (n'-r+1)r^2(\log rd)] + \lambda \cdot [mn'(n'-r+1) + (m+n')(n'-r+1)r(\log rd)]$. Since $\lambda_2 \lesssim \lambda$ and $n' \lesssim r(n'-r+1)$, we have $\lambda_2\, mn'(\log d) \lesssim \lambda(m+n')(n'-r+1)r(\log rd)$. Also, $\lambda_2\, mn'r + \lambda mn'(n'-r+1) \lesssim \lambda mn'(r+(n'-r+1)) \sim \lambda m(n')^2$. Moreover, $\lambda_2(n'-r+1)r^2(\log rd) \lesssim \lambda(n'-r+1)(m+n')r(\log rd)$. Thus, $T_{PLES} \lesssim \lambda \cdot [m(n')^2 + (m+n')(n'-r+1)r(\log rd)]$, for s = 0.

Now consider the case s > 0. We have that $T_2' + T_3' +$

$T'_7 = \lambda_2(\Pi_{i=1}^{s}\mu_i)\cdot[\ mn'(\log d) + m(n')^2 r^s + (m+n')(n'-r+1)r^{s+1}(\Sigma_{i=1}^{s}\mu_i) + (n'-r+1)r\cdot r^{s+1}(\log rd)\ ] = \lambda_2(\Pi_{i=1}^{s}\mu_i)K_1$. Also, $T'_9 = (\lambda_1+\lambda_2)(\Pi_{i=1}^{s}\mu_i)\cdot[\ mn'(n'-r+1)r^s + (m+n')r(n'-r+1)r^s(\log rd) + (m+n')(n'-r+1)r^{s+1}(\Sigma_{i=1}^{s}\mu_i)\ ] = (\lambda_1+\lambda_2)(\Pi_{i=1}^{s}\mu_i)K_2$. Thus, $T_{PLES} \lesssim (\Pi_{i=1}^{s}\mu_i)[\lambda_2 K_1 + (\lambda_1+\lambda_2)K_2]$. We observe that among the terms of $K_1 + K_2$ the following dominance relations hold. First, $mn'(\log d) \lesssim (m+n')r(n'-r+1)r^s(\log rd)$, since $n' \lesssim r(n'-r+1)$. Also, $(n'-r+1)r^{s+2}(\log rd) \lesssim (m+n')r(n'-r+1)r^s(\log rd)$. Thus, $T_{PLES} \sim r^s(\Pi_{i=1}^{s}\mu_i)(\log n' + r(\log rd))\ [m(n')^2 + (m+n')r(n'-r+1)(\log rd + \Sigma_{i=1}^{s}\mu_i)]$, for $s > 0$.//

The dominance relations obtained by this theorem for the maximum computing time of PLES are quite general and may be simplified by making some reasonable assumptions concerning the parameters m, n', and r. Thus, typically we have $n' \sim m$. For example, when solving a single system of linear equations with a square coefficient matrix A we have $n = m$ and $n' = m+1$. The following corollary gives the simplified dominance relation for PLES under this assumption and is easily proved.

Corollary 4.5.3. If the m by n' input matrices C to PLES satisfy $n' \sim m$, then $T_{PLES} \lesssim (\log m + r(\log rd))m\ [m^2+(m-r+1)$

$r(\log rd)]$, if $s = 0$, and $T_{PLES} \lesssim (\log m + r(\log rd))m$ $r^s(\Pi_{i=1}^{s}\mu_i)[m^2 + r(m-r+1)(\log rd + \sum_{i=1}^{s}\mu_i)]$, if $s > 0$.

It is quite often the case that systems of linear equations have an augmented matrix with the number of columns on the order of the number of rows, i.e., $n' \sim m$. Also, it is not unusual to have the rank r the order of m. This is obtained by assuming $m - r \lesssim 1$. These assumptions provide more concise computing times for PLES. We note that $m + \log md \sim m + \log d$.

Corollary 4.5.4. If the m by n' input matrices C to PLES are such that $m - r \lesssim 1$ and $n' \sim m$, then $T_{PLES} \lesssim m^{s+3}$ $(\log md)(\Pi_{i=1}^{s}\mu_i)(m + \log d + \sum_{i=1}^{s}\mu_i)$, for $s \geq 0$.

As a final corrollary, by applying the definition $\mu = \max \mu_i$ ($\mu = 0$ if $s = 0$) we obtain the following crude but compact result.

Corollary 4.5.5. If the m by n' input matrices C to PLES satisfy: $m - r \lesssim 1$ and $n' \sim m$, then $T_{PLES} \lesssim m^{s+3}\ \mu^s(\log md)$ $(\mu + m + \log d\ )$, for $s \geq 0$.

It can be shown that the exact division algorithm (EXDIV) for solving systems of linear equations with integer or polynomial coefficients has a maximum computing time $T_{EXDIV} \lesssim$ $m^{2s+5}(\log md)^2(\Pi_{i=1}^{s}\mu_i)^2$ when applied to augmented matrices C under the same assumptions as in Corollary 4.5.4. The general

superiority of the modular algorithm over the exact division algorithm is thus apparent. In particular, to obtain the dominating function for T_{PLES} from that for T_{EXDIV}, m^{2s+4} is replaced by m^{s+3} and $m(\log md)(\Pi_{i=1}^{s} \mu_i)$ is replaced by $m + \log d + \sum_{i=1}^{s} \mu_i$.

4.6. A Modular Algorithm for Determinant Calculation

In the preceding sections modular algorithms for matrix multiplication and for solving systems of linear equations were given. Not surprisingly, an algorithm, analogous to those algorithms, for computing the determinants of matrices with integer or polynomial entries exists. We present such an algorithm in this section and, as was done for MMPY and PLES, we present it as three separate algorithms: inner, middle, and outer. The outer algorithm PDET applies mod-p mappings to matrices over $I[x_1,\ldots,x_s]$, $s \geq 0$, obtaining image matrices over $GF(p)[x_1,\ldots,x_s]$, $s \geq 0$, to which the middle algorithm CPDET is applied to compute the determinants. The outer algorithm then applies Garner's Method to construct the determinant in $I[x_1,\ldots,x_s]$ from these determinant images. If a matrix over GF(p) is input to CPDET, the inner algorithm CDET is evoked immediately. If a matrix of polynomials over GF(p) is input to CPDET, evaluation mappings are applied and the algorithm applies itself recursively to the images, obtaining determinant images. The determinant is then constructed from these images by incremental interpolation. CDET computes the determinant of a matrix over GF(p) by employing Gaussian elimination. Its similarity to CRRE will be detailed below. The inner algorithm is now presented.

Algorithm 4.6.1. (CDET) Determinant Calculation over GF(p) by Gaussian Elimination

Input: A prime number p and an m by m matrix A over GF(p). The list representing A may not overlap any other list.

Output: d = det(A) obtained by d = CDET(p,A). A is erased.

(1) [Initialize.] d ← 1; t ← 0; B' ← A.

(2) [Begin next elimination step.] B" ← B'; B' ← ().

(3) [Search next column for pivot.] DECAP(F,B"); DECAP(e,F); if e ≠ 0, go to (4); if F ≠ (), B' ← PFL(F,B'); t ← t+1; if B" ≠ (), go to (3); d ← 0; erase B'; return d.

(4) [Recompute d and begin elimination.] d ← CPROD(p,e,d); if F = (), go to (8); e ← CRECIP(p,e); G ← F.

(5) [Transform pivot row.] ALTER(CPROD(p,e,FIRST(G)),G); G ← TAIL(G); if G ≠ (), go to (5).

(6) [Begin transforming next row, if any.] If B" = (), go to (8); DECAP(G,B"); DECAP(u,G); B' ← PFL(G,B'); if u = 0, go to (6); H ← F.

(7) [Recompute next element of row G.] s ← CPROD(p,u,FIRST(H)); v ← CDIF(p,s,FIRST(G)); ALTER(t,G); G ← TAIL(G); H ← TAIL(H); if G ≠ (), go to (7); go to (6).

(8) [Complete determinant calculation, if elimination finished.] Erase F; B' ← INV(B'); if B' ≠ (), go to (2); u ← 0; QR(u,t,2); if t = 1, d ← p-d; return d.

This algorithm is nearly identical to the triangularization phase of CRRE, i.e., steps (1)-(8). The differences lie in the fact that those operations of CRRE not required in determinant calculation are omitted. Thus, the lists I and J are not computed and so I, I', I", J, and z have been eliminated. Also, since the triangularized rows are not retained, the list B has been eliminated. Of course, all quantities introduced for the back substitution have been discarded. The only element of the output (J,I,d,V) of CRRE which has been retained is d. If during the k^{th} pivot operation a pivot element fails to be found in column k in step (3) (i.e., B" = ()), then rank(A) < m and so d = 0 is returned. Otherwise, return occurs in step (8) at the end of the m^{th} pivot operation.

In the m^{th} execution of step (8) $d = \delta(A) = \det(C)$, where $C = A\begin{bmatrix} i_1,\ldots,i_m \\ 1,\ldots,m \end{bmatrix}$, has been computed. We know that, when the rows of A have been permuted by $I_A = (i_1,\ldots,i_m)$ to give C, then $\det(C) = \operatorname{sgn}(I_A)\cdot\det(A)$ or $\det(A) = \operatorname{sgn}(I_A)\cdot\det(C)$ or $\det(A) = \operatorname{sgn}(I_A)\cdot\delta(A)$, where sgn is the signum function defined on permutations. For a permutation $\sigma \in \mathcal{P}$, $\operatorname{sgn}(\sigma) = +1$, when σ is an even permutation, and $\operatorname{sgn}(\sigma) = -1$, when σ is odd. If σ is written as the product of transpositions: $\sigma = \zeta_1\zeta_2\cdots\zeta_t$, then σ is even if and only if t is an even integer; otherwise, σ is odd. In effect, what is

occuring in step (3) of CRRE is the computation of I_A as the product of transpositions, for the row interchanges performed on the matrix are being mirrored in the interchanges of the components of the permutation. Thus, although the permutation I_A is not computed in CDET, the integer t is in step (3). Hence, in step (8) by applying QR the integer t is recomputed as t(modulo 2). Then t = 1 if and only if I_A is odd and in this case d is replaced by $-d = \text{sgn}(I_A) \cdot \delta(A) = \det(A)$. Otherwise, $d = \delta(A) = \det(A)$ has been computed.

Consider now the maximum computing time of CDET applied to a matrix A of rank r. Recall that $T_{CRRE}(m,n,r) \sim mnr$, which is codominant with the maximum time of step (7) of that algorithm. Taking m = n, since steps (7) of CDET and CRRE are identical, $mnr = m^2r$ dominates the time for step (7) and, in fact, for all steps of CDET. So, the maximum computing time for CDET is $\sim m^2r$. Analogous to Theorem 4.3.3, suppose the matrix A defined by formula (1) of Section 4.3, with m = n, is input to CDET. Then it can be shown in the same way that the time for step (7) is $\sim m^2r$. Hence, the maximum computing time of CDET is $\sim m^2r$. We summarize in the following theorem.

Theorem 4.6.2. Let $T_{CDET}(m,r)$ be the maximum computing time of CDET for $A \in M(m,m,GF(p))$ of rank r. Then $T_{CDET} \sim m^2r$.

The middle and outer algorithms apply the following fact in the construction of determinants by interpolation or Garner's Method. Let θ be an induced mapping on matrices over a set S and let $A^* = \theta(A)$. If $D^* = \det(A^*)$, then $D^* = \det(\theta(A)) = \theta(\det(A))$. Hence, D^* is an image of $\det(A)$ and may be used to construct $\det(A)$ by Garner's Method, when θ is a mod-p mapping and A is over $I[x_1,..,x_s]$, $s \geq 0$, or by incremental interpolation, when θ is an evaluation mapping and A is over $GF(p)[x_1,...,x_s]$, $s \geq 1$. We note that PDET and CPDET are like MMPY and MCPMPY in that all images A^* can be used to find images D^* of $D = \det(A)$; there are no rejected mappings in determinant calculation.

We now present the middle algorithm.

Algorithm 4.6.3. (CPDET) Determinant Calculation Using Evaluation Mappings

Input: A prime number p and an m by m matrix A over $GF(p)[x_1,...,x_s]$, $s \geq 0$.

Output: $D = \det(A)$ obtained by D = CPDET(p,A). The list representing A is erased.

(1) [Get number of variables.] If MZERO(A),EQ.1, (D←0; erase A; return D); s ← MCPNV(A).

(2) [Apply Gaussian elimination algorithm.] If s > 0, go to (3); D ← CDET(p,A); return D.

(3) [Initialize evaluation mapping algorithm.] h ← 0; m ← LENGTH(A); J ← (); for i = 1, m do: (J ← PFA(m+1-i,J));

b ← DBRRE(J,A); erase J; i ← 0; E ← PFA(0,PFA(1,0)); D ← 0.

(4) [Apply evaluation mapping Ψ_i.] If i = p, (print error message; stop); A* ← MCPEVL(p,A,i,s).

(5) [Apply algorithm recursively.] D* ← CPDET(p,A*); if D* ≠ 0, h ← 1; if h = 0, go to (7).

(6) [Interpolation step.] D' ← CPINT(p,D,i,D*,E,s); erase D; if s > 1, erase D*; D ← D'.

(7) [Degree test and return.] If CPDEG(E) = b, (erase A, E; return D).

(8) [Initialize next interpolation step.] T ← PFA(1,PFA(1, PFA(CDIF(p,0,a,),0))); U ← CPPROD(p,T,E); erase T,E; E ← U; i ← i+1; go to (4).

A brief explanation of some of the less trivial parts of this algorithm is now given. In step (3) we construct the list J = (1,2,...,m). It can be easily verified that by applying DBRRE to J and A, a bound b on the degree in x_s of det(A) is obtained. Suppose A is over $P^*(\bar{m}_s)$. Then, as has been done several times in the preceding sections, it can be shown that $\deg(\det(A)) \leq mm_s$ by applying Lemma 2.5.1. In fact, $\det(A) \in P^*(mm_1,\ldots,mm_s)$. Hence, since the bound b is computed to be attainable, $b \leq mm_s < m\mu_s$. There then follows b+1 iterations of steps (4)-(8) in which the evaluations, recursive applications, and interpolations are performed. We note that, during the k^{th} iteration, k > 1, the univariate

polynomial E is the polynomial $\Pi_{i=0}^{k-1}(x-i)$, where the evaluation mapping Ψ_i has just been applied. At the $(b+1)^{th}$ execution of step (7), deg(E) = b and so D = det(A) is returned, after erasing E and A.

If rank(A) = r < m, then det(A) = 0. In such cases D* = 0 will always be obtained in step (5) and no interpolations need be applied. Hence, in step (3) the flag h is initially set to 0 and is only reset to 1 in step (5) when a nonzero determinant D* is obtained. Thus, if the first k values of D* are zero, no interpolations are performed, leaving D = 0, which is what would have been computed had the interpolations been performed. Thus, if rank(A) < m, no interpolations are performed and D = 0 is returned.

The following theorem obtains dominance relations on the maximum computing times of CPDET.

Theorem 4.6.4. Let $T_{CPDET}(m,r,\bar{m}_s)$ be the maximum computing time of CPDET for $A \in M(m,m,P^*(\bar{m}_s))$ of rank r. Then, for s = 0, $T_{CPDET} \sim m^2 r$ and, for s = 1, $T_{CPDET} \lesssim m^3 \mu_1 (r+\mu_1)$. If r < m, then $T_{CPDET} \lesssim m^3 (\Pi_{i=1}^{s} \mu_i)(m^{s-1} r + \sum_{i=1}^{s} \mu_i m^{s-i})$, for $s \geq 2$. If r = m, then $T_{CPDET} \lesssim m^{s+1} (\Pi_{i=1}^{s} \mu_i)(m^2 + m\mu_1 + \sum_{i=2}^{s} \mu_i)$, for $s \geq 2$.

Proof: If A is over GF(p), s = 0 and only steps (1) and (2) are executed, each once. Thus, $T_1 \sim T_{MCPNV}(m,m) \sim m^2$, $T_2 \sim T_{CDET}(m,m,r) \sim m^2 r$, and so $T_{CPDET}(m,r,\bar{m}_o) \sim T_2 \sim m^2 r$. Suppose

A is over $GF(p)[x_1,\ldots,x_s]$, $s \geq 1$. The following maximum computing time table is then derived for CPDET.

i	N_i	$T_{i,j}$	T_i
1	1	-	m^2
2	1	-	1
3	1	-	m^2
4	$m\mu_s$	$m^2(\Pi_{i=1}^{s}\mu_i)$	$m^3\mu_s(\Pi_{i=1}^{s}\mu_i)$
5	$m\mu_s$	$T_{CPDET}(m,r_j,\bar{m}_{s-1})$	$\sum_{j=1}^{b+1} T_{CPDET}(m,r_j,\bar{m}_{s-1})$
6:r<m	0	-	0
6:r=m	$m\mu_s$	$jm^{s-1}(\Pi_{i=1}^{s-1}\mu_i)$	$m^{s+1}\mu_s(\Pi_{i=1}^{s}\mu_i)$
7	$m\mu_s$	1, last is $\lesssim m^2(\Pi_{i=1}^{s}\mu_i)$	$m^2(\Pi_{i=1}^{s}\mu_i)$
8	$m\mu_s$	j	$m^2\mu_s^2$

Step (1) has been analyzed above and step (2) is trivial. We noted prior to the theorem that the degree bound b satisifes: $b+1 \leq m\mu_s$ and so the number of executions of steps (4)-(8), which constitute the iteration loop, is $\lesssim m\mu_s$. So, $N_i' = m\mu_s$, for i = 4,5,6,7,8, except $N_6' = 0$ when $r < m$, as explained previously. For step (4), $T_{4,j} \sim T_{MCPEVL}(m,m,\bar{m}_s) \lesssim m^2(\Pi_{i=1}^{s}\mu_i)$ and T_4 is then obtained easily. $T_{5,j} \sim T_{CPDET}(m, r_j,\bar{m}_{s-1})$, which is not explicitly known at this point, where $r_j = \text{rank}(\Psi_j(A))$. Of course, $r_j \leq r$. When r = m in step (6), $T_{6,j} \sim T_{CPINT}(mm_1,\ldots,mm_{s-1},j-1) \lesssim j(\Pi_{i=1}^{s-1}m\mu_i) = jm^{s-1}(\Pi_{i=1}^{s-1}\mu_i)$. Since $j \leq b+1 \leq m\mu_s$, $T_6 = \sum_{j=1}^{b+1} T_{6,j} \leq (b+1)(m\mu_s)m^{s-1}(\Pi_{i=1}^{s-1}\mu_i) \leq$

$(m\mu_s)^2\, m^{s-1}(\Pi_{i=1}^{s-1}\mu_i) = m^{s+1}\mu_s(\Pi_{i=1}^{s}\mu_i)$. In step (7) the erasure of A in the last execution dominates the previous execution times. In step (8) the polynomial E has degree j-1 at the j^{th} execution and is multiplied by a polynomial T of degree 1. Hence, $T_{8,j} \sim j$ and $T_8 \lesssim \sum_{j=1}^{b+1} j \lesssim (b+1)^2 \le (m\mu_s)^2$.

As can be easily seen, $T_1' + T_2' + T_3' + T_7' + T_8' \lesssim T_4'$. For s = 1, $T_6' \lesssim T_4'$ and so $T_{CPDET}(m,r,\bar{m}_1) \lesssim T_4' + T_5' \lesssim m^3\mu_1^2 + \sum_{j=1}^{b+1} T_{CPDET}(m,r_j,\bar{m}_o) \lesssim m^3\mu_1^2 + \sum_{j=1}^{b+1} m^2 r_j \le m^3\mu_1^2 + \sum_{j=1}^{b+1} m^2 r = m^3\mu_1^2 + (b+1)m^2 r \le m^3\mu_1^2 + m\mu_1 \cdot m^2 r = m^3\mu_1(r+\mu_1)$.

Consider now the case r < m. Then $T_6' = 0$ and $T_{PLES} \lesssim T_4' + T_5'$. We wish to show inductively that, for $s \ge 1$, $T_{CPDET}(m, r,\bar{m}_s) \lesssim m^3(\Pi_{i=1}^{s}\mu_i)(m^{s-1}r + \sum_{i=1}^{s}\mu_i m^{s-i})$. This is clearly true for s = 1. Assuming it holds for s-1 replacing s, we show it holds for s. We have $T_{CPDET}(m,r,\bar{m}_s) \lesssim m^3\mu_s(\Pi_{i=1}^{s}\mu_i) + \sum_{j=1}^{b+1} m^3(\Pi_{i=1}^{s-1}\mu_i)(m^{s-2}r_j + \sum_{i=1}^{s-1}\mu_i m^{s-i-1}) \le m^3\mu_s(\Pi_{i=1}^{s}\mu_i) + (b+1)\, m^3(\Pi_{i=1}^{s-1}\mu_i)(m^{s-2}\, r + \sum_{i=1}^{s-1}\mu_i m^{s-i-1}) \le m^3\mu_s(\Pi_{i=1}^{s}\mu_i) + m\mu_s \cdot m^3 (\Pi_{i=1}^{s-1}\mu_i)(m^{s-2}\, r + \sum_{i=1}^{s-1}\mu_i m^{s-i-1}) = m^3(\Pi_{i=1}^{s}\mu_i)[\mu_s + m(m^{s-2}\, r + \sum_{i=1}^{s-1}\mu_i m^{s-i-1})] = m^3(\Pi_{i=1}^{s}\mu_i)(m^{s-1}r + \sum_{i=1}^{s}\mu_i m^{s-i})$, completing the inductive step.

Consider finally the case r = m. Then $T_4 \lesssim T_6'$ and $T_{CPDET} \lesssim T_5' + T_6'$. It will be shown inductively that, for $s \ge 2$, $T_{CPDET}(m,m,\bar{m}_s) \lesssim m^{s+1}(\Pi_{i=1}^{s}\mu_i)(m^2 + m\mu_1 + \sum_{i=2}^{s}\mu_i)$ and that $T_{CPDET}(m,r,\bar{m}_s) \lesssim T_{CPDET}(m,m,\bar{m}_s)$. For s = 2, $T_4' = T_6'$

and so the dominating function obtained above for the case $r < m$ holds also for $r = m$. This function, $m^3 \mu_1 \mu_2 (mr + m\mu_1 + \mu_2)$, clearly satisfies the inductive hypothesis for $r = m$. Thus, we assume it holds for s-1 replacing s and show it holds for s. We have $T_{CPDET}(m,m,\bar{m}_s) \lesssim \sum_{j=1}^{b+1} T_{CPDET}(m,r_j,\bar{m}_{s-1}) + m^{s+1} \mu_s (\Pi_{i=1}^{s} \mu_i) \lesssim \sum_{j=1}^{b+1} T_{CPDET}(m,m,\bar{m}_{s-1}) + m^{s+1} \mu_s (\Pi_{i=1}^{s} \mu_i) \lesssim \sum_{j=1}^{b+1} m^s (\Pi_{i=1}^{s-1} \mu_i)(m^2 + m\mu_1 + \sum_{i=2}^{s-1} \mu_i) + m^{s+1} \mu_s (\Pi_{i=1}^{s} \mu_i) \leq m\mu_s \cdot m^s (\Pi_{i=1}^{s-1} \mu_i)(m^2 + m\mu_1 + \sum_{i=2}^{s-1} \mu_i) + m^{s+1} \mu_s (\Pi_{i=1}^{s} \mu_i) = m^{s+1} (\Pi_{i=1}^{s} \mu_i)(m^2 + m\mu_1 + \sum_{i=2}^{s} \mu_i)$, completing the inductive step.//

By using the definition $\mu = \max_i \mu_i$, Theorem 4.6.4 can be simplified as follows.

Corollary 4.6.5. The maximum computing time functions of CPDET satisfy: for $s = 0$, $T_{CPDET} \sim m^2 r$ and, for $s \geq 1$, $T_{CPDET} \lesssim m^{s+2} \mu^s (r+\mu)$, for $r \leq m$.

Proof: If $r < m$ and $s \geq 2$, $T_{CPDET} \lesssim m^3 \mu^s (m^{s-1} r + \mu \sum_{i=1}^{s} m^{s-i}) \sim m^3 \mu^s (m^{s-1} r + \mu m^{s-1}) = m^{s+2} \mu^s (r+\mu)$. The case $r = m$ follows similarly.//

The main algorithm will require a bound on the integer coefficients of the determinant of a matrix of integers or polynomials over the integers. Such a bound is computed by the following algorithm.

Algorithm 4.6.6. (CBDET) Coefficient Bound for a Determinant

Input: An m by m matrix A over $I[x_1,\ldots,x_s]$, $s \geq 0$.

Output: An L-integer J = CBDET(A) such that J/2 bounds the magnitudes of the integer coefficients of det(A).

(1) [Initialize.] B ← A; m ← LENGTH(B); J ← PFA(2,0).

(2) [Begin to compute maximum norm of next row.] ADV(C,B); K ← 0.

(3) [Compare norm of next row element.] ADV(D,C); L ← PNORMF(D); u ← ICOMP(L,K); if u = 1, (erase K; K ← L); if u ≠ 1, erase L; if C ≠ (), go to (3).

(4) [Recompute coefficient bound.] L ← IPROD(J,K); erase J,K; J ← L; if B ≠ (), go to (2).

(5) [Final computations for bound.] K ← IFACT(m); L ← IPROD(K, J); erase K,J; J ← L; return J.

From Lemma 2.4.1 we have that $J' = m!(\Pi_{i=1}^{m} K_i)$ is a bound on norm(det(A)), where $K_i = \max_{1 \le j \le m}$ norm (A(i,j)). In step (1) J is set to 2. In steps (2)-(4) the K_i are computed. Thus, in the i^{th} execution of step (2) K is set to zero and in the next m executions of step (3) $K = K_i$ is computed, whereupon J is replaced by $J \cdot K_i$ in the i^{th} execution of step (4). When B = () in step (4), $J = 2(\Pi_{i=1}^{m} K_i)$ has been computed. In step (5) J is replaced by m!J and returned. (See [COG70].) Thus, J = 2J' and so J/2 is a bound on the magnitudes of the integer coefficients of det(A).

Theorem 4.6.7. Let $T_{CBDET}(m,d,\bar{m}_s)$ be the maximum computing time of CBDET for $A \in M(m,m,P(d,\bar{m}_s))$, $s \ge 0$. Then T_{CBDET} ≤

$m^2(\log d)(\log md + \Pi_{i=1}^{s}\mu_i) + m^2(\log m)^2$.

Proof: From the preceding remarks, we clearly have $N_1 = N_5 = 1$, $N_2 = N_4 = m$, and $N_3 = m^2$. So, $T_1 \sim T_{LENGTH}(m) \sim m$. Since $T_{2,j} \sim 1$, $T_2 \sim N_2 = m$. In each execution of step (3) $T_{3,j} \sim T_{PNORMF}(d,\bar{m}_s) \precsim (\log d)(\Pi_{i=1}^{s}\mu_i)$, since this function dominates the times for ICOMP and the erasures, which are $\precsim \log d$. So, $T_3 \precsim m^2(\log d)(\Pi_{i=1}^{s}\mu_i)$. $T_{4,j} \sim T_{IPROD}(2d^j,d) \precsim (\log 2d^j)(\log d) \sim j(\log d)^2$, since $|J| \leq 2d^j$ and $|K| \leq d$. Hence, $T_4 \precsim \sum_{j=1}^{m} j(\log d)^2 \precsim m^2(\log d)^2$. For step (5), $T_5 \sim T_{IFACT}(m) + T_{IPROD}(m^m,2d^m) \precsim m^2(\log m)^2 + m^2(\log m)(\log d) = m^2(\log m)(\log m + \log d)$. Combining these dominating functions, we have $T_{CBDET}(m,d,\bar{m}_s) \precsim T_3' + T_4' + T_5' = m^2(\log d)(\Pi_{i=1}^{s}\mu_i) + m^2(\log d)^2 + m^2(\log m)(\log m + \log d) = m^2(\log d)(\log m + \log d + \Pi_{i=1}^{s}\mu_i) + m^2(\log m)^2 = m^2(\log d)(\log md + \Pi_{i=1}^{s}\mu_i) + m^2(\log m)^2$.//

We now present the main algorithm for computing determinants.

Algorithm 4.6.8. (PDET) A Modular Algorithm for Determinant Calculation

Input: An m by m nonzero matrix A over $I[x_1,\ldots,x_s]$, $s \geq 0$.

Output: D = det(A) obtained by D = PDET(A).

Note that this algorithm requires the list PRIME of distinct odd primes.

(1) [Initialize.] D ← 0; V ← MVLIST(A); s ← LENGTH(V); J ←

CBDET(A); I ← PFA(1,0); L ← PRIME; h ← 0.

(2) [Apply mod-p mapping φ_p.] If L = (), (print error message; stop); ADV(p,L); A* ← MMOD(p,A,s); if MZERO(A*) = 1, (erase A*; D* ← 0; go to (4)).

(3) [Apply evaluation mapping algorithm.] D* ← CPDET(p,A*).

(4) [Apply Garner's Method.] If D* ≠ 0, h ← 1; if h = 0, go to (5); D' ← CPGARN(I,D,p,D*,V); erase D; if V≠0, erase D*;D←D'.

(5) [Integer coefficient bound test.] T ← PFA(p,0); U ← IPROD(I,T); erase I,T; I ← U; if ICOMP(I,J) ≠ 1, go to (2).

(6) [Final erasures and return.] Erase V,I,J; return D.

This algorithm is analogous to MMPY, as was mentioned earlier. The initialization, which is accomplished in steps (1)-(3) of MMPY, is done in step (1) in PDET. The similarity of the two algorithms then proceeds step-by-step; that is, steps (2)-(6) of PDET correspond to steps (4)-(8) of MMPY. Some particular aspects of PDET should be explained. We note that in step (1) an L-integer coefficient bound J on the magnitudes of the integer coefficients of det(A) is computed. In fact, J/2 is also a bound. I is set to the L-integer 1. Then a sequence of iterations of steps (2)-(5) follows, each accomplishing an application of a mod-p mapping φ_p, computing a determinant in $GF(p)[x_1,\ldots,x_s]$, applying Garner's Method, and testing I against J. In step (5), before the coefficient

bound test, I is replaced by p·I. At the k^{th} iteration $I = p_1 \cdots p_{k-1}$ is used in step (4) and then I is recomputed as $I = p_1 \cdots p_k$ for the test in step (5).

Suppose the elements of A are in $P(d,\bar{m}_s)$, $s \geq 0$. From Lemma 2.4.1 we find that $\text{norm}(\det(A)) \leq m!d^m \leq (md)^m$ and, by Lemma 2.5.1, $\deg_{x_i}(\det(A)) \leq mm_i < m\mu_i$. Hence, $\det(A) \in P(md)^m, mm_1, \ldots, mm_s)$. It can easily be shown that J is computed such that $J/2 \leq (md)^m$. If k primes p_i are used, then on the $(k-1)^{th}$ iteration $I = p_1 \cdots p_{k-1} \leq J$ and on the k^{th} iteration $I = p_1 \cdots p_k > J$. Thus, $k-1 \sim \log p_1 \cdots p_{k-1} \leq \log J \leq \log(2(md)^m) = \log 2 + m(\log md) \sim m(\log md)$. Hence, $k \lesssim m(\log md)$.

Similar to CPDET, a special test is used to omit the application of Garner's Method when $r < m$ (i.e., $\det(A) = 0$). Hence, in step (1) flag h is set to 0 and is not reset to 1 in step (4) until $D^* \neq 0$ is obtained. Garner's Method will then be applied in every subsequent iteration.

Theorem 4.6.9. Let $T_{PDET}(m,r,d,\bar{m}_s)$ be the maximum computing time of PDET for $A \in M(m,m,P(d,\bar{m}_s))$ of rank r, for $s \geq 0$. Then, for $s = 0$, $T_{PDET} \lesssim m^3(\log md)(\log d + r)$ and, for $s = 1$, $T_{PDET} \lesssim m^3(\log md)\mu_1(\log d + mr + m\mu_1)$. For $s \geq 2$, $T_{PDET} \lesssim m^3(\log md)(\prod_{i=1}^{s}\mu_i)(\log d + m^s r + m\sum_{i=1}^{s}\mu_i m^{s-i})$, if $r < m$, and $T_{PDET} \lesssim m^{s+2}(\log md)(\prod_{i=1}^{s}\mu_i)(\log d + m^2 + m\mu_1 + \sum_{i=2}^{s}\mu_i)$, if

$r = m$.

Proof: The following maximum computing time table is derived for PDET. Note that the i-column specifies the class of matrices to which the corresponding dominating functions apply.

i	N_i	$T_{i,j}$	T_i
1	1	-	$m^2(\log d)(\log md + \Pi_{i=1}^{s}\mu_i) + m^2(\log m)^2$
2	$m(\log md)$	$m^2(\log d)(\Pi_{i=1}^{s}\mu_i)$	$m^3(\log d)(\log md)(\Pi_{i=1}^{s}\mu_i)$
3:s=0	$m(\log md)$	$m^2 r$	$m^3 r(\log md)$
3:s=1	$m(\log md)$	$m^3\mu_1(r+\mu_1)$	$m^4(\log md)\mu_1(r+\mu_1)$
3:s≥2 r<m	$m(\log md)$	$m^3(\Pi_{i=1}^{s}\mu_i)(m^{s-1}r + \sum_{i=1}^{s}\mu_i m^{s-i})$	$m^4(\log md)(\Pi_{i=1}^{s}\mu_i)(m^{s-1}r + \sum_{i=1}^{s}\mu_i m^{s-i})$
3:s≥2 r=m	$m(\log md)$	$m^{s+1}(\Pi_{i=1}^{s}\mu_i)(m^2 + m\mu_1 + \sum_{i=2}^{s}\mu_i)$	$m^{s+2}(\log md)(\Pi_{i=1}^{s}\mu_i)(m^2 + m\mu_1 + \sum_{i=2}^{s}\mu_i)$
4:r<m	$m(\log md)$	1	$m(\log md)$
4:r=m	$m(\log md)$	$jm^s(\Pi_{i=1}^{s}\mu_i)$	$m^{s+2}(\log md)^2(\Pi_{i=1}^{s}\mu_i)$
5	$m(\log md)$	j	$m^2(\log md)^2$
6	1	-	$m(\log md)$

The values of N_i' should be clear from the preceding discussion. Then $T_1 \sim T_{CBDET}(m,d,\bar{m}_s) \lesssim m^2(\log d)(\log md + \Pi_{i=1}^{s}\mu_i) + m^2(\log m)^2$. For step (2), $T_{2,j} \sim T_{MMOD}(m,m,d,\bar{m}_s)$

$\lesssim m^2(\log d)(\Pi_{i=1}^{s}\mu_i)$. In step (3) the dominating functions for the several cases of CPDET are used, with rank(A*) = r* replaced by r = rank(A), since r* ≤ r for each mod-p mapping. In step (4), when r < m, mo applications of Garner's Method occur and so $T_{4,j} \sim 1$. If in step (4) r = m, then $T_{4,j} \sim T_{CPGARN}(p_1\cdots p_{j-1}, mm_1, \ldots, mm_s) \lesssim (\log p_1\cdots p_{j-1})\, m^s \Pi_{i=1}^{s}\mu_i \lesssim jm^s(\Pi_{i=1}^{s}\mu_i)$. Thus, if r < m, $T_4 \lesssim m(\log md)$ and, if r = m, $T_4 \lesssim \sum_{j=1}^{k} jm^s(\Pi_{i=1}^{s}\mu_i) \lesssim k\cdot m(\log md)\, m^s(\Pi_{i=1}^{s}\mu_i) \lesssim m(\log md)\cdot m(\log md)m^s(\Pi_{i=1}^{s}\mu_i) = m^{s+2}(\log md)(\Pi_{i=1}^{s}\mu_i)$. $T_{5,j} \lesssim T_{IPROD}(p_1\cdots p_{j-1}, p_j) \lesssim (\log p_1\cdots p_{j-1})(\log p_j) \lesssim j$. Hence, $T_5 \lesssim \sum_{j=1}^{k} j \lesssim k\cdot m(\log md) \lesssim m^2(\log md)^2$. Finally, in step (6) the time to erase I and J is $\lesssim \log p_1\cdots p_k \sim k \lesssim m(\log md) = T_6'$.

We may now obtain dominance relations for the maximum computing times for PDET. For s ≥ 0, $T_1' = m^2(\log d)(\log md) + m^2(\log d)(\Pi_{i=1}^{s}\mu_i) + m^2(\log m)^2 \lesssim m^3(\log d)(\log md)(\Pi_{i=1}^{s}\mu_i) = T_2'$, since each term of T_1' is clearly dominated by T_2'. Also, $T_6' \lesssim T_5' = m^2(\log m)(\log md) + m^2(\log d)(\log md) \lesssim T_2'$, for each term of T_5' is dominated by T_2'. Thus, for s ≥ 0, $T_{PDET} \sim T_2' + T_3' + T_4'$.

For s = 0, $T_4' \lesssim m^2(\log md)^2 = T_5' \lesssim T_2'$. Thus, $T_{PDET}(m, r, d, \bar{m}_o) \lesssim T_2' + T_3' = m^3(\log d)(\log md) + m^3 r(\log md) = m^3(\log md)(\log d + r)$. For s = 1, $T_4' \lesssim m^3(\log md)^2\mu_1 = m^3(\log m)(\log md)\mu_1 + m^3(\log d)(\log md)\mu_1 \lesssim m^4(\log md)\mu_1(r+\mu_1) + m^3(\log d)(\log md)\mu_1 = T_3' + T_2'$. Hence, for s = 1, $T_{PDET} \sim T_2' + T_3' = m^3(\log md)$

$\mu_1(\log d + mr + m\mu_1)$.

Now suppose $s \geq 2$. If $r < m$, then $T_4' \precsim T_2'$ and so $T_{PDET} \precsim T_2' + T_3' = m^3(\log d)(\log md)(\Pi_{i=1}^{s}\mu_i) + m^4(\log md)(\Pi_{i=1}^{s}\mu_i)(m^{s-1}r + \sum_{i=1}^{s}\mu_i m^{s-i}) = m^3(\log md)(\Pi_{i=1}^{s}\mu_i)(\log d + m(m^{s-1}r + \sum_{i=1}^{s}\mu_i m^{s-i}) = m^3(\log md)(\Pi_{i=1}^{s}\mu_i)(\log d + m^s r + m\sum_{i=1}^{s}\mu_i m^{s-i})$. If $r = m$, then $T_2' \precsim T_4'$ and so $T_{PDET} \precsim T_3' + T_4' = m^{s+2}(\log md)(\Pi_{i=1}^{s}\mu_i)(m^2 + m\mu_1 + \sum_{i=2}^{s}\mu_i) + m^{s+2}(\log md)^2(\Pi_{i=1}^{s}\mu_i) = m^{s+2}(\log md)(\Pi_{i=1}^{s}\mu_i)(\log md + m^2 + m\mu_1 + \sum_{i=2}^{s}\mu_i) \sim m^{s+2}(\log md)(\Pi_{i=1}^{s}\mu_i)(\log d + m^2 + m\mu_1 + \sum_{i=2}^{s}\mu_i)$. Note that we have applied the fact that $\log md + m^2 = \log m + \log d + m^2 \sim \log d + m^2$.//

The following result parallels Corollary 4.6.5 in obtaining more compact but cruder dominance relations for the maximum computing times of PDET. The definition $\mu = \max \mu_i$ is applied.

Corollary 4.6.10. The maximum computing time functions of PDET satisfy: for $s = 0$, $T_{PDET} \precsim m^3(\log md)(\log d + r)$ and for $s \geq 1$, $T_{PDET} \precsim m^3\mu^s(\log md)(\log d + m^s(r+\mu))$, if $r < m$, and $T_{PDET} \precsim m^{s+2}\mu^s(\log md)(\log d + m(m+\mu))$, if $r = m$.

Proof: If $s \geq 1$ and $r < m$, $T_{PDET} \precsim m^3(\log md)(\Pi_{i=1}^{s}\mu)(\log d + m^s r + m\mu\sum_{i=1}^{s}m^{s-i}) \sim m^3\mu^s(\log md)(\log d + m^s r + m\mu\cdot m^{s-1}) = m^3\mu^s(\log md)(\log d + m^s(r+\mu))$. The cases $r = m$ also follow easily.//

4.7. Algorithms for Matrix Inversion and Null Space Basis Generation

The modular algorithm for solving linear equations, presented in earlier sections, may be used for specialized applications. Two of the more important applications are the computation of matrix inverses and null space basis generation. In this section algorithms are presented for performing these computations. We begin with matrix inversion.

Let A be an m by m matrix of integers or polynomials over the integers and let r = rank(A). Assuming r = m, A has an inverse (D,Y), where $D \in I[x_1,\ldots,x_s]$ and $Y \in M(m,m, I[x_1,\ldots,x_s])$. Thus, if B is the m by m identity matrix over $I[x_1,\ldots,x_s]$, then $AY = D \cdot B$. Let C = (A,B) be the augmented matrix of the linear system of equations AX = B and note that rank(C) = rank(A) = m. Thus, when r = m, the system C is consistent and the list G = (D,Y,Z) is obtained by G = PLES(m,C), where AY = DB and Z = (). The pair (D,Y) is an inverse of A. However, if r < m, then rank(A) < rank(C), the system is inconsistent, and G = () is obtained. Thus, one need only construct C = (A,B), where B is the m by m identity matrix, and input (m,C) to PLES to determine if A is nonsingular and, if it is, to compute an inverse for A. This is accomplished by the following algorithm.

Algorithm 4.7.1. (MINV) Matrix Inversion

Input: An m by m nonzero matrix A over $I[x_1,\ldots,x_s]$, $s \geq 0$.

Output: The list X obtained by X = MINV(A). If rank(A) = m, X is the list (D,Y) representing a matrix inverse for A and, if rank(A) < m, X is the null list.

(1) [Initialize.] T ← A; C ← (); j ← 0; P ← PFA(1,0); m ← LENGTH(T); L ← MVLIST(T).

(2) [Construct integer or polynomial 1.] If L = (), go to (3); DECAP(V,L); P ← PFL(V,PFL(P,PFA(0,0))); go to (2).

(3) [Begin next row of augmented matrix C.] ADV(U,T); R ← (); j ← j+1.

(4) [Prefix element of A to row R.] ADV(W,U); R ← PFL(BORROW(W),R); if U ≠ (), go to (4).

(5) [Suffix elements of identity matrix to row R.] k ← j-1; if k > 0, do R ← PFL(0,R) k times; R ← PFL(BORROW(P),R); k ← m-j; if k > 0, do R ← PFL(0,R) k times ; C ← PFL(INV(R),C); if T ≠ (), go to (3); C ← INV(C).

(6) [Compute a matrix inverse.] X ← PLES(m,C); if X = (), go to (7); T = TAIL(X); U = TAIL(T); SSUCC(0,T); erase U.

(7) [Return.] Erase P,C; return X.

Steps (1)-(5) of this algorithm are concerned with constructing the matrix C = (A,B) for input to PLES in step (6), where B is the m by m identity matrix over $I[x_1,\ldots,x_s]$. This requires constructing the identity P = $1 \in I[x_1,\ldots,x_s]$

in step (2). In step (4) the elements of A are borrowed and appended to a row of C after which the elements of the row of the identity matrix are appended, which requires borrowing P for each row. The elements of each row are generated in the order 1,2,...,2m and the rows in the order 1,2,...,m. Thus, inversions of each row are required and also inversion of the list C to obtain the augmented system matrix C. Step (6) applies PLES either to discover that A is singular (i.e., X = ()) or to obtain a matrix inverse from the list X = (D, Y,Z) by redefining X as the list (D,Y).

We now obtain maximum computing time dominance relations for MINV, assuming that in the application of PLES no rejected mod-p or evaluation mappings occur and no substitution test failures occur.

Theorem 4.7.2. Let $T_{MINV}(m,r,d,\bar{m}_s)$ be the maximum computing time of MINV for $A \in M(m,m,P(d,\bar{m}_s))$ of rank r, for $s \geq 0$, assuming that no rejected mappings occur in applying PLES and no substitution test failures occur. Then $T_{PLES} \lesssim m^{s+3}(\log md)(\Pi_{i=1}^{s}\mu_i)(m+\log d + \sum_{i=1}^{s}\mu_i)$.

Proof: Step (1) is executed once and so $T_1 \sim T_{LENGTH}(m) \sim m$. The time for each of the s executions of step (2) is ~ 1 and, since s is not considered variable, $T_2 \sim 1$. Steps (3)-(5) construct C, the time to obtain and append each of the $2m^2$ elements being bounded. Hence, $T_3 + T_4 + T_5 \sim m^2$. For step

(6), $T_6 \sim T_{PLES}(m,2m,m,d,\bar{m}_s) \lesssim m^{s+3}(\log md)(\Pi_{i=1}^{s}\mu_i)(m + \log d + \sum_{i=1}^{s}\mu_i) = T_6'$, for $s \geq 0$, by applying Corollary 4.5.4. In step (7) the time for the erasure is $\sim m^2$. Clearly T_6' dominates the times for the other steps and so $T_{PLES} \lesssim T_6'$, giving the theorem.//

We note that a more compact dominating function for T_{MINV} is the same as that given in Corollary 4.5.5 for PLES.

Now consider null space basis generation for an m by n nonzero matrix A of integer or polynomial entries. Let C be the m by n+1 matrix whose first n columns constitute A and whose final column is a column B of zeros. Since rank(C) = rank(A), C represents a consistent system of linear equations AX = B. Hence, the list G = (D,Y,Z) is obtained by the reference G = PLES(n,C), where Y is a column matrix of zeros and Z is the null list, if rank(A) = n, or a null space basis for A, if rank(A) < n. All that remains to be done is to erase D,Y, and C and obtain Z. These ideas are implemented in the following algorithm.

Algorithm 4.7.3. (NULSP) Null Space Basis Generation

Input: An m by n nonzero matrix A over $I[x_1,\ldots,x_s]$, $s \geq 0$, of rank r.

Output: The list Z obtained by Z = NULSP(A). If r < n, Z is an n by n-r null space basis matrix for A. If r = n, Z is the null list.

(1) [Initialize.] T ← A; C ← ().

(2) [Begin next row of augmented matrix C.] ADV(U,T); R ← ().

(3) [Construct matrix C.] ADV(V,U); R ← PFL(BORROW(V), R); if U ≠ (), go to (3); R ← PFL(0,R); C ← PFL(INV(R), C); if T ≠ (), go to (2); C ← INV(C).

(4) [Obtain null space basis.] n ← LENGTH(FIRST(A)); G ← PLES(n,C); DECAP(D,G); DECAP(Y,G); DECAP(Z,G); erase D, Y,C; return Z.

Steps (1)-(3) construct the augmented system matrix C by borrowing the elements of A and suffixing a final zero to each row. It can be easily shown that the time to do this is ~ mn. The time for step (4) is clearly dominated by the time to apply PLES, which also dominates the time for steps (1)-(3). Thus, $T_6 \sim T_{PLES}(m,n,r,d,\bar{m}_s)$ from which dominance relations for the time for NULSP can be obtained, where we may replace n' = n+1 by n. (See Theorem 4.5.2.) This result is given in the following theorem.

Theorem 4.7.4. Let $T_{NULSP}(m,n,r,d,\bar{m}_s)$ be the maximum computing time of NULSP for $A \in M(m,n,P(d,\bar{m}_s))$, $s \geq 0$, of rank r, assuming that no rejected mappings occur and no false equality test successes occur. Then, if $s = 0$, $T_{NULSP} \lesssim (\log n + r(\log rd))[mn^2 + (m+n)r(\log rd)]$, and, for $s \geq 1$, $T_{NULSP} \lesssim (\log n + r(\log rd))r^s(\Pi_{i=1}^{s}\mu_i)[mn^2 + (m+n)r(n-r+1)(\log rd +$

$\sum_{i=1}^{s} \mu_i)]$.

More compact dominating functions for NULSP can be obtained from Corollaries 4.5.3-4.5.5 for PLES.

CHAPTER V.

EMPIRICAL COMPUTING TIMES

The algorithms of Chapters 3 and 4 have been programmed in ASA Fortran; the programs are listed in the appendix. These programs are to constitute the SAC-1 Linear Algebra System (see [COG71c]), which is one module of the SAC-1 System for Symbolic and Algebraic Calculation. In order to gain some insight into the practical computing capabilities of the algorithms, we now present a series of tables of empirical computing times, in particular, for PDET, PLES, and MINV. These algorithms are applied to matrices with randomly generated integer or polynomial (univariate and bivariate) entries. The cases considered were run on the UNIVAC 1108. Since the cases were run in a multiprogramming environment, precise computing times were difficult to obtain. Consequently, one should be aware that the times in the tables below may be as much as ten percent in error.

Tables 5.1, 5.4, and 5.7 give the times for computing determinants of m by m matrices. Tables 5.2, 5.5, and 5.8 give the times for solving single systems of linear equations with m equations and m unknowns. Tables 5.3, 5.6, and 5.9 give times for inverting m by m matrices. The first three tables were obtained for matrices of randomly generated

integers in the intervals $[-2^k, +2^k]$, k = 4,8,16,32.

Table 5.1

PDET Computing Times (sec.) for Integral Matrices

k \ m	5	10	15
4	0.12	0.51	1.89
8	0.14	0.82	2.84
16	0.21	1.30	4.20
32	0.39	2.33	8.08

Table 5.2

PLES Computing Times (sec.) for Integral Matrices

k \ m	5	10	15
4	0.23	1.09	3.21
8	0.38	1.56	4.49
16	0.57	3.12	8.56
32	1.11	6.58	18.45

Table 5.3

MINV Computing Times (sec.) for Integral Matrices

k \ m	5	10	15
4	0.47	3.26	10.37
8	0.71	4.76	18.35
16	1.10	8.96	33.88
32	2.08	21.86	106.98

The next three tables were obtained for matrices of randomly generated univariate polynomial entries with integer coefficients in the interval [-15, +15] and degrees at most t.

Table 5.4

PDET Computing Times (sec.) for Matrices of Univariate Polynomials

t \ m	4	6	8	10
1	0.20	0.82	1.58	3.91
2	0.32	1.47	2.97	7.78
3	0.53	2.37	4.72	12.35
4	0.68	3.22	6.16	16.42
5	0.81	3.92	7.92	21.56

Table 5.5

PLES Computing Times (sec.) for Matrices of Univariate Polynomials

t \ m	4	6	8	10
1	1.29	3.44	8.21	15.04
2	2.68	7.14	19.21	33.96
3	4.33	12.41	33.01	64.01
4	5.66	16.59	45.60	89.76
5	7.91	22.65	64.24	119.70

Table 5.6

MINV Computing Times (sec.) for Matrices of Univariate Polynomials

t \ m	4	6	8	10
1	2.39	8.95	28.89	60.78
2	4.62	18.84	66.93	151.25
3	7.43	32.15	114.32	--
4	11.25	47.39	--	--
5	14.46	--	--	--

The final three tables were obtained for randomly generated bivariate polynomial entries with integer coefficients in the interval [-15, +15] and degrees in each variable at most t.

Table 5.7

PDET Computing Times (sec.) for Matrices of Bivariate Polynomials

t \ m	3	4	5
1	0.48	0.96	3.30
2	1.79	3.28	12.01
3	3.32	14.64	28.60

Table 5.8

PLES Computing Times (sec.) for Matrices of Bivariate Polynomials

t \ m	3	4	5
1	5.79	11.52	24.79
2	24.80	61.29	115.95
3	46.22	129.63	263.75

Table 5.9

MINV Computing Times (sec.) for Matrices of Bivariate Polynomials

t \ m	3	4	5
1	9.78	26.32	57.89
2	33.17	91.97	242.67
3	62.14	199.31	--

REFERENCES

[ASA64] American Standards Association,"A Programming Language for Information Processing on Automatic Data Processing Systems", Comm. A.C.M., Vol. 7, No. 10 (Oct. 1964), pp. 591-625.

[BAR68] Bareiss, Erwin, H.,"Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination", Math. of Comp. Vol. 22, No. 103 (July 1968), pp. 565-578.

[BIG65] Birkhoff, Garrett and MacLane, Saunders, A Survey of Modern Algebra, Macmillan, New York, 1965.

[BOD59] Bodwig, E., Matrix Calculus, Interscience, New York, 1959.

[BOF66] Borosh, I. and Fraenkel, A. S., "Exact Solutions of Linear Equations with Rational Coefficients by Congruence Techniques", Math. of Comp., Vol 20, No. 93, Jan. 1966.

[BRO68] Brown, W. S.,"The Compleat Euclidean Algorithm", Bell Telephone Laboratories Report, June 1968.

[BRO71] Brown, W. S., "On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors", Jour. ACM, Vol. 18, No. 4 (Oct. 1971).

[COG67] Collins, George E., The SAC-1 List Processing System, University of Wisconsin Computing Center Technical Report, July 1967.

[COG68a] Collins, George E., and James R. Pinkert, The Revised SAC-1 Integer Arithmetic System, University of Wisconsin Computing Center Technical Report No. 9, Nov. 1968.

[COG68b] Collins, George E., The SAC-1 Polynomial System, University of Wisconsin Computing Center Technical Report No. 2, March 1968.

[COG68c] Collins, George E., The SAC-1 Rational Function System, University of Wisconsin Computing Center Technical Report No. 8, July 1968.

[COG69a] Collins, George E., L. E. Heindel, E. Horowitz, M. T. McClellan, and D. R. Musser, The SAC-1 Modular Arithmetic System, University of Wisconsin Computing Center Technical Report No. 10, June 1969.

[COG69b] Collins, George E.,"Computing Time Analyses for Some Arithmetic and Algebraic Algorithms", Proceedings of the 1968 Summer Institute on Symbolic Mathematical Computation (Robert G. Tobey, ed.), IBM Federal Systems Center, June 1969, pp. 195-232.

[COG70] Collins, George E., and Lee E. Heindel, The SAC-1 Polynomial Real Zero System, University of Wisconsin Computing Center Technical Report No. 18, Aug. 1970.

[COG71a] Collins, George E., "The Calculation of Multivariate Polynomial Resultants", Jour. ACM, Vol 18, No. 4 (Oct. 1971).

[COG71b] Collins, George E., The SAC-1 Polynomial Greatest Common Divisor and Resultant System. In preparation.

[COG71c] Collins, George E., and Michael T. McClellan, The SAC-1 Polynomial Linear Algebra System. In preparation.

[FOX64] Fox, L., An Introduction to Numerical Linear Algebra, Clarendon Press, Oxford, 1964.

[GAF59] Gantmacher, F. R., Matrix Theory, Vol. 1, Chelsea, New York, 1959.

[HOG69] Howell, J. A. and Gregory, R. T., "An Algorithm for Solving Linear Algebraic Equations Using Residue Arithmetic", BIT, 9 (1969), pp. 200-234, 324-337.

[KND68] Knuth, D. E., The Art of Computer Programming, Vol. 1 (Fundamental Algorithms), Addison-Wesley, 1968.

[KND69] Knuth, D. E., The Art of Computer Programming, Vol. 2 (Seminumerical Algorithms), Addison-Wesley, 1969.

[LIP69] Lipson, John D., "Symbolic Methods for the Computer Solution of Linear Equations with Applications to Flow-graphs", Proceedings of the 1968 Summer Institute on Symbolic Mathematical Computation (Robert G. Tobey, ed.), June 1969, IBM Federal Systems Center, pp. 233-303.

[LUG62] Luther, H. A. and Guseman, L. F., Jr., "A Finite Sequentially Compact Process for the Adjoints of Matrices over Arbitrary Integral Domains", CACM, Vol. 5, No. 8, (Aug. 1962).

[NEW67] Newman, Morris, "Solving Equations Exactly", J. of Res., N.B.S., Vol. 71B, No. 4, Oct. - Dec., 1967.

[ROS52] Rosser, J. Barkley, "A Method of Computing Exact Inverses of Matrices with Integer Coefficients", J. of Res., N.B.S., Vol. 49, No. 5 (Nov. 1952), pp. 349-358.

[TAI61] Takahasi, H. and Ishibashi, Y., "A New Method for 'Exact Calculation' by a Digital Computer", Information Processing in Japan, Vol. 1 (1961), pp. 28-42.

INDEX TO ALGORITHMS

APPENDIX: FORTRAN PROGRAM LISTINGS

```
      INTEGER FUNCTION CBDET(A)
      INTEGER A,B,C,D,J,K,L,M,U
      INTEGER ICOMP,IFACT,IPROD,LENGTH,PFA,PNORMF
10    B=A
      M=LENGTH(B)
      J=PFA(2,0)
20    CALL ADV(C,B)
      K=0
30    CALL ADV(D,C)
      L=PNORMF(D)
      U=ICOMP(L,K)
      IF (U.NE.1) GO TO 31
      CALL ERLA(K)
      K=L
      GO TO 32
31    CALL ERLA(L)
32    IF (C.NE.0) GO TO 30
40    L=IPROD(J,K)
      CALL ERLA(J)
      CALL ERLA(K)
      J=L
      IF (B.NE.0) GO TO 20
50    K=IFACT(M)
      CBDET=IPROD(K,J)
      CALL ERLA(K)
      CALL ERLA(J)
      RETURN
      END
```

```
      INTEGER FUNCTION CDET(P,A)
      INTEGER A,BP,BPP,D,E,F,G,H,I,T,U
      INTEGER CDIF,CPROD,CRECIP,FIRST,INV,PFL,TAIL
1     BP=A
      D=1
      I=0
2     BPP=BP
      BP=0
3     CALL DECAP(F,BPP)
      CALL DECAP(E,F)
      IF (E.NE.0) GO TO 4
      IF (F.NE.0) BP=PFL(F,BP)
      I=I+1
      IF (BPP.NE.0) GO TO 3
      CDET=0
      CALL MCPERS(BP)
      RETURN
4     D=CPROD(P,E,D)
```

```
      IF (F.EQ.0) GO TO 8
      E=CRECIP(P,E)
      G=F
5     CALL ALTER(CPROD(P,E,FIRST(G)),G)
      G=TAIL(G)
      IF (G.NE.0) GO TO 5
6     IF (BPP.EQ.0) GO TO 8
      CALL DECAP(G,BPP)
      CALL DECAP(J,G)
      BP=PFL(G,BP)
      IF (U.EQ.0) GO TO 6
      H=F
7     T=CDIF(P,FIRST(G),CPROD(P,U,FIRST(H)))
      CALL ALTER(T,G)
      G=TAIL(G)
      H=TAIL(H)
      IF (G.NE.0) GO TO 7
      GO TO 6
8     CALL ERLA(F)
      BP=INV(BP)
      IF (BP.NE.0) GO TO 2
      U=0
      CALL QR(U,I,2)
      IF (I.EQ.1) D=P-D
      CDET=D
      RETURN
      END



      INTEGER FUNCTION CPDET(PP,AA)
      INTEGER A,AA,ASTAR,B,D,DP,DSTAR,E,H,I,J,M,P,PP,RET,S,T,U
      INTEGER CDET,CDIF,CPDEG,CPINT,CPPROD,DBRRE,LENGTH,MCPEVL,MCPNV
      INTEGER MZERO,PFA
      A=AA
      P=PP
      RET=1
      GO TO 10
1     CPDET=D
      RETURN
10    IF (MZERO(A).EQ.0) GO TO 11
      D=0
      CALL MCPERS(A)
      GO TO (1,51), RET
11    S=MCPNV(A)
20    IF (S.GT.0) GO TO 30
      D=CDET(P,A)
      GO TO (1,51), RET
30    H=0
      M=LENGTH(A)
      J=0
      DO 31 I=1,M
31    J=PFA(M+1-I,J)
      B=DBRRE(J,A)
      CALL ERLA(J)
```

```
      I=0
      E=PFA(0,PFA(1,0))
      D=0
40    IF (I.LT.P) GO TO 42
      PRINT 41
41    FORMAT(52H ELEMENTS OF GF(P) EXHAUSTED. ALGORITHM CPDET FAILS.)
      STOP
42    ASTAR=MCPEVL(P,A,I,S)
50    CALL STACK3(A,S,I)
      CALL STACK3(D,E,B)
      CALL STACK2(H,RET)
      RET=2
      A=ASTAR
      GO TO 10
51    DSTAR=D
      CALL UNSTK2(H,RET)
      CALL UNSTK3(D,E,B)
      CALL UNSTK3(A,S,I)
      IF (DSTAR.NE.0) H=1
      IF (H.EQ.0) GO TO 70
60    DP=CPINT(P,D,I,DSTAR,E,S)
      CALL CPERAS(D)
      IF (S.GT.1) CALL CPERAS(DSTAR)
      D=DP
70    IF (CPDEG(E).NE.B) GO TO 80
      CALL MCPERS(A)
      CALL ERLA(E)
      GO TO (1,51), RET
80    T=PFA(1,PFA(1,PFA(CDIF(P,0,I),0)))
      U=CPPROD(P,T,E)
      CALL CPERAS(T)
      CALL CPERAS(E)
      E=U
      I=I+1
      GO TO 40
      END


      INTEGER FUNCTION CPRRE(P,CC)
      INTEGER A,B,C,CC,CSTAR,D,DP,DSTAR,E,EP,H,I,ISTAR,J,JSTAR,N,P
      INTEGER R,RET,RSTAR,S,T,U,V,VP,VSTAR,W,WSTAR,Z,ZP
      INTEGER CDIF,CPDEG,CPDIF,CPINT,CPNEG,CPPROD,CRRE,DBRRE,FIRST
      INTEGER LENGTH,MCPEQ,MCPEVL,MCPINT,MCPMPY,MCPNV,MTRAN,MZCON
      INTEGER MZERO,NULCON,PFA,PFL,VCOMP
      C=CC
      RET=1
      GO TO 10
1     CPRRE=W
      RETURN
10    S=MCPNV(C)
      IF (S.GT.0) GO TO 20
      W=CRRE(P,C)
      GO TO (1,41), RET
20    R=0
```

```
      A=P
      H=0
30    A=A-1
      IF (A.GE.0) GO TO 32
      PRINT 31
31    FORMAT(52H ELEMENTS OF GF(P) EXHAUSTED. ALGORITHM CPRRE FAILS.)
      STOP
32    CSTAR=MCPEVL(P,C,A,S)
      IF (MZERO(CSTAR).EQ.0) GO TO 40
      CALL MCPERS(CSTAR)
      GO TO 30
40    CALL STACK3(C,S,A)
      CALL STACK3(R,I,J)
      CALL STACK3(D,W,E)
      CALL STACK3(H,B,RET)
      C=CSTAR
      RET=2
      GO TO 10
41    WSTAR=W
      CALL UNSTK3(H,B,RET)
      CALL UNSTK3(D,V,E)
      CALL UNSTK3(R,I,J)
      CALL UNSTK3(C,S,A)
      CALL DECAP(JSTAR,WSTAR)
      CALL DECAP(ISTAR,WSTAR)
      CALL DECAP(DSTAR,WSTAR)
      CALL DECAP(VSTAR,WSTAR)
50    RSTAR=LENGTH(JSTAR)
      IF (RSTAR-R) 60,51,70
51    IF (VCOMP(JSTAR,J)) 70,52,60
52    IF (VCOMP(ISTAR,I)) 70,53,60
53    CALL ERLA(JSTAR)
      CALL ERLA(ISTAR)
      GO TO 80
60    CALL ERLA(JSTAR)
      CALL ERLA(ISTAR)
      IF (VSTAR.NE.0) CALL MCPERS(VSTAR)
      IF (S.GT.1) CALL CPERAS(DSTAR)
      GO TO 30
70    IF (R.EQ.0) GO TO 71
      CALL ERLA(J)
      CALL ERLA(I)
      CALL CPERAS(D)
      IF (V.NE.0) CALL MCPERS(V)
      CALL CPERAS(E)
71    R=RSTAR
      J=JSTAR
      I=ISTAR
      D=0
      V=0
      IF (VSTAR.NE.0) V=MZCON(VSTAR)
      E=PFA(0,PFA(1,0))
80    DP=CPINT(P,D,A,QSTAR,E,S)
      IF (S.GT.1) CALL CPERAS(QSTAR)
      IF (V.EQ.0) GO TO 81
```

```
      VP=MCPINT(P,V,A,VSTAR,E,S)
      CALL MCPERS(VSTAR)
81    T=PFA(1,PFA(1,PFA(CDIF(P,0,A),0)))
      EP=CPPROD(P,T,E)
      CALL CPERA$(T)
      CALL CPERAS(E)
      E=EP
      IF (H.EQ.0) GO TO 90
      CALL CPERAS(D)
      D=DP
      IF (V.EQ.0) GO TO 120
      CALL MCPERS(V)
      V=VP
      GO TO 120
90    T=CPDIF(P,D,DP)
      CALL CPERAS(D)
      D=DP
      U=1
      IF (V.EQ.0) GO TO 91
      U=MCPEQ(P,V,VP)
      CALL MCPERS(V)
      V=VP
91    IF (U.EQ.1 .AND. T.EQ.0) GO TO 100
      CALL CPERAS(T)
      GO TO 30
100   N=LENGTH(FIRST(C))
      IF (R.NE.N) GO TO 101
      H=1
      GO TO 110
101   DP=CPNEG(P,D)
      Z=NULCON(N,J,DP,V)
      ZP=MTRAN(Z)
      T=MCPMPY(P,C,ZP)
      U=MZERO(T)
      CALL MCPERS(ZP)
      CALL CPERAS(DP)
      CALL MCPERS(Z)
      CALL MCPERS(T)
      IF (U.EQ.0) GO TO 30
      H=1
110   B=DBRRE(J,C)
120   IF (CPDEG(E).LE.B) GO TO 30
130   CALL CPERAS(E)
      CALL MCPERS(C)
      W=PFL(J,PFL(I,PFL(D,PFL(V,0))))
140   GO TO (1,41), RET
      END
```

```
      INTEGER FUNCTION CRRE(P,A)
      INTEGER A,B,BP,BPP,D,E,F,G,H,HBAR,I,IP,IPP,J,K,L,M,N,P
      INTEGER S,T,U,V,W,X,Z
      INTEGER CDIF,CONC,CPROD,CRECIP,FIRST,INV,LENGTH,PFA,PFL,TAIL
10    BP=A
      M=LENGTH(BP)
      N=LENGTH(FIRST(BP))
      I=0
      J=0
      D=1
      V=0
      B=0
      IP=0
      DO 11 K=1,M
      U=M+1-K
11    IP=PFA(U,IP)
      Z=0
20    Z=Z+1
      BPP=BP
      BP=0
      IPP=IP
      IP=0
30    CALL DECAP(F,BPP)
      CALL DECAP(E,F)
      CALL DECAP(K,IPP)
      IF (E.NE.0) GO TO 40
      IP=PFA(K,IP)
      IF (F.NE.0) BP=PFL(F,BP)
      IF (BPP.NE.0) GO TO 30
      GO TO 80
40    D=CPROD(P,E,D)
      J=PFA(Z,J)
      I=PFA(K,I)
      B=PFL(F,B)
      IF (F.EQ.0) GO TO 80
      E=CRECIP(P,E)
      G=F
50    CALL ALTER(CPROD(P,E,FIRST(G)),G)
      G=TAIL(G)
      IF (G.NE.0) GO TO 50
60    IF (BPP.EQ.0) GO TO 80
      CALL DECAP(G,BPP)
      CALL DECAP(J,G)
      BP=PFL(G,BP)
      IF (U.EQ.0) GO TO 60
      H=F
70    T=CDIF(P,FIRST(G),CPROD(P,U,FIRST(H)))
      CALL ALTER(T,G)
      G=TAIL(G)
      H=TAIL(H)
      IF (G.NE.0) GO TO 70
      GO TO 60
80    BP=INV(BP)
      IP=CONC(INV(IP),IPP)
      IF (BP.NE.0) GO TO 20
```

```
      I=CONC(INV(I),IP)
      IF (BPP.NE.0) CALL ERASE(BPP)
90    IF (B.NE.0) GO TO 91
      U=V
      V=0
      GO TO 150
91    CALL DECAP(H,B)
      H=INV(H)
      HBAR=0
      M=0
      Z=N
      L=J
100   IF (H.EQ.0) GO TO 110
      CALL DECAP(J,H)
      K=FIRST(L)
      IF (Z.NE.K) GO TO 101
      M=PFA(U,M)
      L=TAIL(L)
      GO TO 102
101   HBAR=PFA(U,HBAR)
102   Z=Z-1
      GO TO 100
110   T=V
      HBAR=INV(HBAR)
120   IF (M.EQ.0) GO TO 140
      CALL DECAP(U,M)
      IF (T.EQ.0) GO TO 120
      CALL ADV(S,T)
      H=HBAR
130   IF (S.EQ.0) GO TO 120
      CALL ADV(X,S)
      W=FIRST(H)
      W=CDIF(P,W,CPROD(P,U,X))
      CALL ALTER(W,H)
      H=TAIL(H)
      GO TO 130
140   IF (HBAR.NE.0) V=PFL(HBAR,V)
      GO TO 90
150   IF (U.EQ.0) GO TO 170
      CALL DECAP(T,U)
      S=0
160   CALL DECAP(E,T)
      G=CPROD(P,D,E)
      S=PFA(G,S)
      IF (T.NE.0) GO TO 160
      V=PFL(S,V)
      GO TO 150
170   V=INV(V)
      J=INV(J)
      CRRE=PFL(J,PFL(I,PFA(D,PFL(V,0))))
      RETURN
      END
```

```
      INTEGER FUNCTION DBRRE(J,C)
      INTEGER C,D,E,F,G,H,I,J,K,L,T,U,X,Y
      INTEGER CPDEG,FIRST,MTRAN,PFA,TAIL
10    X=MTRAN(C)
      Y=X
      K=J
      E=0
      F=0
      L=0
      I=0
20    IF (Y.EQ.0) GO TO 50
      CALL ADV(U,Y)
      I=I+1
      IF (K.EQ.0) GO TO 40
      IF (I.NE.FIRST(K)) GO TO 40
      G=0
30    CALL ADV(T,J)
      IF (T.EQ.0) GO TO 31
      D=CPDEG(T)
      IF (D.GT.G) G=D
31    IF (U.NE.0) GO TO 30
      L=PFA(G,L)
      K=TAIL(K)
      GO TO 20
40    CALL ADV(T,J)
      IF (T.EQ.0) GO TO 41
      D=CPDEG(T)
      IF (D.GT.F) F=D
41    IF (U.NE.0) GO TO 40
      GO TO 20
50    CALL DECAP(D,L)
      E=E+D
      IF (D.LT.G) G=D
      IF (L.NE.0) GO TO 50
60    DBRRE=E
      H=F-G
      IF (H.GT.0) DBRRE=DBRRE+H
      CALL MCPERS(X)
      RETURN
      END


      INTEGER FUNCTION MCMPY(P,A,B)
      INTEGER A,B,C,D,E,P,R,S,T,V,W,X,Y
      INTEGER CINV,CPROD,CSUM,FIRST,PFA,PFL,TAIL
1     X=CINV(A)
      Y=CINV(B)
      C=0
      V=X
2     CALL ADV(R,V)
      D=0
      W=Y
3     S=R
      CALL ADV(T,W)
```

```
      E=0
4     E=CSUM(P,E,CPROD(P,FIRST(S),FIRST(T)))
      S=TAIL(S)
      T=TAIL(T)
      IF (S.NE.0) GO TO 4
      D=PFA(E,D)
      IF (W.NE.0) GO TO 3
      C=PFL(D,C)
      IF (V.NE.0) GO TO 2
5     CALL ERASE(X)
      CALL ERASE(Y)
      MCMPY=C
      RETURN
      END


      INTEGER FUNCTION MCPEVL(P,A,B,S)
      INTEGER A,B,C,D,P,S,U,W,X
      INTEGER CPEVAL,INV,PFA,PFL
1     X=A
      C=0
2     CALL ADV(W,X)
      D=0
3     CALL ADV(U,W)
      IF (U.NE.0) U=CPEVAL(P,U,B)
      IF (S.EQ.1) D=PFA(U,D)
      IF (S.NE.1) D=PFL(U,D)
      IF (W.NE.0) GO TO 3
4     C=PFL(INV(D),C)
      IF (X.NE.0) GO TO 2
5     MCPEVL=INV(C)
      RETURN
      END


      INTEGER FUNCTION MCPEQ(P,A,B)
      INTEGER A,B,C,P,S,T,X,Y
      INTEGER CPDIF,FIRST,TAIL
1     X=A
      Y=B
2     CALL ADV(S,X)
      CALL ADV(T,Y)
3     C=CPDIF(P,FIRST(S),FIRST(T))
      IF (C.NE.0) GO TO 4
      S=TAIL(S)
      T=TAIL(T)
      IF (S.NE.0) GO TO 3
      IF (X.NE.0) GO TO 2
      MCPEQ=1
      RETURN
4     CALL CPERAS(C)
      MCPEQ=0
      RETURN
      END
```

```
      SUBROUTINE MCPERS(A)
      INTEGER A,B,C,K,T
      INTEGER COUNT,FIRST,TYPE
1     T=TYPE(FIRST(A))
2     IF (A.EQ.0) RETURN
      K=COUNT(A)-1
      IF (K.EQ.0) GO TO 21
      CALL SCOUNT(K,A)
      RETURN
21    CALL DECAP(B,A)
3     K=COUNT(B)-1
      IF (K.EQ.0) GO TO 31
      CALL SCOUNT(K,B)
      GO TO 2
31    CALL DECAP(C,B)
      IF (T.EQ.1) CALL CPERAS(C)
      IF (B.NE.0) GO TO 3
      GO TO 2
      END


      INTEGER FUNCTION MCPINT(P,A,B,C,D,S)
      INTEGER A,B,C,D,E,F,H,P,S,T,U,V,W,X,Y
      INTEGER CPINT,INV,PFL
1     X=A
      Y=C
      H=0
2     CALL ADV(W,X)
      CALL ADV(T,Y)
      E=0
3     CALL ADV(U,W)
      CALL ADV(V,T)
      F=CPINT(P,U,B,V,D,S)
      E=PFL(F,E)
      IF (W.NE.0) GO TO 3
4     H=PFL(INV(E),H)
      IF (X.NE.0) GO TO 2
5     MCPINT=INV(H)
      RETURN
      END


      INTEGER FUNCTION MCPMDB(A,B)
      INTEGER A,B,E,R,S,T,U,V,W,X,Y,Z
      INTEGER CPDEG
1     X=A
      Y=B
      Z=0
2     CALL ADV(R,X)
      W=Y
3     S=R
      CALL ADV(T,W)
4     CALL ADV(U,S)
```

```
      CALL ADV(V,T)
      IF (U.EQ.0 .OR. V.EQ.0) GO TO 5
      E=CPDEG(U)+CPDEG(V)
      IF (E.GT.Z) Z=E
5     IF (S.NE.0) GO TO 4
      IF (W.NE.0) GO TO 3
      IF (X.NE.0) GO TO 2
      MCPMDB=Z
      RETURN
      END


      INTEGER FUNCTION MCPMPY(P,AA,BB)
      INTEGER A,AA,ASTAR,B,BB,BSTAR,C,CP,CSTAR,D,E,I,J,K,M,P,Q,RET,S,W
      INTEGER CDIF,CPPROD,FIRST,LENGTH,MCMPY,MCPEVL,MCPINT,MCPMDB
      INTEGER MCPNV,MZERO,PFA,PFL,TYPE
      A=AA
      B=BB
      RET=1
      GO TO 10
91    MCPMPY=C
      RETURN
10    M=LENGTH(A)
      Q=LENGTH(B)
20    IF (TYPE(FIRST(A)).NE.0) GO TO 30
      C=MCMPY(P,A,B)
      GO TO 90
30    C=0
      DO 32 I=1,M
      D=0
      DO 31 J=1,Q
31    D=PFL(0,D)
32    C=PFL(D,C)
      IF (MZERO(A).EQ.1 .OR. MZERO(B).EQ.1) GO TO 90
40    S=MCPNV(A)
      K=MCPMDB(A,B)
      I=0
      D=PFA(0,PFA(1,0))
50    IF (I.LT.P) GO TO 52
      PRINT 51
51    FORMAT(53H ELEMENTS OF GF(P) EXHAUSTED. ALGORITHM MCPMPY FAILS.
      STOP
52    ASTAR=MCPEVL(P,A,I,S)
      BSTAR=MCPEVL(P,B,I,S)
60    CALL STACK3(A,B,C)
      CALL STACK3(D,I,K)
      CALL STACK2(S,RET)
      A=ASTAR
      B=BSTAR
      RET=2
      GO TO 10
61    CSTAR=C
      ASTAR=A
      BSTAR=B
```

```
      CALL UNSTK2(S,RET)
      CALL UNSTK3(D,I,K)
      CALL UNSTK3(A,B,C)
      CALL MCPERS(ASTAR)
      CALL MCPERS(BSTAR)
70    CP=MCPINT(P,C,I,CSTAR,D,S)
      CALL MCPERS(C)
      CALL MCPERS(CSTAR)
      C=CP
      IF (I.LT.K) GO TO 80
      CALL CPERAS(D)
      GO TO 90
80    E=PFA(1,PFA(1,PFA(CDIF(P,0,I),0)))
      W=CPPROD(P,D,E)
      CALL CPERAS(D)
      CALL CPERAS(E)
      D=W
      I=I+1
      GO TO 50
90    GO TO (91,61),RET
      END


      INTEGER FUNCTION MCPNV(A)
      INTEGER A,B,C,D
      INTEGER CPNV,FIRST,TYPE
1     B=A
      IF (TYPE(FIRST(B)).EQ.1) GO TO 2
      MCPNV=0
      RETURN
2     CALL ADV(C,B)
3     CALL ADV(D,C)
      IF (D.NE.0) GO TO 4
      IF (C.NE.0) GO TO 3
      GO TO 2
4     MCPNV=CPNV(D)
      RETURN
      END


      INTEGER FUNCTION MDIF(A,B)
      INTEGER A,B,C,D,E,F,X,Y
      INTEGER FIRST,INV,PDIF,PFL,TAIL
1     X=A
      Y=B
      C=0
2     CALL ADV(E,X)
      CALL ADV(F,Y)
      D=0
3     D=PFL(PDIF(FIRST(E),FIRST(F)),D)
      E=TAIL(E)
      F=TAIL(F)
      IF (E.NE.0) GO TO 3
```

```
4     C=PFL(INV(D),C)
      IF (X.NE.0) GO TO 2
5     MDIF=INV(C)
      RETURN
      END


      INTEGER FUNCTION MEQ(A,B)
      INTEGER A,B,C,S,T,X,Y
      INTEGER FIRST,PDIF,TAIL
1     X=A
      Y=B
2     CALL ADV(S,X)
      CALL ADV(T,Y)
3     C=PDIF(FIRST(S),FIRST(T))
      IF (C.NE.0) GO TO 4
      S=TAIL(S)
      T=TAIL(T)
      IF (S.NE.0) GO TO 3
      IF (X.NE.0) GO TO 2
      MEQ=1
      RETURN
4     CALL PERASE(C)
      MEQ=0
      RETURN
      END


      SUBROUTINE MERASE(A)
      INTEGER A,B,C,K
      INTEGER COUNT
1     IF (A.EQ.0) RETURN
      K=COUNT(A)-1
      IF (K.EQ.0) GO TO 11
      CALL SCOUNT(K,A)
      RETURN
11    CALL DECAP(B,A)
2     K=COUNT(B)-1
      IF (K.EQ.0) GO TO 21
      CALL SCOUNT(K,B)
      GO TO 1
21    CALL DECAP(C,B)
      CALL PERASE(C)
      IF (B.NE.0) GO TO 2
      GO TO 1
      END
```

```
      INTEGER FUNCTION MGARN(Q,B,P,A,L)
      INTEGER A,B,C,D,E,L,P,Q,S,T,U,V,X,Y
      INTEGER CPGARN,INV,PFL
1     X=B
      Y=A
      C=0
2     CALL ADV(S,X)
      CALL ADV(T,Y)
      D=0
3     CALL ADV(U,S)
      CALL ADV(V,T)
      E=CPGARN(Q,U,P,V,L)
      D=PFL(E,D)
      IF (S.NE.0) GO TO 3
4     C=PFL(INV(D),C)
      IF (X.NE.0) GO TO 2
5     MGARN=INV(C)
      RETURN
      END


      INTEGER FUNCTION MINV(A)
      INTEGER A,C,I,J,K,L,M,P,R,T,U,V,W,X
      INTEGER BORROW,INV,LENGTH,MVLIST,PFA,PFL,PLES,TAIL
10    T=A
      C=0
      J=0
      P=PFA(1,0)
      M=LENGTH(T)
      L=MVLIST(T)
20    IF (L.EQ.0) GO TO 30
      CALL DECAP(V,L)
      P=PFL(V,PFL(P,PFA(0,0)))
      GO TO 20
30    CALL ADV(U,T)
      R=0
      J=J+1
40    CALL ADV(W,U)
      R=PFL(BORROW(W),R)
      IF (U.NE.0) GO TO 40
50    K=J-1
      IF (K.EQ.0) GO TO 52
      DO 51 I=1,K
51    R=PFL(0,R)
52    R=PFL(BORROW(P),R)
      K=M-J
      IF (K.EQ.0) GO TO 54
      DO 53 I=1,K
53    R=PFL(0,R)
54    C=PFL(INV(R),C)
      IF (T.NE.0) GO TO 30
      C=INV(C)
60    X=PLES(M,C)
      IF (X.EQ.0) GO TO 70
```

```
      T=TAIL(X)
      U=TAIL(T)
      CALL SSUCC(0,T)
      CALL ERASE(U)
70    MINV=X
      CALL PERASE(P)
      CALL MERASE(C)
      RETURN
      END


      INTEGER FUNCTION MMCB(A,B)
      INTEGER A,B,H,I,J,K,L,N,R,S,U,V,X,Y
      INTEGER FIRST,ICOMP,IPROD,ISUM,LENGTH,PFA,PFL,PNORMF,TAIL
10    X=A
      Y=B
      K=0
      N=LENGTH(FIRST(X))
      I=0
      DO 11 L=1,N
11    I=PFL(0,I)
      J=0
      DO 12 L=1,N
12    J=PFL(0,J)
20    CALL ADV(R,X)
      S=I
30    CALL ADV(U,R)
      U=PNORMF(U)
      V=FIRST(S)
      H=ICOMP(U,V)
      IF (H.NE.1) GO TO 31
      CALL ERLA(V)
      CALL ALTER(U,S)
      GO TO 32
31    CALL ERLA(U)
32    S=TAIL(S)
      IF (S.NE.0) GO TO 30
      IF (X.NE.0) GO TO 20
40    CALL ADV(R,Y)
      S=J
50    CALL ADV(U,R)
      U=PNORMF(U)
      V=FIRST(S)
      H=ICOMP(U,V)
      IF (H.NE.1) GO TO 51
      CALL ERLA(V)
      CALL ALTER(U,S)
      GO TO 52
51    CALL ERLA(U)
52    S=TAIL(S)
      IF (S.NE.0) GO TO 50
      IF (Y.NE.0) GO TO 40
60    CALL DECAP(U,I)
      CALL DECAP(V,J)
```

```
      R=IPROD(U,V)
      CALL ERLA(U)
      CALL ERLA(V)
      S=ISUM(K,R)
      CALL ERLA(K)
      CALL ERLA(R)
      K=S
      IF (I.NE.0) GO TO 60
      U=PFA(Z,J)
      V=IPROD(U,K)
      CALL ERLA(U)
      CALL ERLA(K)
      MMCB=V
      RETURN
      END


      INTEGER FUNCTION MMOD(P,A,S)
      INTEGER A,B,C,P,R,S,T,X
      INTEGER CPMOD,INV,PFA,PFL
1     X=A
      B=0
2     CALL ADV(T,X)
      C=0
3     CALL ADV(R,T)
      R=CPMOD(P,R)
      IF (S.EQ.0) C=PFA(R,C)
      IF (S.NE.0) C=PFL(R,C)
      IF (T.NE.0) GO TO 3
4     B=PFL(INV(C),B)
      IF (X.NE.0) GO TO 2
5     MMOD=INV(B)
      RETURN
      END


      INTEGER FUNCTION MMPY(AA,BB)
      INTEGER A,AA,ASTAR,BB,BP,BSTAR,C,CP,CSTAR,D,I,J,M,P,PRIME
      INTEGER Q,S,T,U,V,Z
      INTEGER ICOMP,IPROD,LENGTH,MCPMPY,MGARN,MMCB,MMOD,MTRAN,MVLIST
      INTEGER MZERO,PFA,PFL
      COMMON /TR4/PRIME
10    A=AA
      BP=MTRAN(BB)
      Z=PRIME
      M=LENGTH(A)
      Q=LENGTH(BP)
20    C=0
      DO 22 I=1,M
      D=0
      DO 21 J=1,Q
21    D=PFL(0,D)
22    C=PFL(D,C)
```

```
30    V=MVLIST(A)
      IF (MZERO(A).EQ.1 .OR. MZERO(BP).EQ.1) GO TO 80
      S=LENGTH(V)
      I=PFA(1,0)
      J=MMCB(A,BP)
40    IF (Z.NE.0) GO TO 42
      PRINT 41
41    FORMAT(48H LIST OF PRIMES EXHAUSTED. ALGORITHM MMPY FAILS. )
      STOP
42    CALL ADV(P,Z)
      ASTAR=MMOD(P,A,S)
      BSTAR=MMOD(P,BP,S)
50    CSTAR=MCPMPY(P,ASTAR,BSTAR)
      CALL MCPERS(ASTAR)
      CALL MCPERS(BSTAR)
60    CP=MGARN(I,C,P,CSTAR,V)
      CALL MERASE(C)
      CALL MCPERS(CSTAR)
      C=CP
70    T=PFA(P,0)
      U=IPROD(I,T)
      CALL ERLA(I)
      CALL ERLA(T)
      I=U
      IF (ICOMP(I,J).NE.1) GO TO 40
      CALL ERASE(V)
      CALL ERLA(I)
      CALL ERLA(J)
80    CALL MERASE(BP)
      MMPY=C
      RETURN
      END


      INTEGER FUNCTION MPROD(A,B)
      INTEGER A,B,C,D,E,F,G,R,S,T,V,W,X,Y
      INTEGER CINV,FIRST,INV,MTRAN,PFL,PPROD,PSUM,TAIL
1     X=CINV(A)
      Y=INV(MTRAN(B))
      C=0
      V=X
2     CALL ADV(R,V)
      W=Y
      D=0
3     S=R
      CALL ADV(T,W)
      E=0
4     F=PPROD(FIRST(S),FIRST(T))
      G=PSUM(E,F)
      CALL PERASE(E)
      CALL PERASE(F)
      E=G
      S=TAIL(S)
      T=TAIL(T)
```

```
      IF (S.NE.0) GO TO 4
5     D=PFL(E,D)
      IF (W.NE.0) GO TO 3
6     C=PFL(D,C)
      IF (V.NE.0) GO TO 2
7     CALL ERASE(X)
      CALL ERASE(Y)
      MPROD=C
      RETURN
      END


      INTEGER FUNCTION MSUM(A,B)
      INTEGER A,B,C,D,E,F,X,Y
      INTEGER FIRST,INV,PFL,PSUM,TAIL
1     X=A
      Y=B
      C=0
2     CALL ADV(E,X)
      CALL ADV(F,Y)
      D=0
3     D=PFL(PSUM(FIRST(E),FIRST(F)),D)
      E=TAIL(E)
      F=TAIL(F)
      IF (E.NE.0) GO TO 3
4     C=PFL(INV(D),C)
      IF (X.NE.0) GO TO 2
5     MSUM=INV(C)
      RETURN
      END


      INTEGER FUNCTION MTRAN(A)
      INTEGER A,B,C,D,F,G,I,N,S,T
      INTEGER BORROW,CINV,FIRST,LENGTH,PFA,PFL,TAIL,TYPE
1     G=CINV(A)
      F=G
      C=FIRST(F)
      T=TYPE(C)
      N=LENGTH(C)
      B=0
      DO 15 I=1,N
15    B=PFL(0,B)
2     CALL ADV(C,F)
      S=B
3     CALL ADV(D,C)
      IF (T.EQ.0) CALL ALTER(PFA(D,FIRST(S)),S)
      IF (T.NE.0) CALL ALTER(PFL(BORROW(D),FIRST(S)),S)
```

```
      S=TAIL(S)
      IF (S.NE.0) GO TO 3
      IF (F.NE.0) GO TO 2
4     CALL ERASE(G)
      MTRAN=B
      RETURN
      END


      INTEGER FUNCTION MVLIST(A)
      INTEGER A,B,C,X
      INTEGER PVLIST
1     X=A
2     CALL ADV(B,X)
3     CALL ADV(C,B)
      IF (C.NE.0) GO TO 4
      IF (B.NE.0) GO TO 3
      GO TO 2
4     MVLIST=PVLIST(0)
      RETURN
      END


      INTEGER FUNCTION MZCON(A)
      INTEGER A,B,S,U,X
      INTEGER INV,PFL,TAIL
1     B=0
      S=A
2     CALL ADV(U,S)
      X=0
3     X=PFL(U,X)
      U=TAIL(U)
      IF (U.NE.0) GO TO 3
      B=PFL(X,B)
      IF (S.NE.0) GO TO 2
4     MZCON=INV(B)
      RETURN
      END


      INTEGER FUNCTION MZERO(A)
      INTEGER A,X,Y
      INTEGER FIRST,TAIL
1     X=A
      MZERO=0
2     CALL ADV(Y,X)
3     IF (FIRST(Y).NE.0) RETURN
      Y=TAIL(Y)
      IF (Y.NE.0) GO TO 3
      IF (X.NE.0) GO TO 2
      MZERO=1
      RETURN
      END
```

```
      INTEGER FUNCTION NULCON(N,J,P,W)
      INTEGER E,H,I,J,K,L,N,P,R,S,T,U,W,WP,Z
      INTEGER BORROW,CINV,COUNT,INV,LENGTH,PFL
10    L=J
      WP=CINV(W)
      R=LENGTH(J)
      T=R-LENGTH(W)
      IF (T.EQ.0) GO TO 12
      DO 11 S=1,T
11    WP=PFL(0,WP)
12    WP=INV(WP)
      E=N-R
      Z=0
      CALL ADV(H,L)
      I=0
      K=0
20    I=I+1
      IF (I.GT.N) GO TO 50
      IF (I.EQ.H) GO TO 40
30    K=K+1
      U=0
      T=E-K
      IF (T.EQ.0) GO TO 32
      DO 31 S=1,T
31    U=PFL(0,U)
32    U=PFL(BORROW(P),U)
      T=K-1
      IF (T.EQ.0) GO TO 34
      DO 33 S=1,T
33    U=PFL(0,U)
34    Z=PFL(U,Z)
      GO TO 20
40    CALL DECAP(U,WP)
      IF (U.NE.0) CALL SCOUNT(COUNT(U)-1,U)
      U=INV(CINV(U))
      T=E-LENGTH(U)
      IF (T.EQ.0) GO TO 42
      DO 41 S=1,T
41    U=PFL(0,U)
42    Z=PFL(U,Z)
      IF (L.NE.0) CALL ADV(H,L)
      GO TO 20
50    NULCON=INV(Z)
      RETURN
      END


      INTEGER FUNCTION NULSP(AA)
      INTEGER A,AA,C,D,G,N,R,T,U,V,Y,Z
      INTEGER BORROW,FIRST,INV,LENGTH,PFL,PLES
      A=AA
10    T=A
      C=0
20    CALL ADV(U,T)
```

```
      R=0
30    CALL ADV(V,U)
      R=PFL(BCRROW(V),R)
      IF (U.NE.0) GO TO 30
      R=PFL(0,R)
      C=PFL(INV(R),C)
      IF (T.NE.0) GO TO 20
      C=INV(C)
40    N=LENGTH(FIRST(A))
      G=PLES(N,C)
      CALL DECAP(D,G)
      CALL DECAP(Y,G)
      CALL DECAP(Z,G)
      CALL PERASE(D)
      CALL MERASE(Y)
      CALL MERASE(C)
      NULSP=Z
      RETURN
      END


      INTEGER FUNCTION PDET(AA)
      INTEGER A,AA,ASTAR,D,DP,DSTAR,H,I,J,L,P,PRIME,S,T,U,V
      INTEGER CBDET,CPDET,CPGARN,ICOMP,IPROD,LENGTH,MMOD,MVLIST
      INTEGER MZERO,PFA
      COMMON /TR4/PRIME
      A=AA
10    D=0
      V=MVLIST(A)
      S=LENGTH(V)
      J=CBDET(A)
      I=PFA(1,0)
      L=PRIME
      H=0
20    IF (L.NE.0) GO TO 22
      PRINT 21
21    FORMAT(48H LIST OF PRIMES EXHAUSTED. ALGORITHM PDET FAILS.)
      STOP
22    CALL ADV(P,L)
      ASTAR=MMOD(P,A,S)
      IF (MZERO(ASTAR).EQ.0) GO TO 30
      CALL MCPERS(ASTAR)
      DSTAR=0
      GO TO 40
30    DSTAR=CPDET(P,ASTAR)
40    IF (DSTAR.NE.0) H=1
      IF (H.EQ.0) GO TO 50
      DP=CPGARN(I,D,P,DSTAR,V)
      CALL PERASE(D)
      IF (V.NE.0) CALL CPERAS(DSTAR)
      D=DP
50    T=PFA(P,0)
      U=IPROD(I,T)
      CALL ERLA(I)
```

```
      CALL ERLA(T)
      I=U
      IF (ICOMP(I,J).NE.1) GO TO 20
60    PDET=D
      CALL ERLA(I)
      CALL ERLA(J)
      CALL ERASE(V)
      RETURN
      END


      INTEGER FUNCTION PLES(NN,CC)
      INTEGER C,CC,CSTAR,D,DP,DSTAR,E,EP,G,H,I,ISTAR,J,JSTAR,K,L,N,NN
      INTEGER NP,P,PRIME,R,RSTAR,S,T,U,V,VP,VSTAR,W,WSTAR,X,Y,Z,ZP
      INTEGER CPGARN,CPRRE,FIRST,INV,IPROD,LENGTH,MEQ,MGARN,MMOD
      INTEGER MMPY,MVLIST,MZCON,MZERO,NULCON,PDIF,PFA,PFL,PNEG
      INTEGER TAIL,VCOMP
      COMMON /TR4/PRIME
      N=NN
      C=CC
10    X=MVLIST(C)
      S=LENGTH(X)
      L=PRIME
      R=0
20    IF (L.NE.0) GO TO 22
      PRINT 21
21    FORMAT(49H LIST OF PRIMES EXHAUSTED.  ALGORITHM PLES FAILS.)
      STOP
22    CALL ADV(P,L)
      CSTAR=MMOD(P,C,S)
      IF (MZERO(CSTAR).EQ.0) GO TO 30
      CALL MCPERS(CSTAR)
      GO TO 20
30    WSTAR=CPRRE(P,CSTAR)
      CALL DECAP(JSTAR,WSTAR)
      CALL DECAP(ISTAR,WSTAR)
      CALL DECAP(DSTAR,WSTAR)
      CALL DECAP(VSTAR,WSTAR)
40    RSTAR=LENGTH(JSTAR)
      IF (RSTAR-R) 50,41,60
41    IF (VCOMP(JSTAR,J)) 60,42,50
42    IF (VCOMP(ISTAR,I)) 60,43,50
43    CALL ERLA(JSTAR)
      CALL ERLA(ISTAR)
      GO TO 70
50    CALL ERLA(JSTAR)
      CALL ERLA(ISTAR)
      IF (X.NE.0) CALL CPERAS(DSTAR)
      IF (VSTAR.NE.0) CALL MCPERS(VSTAR)
      GO TO 20
60    IF (R.EQ.0) GO TO 61
      CALL ERLA(J)
      CALL ERLA(I)
      CALL PERASE(D)
```

```
      IF (V.NE.0) CALL MERASE(V)
      CALL ERLA(E)
61    R=RSTAR
      J=JSTAR
      I=ISTAR
      D=0
      V=0
      IF (VSTAR.NE.0) V=MZCON(VSTAR)
      E=PFA(1,0)
70    DP=CPGARN(E,D,P,DSTAR,X)
      IF (X.NE.0) CALL CPERAS(DSTAR)
      IF (V.EQ.0) GO TO 71
      VP=MGARN(E,V,P,VSTAR,X)
      CALL MCPERS(VSTAR)
71    T=PFA(P,0)
      EP=IPROD(T,E)
      CALL ERLA(T)
      CALL ERLA(E)
      E=EP
80    T=PDIF(D,DP)
      CALL PERASE(D)
      D=DP
      U=1
      IF (V.EQ.0) GO TO 81
      U=MEQ(V,VP)
      CALL MERASE(V)
      V=VP
81    IF (U.EQ.1 .AND. T.EQ.0) GO TO 90
      CALL PERASE(T)
      GO TO 20
90    IF (R.LE.N) GO TO 91
      G=0
      CALL PERASE(D)
      GO TO 120
91    NP=LENGTH(FIRST(C))
      DP=PNEG(D)
      ZP=NULCON(NP,J,DP,V)
      T=MMPY(C,ZP)
      U=MZERO(T)
      CALL PERASE(DP)
      CALL MERASE(T)
      IF (U.EQ.1) GO TO 100
      CALL MERASE(ZP)
      GO TO 20
100   J=INV(J)
      IF (FIRST(J).LE.N) GO TO 101
      CALL MERASE(ZP)
      G=0
      CALL PERASE(D)
      GO TO 120
101   Y=0
      Z=0
      H=N-R-1
      U=N
110   U=U-1
```

```
      CALL DECAP(T,ZP)
      IF (H.LT.0) GO TO 113
      Z=PFL(T,Z)
      IF (H.EQ.0) GO TO 112
      DO 111 K=1,H
111   T=TAIL(T)
112   W=T
      T=TAIL(T)
      CALL SSUCC(0,W)
113   Y=PFL(T,Y)
      IF(U.GT.0) GO TO 110
      CALL MERASE(ZP)
      G=PFL(D,PFL(INV(Y),PFL(INV(Z),0)))
120   CALL ERLA(J)
      CALL ERLA(I)
      IF (V.NE.0) CALL MERASE(V)
      CALL ERLA(E)
      CALL ERASE(X)
      PLES=G
      RETURN
      END


      INTEGER FUNCTION VCOMP(H,K)
      INTEGER H,K,S,T,U,V,W
1     S=H
      T=K
2     IF (S.NE.0) GO TO 3
      IF (T.NE.0) GO TO 5
      VCOMP=0
      RETURN
3     IF (T.NE.0) GO TO 4
      GO TO 6
4     CALL ADV(U,S)
      CALL ADV(V,T)
      W=U-V
      IF (W.EQ.0) GO TO 2
      IF (W.GT.0) GO TO 6
5     VCOMP=-1
      RETURN
6     VCOMP=1
      RETURN
      END
```